

IBM

**PowerPC 60x/7xx
Evaluation Board Kit
User's Manual**

92G8622

Third Edition (September 1997)

This edition of the *IBM PowerPC 60x/7xx Evaluation Kit User's Manual* applies to the IBM PowerPC 60x/7xx Evaluation Board Kit and to all subsequent versions of the 60x/7xx Evaluation Board Kit until otherwise indicated in new versions or technical newsletters.

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS MANUAL "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. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

IBM does not warrant that the contents of this publication or the accompanying source code examples, whether individually or as one or more groups, will meet your requirements or that the publication or the accompanying source code examples are error-free.

This publication could contain technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time.

It is possible that this publication may contain references to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only IBM's licensed program. You can use any functionally equivalent program instead.

No part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the written permission of IBM.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

Address comments about this publication to:

IBM Corporation
Department YM5A
P.O. Box 12195
Research Triangle Park, NC 27709

email: ppc400pubs@vnet.ibm.com

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1996, 1997. All rights reserved.

Printed in the United States of America.

4 3 2 1

Notice to U.S. Government Users—Documentation Related to Restricted Rights—Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

Patents and Trademarks

IBM may have patents or pending patent applications covering the subject matter in this publication. The furnishing of this publication does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, United States of America.

The following terms are trademarks of IBM Corporation:

PowerPC 60x/7xx
AIX
AIXwindows
IBM
OS Open
PowerPC
PowerPC Architecture
RISC System/6000
RISCWatch
RISCTrace

Other terms which are trademarks are the property of their respective owners.

Contents

About This Book	xix
Who Should Use This Book.....	xx
How This Book is Organized	xx
Contacting the IBM Embedded Systems Solution Center	xxiii
Related Publications.....	xxiii
Overview of the 60x/7xx EVB	1-1
Introducing the 60x/7xx EVB Hardware Components	1-1
60x/7xx Evaluation Board	1-1
Cables and Power Supply.....	1-1
Introducing the 60x/7xx EVB Software Support Package	1-1
ROM Monitor.....	1-2
RISCWatch Debugger	1-2
IBM High C/C++ Compiler	1-2
OS Open Real-Time Operating System.....	1-2
Dhrystone Benchmark Program.....	1-3
Application Tools.....	1-3
Host System Requirements	2-1
RS/6000 Host System Requirements.....	2-1
PC Host System Requirements.....	2-2
SUN Host System Requirements	2-3
Installing the EVB Software	3-1
RS/6000 Installation (ELF and XCOFF file formats).....	3-1
EVB Software Support Package Installation - RS/6000.....	3-1
RISCWatch Debugger Installation - RS/6000	3-4
PC Installation (ELF file format version only)	3-4
EVB Software Support Package Installation - PC.....	3-4
RISCWatch Debugger Installation - PC	3-7
Sun Installation (ELF file format version only)	3-7
EVB Software Support Package Installation - Sun	3-7
RISCWatch Debugger Installation - Sun.....	3-10
Host Configuration.....	4-1
RS/6000 Host Configuration.....	4-1
Serial Port Setup - RS/6000.....	4-1
Ethernet Setup - RS/6000.....	4-5
ROM Monitor-Debugger Communication Setup - RS/6000	4-7
PC Host Configuration.....	4-7

Serial Port Setup	4-8
Ethernet Setup - PC	4-10
Windows 3.1	4-11
Ethernet Setup - Windows 95.....	4-12
Ethernet Setup - Windows NT 3.51	4-13
ROM Monitor-Debugger Communication Setup - PC	4-13
Sun Host Configuration	4-13
Serial Port Setup - SUN	4-14
Ethernet Setup - SUN	4-14
ROM Monitor-Debugger Communication Setup - SUN.....	4-14
60x/7xx EVB Connectors	5-1
Connecting the 60x/7xx EVB Hardware	5-1
Using a Terminal Emulator	5-4
RS/6000 Terminal Emulation	5-4
PC Terminal Emulation	5-5
Windows 3.1 and Windows NT Terminal Emulation.....	5-5
Windows 95 Terminal Emulation	5-5
SUN Terminal Emulation.....	5-6
Bootting the PowerPC 60x/7xx on the EVB.....	5-7
60x/7xx EVB Hardware.....	6-1
CPU Clock Control Logic	6-2
60x/7xx EVB Memory Map	6-2
Network Address of the Ethernet Controller	6-4
60x/7xx EVB ROM Monitor	7-1
ROM Monitor Source Code	7-1
Communications Features.....	7-2
Bootp and tftp Configuration to support ROM Monitor Loads.....	7-2
RS/6000 bootp and tftp configuration.....	7-2
PC bootp and tftp configuration.....	7-4
Automatic startup for Windows 3.1 and Windows NT 3.51	7-5
Automatic startup for Windows 95.....	7-6
SUN bootp and tftp configuration	7-6
Accessing the ROM Monitor.....	7-8
ROM Monitor Operation	7-8
Monitor Selections and Submenus.....	7-9
Initial ROM Monitor Menu	7-9
Selecting Power-On Tests	7-11
Selecting Boot Devices	7-13
Changing IP Addresses	7-15
Using the Ping Test.....	7-17
Entering the Debugger	7-19

Disabling the Automatic Display	7-22
Displaying the Current Configuration	7-23
Saving the Current Configuration.....	7-24
Setting the Baud Rate for S1 Boots	7-25
S1 Boot	7-27
Exiting the Main Menu	7-29
ROM Monitor User Functions	7-31
Flash Update Utility	7-31
60x/7xx EVB Sample Applications	8-1
Overview.....	8-1
ROM Monitor Flash Image	8-2
Using the Software Samples	8-4
Building and Running the Dhrystone Benchmark	8-6
Building and Running the usr_samp Program	8-7
Building and Running the timesamp Program	8-8
Resolving Execution Problems.....	8-9
Using the Ping Test on the ROM Monitor to Verify Connectivity	8-9
bootp and tftp Servers (Daemons) for ROM Monitor loads.....	8-10
Using OS Open Functions.....	8-10
Application Libraries and Tools	9-1
OS Open Libraries.....	9-1
Using Libraries and Support Software	9-4
Serial Port Support Library.....	9-4
Boot Library(RAM)	9-4
Input/Output Support Library.....	9-4
PowerPC Low-Level Processor Access Support Library	9-5
ROM Monitor Ethernet IP Interface Library.....	9-5
Real-time Clock Interface Support Library	9-5
Diskette Block Device Driver Support Library	9-5
Ethernet Device Driver Support Library	9-5
Software Timer Tick Support Library	9-6
Device Drivers Supplied with the 60x/7xx EVB	9-7
Asynchronous Device Driver.....	9-7
Device Driver Installation.....	9-7
Device Installation	9-8
Opening Asynchronous Communication Ports.....	9-9
Reading and Writing.....	9-9
I/O Control.....	9-10
Polled Asynchronous I/O.....	9-12
Diskette Device Driver	9-12
Device Driver Installation.....	9-12

Device Installation	9-12
Opening Diskette Files	9-13
Reading and Writing	9-13
I/O Control	9-13
QDEVATTR	9-14
QDSKTATTR	9-14
BLKMEDIA_CHANGE	9-14
Diskette Device Driver Example	9-14
Ethernet Device Driver	9-16
Device Driver Installation	9-16
zapatos_enet_setup()	9-17
Device Installation	9-17
Opening and Closing Ethernet Files	9-18
Reading and Writing	9-18
I/O Control	9-19
ENET_SET_CHANNEL	9-19
ENET_CLEAR_CHANNEL	9-19
ENET_QUERY_ADDRESS	9-19
Ethernet Device Driver Example	9-20
Environment Bringup and Initialization	9-22
Board bootstrap	9-22
Environment Initialization	9-22
Tools	9-24
elf2rom and xcofrom	9-24
hbranch	9-26
eimgbld	9-28
nimgbld (XCOFF kits only)	9-29
Creating a loadable diskette (AIX)	9-30
Creating a loadable diskette (Sun OS)	9-30
Creating a loadable diskette (Solaris)	9-30
dsktimg (PC only)	9-31
60x/7xx EVB Function Reference	10-1
Attributes and Threads	10-1
Async Safe Functions	10-1
Cancel Safe Functions	10-2
Interrupt Handler Safe Functions	10-2
Callable from Application Thread Group Functions	10-2
Processors	10-2
60x/7xx EVB Functions	10-2
Program Trace Calls	A-1
Overview	A-1

MSGDATA Structure	A-1
Ptrace Definitions	A-4
RD_ATTACH (30)	A-5
Request data	A-5
Response data	A-5
RD_CONTINUE (7).....	A-6
Request data	A-6
Response data	A-6
RD_DETACH (31).....	A-7
Request data	A-7
Response data	A-7
RD_FILL (105)	A-8
Request data	A-8
Response data	A-8
RD_KILL (8)	A-9
Request data	A-9
Response data	A-9
RD_LDINFO (34)	A-10
Request data	A-10
Response data	A-10
RD_LOAD (101).....	A-12
Request data	A-12
Response data	A-12
RD_LOGIN (103)	A-13
Request data	A-13
Response data	A-13
RD_LOGOFF (104).....	A-14
Request data	A-14
Response data	A-14
RD_READ_D (2)	A-15
Request data	A-15
Response data	A-15
RD_READ_FPR (12)	A-16
Request data	A-16
Response data	A-16
RD_READ_GPR (11).....	A-17
Request data	A-17
Response data	A-17
RD_READ_GPR_MULT(71).....	A-18
Request data	A-18
Response data	A-18
RD_READ_I (1)	A-19
Request data	A-19

Response data	A-19
RD_READ_I_MULT (71)	A-20
Request data	A-20
Response data	A-20
RD_READ_SPR (115)	A-21
Request data	A-21
Response data	A-21
RD_READ_SR (118)	A-22
Request data	A-22
Response data	A-22
RD_STATUS (114)	A-23
Request data	A-23
Response data	A-23
RD_STOP_APPL (113)	A-24
Request data	A-24
Response data	A-24
RD_WAIT (108)	A-25
Request data	A-25
Response data	A-25
RD_WRITE_BLOCK (19)	A-26
Request data	A-26
Response data	A-26
RD_WRITE_D (5)	A-27
Request data	A-27
Response data	A-27
RD_WRITE_FPR (15)	A-28
Request data	A-28
Response data	A-28
RD_WRITE_GPR (14)	A-29
Request data	A-29
Response data	A-29
RD_WRITE_I (4)	A-30
Request data	A-30
Response data	A-30
RD_WRITE_SPR (112)	A-31
Request data	A-31
Response data	A-31
RD_WRITE_SR (119)	A-32
Request data	A-32
Response data	A-32
RL_LDINFO (181)	A-33
Request data	A-33
Response data	A-33

RL_LOAD_REQ(180)	A-34
Request data	A-34
Response data	A-34
ROM Monitor Load Format	B-1
Overview.....	B-1
Section Types.....	B-1
First Section	B-2
Text Section	B-3
Data Section	B-3
Symbol Section	B-3
Boot Header	B-3
Index	X-1

Figures

Figure 5-1. Serial Port Connection.....	5-1
Figure 5-2. Wiring in a Crossover Cable.....	5-2
Figure 5-3. Point-to-Point Ethernet Connection	5-3
Figure 5-4. Ethernet Connection with Hub	5-3
Figure 7-1. ROM Monitor Address Map	7-9
Figure 9-1. elf2rom and xcofrom Output File	9-25
Figure 9-2. Detail of patch file placement.....	9-27
Figure 9-3. hbranch Output Image	9-27

Tables

Table 9-1. OS Open Libraries	9-1
Table 9-2. OS Open Libraries for the 60x/7xx EVB	9-4
Table 9-3. ioctl() Commands for Asynchronous Device Drivers	9-10
Table 10-1. Functions Specific to 60x/7xx EVB	10-2
Table A-1. RD_ATTACH Request Table.....	A-5
Table A-2. RD_ATTACH Response Table.....	A-5
Table A-3. RD_CONTINUE Request Table	A-6
Table A-4. RD_CONTINUE Response Table	A-6
Table A-5. RD_DETACH Request Table	A-7
Table A-6. RD_DETACH Response Table	A-7
Table A-7. RD_FILL Request Table.....	A-8
Table A-8. RD_FILL Response Table	A-8
Table A-9. RD_KILL Request Table.....	A-9
Table A-10. RD_KILL Response Table.....	A-9
Table A-11. RD_LDINFO Request Table.....	A-10
Table A-12. RD_LDINFO Response Table	A-10
Table A-13. RD_LOAD Request Table	A-12
Table A-14. RD_LOAD Response Table	A-12
Table A-15. RD_LOGIN Request Table	A-13
Table A-16. RD_LOGIN Response Table	A-13
Table A-17. RD_LOGOFF Request Table	A-14
Table A-18. RD_LOGOFF Response Table	A-14
Table A-19. RD_READ_D Request Table	A-15
Table A-20. RD_READ_D Response Table.....	A-15
Table A-21. RD_READ_FPR Request Table.....	A-16
Table A-22. RD_READ_FPR Response Table	A-16
Table A-23. RD_READ_GPR Request Table	A-17
Table A-24. RD_READ_GPR Response Table	A-17

Table A-25. RD_READ_GPR_MULT Request Table	A-18
Table A-26. RD_READ_GPR_MULT Response Table.....	A-18
Table A-27. RD_READ_I Request Table	A-19
Table A-28. RD_READ_I Response Table	A-19
Table A-29. RD_READ_I_MULT Request Table	A-20
Table A-30. RD_READ_I_MULT Response Table	A-20
Table A-31. RD_READ_SPR Request Table.....	A-21
Table A-32. RD_READ_SPR Response Table.....	A-21
Table A-33. RD_READ_SR Request Table	A-22
Table A-34. RD_READ_SR Response Table	A-22
Table A-35. RD_STATUS Request Table	A-23
Table A-36. RD_STATUS Response Table	A-23
Table A-37. RD_STOP_APPL Request Table	A-24
Table A-38. RD_STOP_APPL Response Table	A-24
Table A-39. RD_WAIT Request Table	A-25
Table A-40. RD_WAIT Response Table	A-25
Table A-41. RD_WRITE_BLOCK Request Table	A-26
Table A-42. RD_WRITE_BLOCK Response Table	A-26
Table A-43. RD_WRITE_D Request Table	A-27
Table A-44. RD_WRITE_D Response Table	A-27
Table A-45. RD_WRITE_FPR Request Table	A-28
Table A-46. RD_WRITE_FPR Response Table	A-28
Table A-47. RD_WRITE_GPR Request Table.....	A-29
Table A-48. RD_WRITE_GPR Response Table.....	A-29
Table A-49. RD_WRITE_I Request Table	A-30
Table A-50. RD_WRITE_I Response Table.....	A-30
Table A-51. RD_WRITE_SPR Request Table	A-31
Table A-52. RD_WRITE_SPR Response Table	A-31
Table A-53. RD_WRITE_SR Request Table	A-32
Table A-54. RD_WRITE_SR Response Table.....	A-32

Table A-55. RL_LDINFO Request Table	A-33
Table A-56. RL_LDINFO Response Table.....	A-33
Table A-57. RL_LOAD_REQ Request Table	A-34
Table A-58. RL_LOAD_REQ Response Table	A-34

About This Book

This book contains the information you need to install and use the IBM® PowerPC™ 60x/7xx™ Evaluation Board (EVB), a hardware and software development tool for the PowerPC 603e™, 603ev™, 604™, 604e-v1™, 740™, and 750™ 32-bit RISC microprocessors.

Note: Although the reference design supports the 604 family, use of the 604 line of processors is not encouraged for future designs.

Connection of the 60x/7xx EVB to a host system is required for the exercises in this book. Supported host systems include:

- an IBM RISC System/6000™ workstation running AIX™ 3.2.5 (or higher)
- an IBM or compatible PC running one of the following
 - Windows 3.1 (or higher) and a TCP/IP package compliant with the Microsoft Windows Socket API definition
 - Windows 95
 - Windows NT 3.51
- a Sun SPARCstation 5, 10, or 20 workstation running Solaris 2.3 (or higher) or SunOS 4.1.3 (or higher)

The RISC System/6000 kit is available in both ELF and XCOFF file formats. The PC and Sun kits are available in ELF file format only.

The 60x/7xx EVB is based on the IBM PowerPC 604 SMP reference design platform that contains an interchangeable processor card with either a PowerPC 603e, 603ev, 604, 604e-v1, 740, or 750 processor. It also includes a chassis, power supply, 8MB DRAM, 512K L2 cache, IBM 27-82660 PowerPC to PCI Bridge and Memory Controller, SCSI controller, Ethernet controller, 512KB flash memory, PCI/ISA bridge, Business Audio, 2 serial ports, parallel port, floppy diskette drive, time-of-day clock with 4KB NV RAM, and mouse/keyboard controller. The reference design also includes technical specifications and schematics.

The 60x/7xx EVB software includes the ROM Monitor (resident in the flash memory on the board), ROM Monitor source code, IBM's OS Open real time operating system, sample application programs, application development libraries and tools, IBM's High C/C++ compiler, and IBM's RISCWatch, a source-level debugger that runs on the host.

Who Should Use This Book

This book is for hardware and software developers who need to evaluate the 60x/7xx microprocessor and use the debugging features of the 60x/7xx EVB to support software development.

Users should understand hardware and software development tools, concepts, and environments. Specifically, users should understand:

- the host's operating system
- the PowerPC Architecture™ and implementation-specific characteristics of the PowerPC 60x/7xx microprocessor
- C and assembler language programming

How This Book is Organized

This book contains the following chapters and appendixes:

- Chapter 1, "Overview of the 60x/7xx EVB," describes the product, its hardware and software components, and its relationship with the software tools on the host.
- Chapter 2, "Host System Requirements," lists the hardware and software requirements of the host system.
- Chapter 3, "Installing the EVB Software," describes the software installation on the host system.
- Chapter 4, "Host Configuration," describes the steps required to facilitate communications between the host computer and the 60x/7xx EVB.
- Chapter 5, "60x/7xx EVB Connectors," describes the EVB connectors and the procedures for connecting and configuring the 60x/7xx EVB hardware.
- Chapter 6, "60x/7xx EVB Hardware," describes the hardware components and their functions in terms of the overall organization of the 60x/7xx EVB.
- Chapter 7, "60x/7xx EVB ROM Monitor," describes the operations of the ROM monitor.
- Chapter 8, "60x/7xx EVB Sample Applications," contains sample applications to be built, loaded onto the EVB, and run.
- Chapter 9, "Application Libraries and Tools," describes the application libraries and host tools provided with the EVB software.
- Chapter 10, "60x/7xx EVB Function Reference," lists the OS Open functions for the 60x/7xx EVB platform. The function calls are arranged alphabetically by function name.
- Appendix A, "Program Trace Calls," describes the messages for interfacing a debugger on the host system to the ROM monitor on the 60x/7xx EVB.

- Appendix B, “ROM Monitor Load Format,” describes the load format requirements supported by the ROM monitor.

Conventions Used in This Book

This book follows the numeric and highlighting notation conventions based on those used in the RISC System/6000 and AIX publications.

Numeric Conventions

In general, numbers are used exactly as shown. Unless noted otherwise, all numbers are in decimal, and, if entered as part of a command, are entered without format information.

In text, binary numbers are preceded by a “B” followed by the number enclosed in single quotes, for example:

B'010'

In commands, binary numbers are preceded by “0b” or “b” followed by the number, which may be enclosed in single quotes, for example:

0b010 or b'010'

In text, hexadecimal numbers are preceded by an “X” followed by the number enclosed in single quotes, for example:

X'1A7'

In commands, hexadecimal numbers are preceded by “0x” or “x” followed by the number, which may be enclosed in single quotes, for example:

0x1a7 or x'1a7'

In text, the hexadecimal digits A through F appear in uppercase. In commands, these digits are typically entered in lowercase.

Highlighting Conventions

This book uses the following highlighting conventions:





- The names of invariant objects known to the software appear in bold type. In some text, however, such as in lists, no special typographic treatment is used. Examples of such objects include:
 - Function and macro names
 - Data types and structures
 - Constants and flags

Names of objects known to the software must be entered exactly as shown.

- Variable names supplied by user programs appear in italic type. In some text, however, such as in lists, no special typographic treatment is used. Examples of these objects include arguments and other parameters.
- No highlighting appears in code examples.

Syntax Diagram Conventions

Throughout this book, diagrams illustrate the syntax for string formats and commands. The following list shows how to read these diagrams:


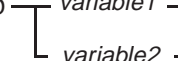

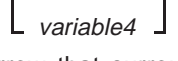
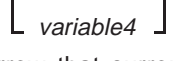

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.
- A  symbol begins a diagram.
- A  symbol indicates continuation of a diagram on the next line.
- A  symbol indicates continuation of a diagram from the previous line.
- A  symbol terminates a diagram.
- Keywords are in regular type, and variables are in italics. Keywords must be typed exactly as shown.
- Keywords or variables on the main path of a diagram are required.

 keyword — *variable1* — *variable2* 


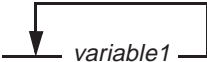

- Keywords or variables shown on branches below the main path are optional.

 keyword  *variable1*  *variable2* 

- Keywords or variables can appear in a stack, indicating that only one item in a stack can be chosen. If an item in a stack is on the main path, you must choose an item from the stack. If all items in a stack are below the main path, you may choose an item from the stack.
- For example, in the following syntax diagram, you must choose either *variable1* or *variable2*. However, because *variable3* and *variable4* are below the main path, neither is required.

 KEYWORD  *variable1*  *variable2*  *variable3*  *variable4* 

- A repeat separator is a returning arrow that surrounds a syntax element or group and shows that the element or group can be repeated.

 KEYWORD  *variable1* 

Contacting the IBM Embedded Systems Solution Center

For information about the 60x/7xx EVB Kit and the IBM family of hardware and software products for embedded system developers, call the IBM Embedded Systems Solution Center at (919) 254-1810.

Please send any comments regarding this document to the following Internet address:

ppc400pubs@vnet.ibm.com

Related Publications

Many of the following publications are included on the CD ROM that comes with the evaluation kit. The others are available from your IBM Microelectronics representative:

- **RISC System/6000 Publications**

IBM RISC System/6000: POWERstation and POWERserver Hardware Technical Information General Architectures, SA23-2643

- **AIX Publications**

This book refers to the following AIX publications. The words "IBM AIX Version 3.2 for RISC System/6000" are actually part of the title of each book; however, in all references to these books, those words are omitted.

Assembler Language Reference, SC23-2197

Commands Reference, Volume 1, SC23-2376

Commands Reference, Volume 2, SC23-2366

Commands Reference, Volume 3, SC23-2367

Commands Reference, Volume 4, SC23-2393

Editing Concepts and Procedures, GC23-2212

- **Embedded Application Binary Interface (EABI) Publications**

PowerPC Embedded Application Binary Interface (EABI)

System V Application Binary Interface, Third Edition, 0-13-0100439-5

System V Application Binary Interface, PowerPC Processor Supplement

- **IBM High C/C++ Publications**

The following list includes the books in the IBM High C/C++ library:

IBM High C/C++ Programmer's Guide for PowerPC, 92G6920

IBM High C/C++ Language Reference for PowerPC, 92G6923

IBM ELF Assembler User's Guide for PowerPC, 92G6921

IBM ELF Linker User's Guide for PowerPC, 92G6922

- **OS Open Publications**

The following list includes the books in the OS Open library:

IBM OS Open Programmer's Reference, Volume 1, 92G6911

IBM OS Open Programmer's Reference, Volume 2, 92G6912

IBM OS Open User's Guide, 92G6897

- **RISCWatch Debugger Publications**

RISCWatch Debugger User's Guide, 13H6964

- **PowerPC 6xx User's Manuals**

PowerPC 602 RISC Microprocessor User's Manual, MPR602UMU-01

PowerPC 603 RISC Microprocessor User's Manual, MPR603UMU-01

PowerPC 603e RISC Microprocessor User's Manual, MPR603EUM-01

PowerPC 604 RISC Microprocessor User's Manual, MPR604UMU-01

- **Evaluation Platform Publications**

PowerPC 604 SMP Reference Design Technical Specification, Release 3.0,
MPRZAPRSU-03

Overview of the 60x/7xx EVB

This chapter introduces the hardware and software in the 60x/7xx EVB kit.

1.1 Introducing the 60x/7xx EVB Hardware Components

The 60x/7xx EVB kit contains the Power PC 604 SMP reference design platform, line cord, serial port and Ethernet cables.

1.1.1 60x/7xx Evaluation Board

For a detailed description of the PowerPC 60x/7xx evaluation board, refer to the PowerPC 604 SMP Reference Design Technical Specification.

1.1.2 Cables and Power Supply

The 60x/7xx EVB kit includes a serial port interface cable for connecting the EVB serial port 1 to a terminal or terminal emulator running on the host.

Note: The Sun version of the EVB kit contains a male-to-male adapter to support connectivity between serial port 1 on the EVB and a serial port on the host.

An Ethernet crossover cable is provided in the kit to support direct Ethernet communication with the host system. A standard 10BaseT Ethernet connector is provided on the EVB. The Ethernet crossover cable is for direct connection to a single host and cannot be used with a hub or a building's Ethernet network.

A power supply line cord is also provided with the 60x/7xx EVB kit.

1.2 Introducing the 60x/7xx EVB Software Support Package

The 60x/7xx EVB software support package consists of the ROM Monitor, ROM Monitor source code, the RISCWatch source level debugger for ROM Monitor and OS Open debug modes, the IBM OS Open real time operating system, several sample programs (including the Dhrystone benchmark program), and application development libraries and tools. In the ELF file format kits, the IBM High C/C++ compiler is also included.

1.2.1 ROM Monitor

The ROM Monitor program for the 60x/7xx EVB is supplied in the 512KB flash memory module on the board. This code initializes the 60x/7xx processor, the memory controller, the PCI bridge, and the controllers for serial and Ethernet communications. By supporting communications with the host computer system, the ROM Monitor provides the means to load applications from the host onto the EVB and to debug them with the RISCWatch source level debugger.

The ROM Monitor is accessed through a terminal (or terminal emulator) attached to serial port 1 on the EVB. The RISCWatch debugger, when in ROM Monitor mode, runs on the host system, communicating with the ROM Monitor through serial port 2 or the Ethernet interface on the 60x/7xx EVB.

The ROM Monitor source code is provided primarily for customers interested in developing their own ROM versions. It is also provided so that debuggers other than RISCWatch may be integrated with the 60x/7xx EVB. Appendix A describes the trace calls that support communication between the RISCWatch debugger on the host and the ROM Monitor running on the 60x/7xx EVB.

1.2.2 RISCWatch Debugger

The RISCWatch source level debugger provides a window-based debugging environment for application programs running on the 60x/7xx EVB. The debugger can be used to load and execute application programs on the evaluation board. Debugger installation and usage for ROM Monitor and OS Open (non-JTAG) targets are addressed in the *RISCWatch Debugger User's Guide* included in the EVB kit. A sample debug session is included with the debugger.

1.2.3 IBM High C/C++ Compiler

The IBM High C/C++ compiler is a globally optimizing compiler developed for the PowerPC family of processors. It produces executable code in Extended Link Format(ELF) file format. The version included in the software support package is a limited capacity version created specifically for the 60x/7xx EVB kit. It supports the compilation, assembly, and linkage of the sample application programs and the ROM Monitor source code. A full featured version of the IBM High C/C++ compiler is available from IBM. For more information call the PowerPC Embedded Systems Solutions Center at (919)254-1810.

1.2.4 OS Open Real-Time Operating System

OS Open is a real-time operating system (RTOS) available for the PowerPC 400 Series, 60x and 7xx processors. OS Open is designed to take full advantage of the power of the IBM PowerPC RISC processors. Also, because the OS Open environment is built in a scalable fashion, it can be configured to meet the functional requirements and memory constraints of a wide variety of embedded systems.

OS Open features:

- Hard real-time support, including deterministic execution, priority inheritance protocols, and priority ceiling protocols
- Board support packages for plug-and-play operation of popular board-level products
- Support for existing American National Standards Institute (ANSI) C and emerging POSIX standards
- Open network interfaces to support embedded systems in heterogeneous environments
- Scalable implementations to meet the requirements and constraints of a variety of embedded systems

The version of OS Open included in the EVB software contains a limited function kernel that limits the number of threads that can be in existence at one time. Additional details can be found in the readme file following software installation. A full function OS Open kernel is available from IBM. Contact the IBM Embedded Systems Solutions Center at (919)254-1810 for additional information.

1.2.5 Dhrystone Benchmark Program

The Dhrystone benchmark is a commonly available integer benchmark. It is included as an example program to be built, loaded onto the evaluation board, and executed. The results of this benchmark may vary based on compiler options and the system environment in which it is run.

1.2.6 Application Tools

Several host-based tools are provided to support ROM and application development on the 60x/7xx EVB.

Host System Requirements

This chapter describes the hardware and software requirements of the host system to which the 60x/7xx EVB is to be connected. Supported host systems include:

- an IBM RS/6000 workstation running AIX 3.2.5 (or higher)
- an IBM or compatible PC running one of the following
 - Windows 3.1 (or higher) and a TCP/IP package compliant with the Microsoft Windows Socket API definition
 - Windows 95
 - Windows NT 3.51
- a Sun SPARCstation 5, 10, or 20 workstation running Solaris 2.3 (or higher) or SunOS 4.1.3 (or higher)

1.1 RS/6000 Host System Requirements

Hardware requirements of the host RS/6000 computer include:

- Approximately 25MB of free disk space. This space is required for the IBM High C/C++ compiler, the 60x/7xx EVB Software Support Package, and the RISCWatch debugger. When planning disk space usage, consider disk space requirements for AIX and any other software packages.
- Two available serial ports, one for terminal emulation and the other for host-to-EVB communications. Only one serial port is required if an Ethernet adapter is available for host-to-EVB communications. For better performance, an Ethernet connection is strongly recommended. Most RS/6000 computers come equipped with two serial ports and an Ethernet adapter. Please consult your RS/6000 literature for more details.
- A graphics display (IBM 6091 or similar), to display debugger screens

The following software must be installed on the host RS/6000 computer to run the debugger that communicates with the ROM Monitor on the 60x/7xx EVB:

- RISCWatch 3.3 or higher (for 7xx processors, RISCWatch 4.0 or higher)
- AIX Version 3.2.5 or higher
- AIX/Windows™ with X11R5 and Motif 1.2

AIX tools used to develop OS Open applications include:

XCOFF

- XL C or CSet++ compiler, for C and C++ programs
- as-emb, assembler for PowerPC assembler language programs
- nimgbld, binary image build tool
- AIX linker/binder, to build OS Open applications for a target system

ELF

- High C/C++ compiler for C programs
- asppc assembler for assembler and C language programs
- eimgbld, binary image build tool
- ELF linker/binder, to build OS Open applications for a target system

IBM and other vendors provide numerous optional software development tools for AIX, including tools for:

- Computer-aided software engineering (CASE)
- Structured analysis and design
- Program understanding
- Code management and version control

1.2 PC Host System Requirements

Hardware requirements of the host PC include:

- IBM or compatible system unit. Minimum requirements: x486 DX2 50/66 MHz with 8 MB of RAM
- VGA/SVGA Display Monitor. Minimum required: VGA 640x480. Recommended: SVGA 1024x768
- Approximately 25MB of free disk space. This space is required for the IBM High C/C++ compiler, the 60x/7xx EVB Software Support Package, and the RISCWatch debugger. When planning disk space usage, consider disk space requirements for Windows and any other software packages.
- Two available serial ports, one for terminal emulation and the other for SLIP host-to-EVB communications. Since PC hardware varies greatly, you should consult your PC literature to determine the number of serial ports available. Only one serial port is required if an Ethernet adapter is available for host-to-EVB communications. For better performance, an Ethernet connection is strongly recommended. Establishing an Ethernet host-to-EVB connection will most likely require the installation of an Ethernet adapter card and some additional connectivity hardware since most PCs do not come equipped for Ethernet communications. That hardware might include any or all of the following:

- For 10BaseT, an Ethernet 10BaseT network transceiver, a twisted pair cable, and a hub. At a minimum, a 10BaseT point-to-point connection will require the Ethernet crossover cable supplied with the EVB kit.

The following software must be installed on the host PC to run the debugger that communicates with the ROM Monitor on the 60x/7xx EVB:

- RISCWatch 3.3 or higher (for 7xx processors, RISCWatch 4.0 or higher)
- Windows 3.1 or higher, Windows 95, or Windows NT 3.51

Windows 3.1 users require a TCP/IP package compliant with the Microsoft Windows Socket API definition. One such compatible TCP/IP package is Trumpet Winsock, a TCP/IP protocol stack available from the www.trumpet.com Internet site. Windows 95 users who want to establish a SLIP host-to-EVB connection over a second serial port, require Trumpet Winsock as well, since the TCP/IP package that comes with Windows 95 does not support SLIP communications. Appropriate installation documentation can be found at the Trumpet site. Users should refer to the documentation for the terms and conditions of using Trumpet Winsock. Information regarding the setup and use of Trumpet Winsock can be found in the subsequent chapter on “Host Configuration”.

Note: Trumpet is not recommended for Windows 95 users already connected to a network since installing Trumpet may cause problems with previously defined networks. If the recommended Ethernet host-to-EVB connection is going to be used (instead of the SLIP host-to-EVB connection), Windows 95 users do **not** need to install Trumpet since the TCP/IP package that comes with Windows 95 can be used to establish the Ethernet connection.

1.3 SUN Host System Requirements

Hardware requirements of the host Sun workstation include:

- Approximately 25MB of free disk space. This space is required for the IBM High C/C++ compiler, the 60x/7xx EVB Software Support Package, and the RISCWatch debugger. When planning disk space usage, consider disk space requirements for the operating system and any other software packages.
- An available serial port for terminal emulation and an Ethernet (Attachment Unit Interface (AUI) or RJ-45) port for host-to-EVB communications. Most Sun SPARCstations come equipped with one serial port and an Ethernet (AUI) port. Consult your Sun literature for additional details.
- Any or all of the following hardware to establish an Ethernet connection between the EVB and the host.

- For 10BaseT, an Ethernet 10BaseT network transceiver, a twisted pair cable, and a hub. At a minimum, a 10BaseT point-to-point connection will require the Ethernet crossover cable supplied with the EVB kit.
- Consult your hardware documentation for additional information.
- A graphics display to display debugger screens

The following software must be installed on the Sun workstation to run the debugger that communicates with the ROM Monitor on the EVB:

- RISCWatch 3.3 or higher (for 7xx processors, RISCWatch 4.0 or higher)
- SunOS 4.1.3 (or higher) or Solaris 2.3 (or higher)
- OpenWindows 3.0 (SunOS 4.1.3) or 3.3 (Solaris 2.3)

Installing the EVB Software

This chapter describes the procedures for installing the EVB software on the host system. Details of the software, its directories and their contents, are also given. Please refer to the section corresponding to your host system.

2.1 RS/6000 Installation (ELF and XCOFF file formats)

2.1.1 EVB Software Support Package Installation - RS/6000

The software support package is installed from diskettes on an AIX host system using the system management interface tool (**smit**).

Before beginning the installation, you must have:

- **EVB for RS/6000** installation diskettes
- RISC System/6000, running AIX Version 3.2.5 or higher
- Superuser privileges on the AIX system

The method used to perform Steps 7 through 20 of the installation procedure depends on your version of **smit**. To select options, use the appropriate method for your version.

- In the X Window version, position the cursor and make selections using the mouse.
- In the character-based version, position the cursor using arrow keys and make selections using function keys.

The following procedure installs the EVB software support package.

1. Log in as **root** or use the AIX **su** command to become the superuser.
2. Use a **cd** command to change to the directory where the install image file will be stored.

Typically, the directory **/usr/sys/inst.images** holds install image files. However, any directory can be used.

3. Insert the EVB installation diskette labeled “1 of *n*” (*n* may vary) into the diskette drive.
4. Run the following **restore** command to read the file **EVB.instal.Z** from the diskette into the working directory.

```
restore -f/dev/rfd0
```

5. Insert the rest of the EVB installation diskettes into the diskette drive when prompted.

6. After the diskettes are read, unpack the file.

uncompress EVB.instal.Z

7. Run the following command to begin the installation via **smit**.

smit install_latest

8. Type the fully qualified path name of the file **EVB.instal** into the **Input device/directory for software** field.

The path includes the directory selected in Step 2, for example,
/usr/sys/inst.images/EVB.instal.

9. Press **Enter**.

10. Position the cursor on the **Software to install** line.

11. Select the **list** button (X Window version) or the **F4=List** function key (character-based version) to display a list of available software.

12. From the list, select the item or items appropriate for your platform and application.

- To install the IBM High C/C++ Compiler, select the highc base item (ELF file format version only).
- To install the complete OS Open distribution, select both the OS Open base and the OS Open platform specific items.

13. Select **OK** to complete the selection process and return to the **Install Software Products at Latest Available Level** window.

14. Ensure that the response for **Automatically install PREREQUISITE software** is "no".

For systems running AIX 4 or later, this field is called **AUTOMATICALLY install requisite software**.

15. Ensure that the response for **OVERWRITE existing version** is "yes".

For systems running AIX 4 or later, this field is called **OVERWRITE same or newer versions**.

16. Ensure that the response for **COMMIT Software** is "yes".

For systems running AIX 4 or later, this field is called **COMMIT software updates**.

17. Begin the installation by selecting **Do** or **OK**.

18. Select **OK** at the **ARE YOU SURE?** screen to continue the installation.

19. When the Command status is **OK**, file installation is complete.

20. Exit **smit**.

The IBM High C/C++ Compiler is installed in the **/usr/highcppc** directory tree and the EVB software support package in the **/usr/osopen** directory tree. It may be necessary to change ownership of these directories, their subdirectories and their contents if other users will require access to them. The **/usr/highcppc/bin** directory contains the files required for the IBM High C/C++ Compiler. Those files include:

- asppc - Assembler for assembler language programs
- ldppc - ELF linker/binder to build applications to be run on the EVB
- hcppc - High C/C++ compiler for C programs
- arppc - ELF library archiver

The **readme** file under the **/usr/highcppc** directory contains the latest information regarding the compiler and should be considered “must reading”.

If you installed the compiler into a directory other than **/usr/highcppc**, edit the **bin/hcppc.cnf** file, and locate the line near the top of the file that reads **HCDIR=/usr/highcppc**. Change this to reflect the directory that the compiler was installed into. Save your changes and exit the editor.

The **/usr/osopen** directory tree contains the files and tools that support OS Open application and ROM development. The **/usr/osopen** subdirectories and their contents are as follows.

- **/bin**

This directory contains several host based utilities used for application and ROM program development.

- elf2rom - creates a ROM image from an ELF executable file
- eimgbld - creates a ROM Monitor loadable image from an ELF executable file
- hbranch - places an absolute branch in the last address of a ROM image
- nimgbld - creates a ROM Monitor loadable image from an Xcoff executable file.
- rambuild - creates an assembler source file that contains the files found in a specified directory
- tracefmt - post-processes OS Open trace snapshots for AIX 3.2.X
- trc41 - post-processes OS Open trace snapshots for AIX 4.1

- **/examples**

This directory contains many example OS Open programs.

- **/PLATFORM**

This directory contains the OS Open platform specific code for the platform included in your EVB kit. The directory is not literally named “PLATFORM”, but rather is named to identify the board and processor that was shipped with your kit. For example, if your platform was the 60x evaluation board, this directory might be named mzap60x.

- README.TXT - contains the latest information regarding this release

- /include - contains OS Open include files
- /ld - contains dynamically loadable modules that can be run from OS Open's OpenShell
- /lib - contains OS Open libraries
- /m4 - contains assembler preprocessor include files
- /openbios - contains the source code for the ROM Monitor (detailed in a later chapter)
- /samples - contains samples programs that can be compiled and run

Considerable effort goes into providing a quality product with consistent documentation. To insure that our customers have the advantage of the latest software features and updated information, README.TXT may contain clarifications and/or additional information and should be considered "must reading".

- **/COMMENT.USER and COMMENT.DOC**

Please take the time to complete these user comment forms. Your feedback and suggestions will help us to improve our products and technical publications. Fax and email instructions are included in each of the files.

2.1.2 RISCWatch Debugger Installation - RS/6000

Please refer to the *RISCWatch Debugger User's Guide* for debugger installation instructions. Be sure to follow the instructions for RS/6000 installation.

2.2 PC Installation (ELF file format version only)

2.2.1 EVB Software Support Package Installation - PC

Before beginning the installation, you must have:

- **EVB for PC** installation diskettes
- PC running Windows 3.1 or higher, Windows 95, or Windows NT 3.51

The following procedure installs the EVB software support package:

NOTE: For Windows NT users, we recommend that you logon as "root".

1. Insert the installation diskette labeled "EVB - PC" and "1 of *n*" (*n* may vary) into diskette drive A:
2. Start Microsoft Windows if it is not active
3. Select Run... from the File pull-down of Program Manager or from the Start menu for Win95/NT
4. Type 'A:INSTALL' to run the installation program
5. Follow the installation program instructions

Once completed, the IBM High C/C++ Compiler is installed in the **\highcppc** directory tree and the EVB software support package in the **\osopen** directory tree. The **\highcppc\bin** directory contains the files required for the IBM High C/C++ Compiler. Those files include:

- **asppc.exe** - Assembler for assembler language programs
- **ldppc.exe** - ELF linker/binder to build applications to be run on the EVB
- **hcpc.exe** - High C/C++ compiler for C programs
- **arppc.exe** - ELF library archiver

The **readme** file under the **\highcppc** directory contains the latest information regarding the compiler and should be considered “must reading”.

The **\osopen** directory tree contains the files and tools that support OS Open application and ROM development. The **\osopen** subdirectories and their contents are as follows.

- **\bin**

This directory contains several host based utilities used for application and ROM program development.

- **elf2rom.exe** - creates a ROM image from an ELF file
- **eimgbld.exe** - creates a ROM Monitor loadable image from an ELF executable file
- **hbranch.exe** - places an absolute branch in the last address of a ROM image
- **rambuild.exe** - creates an assembler source file that contains the files found in a specified directory
- **make.exe** - supports the use of makefiles when building application programs
- **bootpd.exe** - bootp server to support ROM Monitor downloads
- **tftpd.exe** - tftp server to support host-to-EVB file transfers

- **\examples**

This directory contains many example OS Open programs.

- **\PLATFORM**

This directory contains the OS Open platform specific code for the platform included in your EVB kit. The directory is not literally named “PLATFORM”, but rather is named to identify the board and processor that was shipped with your kit. For example, if your platform was the 60x evaluation board, this directory might be named **mzap60x**.

- **README.TXT** - contains the latest information regarding this release
- **\include** - contains OS Open include files
- **\ld** - contains dynamically loadable modules that can be run from OS Open's OpenShell
- **\lib** - contains OS Open libraries
- **\m4** - contains assembler preprocessor include files

- \openbios - contains the source code for the ROM Monitor (detailed in a later chapter)
- \samples - contains sample programs that can be compiled and run

Considerable effort goes into providing a quality product with consistent documentation. To insure that our customers have the advantage of the latest software features and updated information, README.TXT may contain clarifications and/or additional information and should be considered “must reading”.

- **\COMMENT.USER and \COMMENT.DOC**

Please take the time to complete these user comment forms. Your feedback and suggestions will help us to improve our products and technical publications. Fax and email instructions are included in each of the files.

2.2.2 RISCWatch Debugger Installation - PC

Please refer to the *RISCWatch Debugger User's Guide* for debugger installation instructions. Be sure to follow the instructions for PC installation.

2.3 Sun Installation (ELF file format version only)

2.3.1 EVB Software Support Package Installation - Sun

The software support package is installed from diskettes on a Sun host system using the **cpio** and **tar** commands.

Before beginning the installation, you must have:

- **EVB for Sun** installation diskettes
- a Sun SPARCstation 5, 10, or 20 workstation running SunOS 4.1.3 (or higher) or Solaris 2.3 (or higher)
- Superuser privileges on the Sun system

The procedures required for installing the EVB software support package vary depending on the operating system being used. Please follow the instructions corresponding to your operating system.

1. Log in as **root** or use the **su** command to become the superuser
2. Open at least two windows for this procedure
3. Use the **cd** command to change to the **/usr** directory
4. Insert the installation diskette labeled “EVB - Sun” and “1 of *n*” (*n* may vary) into the diskette drive.

Instructions for SunOS 4.1.3 (or higher) only:

5. From the second window run the command:

```
cpio -ivB EVB_os4.tar.Z EVB.tar.Z EVB_hcppc.tar.Z < /dev/rfd0
```

where **/dev/rfd0** is the name of your diskette device.

6. When the system prompts you for a new volume, move to the first window and type **eject** to eject the diskette. Insert the next diskette.
7. Move to the second window and type the name of the diskette drive (**/dev/rfd0**) to continue the process.
8. If prompted for more diskettes, repeat the previous two steps. When finished, type **eject** to remove the final diskette.
9. Return to the first window and verify that the following files are installed under the **/usr** directory:

EVB.tar.Z

EVB_os4.tar.Z

EVB_hcppc.tar.Z

10. Run the following commands to unpack and install the files (**order is important**):

```
zcat EVB.tar.Z | tar xvf -
```

```
zcat EVB_os4.tar.Z | tar xvf -
```

```
zcat EVB_hcppc.tar.Z | tar xvf -
```

Installation for SunOS is complete. The tar.Z files may be removed to recover space.

Instructions for Solaris 2.3 (or higher) only:

11. From the first window type **volcheck**. This creates a file called **/vol/dev/rdiskette0/unlabeled** (the diskette device name).

If the system pops up a message box saying the diskette format is unrecognized, ignore the message and cancel the message box. The name of the file created may be different on your system. You can use the **eject -q** command to see the actual name. The file name returned is the name that should be used in the subsequent steps.

12. From the second window run the command:

```
cpio -ivB EVB.tar.Z EVB_hcppc.tar.Z < /vol/dev/rdiskette0/unlabeled
```

where **/voldev/rdiskette0/unlabeled** is the name of your diskette device.

13. When the system prompts you for a new volume, move to the first window. Type **eject** if the system did not automatically eject the diskette. Insert the next diskette and type **volcheck**.
14. Move to the second window and type the name of the diskette drive (**/vol/dev/rdiskette0/unlabeled**) to continue the process.
15. If prompted for more diskettes, repeat the previous two steps. When finished, type **eject** to remove the final diskette.
16. Return to the first window and verify that the following files are installed under the **/usr** directory.

EVB.tar.Z

EVB_hcppc.tar.Z

17. Run the following commands to unpack and install the files.

zcat EVB.tar.Z | tar xvf -

zcat EVB_hcppc.tar.Z | tar xvf -

Installation for Solaris is complete. The tar.Z files may be removed to recover space.

The IBM High C/C++ Compiler is installed in the **/usr/highcppc** directory tree and the EVB software support package in the **/usr/osopen** directory tree. It may be necessary to change ownership of these directories, their subdirectories and their contents if other users will require access to them. The **/usr/highcppc/bin** directory contains the files required for the IBM High C/C++ Compiler. Those files include:

- **asppc** - Assembler for assembler language programs
- **ldppc** - ELF linker/binder to build applications to be run on the EVB
- **hcppc** - High C/C++ compiler for C programs
- **arppc** - ELF library archiver

The **readme** file under the **/usr/highcppc** directory contains the latest information regarding the compiler and should be considered “must reading”.

If you installed the compiler into a directory other than **/usr/highcppc**, edit the **bin/hcppc.cnf** file, and locate the line near the top of the file that reads **HCDIR=/usr/highcppc**. Change this to reflect the directory that the compiler was installed into. Save your changes and exit the editor.

The **/usr/osopen** directory tree contains the files and tools that support OS Open application and ROM development. The **/usr/osopen** subdirectories and their contents are as follows.

- **/bin**

This directory contains several host based utilities used for application and ROM program

development.

- elf2rom - creates a ROM image from an ELF file
- eimgbld - creates a ROM Monitor loadable image from an ELF executable file
- hbranch - places an absolute branch in the last address of a ROM image
- rambuild - creates an assembler source file that contains the files found in a specified directory
- bootpd - bootp server to support ROM Monitor downloads

- **/examples**

This directory contains many example OS Open programs.

- **/PLATFORM**

This directory contains the OS Open platform specific code for the platform included in your EVB kit. The directory is not literally named "PLATFORM", but rather is named to identify the board and processor that was shipped with your kit. For example, if your platform was the 60x evaluation board, this directory might be named mzap60x.

- README.TXT - contains the latest information regarding this release
- /include - contains OS Open include files
- /ld - contains dynamically loadable modules that can be run from OS Open's OpenShell
- /lib - contains OS Open libraries
- /m4 - contains assembler preprocessor include files
- /openbios - contains the source code for the ROM Monitor (detailed in a later chapter)
- /samples - contains samples programs that can be compiled and run

Considerable effort goes into providing a quality product with consistent documentation. To insure that our customers have the advantage of the latest software features and updated information, README.TXT may contain clarifications and/or additional information and should be considered "must reading".

- **/COMMENT.USER and /COMMENT.DOC**

Please take the time to complete these user comment forms. Your feedback and suggestions will help us to improve our products and technical publications. Fax and email instructions are included in each of the files.

2.3.2 RISCWatch Debugger Installation - Sun

Please refer to the *RISCWatch Debugger User's Guide* for debugger installation instructions. Be sure to follow the instructions for Sun installation.

Host Configuration

Several host configuration steps are required to facilitate communications between the host computer and the evaluation board. These steps are outlined in this chapter. Please refer to the section corresponding to your host system.

3.1 RS/6000 Host Configuration

RS/6000 configuration requires that you be the superuser of the host workstation. This is accomplished by logging in as **root** or by using the AIX **su** command to become the superuser.

3.1.1 Serial Port Setup - RS/6000

The RS/6000 includes two serial ports to support communications via asynchronous data transfer. These ports are labeled S1 and S2 on the back of the RS/6000's system unit. When properly configured, one serial port can be used to connect a terminal emulator running on the host to the ROM Monitor running on the EVB, and the other to provide a **Serial Line Internet Protocol** (or **SLIP**) network interface between the host and the EVB to download applications. This section addresses the proper configuration of the S1 and S2 serial ports to support these connections. Details on setting up the terminal emulator are discussed in a later chapter. In this section, S1 and S2 refer to the respective serial ports on the host RS/6000, and SP1 and SP2 (labeled J26 and J27 on the board) to the respective serial ports on the EVB.

The connection of the terminal emulator running on the host to the ROM Monitor running on the EVB, is made through the S1 serial port on the RS/6000 and the SP1 serial port on the EVB. A connection between the S2 serial port on the host and the SP2 serial port on the EVB, provides a SLIP network interface to download application programs from the host to the EVB. If the recommended Ethernet connection is going to be used, the S2-to-SP2 SLIP connection is optional and does not need to be established.

Proper setup involves the configuration of **tty** devices for both the S1 and S2 serial ports on the host. **tty0** is used for the terminal emulator-to-ROM Monitor connection and **tty1** for the host-to-EVB SLIP connection. It is also necessary to establish a SLIP network interface between S2 on the host and SP2 on the EVB. The following steps should be taken to insure proper S1, S2 configuration:

1. Log in as **root** or the superuser (**su**)

2. Determine if the **tty0**, **tty1** devices already exist

- enter **smit**
- select **Devices**
- select **TTY**
- select **List All Defined TTYs**

Perform step 3 for each tty not listed.

Perform step 4 for each tty listed to insure that it is properly configured.

3. To add a **tty** device

- return to the **TTY** screen
- select **Add a TTY**
- select **tty rs232 Asynchronous Terminal**
- select **sa0** - Serial Port 1 (for ROM Monitor connection) when adding **tty0**
OR **sa1** - Serial Port 2 (for EVB SLIP connection) when adding **tty1**
- select **s1** for the port number when adding **tty0**
- OR **s2** for the port number when adding **tty1**
- insure that the BAUD rate is **9600** when adding **tty0**
OR that the BAUD rate is **38400** when adding **tty1**
- insure that the PARITY is **none**
- insure that the BITS per character is **8**
- insure that the Number of STOP BITS is **1**
- insure that Enable LOGIN is **disabled**

The default settings for all the other fields are satisfactory.

- select **Do** or hit **Enter**

Upon successful completion, a properly configured **tty** device is created and thus, step 4 can be skipped for the particular **tty** (**tty0** or **tty1**) added.

Remember to repeat this step, step 3, if both **tty0** and **tty1** needed to be added.

4. To properly configure a previously defined **tty** device

For systems running **AIX 3** :

- return to the **TTY** screen
- select **Change / Show Characteristics of a TTY**
- select **tty#** (where **#** = **0** or **1**)
- select **Change / Show TTY Program**
- insure that the following fields are set to the indicated values:

TTY	tty#	(#=0 for tty0, 1 for tty1)
TTY type	tty	
TTY interface	rs232	
Description	Asynchronous Terminal	
Status	Available	
Location	00-00-S*-00	(*=1 for tty0, 2 for tty1)
Parent Adapter	sa#	(#=0 for tty0, 1 for tty1)
Port Number	s*	(*=1 for tty0, 2 for tty1)
Terminal Type	dumb	
Enable LOGIN	disable	

The other fields can remain at their default values.

- select **Do** or hit **Enter**
- upon successful completion, select **Done** or hit **PF3** to return to the **TTY** screen
- select **Change / Show Characteristics of a TTY**
- select **tty#** (where # = 0 or 1)
- select **Change/Show HARDWARE TTY Characteristics**
- insure that the BAUD rate is **9600** for **tty0**

OR that the BAUD rate is **38400** for **tty1**

- insure that the PARITY is **none**
- insure that the BITS per character is **8**
- insure that the Number of STOP BITS is **1**
- select **Do** or hit **Enter**

Upon successful completion, the **tty** device is properly configured.

For systems running **AIX 4** or later :

- return to the **TTY** screen
- select **Change / Show Characteristics of a TTY**
- select **tty#** (where # = 0 or 1)
- insure that the following fields are set to the indicated values:

TTY	tty#	(#=0 for tty0, 1 for tty1)
TTY type	tty	
TTY interface	rs232	
Description	Asynchronous Terminal	
Status	Available	
Location	00-00-S*-00	(*=1 for tty0, 2 for tty1)
Parent Adapter	sa#	(#=0 for tty0, 1 for tty1)
Port Number	s*	(*=1 for tty0, 2 for tty1)
Terminal Type	dumb	
Enable LOGIN	disable	

- insure that the BAUD rate is **9600** for **tty0**
OR that the BAUD rate is **38400** for **tty1**
- insure that the PARITY is **none**
- insure that the BITS per character is **8**
- insure that the Number of STOP BITS is **1**

The other fields can remain at their default values.

- select **Do** or hit **Enter**

Upon successful completion, the **tty** device is properly configured.

5. This last step establishes the SLIP network over the **tty1** device between the host and the EVB. It's optional for those using the recommended Ethernet connection for host-to-EVB communications. This step is **not** required for **tty0** since it is being used simply for terminal emulation. Unlike a LAN interface, a SLIP connection is point to point. We first need to specify an IP address for the host and then an IP address for the other end of the SLIP connection, which in this case, is the evaluation board. To do this:

- enter **smit**
- select **Communication Applications and Services**
- select **TCP/IP**
- select **Further Configuration**
- select **Network Interfaces**
- select **Network Interface Selection**
- select **Add a Network Interface**
- select **Add a Serial Line INTERNET Network Interface**
- select **tty1**
- set the INTERNET ADDRESS field to the host IP address. An acceptable value would be **8.1.1.4**
- set the DESTINATION Address field to the evaluation board's IP address. An acceptable value would be **8.1.1.5**

Make a note of the addresses selected for the host and the evaluation board. They will be needed later.

- set the Network MASK to **255.255.240.0**
- insure that ACTIVATE is **yes**
- insure that the TTY PORT is **tty1**
- leave the BAUD RATE field blank
- leave the DIAL STRING field blank

- select **Do** or hit **Enter**

Upon successful completion, the SLIP Network Interface is established over **tty1** and the serial port setup is complete.

If this step fails, insure that a SLIP Network has not already been defined over **tty1**. To make this check, return to the **Network Interface Selection** screen in **smit** and select **List All Network Interfaces**. If **sl1** is listed then a network interface has already been defined for **tty1** and its characteristics may need to be changed. Return to the **Network Interface Selection** screen and select **Change/Show Characteristics of a Network Interface**. Select **sl1** and insure that the fields are set as stated previously in this step. (Note - there is no need to change the IP addresses in the INTERNET ADDRESS and DESTINATION Address fields if they have already been defined, but use of the above mentioned IP addresses is strongly recommended to maintain consistency with the rest of the documentation.) Make a note of the IP addresses chosen since they will be needed later during board setup.

3.1.2 Ethernet Setup - RS/6000

In addition to (or in place of) the SLIP connection, an Ethernet connection can be used for host-to-EVB communications. The Ethernet connection is made through an Ethernet adapter on the host and the 10BaseT connector on the EVB. Ethernet is much faster than SLIP and is recommended when downloading large applications on to the board or when using the RISCWatch debugger.

An Ethernet connection may require additional hardware. The 60x/7xx EVB supports connection via Standard Ethernet, twisted pair (10BaseT).

At a minimum, a 10BaseT connection requires a crossover Ethernet twisted pair cable (included in the kit) for point-to-point communications. If you want more than two nodes, you will need a hub and straight-through twisted pair cables.

Other hardware required will depend on the type of Ethernet adapter you have on your RS/6000 and whether the board is being connected to an existing Ethernet network. *AIX Communications Concepts and Procedures (GC23-2203, two volumes)* has additional information about the management and configuration of a TCP/IP network, including specifics as to how to configure an Ethernet network interface. Some of the basic steps are outlined below. You should consult your network administrator before attempting ethernet setup.

1. The host must be equipped to participate in a 10BaseT Ethernet network. This may require the installation of an Ethernet adapter card for your specific RS/6000 model and, as discussed previously, additional connectivity hardware. Consult the documentation included with the hardware for installation instructions. Most RS/6000 models come with Ethernet adapters already installed. They are labeled ET in the back of the RS/6000 system unit.
2. Assuming the host system is equipped with the appropriate Ethernet adapter, the Ethernet interface must be configured properly. To do this:

- log in as **root** or the superuser (**su**)
- enter **smit**
- select **Communication Applications and Services**
- select **TCP/IP**
- select **Further Configuration**
- select **Network Interfaces**
- select **Network Interface Selection**
- select **Add a Network Interface**
- select **Add a Standard Ethernet Network Interface**

Note - choose “**Standard Ethernet**” as opposed to “IEEE 802.3 Ethernet”. If you receive an error message stating that there is “No available adapter”, go to step 3 and skip the remaining items in this step, step 2.

- select **en0**
- set the INTERNET ADDRESS field to the host IP address. This value must be different from that used for the SLIP interface. It can be set to any convenient value if the Ethernet network is private for 60x/7xx EVB development purposes. An acceptable value would be **7.1.1.4**

Make a note of the IP address selected for the host system. It will be needed later. Note that an IP address for the evaluation board is not required as it was for the point-to-point SLIP network interface. An IP address for the EVB will, however, be required later on for the board setup.

- set the Network MASK field to **255.255.240.0**
- insure that **ACTIVATE** is **yes**
- insure that the Use Address Resolution Protocol is **yes**
- leave the **BROADCAST ADDRESS** blank
- select **Do** or hit **Enter**

Upon successful completion, a properly configured Ethernet interface has been added. The Ethernet setup is complete and step 3 need not be performed.

3. Perform this step only if you received the “No available adapter” error message when trying to **Add a Standard Ethernet Network Interface** in step 2. This message indicates that either the Ethernet adapter is missing (or possibly misplugged) or the Ethernet Network Interface already exists. To determine if the interface already exists:

- return to the **Network Interface Selection** screen in **smit**

- select **Change/Show Characteristics of a Network Interface**

If **en0** is **not** listed, insure that the RS/6000 host does have an Ethernet adapter and, if possible, that it is plugged correctly. If the adapter was misplugged, repeat step 2 to add the Ethernet Network Interface.

If **en0** is listed, then the Ethernet Network Interface already exists. Select **en0** and note the IP address listed for the INTERNET ADDRESS field. This value is the host's Ethernet IP address and will be needed later. If no IP address is listed, choose one. The IP address **7.1.1.4** can be used to maintain consistency with the menus and examples in this document. The Ethernet setup is complete.

3.1.3 ROM Monitor-Debugger Communication Setup - RS/6000

Before the RISCWatch Debugger can be used, some additional steps need to be taken to establish ROM Monitor-Debugger communications. These steps involve an update of the TCP/IP **services** file and a refresh of the TCP/IP **inetd** daemon.

To modify the **/etc/services** file, you need to log in as **root** or the superuser (**su**). The following lines must be added to the file:

```
osopen-dbg 20044/tcp    # for RISCWatch OS Open debug
osopen-dbg 20044/udp    # for RISCWatch rom_mon debug
```

The AIX **refresh -s inetd** command must then be run to inform the **inetd** daemon of the changes made to the **/etc/services** file.

3.2 PC Host Configuration

As stated previously, PC users are required to have a TCP/IP package compliant with the Microsoft Windows Socket API definition. Unlike Windows 95 and Windows NT, Windows 3.1 does not include such a package. To determine if you will need to install a TCP/IP package on Windows 3.1, do the following:

- Select the **Main** icon from the Windows' Program Manager.
- Select the **File Manager** icon.
- Select **File** from the menu bar and choose **Search**.
- Perform a search for **winsock.dll** on your entire hard drive.

If the winsock.dll file exists, you probably have some compliant TCP/IP package already installed. **Workgroup for Windows** is a product that provides such a TCP/IP package. If the winsock.dll file does not exist, you need to install a TCP/IP package compliant with the Microsoft Windows Socket API definition. One such package, Trumpet Winsock, can be downloaded from the following Internet site: **www.trumpet.com**.

Note: Windows 95 users who want to establish a SLIP host-to-EVB connection over a second serial port, require Trumpet Winsock as well, since the TCP/IP package that comes with Windows 95 does not support SLIP communications. Trumpet is not recommended for Windows 95 users already connected to a network since installing Trumpet may cause problems with previously defined networks. If the recommended Ethernet host-to-EVB connection is going to be used (instead of the SLIP host-to-EVB connection), Windows 95 users do **not** need to install Trumpet since the TCP/IP package that comes with Windows 95 can be used to establish the Ethernet connection.

The following information is provided as a **guide** to installing the Trumpet Winsock code. It is not meant to be a replacement to the installation instructions contained at the Trumpet Internet site. It is provided to help clarify items which may be confusing.

1. Go to the Trumpet Software International's web site (<http://www.trumpet.com>) and find the installation information for Trumpet Winsock. You want to download the latest version which can be used for Windows 3.1 (must have 16 bit support). For example, version 3.0 (file twsk30c.exe) is a combined 16 bit/Windows 95 release. This version can be downloaded and used for an evaluation period of 30 days. Use beyond the evaluation period requires a purchase.
2. The downloaded version is usually a single file called a self extracting ZIP file (has an extension *.exe). This file should be installed in a new directory (c:\trumpet, for example) and then executed. Execution is accomplished by going to the newly created directory and entering the name of the file. This will result in the creation of many more files in the new directory.
3. Read any '**README**' files carefully. Ethernet users are interested in directions concerning Packet Drivers because you will not be using a modem and you have already determined that a TCP/IP package does not exist on your system.
4. If the readme file does not direct you to do otherwise, execute 'install.exe' to start the installation process. You will be prompted for any required information. Note that you may be informed that a search will be done to rename any '**winsock.dll**' files found. If you performed this check earlier, this file should not be found anywhere else on your hard drive.
5. If a 'setup' screen appears, you can defer entering any fields until a later time.
6. When installation is complete, reboot the system, and bring up Windows.

3.2.1 Serial Port Setup

Most PCs include two serial ports to support communications via asynchronous data transfer. These ports are sometimes referred to as communication or COM ports. These ports are usually accessed from the back of the system unit. This document refers to them as serial ports S1 and S2. You should consult your PC literature to determine how many serial ports are available on your unit and where they are located.

When properly configured, one serial port can be used to connect a terminal emulator running on the host to the ROM Monitor running on the EVB, and the other to provide a **Serial Line Internet Protocol** (or **SLIP**) network interface between the host and the EVB to download applications. The SLIP host-to-EVB connection is optional if the recommended Ethernet connection is going to be used for host-to-EVB communications. This section addresses the proper configuration of the S1 and S2 serial ports to support these connections. Users should also refer to the Windows on-line help for "Changing Serial Port Settings".

The connection of the terminal emulator running on the host to the ROM Monitor running on the EVB, is made through the S1 serial port on the PC and the SP1 serial port on the EVB. The S1 port must be configured for a baud rate of 9600, 8 data bits, 1 stop bit, and no parity. The proper setting of these parameters is discussed later in the section on terminal emulation.

A connection between the S2 serial port on the host and the SP2 serial port on the EVB, provides a SLIP network interface to download application programs from the host to the EVB. This connection can be used in place of or along with the recommended Ethernet connection.

To establish a SLIP network over the S2 serial port for host-to-EVB communications, define a SLIP interface via the TCP/IP package being used. Since TCP/IP packages for PCs vary, users should consult their TCP/IP literature or their system administrator on how to establish the SLIP interface between the host and the EVB. The following IP addresses are suggested for the SLIP interface:

- PC host (source) : **8.1.1.4**
- Board (destination) : **8.1.1.5**

Make a note of the IP addresses selected since they will be needed later.

Trumpet Winsock users can use the following steps as a guide to establishing the SLIP interface:

1. Open the Trumpet Winsock by double clicking on the Trumpet Winsock icon in the Trumpet Winsock Files program group.
2. If setup was bypassed during installation, your connection should fail. A Trumpet Winsock window comes up indicating your connection status. Select **Setup** from the File menu to open the Setup dialog.
3. Set the **IP address** field to the IP address of the PC host: **8.1.1.4** is suggested to maintain consistency with this document.
4. Select **SLIP** under Drivers and then go to **Dialler settings**.
5. Select the appropriate **COMM port** (COM2 for example) to be used for SLIP communications.
6. Set the **Baud rate** to **38400**.

7. Disable **Hardware handshaking** and make sure **No automatic login** is selected. Use the default settings for the remaining options and/or check the help for more details.
8. Select **OK** from **Dialler Settings** and then **OK** from **Setup**.
9. Edit the **hosts** file found in the installed Trumpet directory to include both the PC host IP address and the board IP address. For example:
 - 8.1.1.4 local_slip
 - 8.1.1.5 evb_slip

After entering all the information, you may need to restart Trumpet Winsock for the network setup to take effect.

Prior to exiting Windows, we recommend terminating Trumpet Winsock (close the application). If you do not follow this recommendation, subsequent Trumpet starts may fail. If this occurs, you will need to reboot your system.

3.2.2 Ethernet Setup - PC

In addition to (or in place of) the SLIP connection, an Ethernet connection can be used for host-to-EVB communications. The Ethernet connection is made through an Ethernet adapter on the host and the 10BaseT connector on the EVB. Ethernet is much faster than SLIP and is recommended when downloading large applications on to the board or when using the RISCWatch debugger.

An Ethernet connection requires additional hardware. The 60x/7xx EVB supports connection via Standard Ethernet, twisted pair (10BaseT). This connection requires that the host PC be equipped with an appropriate Ethernet adapter. The host adapter is not included in the EVB kit. Please consult your PC and adapter documentation for requirements and installation instructions.

At a minimum, a 10BaseT connection requires a crossover Ethernet twisted pair cable (included in the kit) for point-to-point communications. If you want more than two nodes, you will need a hub and straight-through twisted pair cables.

Other hardware required will depend on the type of Ethernet adapter you have on your PC and whether the board is being connected to an existing Ethernet network. Please consult the documentation included with the adapter hardware for additional instructions.

Since TCP/IP packages for PCs vary, users should consult their TCP/IP documentation for information regarding the management and configuration of an Ethernet network interface. Establishment of an ethernet interface requires a host IP address. If the host PC is connected to an existing ethernet network, the host IP address should already be defined. Consult your network administrator on how to obtain the host's ethernet IP address and how to add the EVB to the existing network.

To maintain consistency with this document, the following IP addresses are suggested for the Ethernet interface :

- PC host (source) : **7.1.1.4**
- Board (destination) : **7.1.1.5**

Make a note of the IP addresses selected since they will be needed later.

3.2.2.1 Windows 3.1

Trumpet Winsock users can use the following steps as a **guide** to establishing a local Ethernet interface:

1. Trumpet Software International provides software which works with 'packet drivers'. When you first install your ethernet card, a set of different device drivers are provided. In order to use Trumpet Winsock, you will need to select a 'Packet Driver'. The Kingston ethernet card, provided with some RISCWatch packages, contains a packet driver that can be selected. If you buy an ethernet card that does not contain a packet driver, you can use the help option on the Trumpet menu bar to find out how you may be able to obtain a packet driver from the Internet. We will assume you have already followed the instructions for installing your ethernet card, have installed Trumpet Winsock, and have chosen a packet driver for use with Trumpet.
2. Read any '**README**' files carefully. Pay particular attention to any directions concerning Packet Drivers.
3. Follow the instructions for **Using the Trumpet Winsock over a packet driver** from the main Trumpet Help window. Follow the instructions for **Installing a packet driver and WINPKT**. At the time of this publication, the WINPKT program needed to be extracted from '<ftp://ftp.trumpet.com/winsock/winpkt.com>'. The ndis3pkt package, referred in the help as a replacement for winpkt, does not work unless you have WorkGroups for Windows, or some other windows package that runs NDIS.
4. Using the Trumpet help as a guide, your '**autoexec.bat**' file will need to have two lines added to get the ethernet communications working. The first line starts the packet driver you installed with your ethernet card. The proper name and syntax for this line should be identified in your ethernet card installation guide or in one of the files that came with the packet driver (i.e. the Kingston ethernet card has a '.doc' file that is part of the packet driver that describes how to invoke the driver). The second line to add is '**winpkt 0x60**' (vector 0x60 is usually the default vector to use).
5. After updating the 'autexec.bat' file, reboot the system to execute the changes.
6. From Windows, start Trumpet Winsock by double clicking on the Trumpet Winsock icon in the Trumpet Winsock Files program group.

7. If setup was bypassed during installation, your connection should fail. A Trumpet Winsock window comes up indicating your connection status. Select **Setup** from the File menu to open the Setup dialog.
8. Set the **IP address** field to the IP address of the PC host: **7.1.1.4** is suggested to maintain consistency with this document.
9. Select **Packet driver**, and set the **Vector** to **60**, **Netmask** to **255.255.240.0**, and **Gateway** to **0.0.0.0**.
10. Select **OK**.
11. Edit the **hosts** file found in the installed Trumpet directory to include both the PC host IP address and the board IP address. For example:
7.1.1.4 local_enet
7.1.1.5 evb_enet

After entering all the information, you may need to restart Trumpet Winsock for the network setup to take effect.

Prior to exiting Windows, we recommend terminating Trumpet Winsock (close the application). If you do not follow this recommendation, subsequent Trumpet starts may fail. If this occurs, you will need to reboot your system.

3.2.2.2 Ethernet Setup - Windows 95

A compliant TCP/IP package comes with Windows 95, so no TCP/IP package needs to be installed. If you haven't done so already, install the ethernet card on the host system according to the directions that came with the card.

To set the Host IP address for the ethernet connection:

- select the 'My Computer' icon from the desktop.
- select 'Control Panel'.
- select 'Network'.
- Add the appropriate "Adapter" network component for the ethernet adapter being used (if not already added).
- Add a "Protocol" network component of 'Microsoft - TCP/IP' (if not already added). Specify the IP address (**7.1.1.4** is recommended to maintain consistency with this document) and netmask (**255.255.240.0**) to be used.

Note: The "services" file that must be updated as part of the RISCWatch or evaluation kit installation is in directory "C:\WINDOWS".

The Evaluation Kit software was developed for Windows 3.1. Though it can be run successfully on Windows 95, certain restrictions apply. For example, file IDs need to be restricted to an eight character file name, and a three character file extension, or RISCWatch will not be able to locate source files.

3.2.2.3 Ethernet Setup - Windows NT 3.51

A compliant TCP/IP package comes with Windows NT, so no TCP/IP package needs to be installed. If you haven't done so already, install the ethernet card on the host system according to the directions that came with the card.

To configure TCP/IP for ethernet, double-click on the control panel icon followed by the network icon. Windows NT will prompt you through adding an ethernet adapter and TCP/IP. An IP address of **7.1.1.4** is recommended to maintain consistency with this document. A netmask of **255.255.240.0** should be used.

Note: The "services" file that must be updated as part of the RISCWatch or evaluation kit installation is in directory "C:\WINNT35\system32\drivers\etc".

The Evaluation Kit software was developed for Windows 3.1. Though it can be run successfully on Windows NT, certain restrictions apply. For example, file IDs need to be restricted to an eight character file name, and a three character file extension, or RISCWatch will not be able to locate source files.

3.2.3 ROM Monitor-Debugger Communication Setup - PC

Before the RISCWatch Debugger can be used, some additional steps need to be taken to establish ROM Monitor-Debugger communications. These steps involve an update of the TCP/IP **services** file and a restart of the TCP/IP package for the update to take effect.

Most PC TCP/IP packages place the **services** file under one of the TCP/IP package's subdirectories. Trumpet Winsock users should find the **services** file in the directory where the Trumpet files were installed. Windows 95 users should find the **services** file under "C:\WINDOWS\SERVICES". Windows NT users will find the **services** file under "C:\WINNT35\system32\drivers\etc". Users should consult their TCP/IP documentation or system administrator if they can not locate the file. The following lines must be added to the file:

```
osopen-dbg  20044/tcp    # for RISCWatch OS Open debug
osopen-dbg  20044/udp    # for RISCWatch rom_mon debug
```

For the update to take effect, TCP/IP needs to be re-started. This may require a re-boot of the system and/or a restart of the TCP/IP package.

3.3 Sun Host Configuration

Sun configuration requires that you be the superuser of the host workstation. This is accomplished by logging in as **root** or by using the **su** command to become the superuser.

3.3.1 Serial Port Setup - SUN

The Sun workstation includes two serial ports to support communications via asynchronous data transfer. These ports are labeled Serial A and Serial B on the back of the Sun's system unit. Some SPARCstation models multiplex these two ports into one physical port labeled A/B (Use A if it's available since use of the B port requires a special de-multiplexing cable from Sun). This section refers to these ports as S1 and S2, respectively. When properly configured, one of the serial ports can be used to connect a terminal emulator running on the host to the ROM Monitor running on the EVB. This connection is made through the S1 serial port on the Sun and the SP1 serial port on the EVB.

The S1 port on the host must be configured for a baud rate of 9600, 8 data bits, 1 stop bit, and no parity. The proper setting of these parameters is discussed later in the section on terminal emulation.

3.3.2 Ethernet Setup - SUN

Since all Sun SPARCstations come equipped with an ethernet (or AUI) port, an ethernet connection is used for host-to-EVB communications. The ethernet connection is made through the ethernet port on the host and the 10BaseT connector on the EVB.

An Ethernet connection requires additional hardware. The 60x/7xx EVB supports connection via Standard Ethernet, twisted pair (10BaseT).

At a minimum, a 10BaseT connection requires a crossover Ethernet twisted pair cable (included in the kit) for point-to-point communications. If you want more than two nodes, you will need a hub and straight-through twisted pair cables. Consult the documentation included with the hardware for additional information.

Establishment of an ethernet interface requires a host IP address. If the host SPARCstation is connected to an existing ethernet network, the host IP address should already be defined. Consult your network administrator on how to obtain the host's ethernet IP address and how to add the EVB to the existing network. Make a note of the host's IP address since it will be needed later.

If the host SPARCstation is not connected to an existing ethernet network, then a network between the EVB and the host must be established. The **ifconfig** command can be used to establish such a network. Users should consult their network administrator and Sun documentation for additional information. A host IP address of 7.1.1.4 is suggested to maintain consistency with this document. Make a note of the IP address selected since it will be needed later during board setup.

3.3.3 ROM Monitor-Debugger Communication Setup - SUN

Before the RISCWatch Debugger can be used, the TCP/IP **services** file must be updated to allow ROM Monitor-Debugger communications.

To modify the **/etc/services** file, you need to log in as **root** or the superuser (**su**). The following lines must be added to the file:


```
osopen-dbg 20044/tcp  # for RISCWatch OS Open debug
osopen-dbg 20044/udp  # for RISCWatch rom_mon debug
```


60x/7xx EVB Connectors

For detailed descriptions of connectors and jumpers on the 60x/7xx EVB, refer to the PowerPC 604 SMP Reference Design Technical Specification.

4.1 Connecting the 60x/7xx EVB Hardware

To establish a working environment, the EVB must be connected to a host system. ROM Monitor access requires a connection between the serial port on the board and the S1 (COM1) serial port on the host. Users must also establish a connection for debug and downloading applications from the host to the board. This connection is made over the SLIP or Ethernet network established during host configuration.

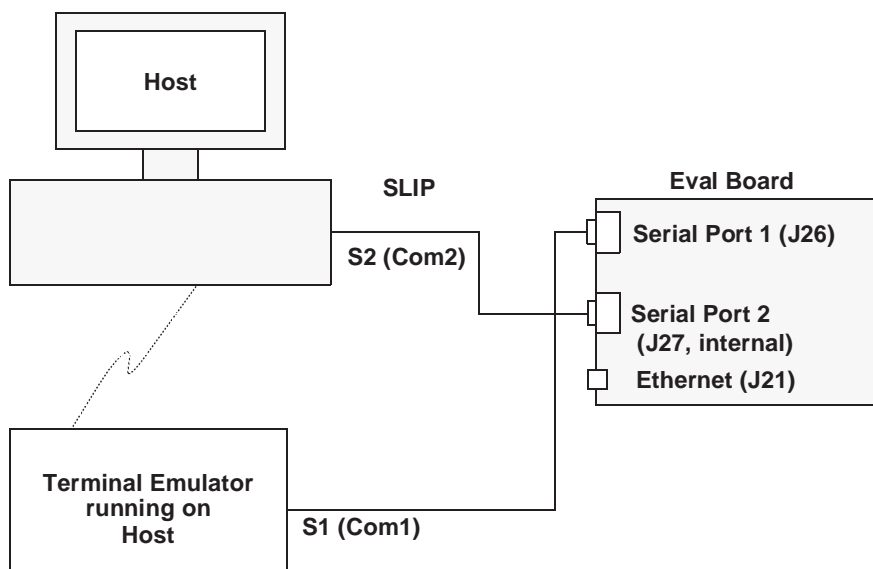


Figure 5-1. Serial Port Connection

Included in the 60x/7xx EVB kit is an interface cable supporting either 9-pin or 25-pin serial port connections. Use either the 9-pin or 25-pin, depending on the type of connector on the host. The cable is for connecting the serial port on the board to a terminal (or to a host running a terminal

emulator). The board supports a second serial connection for communication over SLIP. This requires a 2 x 5 Berg connector cable (not provided) to attach to internal connector J27.

Assuming a terminal emulator running on the host is going to be used for ROM Monitor access, connect the 9-pin serial port connector on one end of a cable to the serial port on the EVB, and the other end of the same cable to the S1 (COM1) serial port on the host. The host end may require a serial port adapter (not supplied) for connectivity. Sun SPARCstation users may require the 25 pin male-to-male adapter (included in the Sun 60x/7xx EVB kit) at the host end. If a SLIP connection is going to be used for host-to-EVB communications, connect a cable (not provided) from internal connector J27 (2 x 5 Berg header) on the EVB to the S2 (COM2) serial port on the host.

The Ethernet connection can be made in two ways. If the connection is to be used exclusively between the host and the EVB, the provided crossover cable can be used to directly connect the two nodes. Otherwise, a 10BaseT hub (not provided) must be used to connect the nodes together.

Note: The Ethernet 10BaseT crossover cable supplied will not work if plugged into a 10BaseT hub.

Figure 5-2 shows the connections and signal assignments required in a crossover cable:

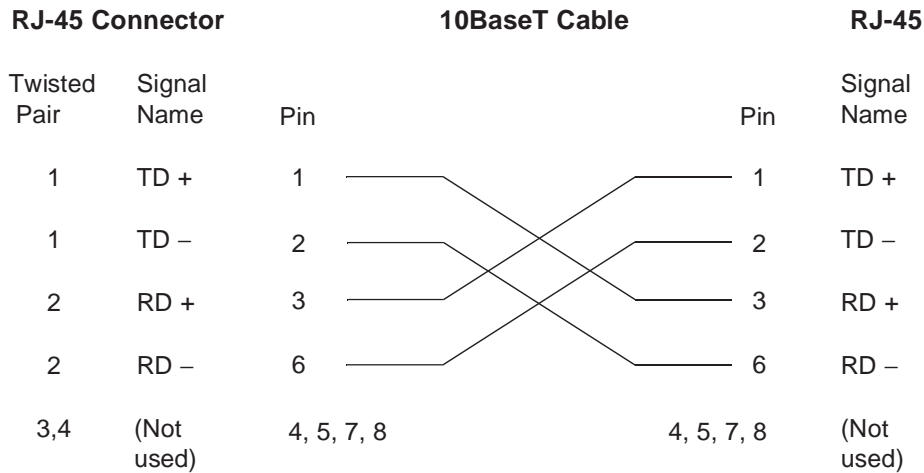


Figure 5-2. Wiring in a Crossover Cable

Figure 5-3 shows a point-to-point Ethernet connection using the provided crossover cable:

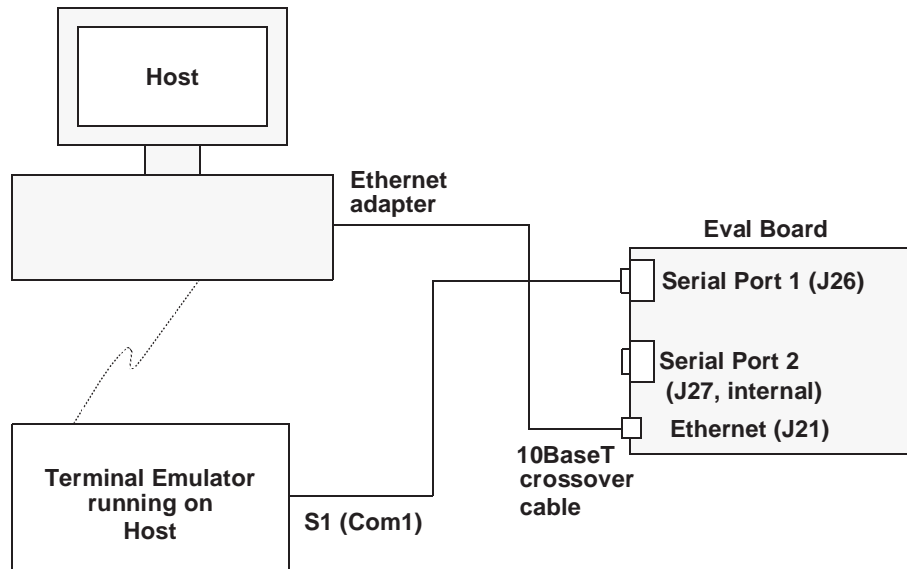


Figure 5-3. Point-to-Point Ethernet Connection

Figure 5-4 shows an Ethernet connection using a hub:

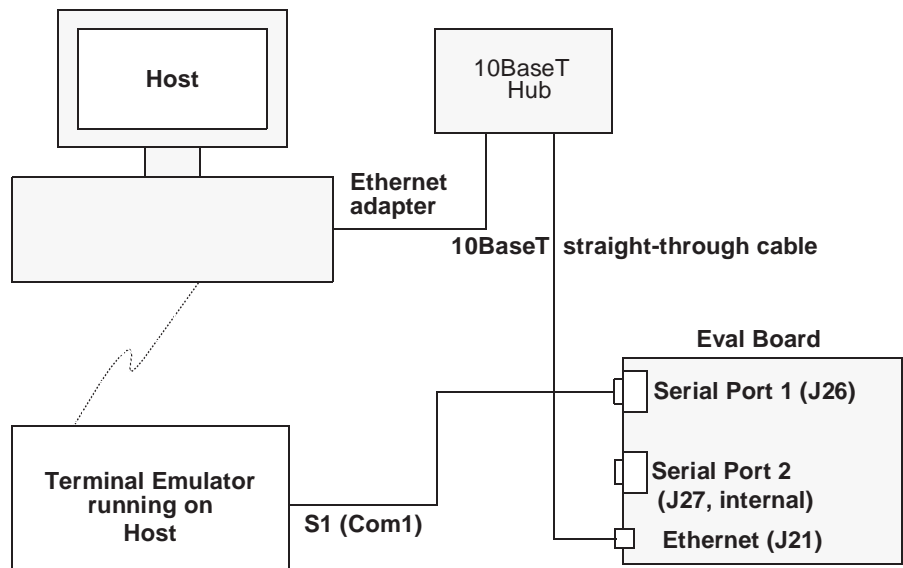


Figure 5-4. Ethernet Connection with Hub

Note: If the connection is going to be made to an existing Ethernet network, users should consult their Network Administrator to insure proper connectivity. Users wanting to use both a SLIP and Ethernet connection can do so, as long as both networks have been configured properly and the proper connections have been made.

4.2 Using a Terminal Emulator

The ROM Monitor transmits/receives data through serial port 1 (J26) on the evaluation board. Access to the ROM Monitor can be achieved by connecting a VT100 (or compatible) terminal directly to J26 on the EVB or by using a terminal emulator running on the host. When using a terminal emulator, access is obtained via a connection between J26 on the EVB and an available serial (or COM) port on the host system.

4.2.1 RS/6000 Terminal Emulation

The AIX Terminal Interface Program (TIP) can be used as a terminal emulator to support communications with the ROM Monitor. When properly configured, TIP connects the host RISC/6000 to a remote system, which in our case is the EVB. To set up TIP, do the following:

- log in as **root** or the superuser (**su**)
- go to the **/etc** directory (**cd /etc**)
- see if the file, **remote**, exists (**ls remote**). If the file does **not** exist, create it.
- using an editor, add the following line to the **remote** file (cut and pasters can find this line in the README.TXT file) :

```
tty0:dv=/dev/tty0:br#9600:el=^U^C^S^Q^D:ie=%$:oe=^D:pa=none:
```

- exit from **root**

TIP configuration is complete. Once all the host-to-EVB connections have been properly made and power has been supplied to the board, TIP can be activated by typing **tip tty0** at the AIX command prompt. After resetting the board, the ROM Monitor main menu should appear in the window where tip was activated. It may be necessary to hit the enter key once or twice to get the menu to appear for the first time. Additional information on TIP can be found in *AIX Communications and Procedures (GC23-2203, two volumes)*.

Some useful escape sequences to know when using TIP include (Note - it may be necessary to hit the **Enter** key before entering these escape sequences.):

- **~?** - help for TIP
- **~CTRL-D** - instructs the TIP command to terminate the connection and exit
- **~#** - sends a break to the remote system
- **~s script** - starts recording of transmissions made by the remote system

Recordings are made in the default **tip.record** file in the user's current directory

- **~s !script** - stops recording of transmissions made by the remote system

Note - If a terminal emulator other than TIP is used, it must be configured for 9600 baud, eight bits per character, one stop bit, and no parity.

4.2.2 PC Terminal Emulation

4.2.2.1 Windows 3.1 and Windows NT Terminal Emulation

Once all the host-to-EVB connections have been properly made and power has been supplied to the board, the Windows Terminal program can be used as a terminal emulator to support communications with the ROM Monitor. To do this:

- from Windows Program Manager, select Accessories
- select Terminal
- select Settings
- select Communications
- select COM1 (or the appropriate COM port used for S1 serial port set-up)
- select Baud Rate **9600**, Data Bits **8**, Stop Bits **1**, Parity **None**
- select Flow Control **Xon/Xoff**
- select OK

After resetting the board, the ROM Monitor menu should appear in the Terminal window. If it does not, check for proper connectivity between the host and the board. If the ROM Monitor menu still does not appear, insure that the COM port has been properly enabled. This can be done by using the configuration utility on the host PC (see your PC documentation for more details).

4.2.2.2 Windows 95 Terminal Emulation

Once all the host-to-EVB connections have been properly made and power has been supplied to the board, the Windows 95 HyperTerminal program can be used as a terminal emulator to support communications with the ROM Monitor. The steps for setting up the terminal emulator connected to COM1 are as follows:

- select 'Start' from the Windows 95 task bar
- select 'Programs'
- select 'Accessories'
- select 'HyperTerminal'
- If you see a window that says "You need to install a modem before you can make a connection. Would you like to do this now?" click on "No", you do not need a modem for the evaluation board.
- select the 'Hypertm' icon
- enter a name, for example "evb" and select an icon

- select the following:
Connect using Direct to Com 1(default)
Bits per second - **9600**
Data bits - **8** (default)
Parity - **None** (default)
Stop Bits - **1** (default)
Flow Control - **Xon/Xoff**
- select 'OK'

After resetting the board, the ROM Monitor menu should appear in the HyperTerminal window. If it does not, check your HyperTerminal settings and ensure proper connectivity between the host and the board.

4.2.3 SUN Terminal Emulation

The Terminal Interface Program (TIP) can be used as a terminal emulator to support communications with the ROM Monitor. When properly configured, TIP connects the host Sun SPARCstation to a remote system, which in our case is the EVB. To set up TIP, do the following:

- log in as **root** or the superuser (**su**)
- go to the **/etc** directory (**cd /etc**)
- see if the file, **remote**, exists (**ls remote**). If the file does **not** exist, create it.
- using an editor, add the following line to the **remote** file (cut and pasters can find this line in the README.TXT file) :

tty0:dv=/dev/ttya:br#9600:el=^U^C^S^Q^D:ie=%\$:oe=^D:pa=none:
- **exit** from root

TIP configuration is complete. Once all the host-to-EVB connections have been properly made and power has been supplied to the board, TIP can be activated by typing **tip tty0** at the command prompt. After resetting the board, the ROM Monitor main menu should appear in the window where tip was activated. It may be necessary to hit the enter key once or twice to get the menu to appear for the first time. If the ROM Monitor menu does not appear, consult your System Administrator - the ttya device may need to be modified. Additional information on TIP can be found in the online man pages by typing **man tip**.

Some useful escape sequences to know when using TIP include (Note - it may be necessary to hit the **Enter** key or **CTRL-D** before entering these escape sequences.):

- **~?** - help for TIP
- **~CTRL-D** - instructs the TIP command to terminate the connection and exit
- **~#** - sends a break to the remote system
- **~s script** - starts recording of transmissions made by the remote system

Recordings are made in the default **tip.record** file in the user's current directory

- **~s !script** - stops recording of transmissions made by the remote system

Note - If a terminal emulator other than TIP is used, it must be configured for 9600 baud, eight bits per character, one stop bit, and no parity.

4.3 Booting the PowerPC 60x/7xx on the EVB

When the connectors have been installed and power is applied to the 60x/7xx EVB, pressing the Reset switch causes the processor and the communications controllers to reset. After the ROM monitor initializes the 60x/7xx EVB, the monitor menu is displayed if a properly configured terminal (or terminal emulator) is attached to serial port 1 (J26) of the EVB. Details of ROM Monitor operation are provided in a later chapter.

60x/7xx EVB Hardware

The 60x/7xx EVB includes the PowerPC 604 SMP Reference Design Platform which contains the following features:

- 1) PowerPC 603e, 603ev, 604e-v1, 740, or 750 processor on a removable 256 pin DIMM daughter card.
- 2) IBM27-82660 Bridge chipset.
- 3) Memory
 - 8M DRAM, eight SIMM sockets, up to 256MB
 - 512KB flash
 - 512KB L2 cache (expandable to 1MB)
- 4) Battery-backed real-time clock with 4KB NVRAM
- 5) Two 16550-type serial ports
- 6) 3.5"x1.44MB floppy disk drive
- 7) Ethernet 10BaseT on RJ-45 connector
- 8) One enhanced bidirectional parallel port
- 9) Two PCI and three ISA connectors
- 10) Business Audio
- 11) PS/2 compatible keyboard and mouse controller

For detailed descriptions of 60x/7xx EVB hardware, refer to the PowerPC 604 SMP Reference Design Technical Specification.

Note: The section of the *PowerPC 604 SMP Reference Design Technical Specification* titled *System Firmware* refers to the firmware contained in the original desktop reference platform. This firmware has been replaced with the ROM monitor described in this document.

5.0.1 CPU Clock Control Logic

The 60x EVB CPU speed is configurable by changing resistors on the processor daughter card. However, some versions of the Frequency Select PAL do not allow you to configure processors to run at 150Mhz as shown in the table titled “CPU Clock Control Logic” of the PowerPC 604 SMP Reference Design Technical Specification. The latest Frequency Select PAL allows processor daughter cards to be modified to run at 150Mhz.

5.1 60x/7xx EVB Memory Map

The memory map of the 60x/7xx EVB is as follows:

PowerPC 60x/7xx Reference Platform Memory Map	
FFE0 0000 -> FFFF FFFF (Note 1) 2M - ROM address space	
FFF0 0000 -> FFF7 FFFF	Boot Flash, 512K
FFFF FFF0	Flash Write Addr/Data
FFFF FFF1	Flash Lock Out
C000 0000 -> C07F FFFF (dbat 3) 8M - PCI Memory, Non-cacheable	
C004 0000 -> C07F FFFF	MPIC Register Space
BFF0 0000 -> BFFF FFFF (dbat 2) 1M - PCI Interrupt Ack., Non-cacheable	
BFFF EFF0	System Error Address
BFFF FFF0	Interrupt Vector

PowerPC 60x/7xx Reference Platform Memory Map (Continued)

8000 0000 -> 80FF FFFF (dbat 1) 16M - I/O Space, Non-cacheable	
8000 0000 -> 8000 000F	DMA1
8000 0020 -> 8000 0021	INT1 Control and Mask
8000 0040 -> 8000 004F	Timer Counter 1
8000 0060 -> 8000 006F	Keyboard/Mouse
8000 0070 -> 8000 007F	RTC/NVRAM/Timer
8000 0080 -> 8000 009F	DMA
8000 00A0 -> 8000 00AF	INT2Control and mask
8000 00C0 -> 8000 00DF	DMA2
8000 0200 -> 8000 022F	Business Audio
8000 02F8 -> 8000 02FF	PC87332 Serial Port 2
8000 0330 -> 8000 0331	Business Audio (MIDI)
8000 0398	Super I/O Index Address
8000 0399	Super I/O Data Address
8000 03BC -> 8000 03BE	PC87332 Parallel Port
8000 03F0 -> 8000 03F7	PC87332 Floppy Disk Controller
8000 03F8 -> 8000 03FF	PC87332 Serial Port 1
8000 0400 -> 8000 04FF	DMA
8000 0534 -> 8000 0538	Business Audio
8000 0808	HDD Light
8000 080C -> 8000 087F	Misc. Registers
8000 0880 -> 8000 0881	SIMM Presence Detect
8000 0CF8	PCI/BCR Configuration Address
8000 0CFC	PCI/BCR Configuration Data
8010 0000 -> 8010 001F	Ethernet Register Space
8080 0800 -> 8080 08FF	82378ZB PCI to ISA Bridge PCI Cfg Regs
8080 1000 -> 8080 10FF	53C810 SCSI Controller PCI Cfg Regs
8080 2000 -> 8080 20FF	PCI Slot 1 (lower) PCI Cfg Regs
8080 4000 -> 8080 40FF	PCI Slot 2 (upper) PCI Cfg Regs
8080 8000 -> 8080 80FF	MPIC PCI Cfg Regs
8081 0000 -> 8081 00FF	79C970 Ethernet Cntr PCI Cfg Regs
0000 0000 -> 0FFF FFFF (dbat 0) 256M - DRAM Space, cacheable	
0000 0000 -> 0000 1FFF	Vector area
0000 2000 -> 0002 1FFF	ROM Monitor Area
0002 2000 -> 007F FFFF	Application area (default)

Notes:

- 1) The ROM area is not mapped with a dbat register.

5.2 Network Address of the Ethernet Controller

The EVB Ethernet controller, an AMD79C970A, has been assigned a unique six-byte network address. This address, also known as the media access control or MAC address, may need to be known by customers using the EVB to develop their own ROM versions.

The easiest way to obtain its value is to hook up a terminal (or terminal emulator) to the EVB serial port (as explained in the previous chapter) and bring up the ROM Monitor. After selecting option 7 to display the configuration, the controller's network address is displayed in the Ethernet boot source's *hwaddr* field as twelve hex characters (six bytes).

The ROM Monitor returns the MAC address in its board configuration function. Samples code showing how this is done can be found in the *usr_samp.c* file in the OS Open samples directory.

Another way to obtain the address, is to search the Vital Product Data (VPD) area in ROM where the network address is stored. The VPD fields consist of ASCII strings identifying the type of field, a length byte specifying the length of the associated data, and the data itself. The VPD begins at address 0xFF1 FE00 and is marked by field “*VPD” with 0 bytes of associated data. The network address is marked by “*NA” with six bytes of associated data (the network address). Finally, the end of the VPD is marked with “*END”. To extract the network address, a program would typically start at 0x0FF1 FE00, scan for “*NA”, verify the next byte is 0x6, and treat the next six bytes as the network address.

60x/7xx EVB ROM Monitor

This chapter describes the 60x/7xx EVB ROM Monitor program. This ROM resident program provides chip (and board level) initialization and a user interface menu that supports board diagnostics, program downloads, and debug.

7.1 ROM Monitor Source Code

The ROM Monitor source code is provided for ROM development purposes. This code is separate from the sample applications described in Chapter 8. The code is loosely organized by function in the following subdirectories and files within the **/usr/osopen/PLATFORM/openbios** directory (**\osopen\PLATFORM\openbios** for PC users).

- Makefile Top level makefile to create ROM monitor image (RS/6000 & SUN)
- makefile.mak Top level makefile to create ROM monitor image (PC)
- devTab.c Handles boot device definitions
- dsktLib/ Diskette routines
- include/ C include files
- m4/ assembler preprocessor include files
- ppcLib/ C callable functions to access PowerPC special instructions
- enetLib/ Ethernet chip specific code
- ioLib/ I/O helper functions
- miscLib/ Miscellaneous routines used for ROM monitor
- s1Lib/ Serial Port interface routines
- s1ldLib/ Code to support S1 serial port downloads
- dbLib/ Ptrace debug interface routines
- entry.s Processor and C environment initialization
- lib/ Repository for intermediate libraries
- netLib/ IP and UDP processing functions
- slipLib/ SLIP implementation
- align_h.s Alignment handling code
- mapfile1 Mapfile to specify ROM Monitor linkage directives
- bios_***.map Load map of the ROM Monitor version *** shipped with the EVB
- flash/ Code to support re-programming the flash memory

7.2 Communications Features

The 60x/7xx EVB ROM Monitor runs as part of the boot code in the flash memory on the board. The monitor communicates with an asynchronous terminal (or terminal emulator) attached to serial port 1 (SP1) on the EVB, through which the user accesses the monitor menu. The 60x/7xx EVB can download applications and communicate with the host debugger through serial port 2 (SP2) or the Ethernet adapter, depending on which devices are enabled. Communications between SP2 and the host use the Serial Link Internet Protocol (SLIP), while Ethernet communications use the Internet Protocol (IP) over standard Ethernet. The 60x/7xx EVB also supports the downloading of programs via serial port 1 (SP1). To use this feature, a VT100 terminal emulator that supports binary file transfers (such as *kermit*) must be used on the host system.

7.3 Bootp and tftp Configuration to support ROM Monitor Loads

Both the debugger and the ROM Monitor can be used to load applications onto the board. Details on how to use the debugger can be found in the *RISCWatch Debugger User's Guide*. To use the facilities of the ROM Monitor for downloading applications to the evaluation board, the host workstation must be configured to support the **bootp** protocol and **tftp** daemons. The configuration consists of two parts. The **bootptab** file on the host must be customized to match system requirements, and the **bootp** and **tftp** daemons (or servers) must be made available.

7.3.1 RS/6000 bootp and tftp configuration

To modify the */etc/bootptab* file, you need to log in as **root** or the superuser (**su**). Entries describing the evaluation board to the host workstation must be added to this file. Complete details describing the bootptab file format are available in the *AIX Command Reference* under "bootpd". File entries suitable for our purposes are shown below.

```
slipc:hd=/usr/osopen/PLATFORM/samples:bf=boot.img:bs:ip=8.1.1.5:sm=255.255.255.255
enetc:ht=ethernet:hd=/usr/osopen/PLATFORM/samples:bf=boot.img:bs:ip=7.1.1.5:sm=255.255.255.255:ha=xxxxxxxxxxxx
```

Each of the entries, *slipc* and *enetc*, should be entered on a single line. The value of the Ethernet hardware address field in the *enetc* entry, *ha=xxxxxxxxxx*, should match the twelve character hardware address listed for the Ethernet Boot Source on the ROM Monitor menu.

Both connections use the file */usr/osopen/PLATFORM/samples/boot.img* as the source for the application image to be downloaded onto the board. Be sure that the **ht=ethernet** keyword is used for the Ethernet connection entry and that the IP addresses are those of the evaluation board. Note that the IP address in the *slipc* entry must match that of the IP address assigned to the board during serial port set-up. Since a board IP address was not required for Ethernet set-up, the IP address used in the *enetc* entry defines the IP address of the board for the Ethernet connection. If the suggested bootptab entries are used, 7.1.1.5 would be the board's Ethernet IP address. Take note of the board's IP addresses, since they must be made known to the ROM Monitor.

To start the **bootp** and **tftp** daemons on systems running AIX 3, do the following:

- log in as **root** or the superuser (**su**)
- enter **smit**
- select **Diskless Workstation Management and Installation**
- select **Start Daemons on Server**
- select **Start BOOTP Daemon**
- select **Do** or hit **Enter**

Upon successful completion, **bootp** configuration is complete. Continue for **tftp**:

- select **Done** or hit **PF3**
- select **Cancel** or hit **PF3** to return to the **Start Daemons on Server** screen
- select **Start TFTP Daemon**
- select **List**

If “**tftp udp**” is not on the list, **tftp** has already been started for the workstation. The configuration steps are complete. Select **Exit** to leave **smit**.

- select “**tftp udp**”
- select **Do** or hit **Enter**
- You should be at the **Add an inetd Subserver** screen. The defaults listed are acceptable.
- select **Do** or hit **Enter**

Upon successful completion, **tftp** configuration is complete. Select **Exit** to leave **smit**

To start the **bootp** and **tftp** daemons on systems running AIX 4, do the following:

- log in as **root** or the superuser (**su**)
- enter **smit**
- select **Processes and Subsystems**
- select **Subservers**
- select **Start a Subserver**
- select **bootps**
- select **OK**

Upon successful completion, **bootp** configuration is complete. Select **Done** and continue for **tftp**.

- select **Start a Subserver**
- select **tftp**
- select **OK**
- select **Done**

Upon successful completion, **tftp** configuration is complete. Select **Exit** to leave **smit**

7.3.2 PC bootp and tftp configuration

Not all TCP/IP packages include the bootpd and tftpd servers required for ROM Monitor downloads. For this reason both the bootpd and tftpd servers have been included in the EVB software package under the \osopen\bin directory. These servers can be installed and used in conjunction with Windows Socket compliant TCP/IP packages such as Trumpet Winsock and those that come with Windows 95 and Windows NT.

Since TCP/IP packages vary greatly, this section should be used only as a **guideline** for bootp and tftp set-up. Users should consult their TCP/IP documentation for specific details.

Configuration consists of two parts. The **bootptab** and **services** files on the host must be customized to match system requirements, and the bootpd and tftpd servers must be made available. If you choose to use the bootpd and tftpd servers provided with this package, you will need to modify your **autoexec.bat** file to specify the location of the bootptab and services files. This is accomplished by adding a line that sets up an ETC constant to the directory where the bootptab and services files are located (ie. SET ETC=C:\TRUMPET for Windows 3.1/Windows 95 Trumpet users, ETC=C:\WINDOWS for Windows 95 users, ETC=C:\WINNT35\system32\drivers\etc for Windows NT 3.51).

A sample bootptab file, \osopen\PLATFORM\samples\bootptab.sam, is included with the EVB software. This file can be copied to the ETC directory set in the autoexec.bat file and modified appropriately. Note that the bootptab file in the ETC directory must be named **bootptab** with no file extension. Entries describing the evaluation board to the host PC must be added to the bootptab file.

When creating or modifying the bootptab file, the following rules apply.

- blank lines and lines beginning with “#” are ignored.
- each entry must be entered on a single line.
- each entry must start with a hostname followed by the legends (see the sample bootptab file for legend descriptions).
- use “:” to separate each legend and leave no spaces between legends.
- user must supply the host ip address via the “ip” legend.
- if the “hd” (home directory) & “bf” (bootfile) legends are not provided for a particular entry, the first defined “hd” and “bf” legends in the bootptab file will be taken as default.

File entries similar to those below would be suitable.

```
slipc:hd=\osopen\PLATFORM\samples:bf=boot.img:bs:ip=8.1.1.5:sm=255.255.255.255
enetc:ht=ethernet:hd=\osopen\PLATFORM\samples:bf=boot.img:bs:ip=7.1.1.5:sm=255.255.255.255:ha=xxxxxxxxxxxx
```

Each of the entries, slipc and enetc, should be entered on a **single** line. The value of the Ethernet hardware address field in the enetc entry, ha=xxxxxxxxxxxx, should match the twelve character hardware address listed for the Ethernet Boot Source on the ROM Monitor menu.

Both connections use the file `\osopen\PLATFORM\samples\boot.img` as the source for the application image to be downloaded onto the board. Be sure that the **ht=ethernet** keyword is used for the Ethernet connection entry and that the IP addresses are those of the evaluation board. Note that the IP address in the slipc entry must match that of the IP address assigned to the board during serial port set-up. Since a board IP address was not required for Ethernet set-up, the IP address used in the enetc entry defines the IP address of the board for the Ethernet connection. If the suggested bootptab entries are used, 7.1.1.5 would be the board's Ethernet IP address. Take note of the board's IP addresses, since they must be made known to the ROM Monitor.

The **services** file (no file extension) must also exist in the ETC directory set in the autoexec.bat file. It must be updated with the port and protocol information for the bootpd and tftpd servers. To use the servers provided with this package, the following entries must be included in the services file.

bootps	67/UDP
bootpc	68/UDP
tftp	69/UDP

For the update to take effect, TCP/IP needs to be re-started. This may require a re-boot of the system and/or a restart of the TCP/IP package. After that, the bootpd and tftpd servers are ready for use.

7.3.2.1 Automatic startup for Windows 3.1 and Windows NT 3.51

Users may find it convenient to have the bootpd and tftpd servers brought up automatically when entering Windows. To do this for Windows 3.1, the bootpd and tftpd servers should be added to your Windows environment Startup window using the following procedure.

- With Windows running, select the Program Manager and open the Startup window.
- Using the File pulldown menu on the Program Manager, select New to bring up a New Program Object window.
- From the New Program Object window, select Program Item and OK to open the Program Item Properties window. The Program Item Properties window requires that you provide Description, Command Line and Working Directory values. The following example shows one possible configuration.

Description: BOOTPD

Command Line: BOOTPD -C D -H 7.1.1.4

Working Directory: D:\OSOPEN\BIN

In the above example, the command line specifies how to invoke the bootpd server, and the working directory specifies where to find the bootpd server program (bootpd.exe). The -C parameter is used to specify a drive letter that is used in conjunction with bootptab file entries. Because the colon is used as a delimiter in bootptab file entries, the -C parameter is used as a mechanism by the bootpd server to concatenate a drive letter to the beginning of the hd: field. If the -C option is not specified, the current drive will be used as a default. The -H parameter is used to specify the Ethernet or slip IP address of the host PC (set during host configuration) to the bootpd server.

Use the same procedure to set up the tftpd server. In this case, the Program Item Properties window entries will describe information used for the tftpd server. The following example shows a possible configuration.

Description: TFTP

Command Line: TFTP

Working Directory: D:\OSOPEN\BIN

If you do not wish to have the bootpd and tftpd servers run automatically upon entering Windows, they can be run individually from the Windows Program Manager, File, Run menu. Note that TCP/IP must be up and running before the servers can be run.

7.3.2.1 Automatic startup for Windows 95

You may choose to run "BOOTPD.EXE" and "TFTP.EXE" automatically every time that Windows 95 is started or you can run these programs only when needed. To make these program run automatically every time Windows 95 is started perform the following steps.

- Select 'Start' from the Windows 95 task bar.
- Select 'Settings'.
- Select 'Taskbar'.
- Select 'Start Menu Programs'.
- Select 'Add...'.
- In the command line field enter the following:
BOOTPD -c C -h 7.1.1.4
(Where "C" is the driver letter containing the boot image and "7.1.1.4" is host IP address)
- Select 'Next'.
- In the 'Select Program Folder' window, select the 'Programs/Startup' folder.
- Select 'Next'.
- Select 'Finished'.
- To start "TFTP" follow the above steps, but enter the following in the command line field:
TFTP.

The BOOTP and TFTP demons will be started automatically upon the next restart of Windows 95.

7.3.3 SUN bootp and tftp configuration

The Solaris and SunOS operating systems both provide a tftpd server but do not provide a bootpd server. For this reason a bootpd server has been included in the EVB software package under the /usr/osopen/bin directory.

A sample bootptab file, `/usr/osopen/PLATFORM/samples/bootptab.sam`, is included with the EVB software. This file should be copied to the `/etc` directory and renamed **bootptab** if a bootptab file does not already exist. You will need to log in as root or the superuser (su) to update or add files in the `/etc` directory. Entries describing the evaluation board to the host PC must be added to the bootptab file.

When creating or modifying the bootptab file, the following rules apply.

- blank lines and lines beginning with “#” are ignored.
- each entry must be entered on a single line.
- each entry must start with a hostname followed by the legends (see the sample bootptab file for legned descriptions).
- use “:” to separate each legend and leave no spaces between legends.
- user must supply the host ip address via the “ip” legend.
- if the “hd” (home directory) & “bf” (bootfile) legends are not provided for a particular entry, the first defined “hd” and “bf” legends in the bootptab file will be taken as default .

File entries similar to those below would be suitable.

```
slipc:hd=/usr//osopen/PLATFORM/samples:bf=boot.img:bs:ip=8.1.1.5:sm=255.255.255.255
enetc:ht=ethernet:hd=/usr/osopen/PLATFORM/samples:bf=boot.img:bs:ip=7.1.1.5:sm=255.255.255.255:ha=xxxxxxxxxxxx
```

Each of the entries, slipc and enetc, should be entered on a **single** line. The value of the Ethernet hardware address field in the enetc entry, ha=xxxxxxxxxxxx, should match the twelve character hardware address listed for the Ethernet Boot Source on the ROM Monitor menu.

Both connections use the file `/usr/osopen/PLATFORM/samples/boot.img` as the source for the application image to be downloaded onto the board. Be sure that the **ht=ethernet** keyword is used for the Ethernet connection entry and that the IP addresses are those of the evaluation board. Note that the IP address in the slipc entry must match that of the IP address assigned to the board during serial port set-up. Since a board IP address was not required for Ethernet set-up, the IP address used in the enetc entry defines the IP address of the board for the Ethernet connection. If the suggested bootptab entries are used, 7.1.1.5 would be the board's Ethernet IP address. Take note of the board's IP addresses, since they must be made known to the ROM Monitor.

To start the bootpd and tftpd servers:

- log in as **root** or the superuser (**su**)
- ensure that the following entries are included in the `/etc/services` file.
 - bootps 67/udp
 - bootpc 68/udp
 - tftp 69/udp
- ensure that the tftp entry in the `/etc/inetd.conf` file is uncommented and modify as follows.

- `tftp dgram udp wait root /usr/etc/in.tftpd`
`in.tftpd -s /`
- add an entry for the bootpd server in **/etc/inetd.conf** as follows.
 - `bootps dgram udp wait root /usr/osopen/bin/bootpd bootpd -i`
- reconfigure inetd for the updates made to the inetd.conf file. First find the process id for inetd .
 - `ps -ef | grep inetd` (Solaris)
 - `ps -auex | grep inetd` (SunOS)
- Then send a hangup signal to reconfigure inetd.
 - `kill -HUP <process id>`

Bootp and tftp configuration is complete.

7.4 Accessing the ROM Monitor

The ROM Monitor expects a real or emulated VT100 type ASCII display attached to serial port 1 with line protocol parameters of 9600 baud, eight bits per character, no parity, and one stop bit. Once the terminal connected to SP1 is configured properly, you can access the ROM Monitor menu options, use the ping test, and load an application onto the evaluation board.

The ROM Monitor also provides the interface to the RISCWatch debugger. This facility, along with the image download process, is accessed via an IP network connection to the host workstation. Network configuration of the host was discussed earlier in the chapter on host configuration. The actual connection is either via SLIP (Serial Link Interface Protocol) running on serial port 2 at speeds up to 56K baud, or via standard Ethernet using the 10BaseT connector on the evaluation board.

7.5 ROM Monitor Operation

The ROM Monitor requires a block of DRAM for its operation and makes some assumptions about applications loaded on the board. Some of these assumptions may be disregarded if you do not need the ROM Monitor to interface with a debugger or otherwise support communication between the host workstation and the EVB.

Applications wishing to coexist with the ROM Monitor must observe the following constraints.

- Provide exception vectors for application events starting at address 0x0000 0000. For example, an application's external interrupt handler should be located at 0x0000 0500. This is handled for you when using OS Open.
- Use storage addresses between 0x0002 2000 and the end of DRAM only, except for application vectors.
- Do not reset the IP bit (bit 25) in the MSR.

- Do not start applications lower than address 0x0002 2000.

Figure 7-1 shows the address map of the evaluation board under control of the ROM Monitor.

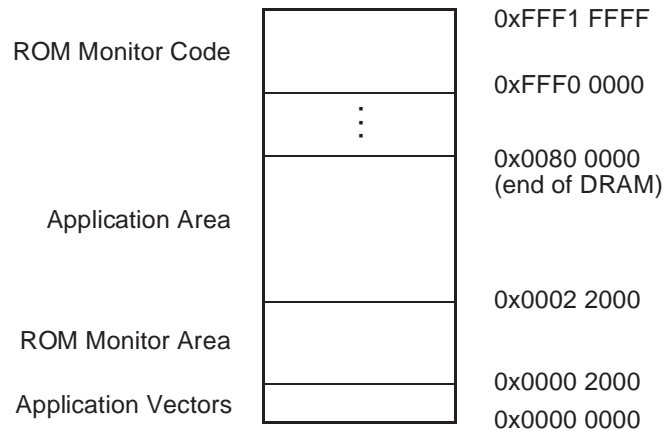


Figure 7-1. ROM Monitor Address Map

7.6 Monitor Selections and Submenus

At this point it is assumed that the host has been properly configured, all board connections have been made, power has been supplied, and the terminal emulator running on the host has been configured and started successfully. The main menu, shown below, is displayed after the 60x/7xx EVB has been reset and the ROM Monitor completes initialization. Note that some of the values you see, in particular the ROM Monitor version, the IP addresses, and the Ethernet controller's hardware address, may differ with those shown below.

Each menu option is described separately in the following sections. "Local" in the context of the ROM Monitor IP addressing means the IP address assigned to the evaluation board, while "remote" means the IP address assigned to the host workstation. Using option 8 to save changes made to the configuration will allow the new values to persist beyond subsequent power-ons or resets. The ROM Monitor supports this by storing its configuration data in NVRAM.

7.6.1 Initial ROM Monitor Menu

The following menu is displayed after the board has been reset.

```
PPC603 2.1 ROM Monitor (8/2/96)

----- System Info -----
603e processor, PVR = 00060301
Processor speed   = 133 MHz
Bus speed        = 66 MHz
Amount of DRAM   = 8 MB
```

```

SIMM slots:  00  00  00  00  00  00  04  04
-----

--- Device Configuration ---
Power-On Test Devices:
    000 Enabled    System Memory [RAM]
    001 Enabled    Ethernet  [ENET]
    004 Enabled    Serial Port 2 [S2]
-----

Boot Sources:
    001 Enabled    Ethernet  [ENET]
                    local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
    002 Enabled    Floppy Disk Controller [FDC]
    004 Enabled    Serial Port 2 [S2]
                    local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
    005 Enabled    Serial Port 1 [S1]
                    Baud = 9600
-----

Debugger : Disabled
-----

1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->

```


7.6.2 Selecting Power-On Tests

Option 1 in the main menu selects power-on tests. These tests are run when the menu exits and before the ROM loader begins the bootp processing.

```
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->1
```

When option 1 is selected, the following submenu is displayed.

```
--- ENABLE AND DISABLE POWER-ON TESTS ---
Power-On Test Devices:
  000  Enabled  System Memory [RAM]
  001  Enabled  Ethernet [ENET]
  004  Enabled  Serial Port 2 [S2]
-----
select device to change ->
```

Selecting a test toggles its testing status. For example, since the System Memory test is enabled in the above menu, selecting 0 at the prompt disables it.

```
select device to change ->0          [Selects system memory]
```

After the selection has been made, the new setting is displayed, followed by the main menu.

```
select device to change ->0
[RAM] test is disabled              [Message describing change]

--- Device Configuration ---
Power-On Test Devices:
  000  Disabled  System Memory [RAM]
  001  Enabled  Ethernet [ENET]
  004  Enabled  Serial Port 2 [S2]
-----
Boot Sources:
  001  Enabled  Ethernet [ENET]
           local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
  002  Enabled  Floppy Disk Controller [FDC]
  004  Enabled  Serial Port 2 [S2]
           local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
  005  Enabled  Serial Port 1 [S1]
           Baud = 9600
```

```
-----  
Debugger : Disabled  
-----  
1 - Enable/disable tests  
2 - Enable/disable boot devices  
3 - Change IP addresses  
4 - Ping test  
5 - Toggle ROM monitor debugger  
6 - Toggle automatic menu  
7 - Display configuration  
8 - Save changes to configuration  
9 - Set baud rate for s1 boot  
0 - Exit menu and continue  
->
```

Remember to use Option 8 to save any configuration changes that you may have made. If the changes are not saved, they will be lost upon an exit from the menu or upon a board reset.

7.6.3 Selecting Boot Devices

Option 2 in the main menu enables and disables boot devices.

```
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->2
```

When option 2 is selected, the following submenu is displayed.

```
--- ENABLE AND DISABLE BOOT DEVICES ---
Boot Sources:
    001   Enabled      Ethernet  [ENET]
                                local=7.1.1.5  remote=7.1.1.4
hwaddr=1000abcdef55
    002   Enabled      Floppy Disk Controller [FDC]
    004   Enabled      Serial Port 2 [S2]
                                local=8.1.1.5  remote=8.1.1.4
hwaddr=ffffffffffff
    005   Enabled      Serial Port 1 [S1]
                                Baud = 9600
-----
select device to change ->
```

Selecting a device toggles its boot status. Selecting 4, for example, would disable Serial Port 2 as a boot device.

```
select device to change ->4           [Selects serial port]
```

After the selection has been made, the new setting is displayed, followed by the main menu.

```
select device to change ->4
[S2] boot is disabled                 [Message describing change]

--- Device Configuration ---
Power-On Test Devices:
    000 Disabled      System Memory [RAM]
    001 Enabled       Ethernet  [ENET]
    004 Enabled       Serial Port 2 [S2]
-----
Boot Sources:
```

```

001  Enabled  Ethernet  [ENET]
      local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
002  Enabled  Floppy Disk Controller [FDC]
004  Disabled Serial Port 2 [S2]
      local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
005  Enabled  Serial Port 1 [S1]
      Baud = 9600
-----
Debugger : Disabled
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->

```

When the user selects option 0 and exits from the monitor menu, the monitor attempts a boot of the application image on the host using the enabled boot sources in the order they are listed. In the above example, a boot would be attempted over Ethernet since it is the first boot source enabled. If more than one boot source is enabled, an attempt to boot over the first enabled device will be made. If that attempt fails, a boot over the next enabled device is attempted.

7.6.4 Changing IP Addresses

Option 3 in the main menu allows users to change the IP addresses for the EVB and the host workstation. These addresses are used for bootp processing, debugger communications, and in the host connectivity “ping” test. **Note** - the local IP address is that of the board and the remote IP address is that of the host workstation. The IP addresses must match those set during host configuration.

```
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->3
```

When option 3 is selected, the following submenu is displayed.

```
--- CHANGE IP ADDRESS ---
Device List:
  001  Enabled   Ethernet   [ENET]
        local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
  004  Disabled  Serial Port 2 [S2]
        local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
-----
select device to change ->
```

Select the appropriate device.

```
select device to change ->1           [Selects Ethernet]
```

When a valid device is selected, the following submenu is displayed.

```
1 - Change local address
2 - Change remote address
0 - Return to main menu
->
```

Make the appropriate selection. To change the board’s IP address, you would select option 1, Change local address.

```
->1                               [Selects the local address]
Current IP address = (7.1.1.5      [Displays the current value]
Enter new IP address ->Enter IP address in dot notation (e. g., 8.1.1.2)
```

Now enter the new IP address in dotted decimal notation.

7.1.1.5

After the selection has been entered, the new configuration is displayed, followed by the main menu.

```
--- Device Configuration ---
Power-On Test Devices:
  000 Disabled   System Memory [RAM]
  001 Enabled    Ethernet   [ENET]
  004 Enabled    Serial Port 2 [S2]
-----
Boot Sources:
  001 Enabled    Ethernet   [ENET]
                local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
  002 Enabled    Floppy Disk Controller [FDC]
  004 Disabled   Serial Port 2 [S2]
                local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
  005 Enabled    Serial Port 1 [S1]
                Baud = 9600
-----
Debugger : Disabled
-----
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Toggle ROM monitor debugger
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
  0 - Exit menu and continue
->
```

This option should be repeated to set all of the IP addresses to their appropriate values. If the suggested IP addresses are being used, the local and remote addresses for both the Ethernet and the Serial Port should match those in the above menu. Remember to save any configuration changes via option 8.

7.6.5 Using the Ping Test

Option four in the main menu selects the ping test. The ping test can be used for a basic assurance test of IP connectivity to the host workstation. It should be performed after setting the IP addresses to insure host-to-EVB communications. If the ping test fails, users can not load applications on to the board. The local and remote addresses for the specified device are used for the source and destination of the ICMP ping packets.

```
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->4
```

When option 4 is selected, the current configuration is displayed, followed by another command prompt.

```
--- PING TEST ---
Device List:
    001  Enabled  Ethernet  [ENET]
           local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
    004  Disabled  Serial Port 2 [S2]
           local=8.1.1.5  remote=8.1.1.4
hwaddr=ffffffffffff
-----
select device to ping ->
```

Select the appropriate device to ping (in this case only Ethernet is enabled).

```
select device to ping ->1           [selects the Ethernet port]
```

If the board is able to successfully ping the host, a message similar to the following should appear.

```
Using [ENET] to ping. press any key to stop.
PING 7.1.1.4 56 data bytes
78 bytes from 7.1.1.4: icmp_seq=0 ttl=255 time=2 ms
78 bytes from 7.1.1.4: icmp_seq=2 ttl=255 time=1 ms
```

Hitting any key terminates the ping test. The main menu is redisplayed following the PING status report.

```
--- 7.1.1.4 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
 1 - Enable/disable tests
 2 - Enable/disable boot devices
 3 - Change IP addresses
 4 - Ping test
 5 - Toggle ROM monitor debugger
 6 - Toggle automatic menu
 7 - Display configuration
 8 - Save changes to configuration
 9 - Set baud rate for s1 boot
 0 - Exit menu and continue
->
```

If the ping test fails,

- Verify that the local and remote IP addresses are set correctly. The local IP address should be that of the board and the remote IP address should be that of the host. These IP addresses were assigned during host configuration (see earlier chapter).
- Verify that the cables are connected properly.
- Verify TCP/IP is running on the host.

Note - The ROM Monitor will not respond to an inbound ping test from the host unless the ROM Monitor is in Debug mode (via options 5 and 0) or the ROM Monitor ping test is active on the EVB at the same time (via option 4).

7.6.6 Entering the Debugger

Option 5 toggles the feature of the ROM Monitor that allows communication with the host based source level debugger. Debugging may be enabled/disabled, and saved as part of the configuration using option 8. The debugger is not actually called by the monitor until after the user exits the main menu by selecting option 0 (exit and continue).

```
--- Device Configuration ---
Power-On Test Devices:
  000 Disabled   System Memory [RAM]
  001 Enabled    Ethernet   [ENET]
  004 Enabled    Serial Port 2 [S2]
-----
Boot Sources:
  001 Enabled    Ethernet   [ENET]
                local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
  002 Enabled    Floppy Disk Controller [FDC]
  004 Disabled   Serial Port 2 [S2]
                local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
  005 Enabled    Serial Port 1 [S1]
                Baud = 9600
-----
Debugger : Disabled
-----
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Toggle ROM monitor debugger
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
  0 - Exit menu and continue
->5
ROM monitor debugger will be active on exit
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Toggle ROM monitor debugger
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
  0 - Exit menu and continue
->7
```

```
--- Device Configuration ---
Power-On Test Devices:
```

```

000 Disabled System Memory [RAM]
001 Enabled Ethernet [ENET]
004 Enabled Serial Port 2 [S2]
-----
Boot Sources:
001 Enabled Ethernet [ENET]
      local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
002 Enabled Floppy Disk Controller [FDC]
004 Disabled Serial Port 2 [S2]
      local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
005 Enabled Serial Port 1 [S1]
      Baud = 9600
-----
Debugger : Enabled (on exit)
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->0
PowerPC ROM Monitor Debugger

Waiting

for debug command...

Press any key to exit

```

Use option 8 to save the state of the ROM Monitor debugger. This option in combination with option 6, “Toggle automatic menu”, can be used to configure the EVB to automatically wait for the debugger to attach after power-on.

After enabling the ROM Monitor debugger (via option 5) and selecting option 0, the RISCWatch debugger can be started on the host and used to load an application onto the EVB. This is assuming the RISCWatch environment file has been updated for ROM Monitor communications. Once loaded successfully, the application can be run from the debugger.

The *RISCWatch Debugger User's Guide* contains more information on how to use the debugger to load and execute files with the ROM Monitor as a non-JTAG target. At this point, it is recommended that users become familiar with the debugging environment by following the “Quick Start” sample debug session in the debugger's User's Guide. This session takes a user through the basics, including how to use the debugger to load and run applications on the board.

7.6.7 Disabling the Automatic Display

Option 6 in the main menu disables the automatic monitor display when the EVB boots up. After option 6 has been selected and the configuration has been saved (via Option 8), the menu display is disabled but continues to function until the user exits from the main menu. Following the next power-on or reset, the menu is no longer automatically displayed. This allows the user's image to be downloaded automatically with no menu input required. This feature also allows a user to download an application with no cable connected to the serial port 1 on the EVB (that is, without a terminal emulator).

After the automatic menu display has been disabled, the main menu can be accessed (assuming a terminal emulator is attached successfully to SP1 on the EVB) by pressing any key during the first five seconds that the EVB is booting. Otherwise, application download processing starts without displaying the main menu.

7.6.8 Displaying the Current Configuration

Option 7 displays the current configuration.

```
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->7

--- Device Configuration ---
Power-On Test Devices:
  000 Disabled   System Memory [RAM]
  001 Enabled    Ethernet   [ENET]
  004 Enabled    Serial Port 2 [S2]
-----
Boot Sources:
  001 Enabled    Ethernet   [ENET]
                    local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
  002 Enabled    Floppy Disk Controller [FDC]
  004 Disabled   Serial Port 2 [S2]
                    local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
  005 Enabled    Serial Port 1 [S1]
                    Baud = 9600
-----
Debugger : Enabled (on exit)
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->
```

When a menu operation is selected to alter configuration settings, the current configuration is automatically redisplayed.

7.6.9 Saving the Current Configuration

Option 8 saves the current configuration for subsequent power-ons/resets..

```
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->8
Configuration has been saved
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->
```

The configuration is saved in the NVRAM on the evaluation board and is retained until a new configuration is subsequently saved.

7.6.10 Setting the Baud Rate for S1 Boots

Option 9 provides a mechanism for setting the baud rate to be used by serial port 1 when it is used as a device to download programs. Downloading over serial port 1 requires the use of a VT100 terminal emulator that supports **kermit** binary file transfer over serial port 1. RS/6000 and Sun users should note that the TIP terminal emulator does not support kermit binary file transfers. Windows 3.1 users can use the Windows Terminal program to perform kermit binary file transfers, but the baud rate is limited to 19 200. Windows 95 users can use HyperTerminal to perform kermit file transfers at up to 115 200 baud. The kermit terminal emulator, available as shareware from the <http://www.columbia.edu/kermit> Internet site, can be used on any of the supported hosts to download programs over serial port 1 at speeds up to 115 200 baud.

```
--- Device Configuration ---
Power-On Test Devices:
    000 Disabled    System Memory [RAM]
    001 Enabled     Ethernet   [ENET]
    004 Enabled     Serial Port 2 [S2]
-----
Boot Sources:
    001 Enabled     Ethernet   [ENET]
                        local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
    002 Enabled     Floppy Disk Controller [FDC]
    004 Disabled     Serial Port 2 [S2]
                        local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
    005 Enabled     Serial Port 1 [S1]
-----
Debugger : Enabled (on exit)
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->9

Select a baud rate for S1 boot
1 -          9600
2 -         19200
3 -         28800
4 -         38400
5 -         57600
6 -        115200
=>4
```

```

--- Device Configuration ---
Power-On Test Devices:
  000 Disabled   System Memory [RAM]
  001 Enabled    Ethernet   [ENET]
  004 Enabled    Serial Port 2 [S2]
-----
Boot Sources:
  001 Enabled    Ethernet   [ENET]
           local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
  002 Enabled    Floppy Disk Controller [FDC]
  004 Disabled   Serial Port 2 [S2]
           local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
  005 Enabled    Serial Port 1 [S1]
                           Baud = 38400           [download baud rate
appears here]
-----
Debugger : Disabled (on exit)
-----
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Toggle ROM monitor debugger
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
  0 - Exit menu and continue
->

```

Use Option 8 to save the selected speed after reset and power-on.

7.6.11 S1 Boot

To perform an S1 boot you must have a terminal emulator which supports kermi file transfer. The file must be a valid boot image and must be sent in binary mode. If you have selected to use a baud rate other than 9600, you must set the terminal emulator to run at that speed before loading the file and set the speed back to 9600 after the down-load is complete. The following example shows loading the "usr_samp.img" file.

```
--- Device Configuration ---
Power-On Test Devices:
  000 Disabled System Memory [RAM]
  001 Disabled Ethernet [ENET]
  004 Disabled Serial Port 2 [S2]
-----
Boot Sources:
  001 Disabled Ethernet [ENET]
        local=7.1.1.5 remote=7.1.14 hwaddr=1000abcdef55
  002 Disabled Floppy Disk Controller [FDC]
  004 Disabled Serial Port 2 [S2]
        local=8.1.1.5 remote=8.1.1.4 hwaddr=ffffffffffff
  005 Enabled Serial Port 1 [S1]
        Baud = 38400
-----
Debugger: Disabled
-----
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Toggle ROM monitor debugger
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
  0 - Exit menu and continue
->0
Booting from [S1] Serial Port 1...

PLEASE NOTE: You must now...

  a. Exit from terminal emulation mode
  b. Modify the baud rate of your host session
  c. Transmit a file to the target in binary mode
  d. Reset the host baud rate to 9600
  e. Reenter terminal emulation mode
  f. Hit enter to execute the downloaded program
```

At this point kermi users must get to the terminal emulator command mode and change the line speed to match what was selected by option 9 and tell the terminal emulator to send the file in binary format.

```
^lc (Cntrl-lc)
(Back at waterdeep)
C-Kermit>set speed 38400
/dev/tty0, 38400 bps
C-Kermit>set file type bin
```

You can now load the file.

```
C-Kermit>send usr_samp.img
SF
Type escape character (^\\) followed by:
X to cancel file, CR to resend current packet
Z to cancel group, A for status report
E to send Error packet, Ctrl-C to quit immediately:

Sending: usr_samp.img => USR_SAMP.IMG
Size: 164864, Type: binary
.....
.....
.....
.... [OK]
ZB
```

When loading is completed, you must change the baud rate back to 9600 bps before continuing.

```
C-Kermit>set speed 9600
/dev/tty0, 9600 bps
```

After setting the baud rate back to 9600 bps, re-connect to your terminal emulator and press enter to complete the down-load.

```
C-Kermit>con
Connecting to /dev/tty0, speed 9600.
The escape character is Ctrl-\\ (ASCII 28, FS)
Type the escape character followed by C to get back,
or followed by ? to see other options

Loaded successfully ...
Entry point at 0x22f20 ...

Hello 603e user!

Your ROM Monitor version is : 2.1

Your 604 Evaluation Board has 33554432 bytes of DRAM installed.

Your Ethernet controller's network address is : 1000abcdef55

usr_samp done!
```

Assuming the S1 boot baud rate has been set to 38400 and option 0 has been selected to exit the ROM Monitor menu and initiate a load, Windows 95 HyperTerminal users can initiate the kermit binary file transfer by performing the following steps.

- Select **Call** and then **Disconnect**.
- Select **File, Properties, Configure** and set the baud to match the baud rate set via ROM Monitor option 9. In this case, it is 38400.
- Select **OK** and **OK** again.
- Select **Call** and then **Connect**.
- Select **Transfer, Send File** and type the filename of the file to load. Set the **Protocol** to Kermit.
- Select **Send**.

Upon successful completion of the transfer, the baud rate must be changed back to 9600.

- Select **Call** and then **Disconnect**.
- Select **File, Properties, Configure** and set the baud to 9600.
- Select **OK** and **OK** again.
- Select **Call** and then **Connect**.
- Hit **Enter** to complete the down-load sequence.

7.6.12 Exiting the Main Menu

Option 0 exits from the main menu, leaving the monitor active. If the debugger is active prior to selecting option 0, the ROM Monitor waits for the user to start the debugger on the host. In all other cases, option 0 initiates an attempt by the ROM Monitor to load an application from the host to the EVB over the enabled boot device(s). When downloading over the Ethernet or SLIP (S2), the host bootp and tftp configuration must be completed for the ROM Monitor to load successfully. Once loaded successfully, the application is executed.

When serial port 1 is used, the ROM Monitor requires the user to follow additional instructions to complete the download. The example shown here describes the sequence required when programs are downloaded over serial port 1.

```
--- Device Configuration ---
Power-On Test Devices:
  000 Disabled   System Memory [RAM]
  001 Disabled   Ethernet   [ENET]
  004 Disabled   Serial Port 2 [S2]
-----
Boot Sources:
  001 Disabled   Ethernet   [ENET]
                    local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
  002 Disabled   Floppy Disk Controller [FDC]
  004 Disabled   Serial Port 2 [S2]
                    local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
  005 Enabled    Serial Port 1 [S1]
                    Baud = 38400
```

```

-----
Debugger : Enabled (on exit)
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Toggle ROM monitor debugger
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->0
Booting from [S1] Serial Port 1...

PLEASE NOTE: You must now...

a. Exit from terminal emulation mode
b. Modify the baud rate of your host session
c. Transmit a file to the target in binary mode
d. Reset the host baud rate to 9600
e. Re-enter terminal emulation mode
f. Hit enter to execute the downloaded program

```

The ROM Monitor will now wait for you to follow the above steps. The idea is that you must temporarily modify the terminal emulation session baud rate to match the baud rate expected by the ROM Monitor for the serial port 1 download. The file must then be transferred to the EVB from the host. The baud rate is restored to 9600 so that terminal emulation support can function after the program has been downloaded, The ROM Monitor will wait for you to restore the baud rate (9600) and hit enter prior to executing the downloaded program. This prevents any program I/O from being lost or incorrectly displayed when it begins execution.

The following is an example of what you might see when the program is allowed to run.

```

Loaded successfully ...
Entry point at 0x23130 ...
.
.
.

```

7.7 ROM Monitor User Functions

The ROM Monitor contains several functions that are available to user programs. The prototypes of these functions can be found in the `usr_func.h` file in the **/usr/osopen/PLATFORM/include** directory (**\osopen\PLATFORM\include** for PC users). These functions include:

- `send_packet_on_bootdev()` - allows an IP packet to be sent over the device that was used to load the application program (either the Ethernet or the second serial port, SP2).
- `sh_register()` - used to register a function that will be called when an IP packet is received by the ROM Monitor over the boot device.
- `get_board_cfg()` - reads the configuration data associated with the board.
- `enet_send_macframe()` - allows a frame to be sent over the Ethernet.
- `enet_register()` - allows the user to register an IP address for the Ethernet (an IP address different from that assigned to the ROM Monitor) and to specify a function to be called when a frame arrives for that address.
- `enetisThere()` - determines if the Ethernet chip is present on the board.
- `enetInit()` - initializes the Ethernet.
- `getchar()` - reads one character at a time from the keyboard buffer over the first serial port (SP1).
- `s1putchar()` - writes one character to the first serial port (SP1).

Applications must follow a predefined protocol to access ROM Monitor user functions. An example showing the proper calling procedures are included in the `usr_samp.c` sample program in the **samples** directory. This sample program calls the `get_board_cfg()` ROM Monitor function to determine the amount of DRAM installed on the board. This program will be run as a sample program in the next chapter.

7.8 Flash Update Utility

The **openbios/flash** directory contains all the code you need to re-program the flash memory on the EVB. This utility takes a binary image file targeted for the ROM as input, and generates a loadable file that will re-program the flash memory with the data in the binary input file. The file can then be loaded by an existing ROM Monitor version (which will be over-written upon successful completion of the loaded program) or via RISCWatch JTAG.

IMPORTANT: Please see the `readme.txt` file in the `openbios/flash` directory for important information regarding the use of this tool.

Be aware that if you use the ROM Monitor bootp or the RISCWatch ROM Monitor mode download process to re-program the flash, and the program loaded contains errors that will not allow you to download images in the same manner, your flash may be corrupted and rendered useless. In this case you will need to use RISCWatch JTAG or a ROM burner to re-program the flash.

RISCWatch JTAG users will find a RISCWatch command file, **rw_flash.cmd** in the openbios/flash directory. This command file can be used to prepare the EVB, load the flash update program containing the new binary image to program into the ROM, and start it running. This method can be used to program new flash parts, or to re-program a corrupted flash part when normal ROM Monitor downloads are not possible or inconvenient. When using this command file, RISCWatch **must** be used in JTAG mode.

60x/7xx EVB Sample Applications

This chapter describes the steps necessary to build and run the sample programs included in the 60x/7xx EVB software support package. This code is separate from ROM monitor code described in Chapter 7.

8.1 Overview

In the High C version of the EVB kit, the sample application programs are compiled, assembled, and linked using the IBM High C/C++ compiler, assembler, and linker. OS Open libraries are used during the link step to create an executable file in ELF format. This file includes the OS Open bootstrap code as well as other OS Open functions and is referred to as a boot file. One of the tools provided in the software support package, **eimgbld**, is then used to convert the boot file into the format used by the ROM Monitor to load programs onto the evaluation board (see Appendix B for more information on the ROM Monitor load format). The output of the **eimgbld** step is a file referred to as a boot image file.

Processing is similar for the RISC/6000 XCOFF version of the EVB kit. Programs are compiled, assembled, and linked using the XCOFF XLC compiler, assembler, and linker, and the boot image files are created using the **nimgbld** tool supplied with the EVB software.

There are several ways to load and execute a boot image file. One way is to use the ROM Monitor to load and execute the file. Network loads over Ethernet or SLIP require that the host contain the bootp and tftp servers and be properly configured to support the bootp and tftp protocols (see the previous chapters on host configuration and ROM Monitor setup). Loads over serial port 1 require a terminal emulator that supports the kermit transfer protocol. A ROM Monitor load is initiated via option 0 from the ROM Monitor main menu.

Another way to load and execute the boot image file is to use the RISCWatch debugger in ROM monitor mode. To bring up RISCWatch in ROM Monitor mode (see the *RISCWatch Debugger User's Guide* for details), you must update the RISCWatch environment file for ROM Monitor communications, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0) and then start up RISCWatch on the host system. The RISCWatch **load image** command can then be used to load the boot image file onto the board. Once loaded successfully, the **attach 42** and **logoff** commands can be issued to execute the program. The **attach 42** command informs the ROM Monitor that a process will be running and the **logoff** command tells the ROM Monitor to exit debug mode and start the execution of the program. After program execution, users should quit and restart RISCWatch before loading another boot image file to run. Without quitting RISCWatch, subsequent boot image execution can not be guaranteed. (Note: RISCWatch also provides

the means to load a boot file (as opposed to a boot image file) via its **load file** command. See the “Running Your Programs” section in the *RISCWatch Debugger User's Guide* for additional information. This section also describes the steps required to load and debug boot and boot image files.)

8.2 ROM Monitor Flash Image

The flash memory on the EVB comes preprogrammed with a specific version of the ROM Monitor. This version may not be latest version of the ROM Monitor. To run the samples in the software support package, the latest version should be used. The latest version of the ROM Monitor is included in the software support package in the file:

- **/usr/osopen/PLATFORM/openbios/lib/rom_***.img** (RS6K & SUN)
- **\osopen\PLATFORM\openbios\lib\rom_***.img** (PC)

where *** is equal to the ROM Monitor version. If the *** version number of the ROM Monitor in the software support package does not match the version number displayed by the monitor when it comes up on the board, you can load the more recent version of the monitor provided in the software support package to re-program the flash memory.

The **rom_***.img** file can be loaded using the ROM Monitor or the RISCWatch debugger. For it to load properly upon the selection of ROM Monitor option 0, it must be copied to **boot.img** if the suggested bootptab entry was used (see the previous chapter on bootptab configuration).

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the following RISCWatch commands to load and execute the **rom_***.img** image file:

- **load image /usr/osopen/PLATFORM/openbios/lib/rom_***.img** (RS6K & SUN)
- **load image \osopen\PLATFORM\openbios\lib\rom_***.img** (PC)
- **attach 42**
- **logoff**

You will see screen information similar to that shown below. Lines preceded by “\$\$” are annotation for this example and do not appear on the screen.

```
$$ Standard ROM Monitor load screen below
PPC603 1.2 ROM Monitor (9/5/95)
$$ Version 1.2 already installed corresponds to rom_12.img
```

```
----- System Info -----
603e processor, PVR = 00060301
Processor speed   = 133 MHz
Bus speed        = 66 MHz
Amount of DRAM   = 8 MB
SIMM slots:  00 00 00 00 00 00 04 04
-----
```



```

--- Device Configuration ---
Power-On Test Devices:
  000 Disabled System Memory [RAM]
  001 Enabled Ethernet [ENET]
  004 Enabled Serial Port 2 [S2]
-----
Boot Sources:
  001 Enabled Ethernet [ENET]
                                local=7.1.1.5 remote=7.1.1.4
hwaddr=1000abcdef55
  004 Disabled Serial Port 2 [S2]
                                local=8.1.1.5 remote=8.1.1.4 hwaddr=fffffffffff
  004 Disabled Serial Port 1 [S1]
                                Baud = 38400
-----
Debugger: Disabled
-----
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Toggle ROM monitor debugger
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
  0 - Exit menu and continue
->0
$$ Selection of 0 causes evaluation board to be loaded. Previous
$$ arrangements must have been made to place the new ROM Monitor
$$ image (for ex. /usr/osopen/PLATFORM/openbios/lib/rom_13.img) in the
$$ place where bootp expects to find it (for ex. boot.img)
Booting from [ENET] Ethernet...
Sending bootp request ...

Loading file "/usr/osopen/PLATFORM/samples/boot.img" ...
Sending tftp boot request ...
Transfer Complete ...
Loaded successfully ...
Entry point at 0x23028 ...

$$ following information is from the ROM Monitor update program
##### IBM 6XX Evaluation Kit FLASH Update #####
ROM Monitor Version 1.3

$$ The universally administered hardware address for the Ethernet
$$ controller is kept in the flash ROM and is displayed here.

```

```

$$ Do not change this value for normal ROM Monitor updates
Network Address =
1000abcdef55

Do you wish to change Network Address? (y or n) n

$$ Heed the following warning. The ROM Monitor image could be
$$ rendered unusable and the board useless until the flash ROM is
$$ replaced.
    WARNING: You are about to re-program your ROM Monitor FLASH
              image. Do NOT turn off power or press reset
              until this procedure is completed. Otherwise
              the card may be permanently damaged!!!

Do you wish to continue? (y or n) y

Verifying new FLASH Image...
131072 matches, 0 mismatches

Update complete!
All done!

```

8.3 Using the Software Samples

The sample application programs are in **/usr/osopen/PLATFORM/samples** (**\osopen\PLATFORM\samples** for PC users). It is recommended that users first build and run the Dhrystone `usr_samp`, and `timesamp` sample programs as detailed below, to become familiar with the working environment. These sample programs use **basic_os.c** to provide a minimal OS Open configuration.

Additional details regarding the sample programs and application development in general can be found in the “Developing OS Open Applications” chapter in the *IBM OS Open User’s Guide*. That chapter should be referenced for instructions on building and running the `applprog`, `benchmk`, `mailsamp`, and `cat` sample programs.

The sample makefile contains the directives needed to build all the sample programs. It is suggested that this makefile be used as the starting point for building subsequent user applications.

Before attempting to build the samples, ensure the **osopen/bin** directory and the directory that contains the compiler, are part of your execution path (these steps should be modified accordingly based on where the compiler and the software support package were actually installed).

For **RS/6000** and **SUN** hosts :

- issue the command:
`export PATH=$PATH:/usr/osopen/bin:/usr/highcppc/bin`

OR (to update your PATH permanently)

- Edit `~/.profile` using an editor such as `vi`.
- Add `PATH=$PATH:/usr/osopen/bin:/usr/highcppc/bin` as a line in your profile before the line "export PATH".
- Run `~/.profile` to update your profile.

For PC hosts:

- Edit AUTOEXEC.BAT using an editor such as `e` (you should back this file up before editing).
- If the following statement is missing, add it to the end of the file.
`SET PATH=C:\highcppc\bin;C:\osopen\bin;%PATH%;`
- Run AUTOEXEC.BAT to update your path.

NOTE: The "make" utility supplied with your evaluation kit may not run under a Windows NT command prompt that is started by "cmd.exe". To avoid potential problems, start a DOS command prompt using the command "COMMAND.COM" and compile from there. Also, some Windows 95 users may receive a 'Program Requires MS-DOS Mode' pop-up message when compiling. To prevent this annoying message from occurring, select 'Properties' for the MS-DOS window you are compiling from, then select 'Advanced' and ensure that the 'Suggest MS-DOS mode as necessary' box is not checked.

8.3.1 Building and Running the Dhrystone Benchmark

The Dhrystone benchmark is a commonly available integer benchmark. Since the main loop of this benchmark fits into the caches of many processors, its validity as a predictor of system performance may be suspect. It is included here as an example of an application to be built, loaded onto the evaluation board, and executed.

To build the Dhrystone benchmark, enter the command "**make dhry**" from the command line while in the **samples** directory. The makefile will compile the Dhrystone source files, link the resulting object files with the support libraries, and produce the boot file, **dhry**, and the boot image file, **dhry.img**.

If the bootptab entry suggested in the chapter on "Host Configuration" was used, then **dhry.img** must be renamed or copied to **boot.img** in order to be selected by the ROM Monitor load process. Select option 0 from the ROM Monitor screen to load and run the image.

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the RISCWatch **load image** command to load the **dhry.img** file. Once successfully loaded, issue the **attach 42** and **logoff** commands to return control to the ROM Monitor and initiate the run.

You should see the following messages (or ones like them) appear on the ROM monitor screen. Explanations preceded by ## do not appear on the screen but are added here as clarification.

```
Booting from [ENET] Ethernet...
Sending bootp request ...
## This requests the Host workstation to return the name of the boot image

Loading file "/usr/osopen/PLATFORM/samples/boot.img" ...
Sending tftp boot request ...
## Having obtained the file name, the ROM monitor uses tftp to retrieve
the file from the
## host workstation
Transfer Complete ...
Loaded successfully ...
Entry point at 0x23a18 ...
## Having loaded an image, the ROM monitor is now transferring control to
the application
## subsequent messages are from the application

Dhrystone Benchmark, Version 2.1 (Language: C)
Program compiled without 'register' attribute
Please give the number of runs through the benchmark:
```

At this point, enter the number of desired iterations. The test is designed not to give results if the selected iterations completes in less two seconds, so pick a large number (≥ 200000). After the test completes, a check screen will be displayed, followed by the benchmark results. The results may vary based on the system environment.

8.3.2 Building and Running the `usr_samp` Program

The `usr_samp.c` program is included as a sample to be built and run on the EVB. It's a simple program that shows how to properly call the `get_board_cfg()` ROM Monitor user function to determine the ROM Monitor version, the amount of DRAM installed on the board and the Ethernet controller's MAC address. Developers interested in using any of the ROM Monitor user functions should use this program as a guide.

To build the `usr_samp` program, enter the command "**make usr_samp**" from the command line while in the **samples** directory. The makefile will compile the `usr_samp.c` file, link the resulting object file with the support libraries, and produce the boot file, **usr_samp**, and the boot image file, **usr_samp.img**.

If the suggested bootptab was used, then **usr_samp.img** must be renamed or copied to **boot.img** in order to be selected by the Rom Monitor load process. Select option 0 from the ROM Monitor screen to load and run the image.

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the RISCWatch **load image** command to load the **usr_samp.img** file. Once successfully loaded, issue the **attach 42** and **logoff** commands to return control to the ROM Monitor and initiate the run.

You should see the following messages (or ones like them) appear on the ROM Monitor screen.

```
Booting from [ENET] Ethernet...
Sending bootp request ...

Loading file "/usr/osopen/PLATFORM/samples/boot.img" ...
Sending tftp boot request ...
Transfer Complete ...
Loaded successfully ...
Entry point at 0x23e48 ...

Hello 603e user!

Your ROM Monitor version is : 2.1

Your 603e Evaluation Board has 8388608 bytes of DRAM installed.

Your Ethernet controller's network address is : 1000abcdef55

Your processor bus speed is 66 Mhz

Your processor internal speed is 133 Mhz

usr_samp done!
```

The DRAM amount listed should match the amount installed on the board.

8.3.3 Building and Running the timesamp Program

The **timesamp.c** program is included as a sample to be built and run on the EVB. This program is an example of how to properly time a particular function or benchmark. The user must know and define the time base frequency (the number of times the time base register is updated per second) in the **timesamp.c** to ensure the timing calculations are accurate.

To build the **timesamp** program, enter the command "**make timesamp**" from the command line while in the **samples** directory. The makefile will compile the **timesamp.c** file, link the resulting object file with the support libraries, and produce the boot file, **timesamp**, and the boot image file, **timesamp.img**.

If the suggested bootptab was used, then **timesamp.img** must be renamed or copied to **boot.img** in order to be selected by the Rom Monitor load process. Select option 0 from the ROM Monitor screen to load and run the image.

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the RISCWatch **load image** command to load the **timesamp.img** file. Once successfully loaded, issue the **attach 42** and **logoff** commands to return control to the ROM Monitor and initiate the run.

You should see the following messages (or ones like them) appear on the ROM Monitor screen.

```
Booting from [ENET] Ethernet...
Sending bootp request ...

Loading file "/usr/osopen/PLATFORM/samples/boot.img" ...
Sending tftp boot request ...
Transfer Complete ...
Loaded successfully ...
Entry point at 0x23e48 ...

Please give the number of runs through the benchmark:
```

At this point, enter the desired number of runs through the function or benchmark being timed. In this sample, the function being timed should execute for approximately a second, so a number between 1 and 10 would suffice.

8.4 Resolving Execution Problems

Configuration errors in the network or bootp tables cause most of the problems with running the sample applications. This section contains information that will aid users in identifying common problems.

8.4.1 Using the Ping Test on the ROM Monitor to Verify Connectivity

If the ping test fails, verify that TCP/IP is running on the host system and that the IP addresses on the selected interface are correct. The local address refers to the IP address of the evaluation board, and the remote refers to the host workstation address. The host workstation address must match the one selected during configuration of the host network interface. Also consult your TCP/IP documentation to insure proper network configuration.

8.4.2 bootp and tftp Servers (Daemons) for ROM Monitor loads

Insure that the bootp and tftp servers are started on the host workstation. If possible, use the **tftp** command from another workstation to retrieve the load image. If this fails, make sure the image exists in the target directory and that it is readable by “others”. If the tftp transfer succeeds, check the bootptab entry in the **bootptab** file to insure that it specifies the correct interface and IP address of the evaluation board.

8.5 Using OS Open Functions

OS Open provides the following major classes of functions for the embedded programming environment:

- **Thread management**
The unit of execution context for OS Open is the thread as defined by POSIX standards. Functions are provided to create threads with various scheduling and execution attributes. To manage the execution environment, serialization and synchronization primitives are part of OS Open. The system also provides functions to associate data with specific threads.
- **Storage management**
OS Open supports variable block allocations in the form of a heap. Functions are provided to extend the heap, query heap usage, and allocate storage to meet alignment constraints. OS Open also provides an independent storage management mechanism to allocate fixed blocks of storage in constant time.
- **Interrupt and fault support**
OS Open provides functions to attach user-written code to any of the processor exceptions and interrupts. Most of the functions of OS Open can be used in these interrupt handlers, except for those functions that suspend execution or are valid only in the context of an executing thread. When the underlying hardware platforms support it, OS Open platform-specific libraries provide additional functions to attach user-written code to external interrupts supported on the platforms.
- **Clock and timer management**
OS Open functions provide time-of-day clock support and the ability to create, use, and destroy timers. These timers can be one-time or periodic.
- **Device support**
OS Open functions support the installation of user-written device drivers to provide character special files, block special files, and logical file systems. Low-level POSIX I/O (read, write) as well as ANSI C stream (fget, fput) functions are provided for device and regular file access.

- **ANSI C library support**
OS Open provides a comprehensive set of ANSI C functions, providing support for string manipulation, memory management, string-to-number conversion, input/output, nonlocal jumps, and variable arguments.
- **Pseudo device driver support**
OS Open provides several functions, such as TTY and DOS file system functions, that are installed and managed like device drivers, but they do not manipulate actual hardware nor do they have platform or device dependencies.

OS Open provides functions that create and manage TCP/IP sockets. Network interface functions for Token Ring, Ethernet, and Serial Line Interface Protocol (SLIP) are also provided. With the TCP/IP protocol stack and network interfaces, additional functions are provided that implement several popular networking utilities, such as ping, ifconfig, ftp, and telnet.

- **Debug functions and kernel abstract data types**
OS Open provides functions that set, clear, and query breakpoints. OS Open features an internal circular trace buffer for operating system and user events. Also, functions are provided that dump kernel data objects in a readable form.

Application Libraries and Tools

This chapter describes some of the application libraries and tools available in the EVB software support package. See the OS Open *User's Guide* and *Programmer's Reference* for additional information.

9.1 OS Open Libraries

The OS Open operating system comprises a real-time executive and optional libraries of functions and macros.

The real-time executive provides a operating system core for embedded applications. Depending on an application's requirements, an embedded application may also incorporate one or more optional libraries.

This modular approach enables embedded system developers to scale an OS Open operating system to match their application requirements. Because unneeded features are not present, an OS Open configuration can provide savings in system hardware, initialization and reset time, and program size.

Table 9-1 summarizes the OS Open libraries, described in the OS Open User's Guide and in this user's guide. For detailed descriptions of the OS Open functions and macros, refer to the *OS Open Programmer's Reference*.

Table 9-1. OS Open Libraries

Library	File Name	Platforms
Alignment Exception Support Library	alignLib.a	Common
ANSI C Library	cLib.a	Common
ANSI C Math Library	mathLib.a	Common
ANSI C I/O Library	fsLib.a	Common
Block Buffer Library	bbuffLib.a	Common
Bios Ethernet Library	benetLib.a	60x/7xx EVB
Boot Library(DRAM)	bootLib.a	60x/7xx EVB

Table 9-1. OS Open Libraries

Library	File Name	Platforms
C++ runtime support (High C++™ support) Library	cppLib.a, crt1.o, crtn.o, mwdctor.o	ELF
C++ runtime support (C Set ++™ support) Library	cpprtLib.a	XCOFF
Card Services/enabler software layer for PCMCIA support	csLib.a	Common
Clock Support Library and NV-RAM	clockLib.a	60x/7xx EVB
Debug Support Library	dbLib.a	Common
Device and File Support Library	devLib.a	Common
DOS File System Support Library	fatLib.a	Common
Diskette Support Library	dsktLib.a	60x/7xx EVB
Dynamic Loader Library	ldrLib.a	Common
Ethernet Support Library	enetLib.a	60x/7xx EVB
File Transfer Protocol Support Library	ftpLib.a	Common
Floating Point Library	fpeLib.a	Common
Input/output Support Library	ioLib.a	60x/7xx EVB
Kernel Abstract Data Types Library	kadtLib.a	Common
Network Support Library	netLib.a	Common
NFS Support Library	nfsLib.a	Common
OpenShell	shell.o	Common
PCMCIA ATA/IDE Hard disk device driver	pataLib.a	Common
PowerPC Low Level Access Support Library	ppcLib.a	60x/7xx EVB
Queue Library	queLib.a	Common
RAM Disk Library	ramdLib.a	Common
Rate Monotonic Scheduling (RMS) Library	rmsLib.a	Common
Remote Source Level Debug Library	rsldLib.a	Common
Ring Buffer Library	rngLib.a	Common
RPC Support Library	rpcLib.a	Common
Runtime Library	runlib.a	Common
SCSI Support Library	scsiLib.a	Common

Table 9-1. OS Open Libraries

Library	File Name	Platforms
Serial Support Library	asyncLib.a	60x/7xx EVB
Socket Services for PCMCIA support	ssLib.a	Common
Symbol Support Library	symLib.a	Common
TCP/IP Protocol Support Library	tcpiLib.a	Common
Telnet Daemon Support Library	tnetdLib.a	Common
Telnet Client Support Library	telnet.o	Common
The Real-time Executive	rtx.o, rtxLib.a	Common
OS Open Minimal Kernel	rtxmin.o	Common
OS Open Kernel Extensions for the minimal kernel	rtxext.o	Common
Timer Tick Support	tickLib.a	60x/7xx EVB
Trivial File Transfer Protocol	tftp.o	Common
TTY Support Library	ttyLib.a	Common
XL C Compiler Support Library	xlLib.a	XCOFF

The real-time executive, the only required component in an OS Open operating system, provides a full set of basic operating system services.

- Thread management
- Virtual memory management for OS Open with Virtual Memory
- Storage management
- Signals
- Clocks and timers
- Interrupt and fault handling
- Message queues
- Semaphores
- Trace buffer support
- Miscellaneous services

The C functions for the real-time executive functions are in two libraries, **rtx.o** and **rtxLib.a**. The **rtx.o** library contains the OS Open real-time executive. The **rtxLib.a** library contains interface routines to OS Open functions, and is linked with application programs to resolve calls to the real-time executive.

9.2 Using Libraries and Support Software

The object libraries specific to the 60x/7xx EVB are described below.

Table 9-2. OS Open Libraries for the 60x/7xx EVB

Library	File Name
Boot Library(RAM)	bootLib.a
Diskette Block Device Driver Support Library	dsktLib.a
Ethernet Device Driver Support Library	enetLib.a
Input/Output Support Library	ioLib.a
PowerPC Low Level Access Support Library	ppcLib.a
Real-time Clock Interface Support Library	clockLib.a
ROM Monitor Ethernet Interface Library	benetLib.a
Serial Support Library	asyncLib.a
Software Timer Tick Support Library	tickLib.a

9.2.1 Serial Port Support Library

This library supports the serial ports on the 60x/7xx EVB. Use in conjunction with the function provided by **devLib.a** and **fsLib.a** to provide a high level I/O interface to application programs. The serial port support functions reside in the **asyncLib.a** library.

9.2.2 Boot Library(RAM)

This library contains the OS Open bootstrap program for the appropriate platform. The boot library performs initial processing to prepare the completed application program for execution on the platform. For the 60x/7xx EVB, this processing includes moving the loaded program such that real addresses correspond with addresses assumed by the language development tools. The boot library for the 60x/7xx EVB also dynamically determines available heap space and prepares the symbol table for use by OS Open symbol management routines. The boot library does not export any functions.

9.2.3 Input/Output Support Library

The input/output functions reside in the **ioLib.a** library. To initialize the I/O subsystem, you must call **ioLib_init()** (normal mode) or **dbg_ioLib_init()** (ROM Monitor debug/ethernet) before performing any I/O other function.

9.2.4 PowerPC Low-Level Processor Access Support Library

The low-level access support library contains C-callable versions of the special PowerPC instructions. A few of the sample programs use these functions to manipulate the PowerPC 60x/7xx's special registers. These functions provide access to processor instructions not generated by compilers. For example, device drivers often have a requirement to control data caching, disable interrupts, synchronize I/O, and other processor and platform-specific operations. The low-level access support functions reside in the **ppcLib.a** library. Since there are special registers that are not implemented on all PowerPC processor types, not all the functions in **ppcLib.a** work for all PowerPC processor types. In Chapter 10, "60x/7xx EVB Function Reference", within each explanation for a function there is a list of processor types that the function is valid for.

9.2.5 ROM Monitor Ethernet IP Interface Library

This library contains routines allowing access to the ROM Monitor's Ethernet IP interface. These functions allow the Ethernet to be simply configured with a unique IP address for use with TCP/IP functions. The ROM Monitor Ethernet IP Interface functions reside in **benetLib.a** library. The **benetLib.a** functions are only available with OS Open without Virtual Memory.

9.2.6 Real-time Clock Interface Support Library

This library contains routines to read and set the 60x/7xx EVB battery-backed real-time clock. These functions are not to be confused with the real-time clock functions provided directly by OS Open when the system is running. The real-time clock interface support functions reside in the OS Open's **clockLib.a** library and are available to perform the following features:

Set the OS Open clock from the real-time clock.

Set the real-time clock from user-supplied data.

Calibrate the real-time clock chip.

Read and write NV-RAM in the clock chip.

9.2.7 Diskette Block Device Driver Support Library

This library provides the support for sector level access to the diskette drive on the workstation platform by using the OS Open block device driver interface. The diskette block device driver support functions reside in the **dsktLib.a** library.

9.2.8 Ethernet Device Driver Support Library

This library provides support for the integrated Ethernet of the 60x/7xx EVB. The Ethernet device driver support functions reside in the **enetLib.a** library.

9.2.9 Software Timer Tick Support Library

The OS Open system requires a periodic call to **timertick_notify()** to maintain internal clocks and timer functions. The **tickLib.a** library contains an implementation of the **timertick_notify()** function for PowerPC architecture machines. Timer tick support functions reside in the **tickLib.a** library.

9.3 Device Drivers Supplied with the 60x/7xx EVB

Device drivers provided with the 60x/7xx EVB include.

- Asynchronous
- Diskette
- Ethernet

Examples and references are provided where appropriate.

For more information about any of the OS Open functions mentioned in this chapter, refer to the *OS Open Programmer's Reference*.

9.3.1 Asynchronous Device Driver

The asynchronous device driver supports the asynchronous communication port found on the . Following is a brief functional description of the device driver.

- Support from 50 baud.
- Full duplex modem line control discipline.
- Overrun error, parity error, and framing error detection.
- BREAK interrupt detection.
- Support for data length of 5, 6, 7, and 8 bits.
- Support for 1, 1.5 and 2 stop bits.
- Support for receive and transmit parity.
- Support for odd and even parity.
- Support for transmitting BREAK.
- Support for 16 byte FIFO in the universal asynchronous receiver transmitter (UART).
- Programmed I/O (PIO) interrupt-driven slave communication.
- Interrupt driven input/output.
- Polled output functions.

Since only full duplex modem line control discipline is supported, connection between the asynchronous port and another device must be made through a "NULL" modem. A NULL modem is a device that crosses transmitted data and received data pins to enable communication. The only time a NULL modem is not necessary is when connection is made to a real modem device.

9.3.1.1 Device Driver Installation

The asynchronous device driver is installed by calling **driver_install()**. Following is an example of asynchronous device driver installation.

```
#include <sys/asyncLib.h>
int devhandle;
rc=driver_install(&devhandle, async_init);
```

async_init() is declared in the file **<sys/asyncLib.h>** as follows.

```
int async_init(driver_t *dsw, va_list vargs)
```

Upon successful installation, **driver_install()** returns 0; otherwise –1 is returned. For more information on **driver_install()**, refer to the *OS Open Programmer's Reference*.

9.3.1.2 Device Installation

After the asynchronous device driver is installed, named devices can be created using **device_install()**. Following is an example of device installation.

```
rc=device_install("/dev/s0", CHRTYPE, devhandle, 1, 128,  
128,1843200,0x80003f8,1,EXT_IRQ_COM1);
```

For device installation, *devhandle* is the value obtained from the **driver_install()**. Device type CHRTYPE is defined in **<sys/devDriver.h>**.

Additional parameters passed in the **device_install()** call are as follows.

Parameter	Meaning
Fourth Parameter	Port number to be installed (1 or 2)
Fifth Parameter	Size of write buffer
Sixth Parameter	Size of read buffer
Seventh Parameter	Input clock for the divisor
Eight Parameter	UART base register address 0x8003F8 or 0x8002F8
Ninth Parameter	UART register address delta, always 1
Tenth Parameter	Interrupt IRQ_MIN < event < IRQ_MAX (0 < event < 31)

These are positional parameters.

Note: Write and read buffer sizes indicate number of characters that can be buffered in the device driver.

Upon successful installation, **device_install()** returns 0; otherwise –1 is returned. When the device is installed, error reporting for the device is turned off and xon/xoff pacing is enabled. For more information on **device_install()**, refer to the *OS Open Programmer's Reference*.

9.3.1.3 Opening Asynchronous Communication Ports

After the device is installed, the **open()** system call can be used to open a particular device. Following is an example of the **open()** system call used against the asynchronous port.

```
fd1=open("/dev/s0", O_RDWR, asyncParityNone, asyncParityOdd,  
        asyncStopBits1, asyncDataBits8, 9600);
```

Additional parameters passed in **open()** are as follows.

Parameter	Meaning
First Parameter	Check/generate parity flag. Valid values are: <code>asyncParityNone</code> and <code>asyncParityGen_Check</code>
Second Parameter	Parity type. Valid values are <code>asyncParityEven</code> and <code>asyncParityOdd</code> . Because parameters are positional, this parameter must be specified even if parity is not used.
Third Parameter	Number of stop bits. Valid values are <code>asyncStopBits1</code> and <code>asyncStopBits2</code> .
Fourth Parameter	Data length. Valid values are <code>asyncDataBits5</code> , <code>asyncDataBits6</code> , <code>asyncDataBits7</code> , and <code>asyncDataBits8</code> .
Fifth Parameter	Baud rate. Valid values range from 50 baud.

These are positional parameters. All parameter constants can be found in **<sys/ioctl.h>**.

Note: The *oflag* parameter, `O_RDWR` in this example, which is passed in the **open** call, is ignored by the device driver. When successful, **open()** returns a file descriptor, otherwise `-1` is returned. **open()** can be called multiple times against the same asynchronous port. Communication parameters passed during the last **open()** call are set in the asynchronous port. For more information on **open()**, refer to the *OS Open Programmer's Reference*.

9.3.1.4 Reading and Writing

After successfully installing and opening the asynchronous port, **read()** and **write()** calls can be issued against that port. Multiple threads can issue **read()** and **write()** calls to the same port at the same time. However, simultaneous **read()** calls issued to the same port may block or be processed in an unexpected order. For these instances, thread scheduling and synchronization must be handled by the application.

Following is an example of **read()** and **write()** calls.

```
rc=write(fd1, "\nOS Open Real-time Executive\n", 29);  
rc=read(fd1, buffer, 10);
```

fd1 is the value obtained from the **open()** call.

Note: For more information on **read()** and **write()**, refer to the *OS Open Programmer's Reference*.

9.3.1.5 I/O Control

An **ioctl()** call issued against asynchronous device driver accepts the commands listed in Table 9-3. All parameter constants can be found in **<sys/ioctl.h>**

Table 9-3. ioctl() Commands for Asynchronous Device Drivers

Command	Parameters	Explanation
ASYNCBAUDSET	Value from 50	Sets baud rate
ASYNCBAUDGET	Pointer to integer	Returns baud rate
ASYNCTRIGSET	asyncFifoTrigger1, asyncFifoTrigger4, asyncFifoTrigger8, asyncFifoTrigger14	Sets FIFO trigger level for asynchronous port
ASYNCTRIGGET	Pointer to integer	Returns current trigger level
ASYNCBREAKSET	None	Starts sending BREAK on port
ASYNCBREAKCLR	None	Stops sending BREAK on port
ASYNCSTICKGET	Pointer to integer	Returns the way the parity bit is interpreted by the port
ASYNCSTICKZERO	None	Disables stick parity
ASYNCSTICKONE	None	Parity interpretation tracks even/odd parity
ASYNCRERRORGET	Pointer to integer	Returns and clears read error conditions. Values are defined in asyn- cLib.h
ASYNCWERRORGET	Pointer to integer	Returns and clears write error conditions. Values are defined in asyn- cLib.h
ASYNCERROREN	None	Enables error reporting
ASYNCERRORDIS	None	Disables error reporting. All pending errors are cleared
ASYNCERRORGET	Pointer to integer	Returns error reporting enabled flag
ASYNCLENGET	Pointer to integer	Returns current data length
ASYNCLENSET	asyncDataBits5, asyncDataBits6, asyncDataBits7, asyncDataBits8	Sets data length
ASYNCSTOPGET	Pointer to integer	Returns number of stop bits

Table 9-3. ioctl() Commands for Asynchronous Device Drivers

Command	Parameters	Explanation
ASYNCSTOPSET1	None	Sets number of stop bits to 1
ASYNCSTOPSET1_5	None	Sets number of stop bits to 1.5
ASYNCSTOPSET2	None	Sets number of stop bits to 2
ASYNCPARITYNONE	None	Disable parity
ASYNCPARITYGEN	None	Enable parity
ASYNCPARITYSGET	Pointer to integer	Return parity status (enabled/disabled)
ASYNCPARITYODD	None	Sets parity to odd
ASYNCPARITYEVEN	None	Sets parity to even
ASYNCPARITYGET	Pointer to integer	Returns parity type
ASYNCXONENABLE	None	Enables XON/XOFF flow control
ASYNCXONDISABLE	None	Disables XON/XOFF flow control
ASYNCXONGET	Pointer to integer	Returns XON/XOFF flow control status
ASYNCMODEMSTAT	Pointer to integer	Returns modem status
ASYNCFLUSHIN	None	Flushes input buffer
ASYNCFLUSHOUT	None	Flushes output buffer
ASYNCDRAIN	None	Blocks until all characters in output buffer have been transmitted
ASYNCIGNBREAK	None	Ignores break interrupts
ASYNCsigBREAK	None	Sends SIGINT on reception of break condition
ASYNCERRBREAK	None	Returns error from read upon reception of break condition. 0x00 is placed in the receive buffer at the position where break occurred.

Following is an example of an **ioctl()** call issued against an asynchronous device.

```
rc=ioctl(fd1, ASYNCXONDISABLE);
if (rc !=0) printf("ioctl failure\n");
```

fd1 is the value obtained from the **open()** call.

9.3.1.6 Polled Asynchronous I/O

A function is provided for polled output to s1 and s2 serial port.

```
int s1dbprintf(unsigned long uart_clock, unsigned char *base_reg,
int reg_delta, event_t event, const char *format, ...)
int s2dbprintf(unsigned long uart_clock, unsigned char *base_reg,
int reg_delta, event_t event, const char *format, ...)
```

The parameters passed to these functions are identical to **printf()** except for *uart_clock*, *base_reg*, *reg_delta*, and *event*. *uart_clock* specifies the clock speed, *base_reg* specifies the address of the base UART register, *reg_delta* specifies the address space between UART registers, and *event* specifies the external interrupt level. Because polled I/O transmits characters synchronously, these functions may be called from first level interrupt handlers (FLIHs) or a user-supplied panic function. Since the function waits until the characters are actually sent before returning, use of this with long strings can significantly affect the timing of calling programs.

9.3.2 Diskette Device Driver

The diskette driver provides a block device driver interface for the diskette drive on 60X EVB. Function highlights are:

- Support for 720KB, and 1.44MB media. The media type is automatically determined by the device driver.
- Support for media changed line allows file systems to be notified when diskette is removed or changed.
- Media access deferred until device read time.

9.3.2.1 Device Driver Installation

The diskette device driver is installed by calling **driver_install()**. Following is an example of diskette device driver installation.

```
#include <dsktLib.h>
int devhandle;
rc=driver_install(&devhandle, dskt_init);
```

Upon successful installation **driver_install()** returns 0, otherwise -1 is returned. For more information on **driver_install()**, refer to the *OS Open Programmer's Reference*.

9.3.2.2 Device Installation

After the diskette device driver is installed, named diskette devices can be installed using **device_install()**. Following is an example of device installation.

```
rc=device_install("/dev/dskt0", BLKTYPE, devhandle,
unit_number); /* integer specifying which of 4
possible
diskette units supported by the
```

```
controller chip */
```

For the 60x/7xx EVB platform, one diskette is supported at unit position 0. For device installation, *devhandle* is the value obtained from the **driver_install()**. Device type BLKTYPE is defined in **<sys/devDriver.h>**.

Upon successful installation, **device_install()** returns 0; otherwise -1 is returned. When the device is installed, files may be opened against the diskette device.

9.3.2.3 Opening Diskette Files

After the device is installed, the **open()** system call can be used to open a particular device. Following is an example of the **open()** system call used against a diskette device.

```
fd1=open("/dev/dskt0", O_RDWR );
```

When successful, **open()** returns the open file descriptor; otherwise -1 is returned. For more information on **open()**, refer to the *OS Open Programmer's Reference*.

9.3.2.4 Reading and Writing

After successfully installing and opening the diskette device, the **strategy()** function is used to read and write blocks. The diskette is treated as a linear collection of 512-byte blocks. The device driver translates addresses to or from the cylinder, head, or sector. An example of reading, clearing, and writing a block is shown below.

```
unsigned char myblock[512];
block_req_t   bk_request;

bk_request.request_type = BLOCK_READ;
bk_request.block_buffer = myblock;
bk_request.block_id = 50;           /* read block 50 from
diskette */
bk_request.block_count = 1;
rc = strategy(fd1, &bk_request);
if (rc != 0) trouble();
memset(myblock, 0x00, sizeof(myblock));
bk_request.request_type = BLOCK_WRITE;
rc = strategy(fd1, &bk_request);
if (rc != 0) trouble();
```

fd1 is the value obtained from the **open()** call.

For more information on **strategy()**, refer to the *OS Open Programmer's Reference*.

9.3.2.5 I/O Control

The **ioctl()** call issued against the diskette device driver accepts following commands. In each of these commands, *fd* is the value obtained from the **open()** call.

9.3.2.6 QDEVATTR

This command queries the characteristics of the block device. This ioctl is common across all block device drivers.

```
block_dev_attr_t block_attr;
rc = ioctl(fd, QDEVATTR, &block_attr);
```

The REMOVABLE_MEDIA mask can be used to test the returned attribute, *block_attr*, to determine if the media is removable. REMOVABLE_MEDIA is defined in **<sys/blockdev.h>**.

9.3.2.7 QDSKTATTR

This command queries the characteristics of the diskette in the drive.

```
dsktattr_t dskt_attr;
rc = ioctl(fd, QDSKTATTR, &dskt_attr);
```

By masking the returned attribute, *dskt_attr*, with one of the following constants, this command can be used to determine the characteristics of the diskette in the drive.

- The DSKT_VALID mask can be used to test the returned attribute, *dskt_attr*, to determine if there is a valid diskette in the drive.
- The DSKT_WP mask can be used to test the returned attribute, *dskt_attr*, to determine if a diskette is write protected.
- The DSKT_DMASK mask can be used to test the returned attribute, *dskt_attr*, to determine whether the diskette type is 720KB (MPT720MB) or 1.44MB (M1PT44MB).

These constants are defined in **<dsktLib.h>**.

9.3.2.8 BLKMEDIA_CHANGE

This command queries the drive to check if the diskette has been removed and possibly replaced. The driver determines this by examining the drive changed line triggered by the removal of the diskette from the drive mechanism.

```
rc = ioctl(fd, BLKMEDIA_CHANGE );
```

The return code from the **ioctl()** call indicates one of the following conditions:

- | | |
|----|--------------------------------------------------------------|
| 0 | Diskette has not been changed. |
| 1 | Diskette changed, but a valid diskette is now in the drive. |
| -1 | Diskette removed or an invalid diskette is now in the drive. |

9.3.2.9 Diskette Device Driver Example

Following is a complete diskette device driver example.

```
#include <fcntl.h> /* File control flags */
#include <sys/ioctl.h> /* ioctl subcommand defines */
```



```

#include <unistd.h>                /* ioctl definition */
#include <sys/devLib.h>            /* device library definitions
*/
#include <sys/blockdev.h>          /* block device definitions */
#include <stdio.h>
#include <sys/devDriver.h>         /* device driver definitions */
#include <dsktLib.h>               /* diskette definitions */
/* This Example tests the ioctl() subcommands of the diskette
device driver */
int dskt_ioctl_example(void)
{
    block_dev_attr_t block_attr;
    int diskettefd;
    dsktattr_t dskt_attr;
    int rc, devhandle;
    unsigned int temp;
    rc=driver_install(&devhandle, dskt_init);
    if (rc != 0) printf("driver install fail for dskt\n");
    rc=device_install("/dev/dskt0", BLKTYPE, devhandle, 0);
    if (rc != 0) printf("device install fail for dskt0\n");
    diskettefd = open("/dev/dskt0", O_RDWR);
    if (diskettefd < 0) printf("open fail for dskt0\n");
    rc = ioctl(diskettefd,QDEVATTR, &block_attr);
    if (rc != 0) printf("ioctl(QDEVATTR) fail, rc = %d\n",rc);
    temp = block_attr & REMOVABLE_MEDIA;
    if (temp != REMOVABLE_MEDIA)
        printf("Removable media flag fail, attr = %u\n",block_attr);
    printf("Place a WRITE PROTECTED DOS formatted disktte\n");
    printf("in the diskette drive and press Enter\n");
    fflush(stdin);
    getchar();
    rc = ioctl(diskettefd,BLKMEDIA_CHANGE);
    if (rc == -1) printf("Block media failure\n");
    rc = ioctl(diskettefd,QDSKTATTR, &dskt_attr);
    if (rc != 0) printf("ioctl QDSKTATTR fail\n");
    temp = dskt_attr & DSKT_VALID;
    if (temp != DSKT_VALID) printf("DSKT_VALID flag fail, attr =
%u\n",dskt_attr);
    temp = dskt_attr & DSKT_WP;
    if (temp != DSKT_WP)
        printf("Diskette is NOT write protected, attr =
%d\n",dskt_attr);
    temp = dskt_attr & DSKT_DMASK;
    if (temp == MPT720MB) printf("720KB density\n");
    else if (temp == M1PT44MB) printf("1.44MB density\n");
    else printf("Invalid density\n");
}

```

9.3.3 Ethernet Device Driver

The Ethernet device driver is a character device driver supporting packet level read/writes to the integrated Ethernet controller. The driver features the ability to open multiple files. Each file receives packets for a specific standard Ethernet or 802.3 address.

Function highlights are:

- Up to 8 receive channels
- Size of receive buffer pool determined by user at driver install time.

9.3.3.1 Device Driver Installation

The Ethernet device driver is installed by calling the **driver_install()** function. Following is an example of Ethernet device driver installation.

```
#include <enet.h>
int devhandle;
char mac_array[6];
#define ENET_RECEIVE_BUFFERS 128
static char enet_buffer[ENET_RECEIVE_BUFFERS*296 + 1568 + 31];

/* vv For OS Open with Virtual Memory ONLY! vv */
void * io_memory;
board_config_ptr = (struct board_cfg_data *)_callxlc(BOARD_CFG_FP);
vm_translate(pthread_gettkernel_np(), boadr_cfg_ptr, &mem);
vm_allocate(pthread_gettkernel_np(), &mem,
    (void *)((unsigned long)board_cfg_ptr&0xFFFFF000),
    0x2000, 0,
    VM_PROTECT_WRITE | VM_PROTECT_READ |
    VM_ATTR_CACHEON);
vm_addrealpages(pthread_gettkernel_np(), mem, 0x2000,
    (void *)((unsigned long)board_cfg_ptr&0xFFFFF000),
    VM_ATTR_IOSPACE);
vm_allocate(pthread_tggetkernel_np(), &io_memory, (void*)0x80100000,
    0x1000, 0,
    VM_PROTECT_ALL);
vm_addrealpages(pthread_tggetkernel_np(), io_memory, 0x1000,
    (void *) 0x80100000, VM_ATTR_IOSPACE);
/* ^^ For OS Open with Virtual Memory ONLY! ^^ */

zapatos_enet_setup()
rc=driver_install(&devhandle, enet_init,
    /enet_intlvl, /* Interrupt level, typically 22
    (EXT_IRQ_MPIC_ETHERNET)*/
    register_base, /* NIC base register address, typically
0x80100000*/
    pci_mem_addr, /* typically 0x80000000 */
    num_blocks, /* number of receive buffers allocated for
```

```

driver use */
enet_buffer,      /* must be doubleword aligned */
mac_array);      /* location of universal MAC address */

```

pci_mem_addr specifies the amount to add to a system memory address to create the address that the Ethernet chip should use to access system memory using PCI bus master cycles.

enet_buffer points to a physically contiguous buffer memory location for the device driver to use. The buffer must be at least $296 * \text{numblocks} + 1568$ bytes, 32 byte aligned, and must not cross a 16 megabyte address boundary. If *pci_mem_addr* is NULL, the device driver will attempt to allocate the needed buffer space based on the value of *num_blocks*.

Note: The device driver can not allocate memory that is guaranteed to be physically contiguous in OS Open with Virtual Memory, so in this case *enet_buffer* must point to the buffer to be used.

mac_array points to the 6 byte ethernet hardware address. Typically this value is obtained from the ROM Monitor's **get_board_cfg()** function. If *mac_array* is NULL, the Ethernet device driver will use the hardware address that is stored in the Ethernet serial PROM that is installed in the 60x7xx EVB.

Upon successful installation, **driver_install()** returns 0; otherwise -1 is returned. For more information about the **driver_install()** function, refer to the *OS Open Programmer's Reference* and the OS Open samples thread0.c file.

9.3.3.2 zapatos_enet_setup()

Before the Ethernet device driver is installed, the function **zapatos_enet_setup()** must be called to map the Ethernet chip to system address 0x8010 0000 (I/O space). With OS Open with Virtual Memory the I/O space for the Ethernet controller must be allocated as VM_ATTR_IOSPACE before **zapatos_enet_setup()** is called. Refer to the OS Open sample file thread0.c for an examples. **zapatos_enet_setup()** takes no parameters and returns 0 if successful;

9.3.3.3 Device Installation

After the Ethernet device driver is installed, Ethernet devices can be installed using the **device_install()** function. Following is an example of device installation.

```
rc=device_install("/dev/en0", CHRTYPE, devhandle);
```

For device installation, *devhandle* is the value obtained from the **driver_install()**. Device type CHRTYPE is defined in **<sys/devDriver.h>**.

Upon successful installation, **device_install()** returns 0; otherwise -1 is returned. At this point, files may be opened against the Ethernet device.

9.3.3.4 Opening and Closing Ethernet Files

After the device is installed, the **open()** system call can be used to open a particular device. Following is an example of the **open()** system call used to open an Ethernet port.

```
fd1=open( "/dev/en0", O_RDWR );
```

When successful, **open()** returns the open file descriptor; otherwise -1 is returned. **open()** can be called multiple times against the same Ethernet device.

When using the **close()** function, the call to the driver-specific **close()** is deferred until all open files on the device are closed. This means that when an Ethernet file is closed, the channel address associated with the file will not be freed if another Ethernet file is open. Be aware that if the Ethernet interface has been connected to the TCP/IP protocol stacks via **enet_attach()**, there will always be a file open against the Ethernet device, and therefore no channel addresses will be freed even if all the files the application opened are closed. To insure that the channel address will be freed, the **ENET_CLEAR_CHANNEL ioctl()** should always be called for an Ethernet file before closing it.

For more information about the **open()** and **close()** functions, refer to the *OS Open Programmer's Reference*.

9.3.3.5 Reading and Writing

After successfully installing and opening the Ethernet port, the **write()** function can be issued. The write buffer must contain a complete Ethernet packet. The universally administered address that was found in the ISA card read only storage (ROS) passed to **driver_install()** will be copied into the source address field by the device driver. There are prototype Ethernet header structures for both standard Ethernet and 802.3 Ethernet packets in **<enet.h>**. Note that packets must be between 60 and 1514 byte in length (inclusive).

Before reading from the Ethernet file, an additional step must be performed. The Ethernet device driver supports up to 8 receive channels. What this means is that up to 8 files can be open for read or read/write simultaneously, and files will receive only those packets that have been selected for them. Packet selection is by packet type, in the case of standard Ethernet, and by destination SAP in the case of 802.3 Ethernet. The selection address is set with the **ioctl ENET_SET_CHANNEL** command, discussed below.

fd1 is the value obtained from the **open()** call.

```
fd1 = open( "/eno", O_RDWR );
ioctl( fd, ENET_SET_CHANNEL, 5, 2 );
/* send packet from buffer */
write( fd, buffer, count );
/* get received packet into buffer */
read( fd, buffer, count );
close( fd );
```

For more information on **read()** and **write()** functions, refer to the *OS Open Programmer's*

Reference.

9.3.3.6 I/O Control

The **ioctl()** call issued against the Ethernet device driver accepts the following commands. In each of these commands, *fd* is the value obtained from the **open()** call.

9.3.3.7 ENET_SET_CHANNEL

This command sets the receive channel address of the file. Once set, a receive channel address cannot be used in a subsequent **ioctl** **ENET_SET_CHANNEL** command unless it is first cleared with the **ioctl** **ENET_CLEAR_CHANNEL** command.

```
rc = ioctl(fd, ENET_SET_CHANNEL,
           packet_type, /* packet type is an unsigned integer
                        containing the channel address */

           type_length); /* specifies how many of the
                        least significant bytes of
the
                        packet type are to be used.
                        Only values 1 and 2 are
valid. */
```

A word about packet addresses. For standard Ethernet, the packet type is a 2-byte field right after the hardware source address. If *type_length* is 2, the *packet_type* parameter is assumed to refer to a standard Ethernet packet type. For a *type_length* of 1, the *packet_type* is assumed to contain a 1-byte destination SAP.

The incoming packets are differentiated as follows: For 802.3, there is a length field immediately after the source address. By convention, Ethernet packets are 1500 bytes or less, and valid Ethernet types are > 0x600. Hence, if the field after the source address is less than 0x600, the packet is assumed to be an 802.3 packet, and the 1 byte *packet_type* is compared against the destination SAP. Some reserved type values should not be generally used. They are defined in the file <netinet/if_ether.h>.

9.3.3.8 ENET_CLEAR_CHANNEL

This command clears the receive channel address of the file. This enables the device driver to free up internal resources and return any unread packets on this channel to the receive buffer pool. Once the receive channel address is cleared, it can be used again with the **ioctl** **ENET_SET_CHANNEL** command. The file can then be set to another receive channel as well.

```
rc = ioctl(fd, ENET_CLEAR_CHANNEL);
```

9.3.3.9 ENET_QUERY_ADDRESS

This **ioctl** command retrieves the universally administered address that was assigned during

device_install.

```
unsigned char ua_address[6];
rc = ioctl(fd, ENET_QUERY_ADDRESS, ua_address);
```

The address is copied into the area supplied as the first data parameter to this ioctl.

9.3.3.10 Ethernet Device Driver Example

Following is an Ethernet device driver example.

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/devLib.h>
#include <sys/devDriver.h>
#include <sys/ioctl.h>
#include <enet.h>
#define ENET_LENGTH 256
#define PACKET_TYPE 0x7C0
#define ENET_INTLVL 11
#define ENET_REGBASE 0xEA00007
#define ENET_REGDELTA 8
#define ENET_BUFFERS 30
#define ENET_SRAMSZ 8
/* physical Ethernet addresses */
static char target_address[6] = {0x08, 0x00, 0x5A, 0x4D, 0x03,
0x9C};
static char host_address[6] = {0x08, 0x00, 0x5A, 0xCD, 0x00, 0xF5};
/* Install the Ethernet device driver and an Ethernet device, and
open an */
/* Ethernet file. Set the receive channel address for the file,
build a */
/* frame, and write it out to the predetermined
target. */
/* The target will have set up an Ethernet file with the same
receive */
/* channel address on its end, and will use read() (possibly
in */
/* conjunction with select()) to receive the
frame. */
int enet_example(void)
{
    int rc;
    int devhandle;
    int enet_fd;
    int enet_length = ENET_LENGTH;
    int frame_number = 1;
    struct enet_frame *e_frame_ptr;
    rc = driver_install(&devhandle, enet_init, ENET_INTLVL,
```

```

ENET_REGBASE, ENET_REGDELTA, 0, 0, ENET_BUFFERS, ENET_SRAMSZ,
host_address);
    if (rc != 0)
    {
        perror("Driver install failure for Enet");
        return(-1);
    }
    rc = device_install("/dev/en0", CHRTYPE, devhandle);
    if (rc != 0)
    {
        perror("Device install failure for Enet");
        return(-1);
    }
    enet_fd=open("/dev/en0", O_RDWR);
    if (enet_fd == -1)
    {
        perror("Ethernet open failure\n");
        return(-1);
    }
    rc = ioctl(enet_fd, ENET_SET_CHANNEL, PACKET_TYPE, 2);
    if (rc != 0)
    {
        perror("Ethernet ENET_SET_CHANNEL ioctl failure\n");
        return -1;
    }
    printf("\nEthernet file opened with file descriptor %d on
channel %x\n", \
        enet_fd, PACKET_TYPE);
    /* build frame */
    e_frame_ptr = (struct enet_frame *) malloc(enet_length);
    memcpy(e_frame_ptr->dest_addr,
        target_address, sizeof(target_address));
    e_frame_ptr->type = PACKET_TYPE; /* set type */
    sprintf(e_frame_ptr->enet_data, \
        "Frame number %d from physical addr 0x%2X%2X%2X%2X%2X%2X,
\
        channel 0x%X", frame_number, \
        host_address[0], host_address[1], host_address[2], \
        host_address[3], host_address[4], host_address[5], \
        e_frame_ptr->type);
    /* write frame */
    rc = write(enet_fd, e_frame_ptr, enet_length);
    free(e_frame_ptr);
    if (rc != enet_length)
    {
        perror("Enet write failure");
        return(-1);
    }
    return(0);

```

}

9.4 Environment Bringup and Initialization

The following section describes the processing that occurs when the evaluation board environment is initialized.

Upon power-up or reset the ROM Monitor initializes the processor and other peripherals on the board. If a ROM Monitor load is attempted (via option 0), all enabled power-on tests are executed and, following their completion, a bootp request is sent to the host. This request involves an exchange of UDP packets corresponding to the bootp protocol. In essence, the ROM Monitor asks for and is supplied with the name of the boot image file on the host workstation. **tftp** (Trivial File Transfer Protocol) is then initiated by the ROM Monitor to transfer the boot image to the evaluation board.

Once the file has been transferred, two simple checks are made. A “magic number” in the boot image’s 32-byte header verifies that the image is one that can be loaded by the ROM Monitor (ie., a file created by the `eimgbld` or `nimgbld` tool - see appendix B for details of the load format). The ROM Monitor also checks that the supplied boot image’s start address does not overlay sections of reserved DRAM. After the load is complete, control is transferred to the specified entry point in the boot image, which is in the bootstrap program.

When using RISCWatch’s **load image** command to load a boot image file, the debugger strips off the file’s 32-byte header and loads the remaining bytes of the file onto the board. The start address of the load is designated in bytes 4-8 of the header. Once loaded, the IAR register is set to the boot image’s entry point as defined in bytes 16-19 of the header. This entry point is in the bootstrap code. See the “Running Your Programs” section in the *RISCWatch Debugger User’s Guide* for additional information on loading files.

9.4.1 Board bootstrap

The source for OS Open’s bootstrap code is included in the **samples\bootLib** directory. The bootstrap program performs the following functions:

1. Unpacks the boot image format, placing the **.text** and **.data** sections in the addresses specified at link time.
2. Modifies the kernel configuration block with new heap size and start address.
3. Sets the **.bss** section to zeros, in accordance with ANSI C requirements.

9.4.2 Environment Initialization

OS Open requires information about the system environment at initialization. The following source files, which are included with the samples, are used to supply that information and to establish the working environment.

- `basic_os.c` - contains pieces of `config.c`, `io_init.c`, `panic.c`, `thread0.c`, and `utils.c` to provide a minimal OS Open configuration.

- `config.c` - configures the OS Open kernel
- `io_init.c` - initializes OS Open's I/O subsystem
- `network.c` - configures the host names and addresses for your environment
- `panic.c` - provides a sample panic function
- `thread0.c` - configures various features of OS Open (networking, remote debugger, etc.)
- `utils.c` - provides some useful utilities such as `dir()` to produce a directory listing

Additional information can be found in the “Configuring the OS Open Operating System” and “Developing OS Open Applications” chapters in the *IBM OS Open User's Guide*.

9.5 Tools

Several host tools are provided to assist you in using the EVB support package or creating your own applications for the PowerPC 60x/7xx. The tools can also be used for ROM program development.

9.5.1 elf2rom and xcofrom

elf2rom takes an ELF format executable file (output from the linker/binder), extracts the text and data sections, and writes them to a binary file for use as input to a ROM programmer. This tool can be used by those who wish to modify the ROM Monitor source code and create a new flash memory binary file for use with a ROM programmer or the flash update utility included with EVB software.

xcofrom works similarly on XCOFF file format input files.

Syntax:

```
elf2rom [-v] [-d] [-p] [-s size] [-i offset] [-o output_file] input_elf
```

```
xcofrom [-v] [-d] [-p] [-s size] [-i offset] [-o output_file] input_xcoff
```

Description:

The program takes the input file *input_elf*, for **elf2rom** (which is assumed to be an ELF file output from the linker) and *input_xcoff* for **xcofrom** (which is assumed to be an XCOFF file output from the linker), extracts the text and data sections, and writes them to the file, *output_file*. There are several optional flags that can affect elf2rom and xcofrom processing. They are described below.

- | | |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -v | The verbose flag causes information about the generated output file to be written to stderr at the completion of the utility. This information includes the sizes and origins of the various sections and entry point. |
| -d | The debug flag will cause the symbol information from the input ELF file to be included after the data section in the output binary file. |
| -p | The promotion flag causes the data section to be aligned on a full word boundary if possible. This alignment facilitates full word moves of data to the appropriate target address without causing alignment exceptions. |
| -s | The size flag causes the output binary file to be padded to a particular size. This option is useful if it is necessary to create binary files that are the same size as a target ROM device. Error messages are generated if the generated image exceeds the specified size. |
| -i offset | The info flag places an information block into the output binary file at the specified offset. Since this info block overlays what is currently in the file at the specified offset, space should be reserved for its placement. The info block contains the following fields. |

long block_id	Magic Number 0xBFAB0030
long entry_point	entry point of image
long toc_ptr	used for XCOFF; not used for ELF
long text_size	size of text section in bytes also offset from beginning of image to data section
long text_p_addr	text origin address as generated in ELF module
long data_size	size of data section
long data_p_addr	data origin as specified in generated ELF module
long bss_size	size of bss section
long bss_p_addr	bss origin as specified in generated ELF module
long num_syms	number of symbols from symbol section only valid if debug flag is set)
long sym_p_addr	address of symbol table. Calculated as text origin + offset of symbols with created ROM image
long text_offset	offset of text section from beginning of original ELF file. This information is required by certain debuggers
-o output_file	Allows the specification of an output file name. The default name is input_elf.img.
input_elf	This is simply the ELF binary input file. (elf2rom only)
input_xcoff	This is simply the XCOFF binary input file. (xcofrom only)

The following picture shows the relationship of the various sections in the produced output file. The figure assumes that the info block flag [-i] was specified with an offset of 0x00.

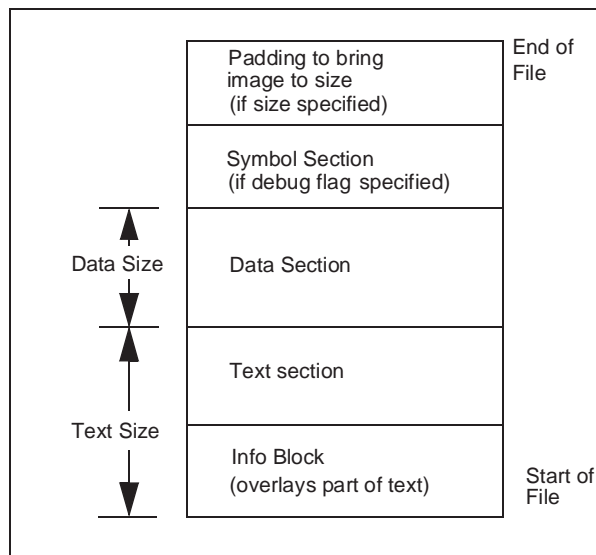


Figure 9-1. elf2rom and xcofrom Output File

Users can find an example of using elf2rom in the ROM Monitor's Makefile under **osopen/PLATFORM/openbios**.

9.5.2 hbranch

hbranch places a branch at the end of a ROM image. **hbranch** can also be used to store a communication device's network address in the ROM's Vital Product Data (VPD) area.

Syntax:

```
hbranch [-v] [-s size] [-n net_addr] input_image
```

Description:

The program takes the input file *input_image* (which must be the output of elf2rom, xcofrom, nimgbld, or eimgbld with an information block at 0x0 relative) pads it to size *size* and writes a relative branch to the entry point recorded in the end of the image. The entry point must be a label, not a function descriptor. There are several optional flags that can affect **hbranch** processing. They are described below.

-v	The verbose flag causes information about the generated output image to be written to <i>stderr</i> at the completion of the utility. This information includes entry point information.
-s size	The size flag causes the image to be padded to a particular size. This facility is useful if it is necessary to create binary images that are the same size as a target ROM device.
-n net_addr	The network address flag stores net_addr, a 12 hex character network address (the media access control (MAC) address), in the VPD area in ROM. The ROM Monitor uses this option to store the EVB's ethernet controller's network address in its VPD.
-p patch_file	The patch file flag causes the file <i>patch_file</i> to be placed into the image just before the final branch and logically inserted into the instruction stream between the branch at the end of the file and the entry point. The patch file is inserted into the image "as is" and will usually contain the binary representation of position independent executable instructions. See Figure 9-2 for the details as to how normal hbranch processing is changed by a patch file.
input_image	This is simply the source image file. The output is written to <i>stdout</i> .

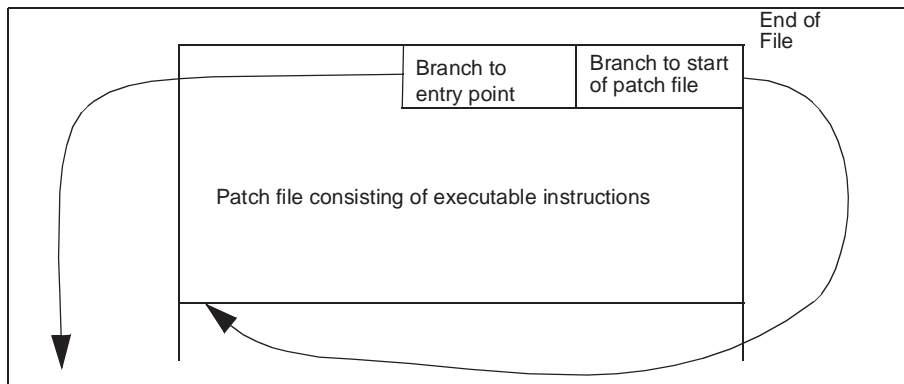


Figure 9-2. Detail of patch file placement

Figure 9-3 shows the relationship of the various sections in the produced output image.

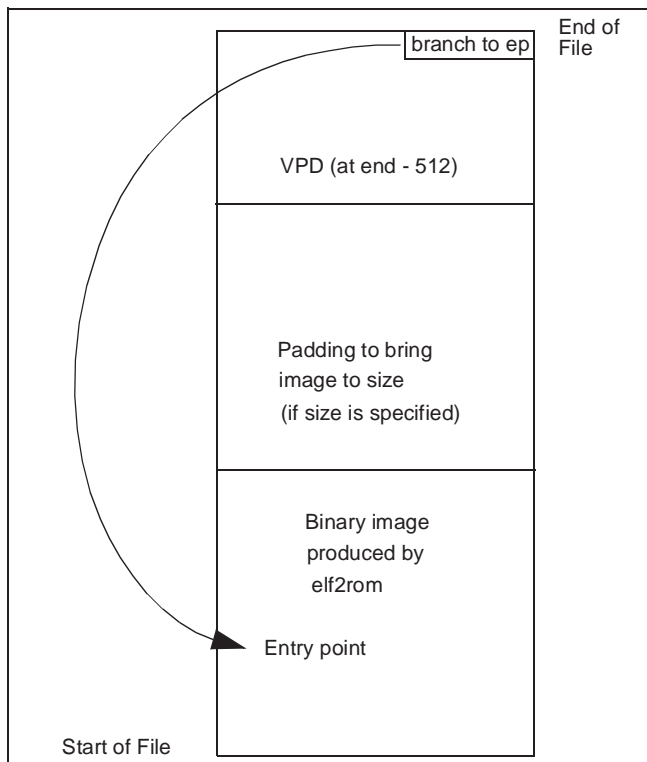


Figure 9-3. hbranch Output Image

Users can find an example of using hbranch in the ROM Monitor's Makefile under **osopen/PLATFORM/openbios**.

9.5.3 eimgbld

The **eimgbld** tool converts an output file from the linker/binder into the format used by the ROM Monitor to load programs from the host onto the evaluation board. The ELF file must be an otherwise executable file, with the text and data addresses bound at link time. Since the entry point of the ELF file will be used by the ROM loader, it must point to a suitable bootstrap.

Syntax:

eimgbld: [-D -P -S -v -b addr -m m_file -o o_file -s s_file -x x_file] input_elf

Description:

The program takes the input file *input_elf* (which must be the final ELF executable file produced from the build process) and converts it into the load format used by the ROM Monitor. There are several optional flags that can affect **eimgbld** processing. They are described below.

-D	Set debug flag. A flag is set in the image causing the ROM Monitor debugger to be invoked immediately after the image is loaded.
-P	Creates output image in PReP format. PReP format is used by some PowerPC platforms.
-S	Suppress symbol information. Specifying this flag will prevent the symbol table from being included in the image.
-v	Verbose option. Directs information about the produced image to stderr.
-b addr	Set the symbol start location to address, addr.
-m m_file	Specify the ROM address map file. The format of this file is two addresses on each line (start address and ending address separated by a “,”).
-o o_file	Allows the specification of an output file name. The default name is input_elf.img.
--s s_file	Restrict symbol table to names in specified file, s_name. The format of this file is one symbol on each line.
-x x_file	Suppress section names listed in specified file, x_name. The format of this file is one section name on each line.

Users can find an example of using eimgbld in the sample Makefile under **osopen/PLATFORM/samples**.

9.5.4 nimgbld (XCOFF kits only)

The input file to **nimgbld** must be an XCOFF file with space reserved after the entry point for the load information block (see Appendix B for more details). RS6000 users can find an example of using **nimgbld** in the Makefile under `/usr/osopen/PLATFORM/samples`.

The **nimgbld** tool converts an output file from the linker/binder into the format used by the ROM Monitor to load programs from the host onto the evaluation board. The XCOFF file must be an otherwise executable file, with the text and data addresses bound at link time. Since the entry point of the XCOFF file will be used by the ROM loader, it must point to a suitable bootstrap.

Syntax:

```
nimgbld: [ -D -P -S -v -o o_file -s s_file -x x_file] input_xcoff
```

Description:

The program takes the input file *input_xcoff* (which must be the final XCOFF executable file produced from the build process) and converts it into the load format used by the ROM Monitor. There are several optional flags that can affect **nimgbld** processing. They are described below.

-v	Verbose option.
-D	Set debug flag. A flag is set in the image causing the ROM Monitor debugger to be invoked immediately after the image is loaded.
-P	Creates output image in PReP format. PReP format is used by some PowerPC platforms.
-S	Suppress symbol information. Specifying this flag will prevent the symbol table from being included in the image.
-o o_file	Allows the specification of an output file name. The default name is <code>input_elf.img</code> .
--s s_file	Restrict symbol table to names in specified file, <code>s_name</code> . The format of this file is one symbol on each line.

The input file must be an XCOFF file with space reserved after the entry point for the load information block (see Appendix B for more details).

9.5.5 Creating a loadable diskette (AIX)

The **dd** AIX command converts and copies a file. **dd** can be used to create loadable diskettes for those evaluation platforms supporting load from diskette. Following is an example of the use of **dd** on AIX.

```
dd conv=sync if=input_file of=output_file ibs=8d obs=8b
```

input_file is the name of the image file created from nimgbld or eimgbld.

output_file is the output file that **dd** will write to. With AIX /dev/fd0 would be used.

Note: Refer to your appropriate host operating system command reference manual for a full description of the **dd** command and it's options.

9.5.6 Creating a loadable diskette (Sun OS)

The **dd** Sun OS command converts and copies a file. **dd** can be used to create loadable diskettes for those evaluation platforms supporting load from diskette. Following is an example of the use of **dd** on Sun OS.

```
dd conv=sync if=input_file of=output_file ibs=8d obs=8b
```

input_file is the name of the image file created from nimgbld or eimgbld.

output_file is the output file that **dd** will write to. With Sun OS /dev/rfd0 would be used.

After the bootable disk has been created type *eject* to remove the disk from the workstation.

Note: Refer to your appropriate host operating system command reference manual for a full description of the **dd** command and it's options.

9.5.7 Creating a loadable diskette (Solaris)

The **dd** Solaris command converts and copies a file. **dd** can be used to create loadable diskettes for those evaluation platforms supporting load from diskette. Use the following procedure.

- Insert the target diskette into the workstations diskette drive
- Issue the **volcheck** command
- Type **eject -q** to obtain the diskette device name. The name returned should look something like **/vol/dev/rdiskette0/unlabeled**.
- Issue the following **dd** command to write the converted image file to the diskette in the proper format.

```
dd conv=sync if=input_file of=output_file ibs=8d obs=8b
```

input_file is the name of the image file created from nimgbld or eimgbld.

output_file is the output file that **dd** will write to. With Solaris **/vol/dev/rdiskette0/unlabeled** fd0 would be used.

After the bootable disk has been created type *eject* to remove the disk from the workstation.

Note: If the diskette device name returned from `eject -q` command is `/vol/dev/rdiskette0/unnamed_floppy` and the boot attempt from the ROM Monitor menu failed, repeat all of the above steps with the failing diskette. Ensure the that the diskette device name specified for `output_file` is the same as that returned from the **eject -q** command.

Note: Refer to your appropriate host operating system command reference manual for a full description of the **dd** command and it's options.

9.5.8 dsktimg (PC only)

The **dsktimg** tool for the PC moves a file onto a 1.44 Mb diskette, treating the diskette as a raw block device. This utility is used to create loadable diskettes for those evaluation platforms supporting load from diskette.

Syntax:

`dsktimg: [-d drive_spec -q] input_img`

Description:

The program takes the input file *input_img* (which must be the final boot image file produced from **eimgbld**) and moves it to the diskette. There are several optional flags that can affect **dsktimg** processing. They are described below.

<code>-q</code>	Quiet. All informational and error messages are suppressed.
<code>-d drive_spec</code>	Can be set to either <i>a:</i> or <i>b:</i> to specify which diskette drive is to be used.

PC users should look in the `Makefile.mak` file under `\osopen\PLATFORM\samples` for an example.

60x/7xx EVB Function Reference

This chapter describes the OS Open functions for the 60x/7xx EVB platform. The function calls and macros are arranged alphabetically by name. For information about the effective use of some of these functions, refer to the microprocessor's User's Manual.

All descriptions contain the following sections:

- Synopsis
- Library
- Description
- Errors
- Attributes
- Processors

Examples and references are provided or referenced where appropriate.

10.1 Attributes and Threads

Functions and macros have attributes that affect thread execution. Depending on their behavior, functions may or may not be “async safe,” “cancel safe,” and “interrupt handler safe.”

10.1.1 Async Safe Functions

An async safe function may be entered by two or more concurrently executing threads, with each thread getting the correct results.

Functions that operate only on disjoint or local data objects are reentrant, and are therefore async safe. For example, **ppcCntlzw()** operates only on its arguments, making it reentrant and therefore async safe.

Functions that operate on common or global data objects may use serialization techniques, such as mutexes and semaphores, within the functions to ensure async safe operation. **enet_send_packet()** uses the functions **semwait()** and **sempost()** to force serialization. Refer to the *OS Open User's Guide* for more information about the use of mutexes and semaphores.

10.1.2 Cancel Safe Functions

The cancel safe attribute is important only to threads executing in deferred cancelability mode (the cancel state is enabled; the cancel type is deferred).

A thread executing in deferred cancelability mode can execute a cancel safe function without being canceled. If the same thread executes a non-cancel safe function, the thread may or may not be canceled during execution of the function.

10.1.3 Interrupt Handler Safe Functions

An interrupt handler safe function may be called by a first level interrupt handler (FLIH).

10.1.4 Callable from Application Thread Group Functions

This attribute is only a concern when running OS Open with Virtual Memory. A function that is callable from an application thread group may be called from all thread groups. A function not callable from an application thread group will cause an exception if called from any thread group other than the kernel thread group.

10.1.5 Processors

The list of processors shows the PowerPC processor the function or macro is valid for.

Note: Functions and macros valid for 603 are valid for the 603, 603e, 603ev, 740, and 750 processors.

10.2 60x/7xx EVB Functions

Descriptions of the functions provided specifically to support the 60x/7xx EVB are listed in Table 10-1:

Table 10-1. Functions Specific to 60x/7xx EVB

Function or Macro	Description	Page
async_init()	Installs the asynchronous device driver	10-10
biosenet_attach()	Attaches the Ethernet to an IP address	10-11

Table 10-1. Functions Specific to 60x/7xx EVB

Function or Macro	Description	Page
clock_set()	Sets the OS Open POSIX clock to the value obtained from the battery operated real time clock	10-13
clockchip_set()	Sets the real-time clock to the number of seconds since 00:00:00 January 1st, 1970 UTC	10-14
cpu_bus_speed()	Returns the cpu bus speed	10-15
dbg_ioLib_init()	Initializes the I/O library	10-16
dcache_flush()	Flushes cache lines, beginning at the effective address and continuing for a specified number of bytes	10-17
dcache_invalidate()	Invalidates cache lines beginning at the effective address and continuing for a specified number of bytes	10-18
dma_disable()	Inhibits DMA activity on a channel	10-19
dma_setup()	Initializes the DMA channel registers for subsequent DMA slave transfers	10-20
dma_status()	Returns the DMA status for the channel specified by channel	10-21
dskt_init()	The diskette device driver	10-22
enet_init()	The Ethernet device driver initialization function	10-23
ext_int_disable()	Disables the interrupt level specified by an event	10-24
ext_int_enable()	Enables the interrupt level specified by an event	10-25

Table 10-1. Functions Specific to 60x/7xx EVB

Function or Macro	Description	Page
ext_int_install()	Installs a first level interrupt handler (FLIH) for an event.	10-26
ext_int_query()	Returns information about the FLIH	10-27
int_install()	Installs a first level interrupt handler (FLIH) for an event.	10-28
int_query()	Returns information about the FLIH	10-29
ioLib_init()	Initializes I/O library	10-30
ppcAbend()	Executes an invalid opcode forcing a program check interrupt	10-31
ppcAndMsr()	ANDs a value with the contents of the MSR	10-32
ppcCntlzw()	Counts consecutive leading zeros in a value	10-33
ppcDcbf()	Copies the cache block back to main storage (if the block resides in cache and has been modified with respect to main storage) and then invalidates the cache block	10-34
ppcDcbi()	Invalidates a cache block, discarding any modified contents if the block is valid in cache	10-35
ppcDcbst()	Copies a cache block, discarding any modified contents if the block is valid in cache	10-36
ppcDcbz()	Sets a cache block to 0	10-37
ppcEieio()	Ensures that all storage references before the call finish before any storage references after the call start	10-39

Table 10-1. Functions Specific to 60x/7xx EVB

Function or Macro	Description	Page
ppcHalt()	Is a one instruction spin loop, effectively putting the processor in an enabled wait at the point of invocation	10-40
ppcIcbi()	Invalidates an instruction cache block	10-41
ppcIsync()	Causes the processor to discard any instructions that may have been prefetched	10-42
ppcMfdabr()	returns the value of the Data Address Breakpoint Register (DABR)	10-43
ppcMfdar()	Returns the value of the processor DAR register	10-44
ppcMfdbat0l() - ppcmfdbat3l()	Returns the value of their respective lower data BAT register.	10-45
ppcMfdbat0u() - ppcmfdbat3u()	Returns the value of their respective upper data BAT register.	10-46
ppcMfdec()	Returns the value of the decremter.	10-47
ppcMfdsisr()	Returns the value of the data storage interrupt status register (DSISR)	10-48
ppcMfear()	Returns the value of the external access register (EAR)	10-49
ppcMfesasrr()	Returns the value of the ESA Save/Restore Register	10-50
ppcMfgpr1()	Returns the current value of GPR(1)	10-51
ppcMfgpr2()	Returns the current value of GPR(2)	10-52
ppcMfhid0()	Returns the value of the hardware implementation-dependent register 0 (HID0)	10-53

Table 10-1. Functions Specific to 60x/7xx EVB

Function or Macro	Description	Page
ppcMfhid1()	Returns the value of the hardware implementation-dependent register 1 (HID1)	10-54
ppcMfiabr()	Returns the value of the instruction address breakpoint register (IABR)	10-55
ppcMfibat0l() - ppcmfibat3l()	Returns the value of their respective lower instruction BAT register.	10-56
ppcMfibat0u() - ppcmfibat3u()	Returns the value of their respective upper instruction BAT register.	10-57
ppcMfmmrc0()	Returns the value of the monitor control register 0 (MMCR0)	10-58
ppcMfmsr()	Returns the value of the MSR	10-59
ppcMfpir()	Returns the value of the processor identification register (PIR)	10-60
ppcMfpmc1()	Returns the value of the performance monitor counter register 1 (PMC1)	10-61
ppcMfpmc2()	Returns the value of the performance monitor counter register 2(PMC2)	10-62
ppcMfpvr()	Returns the value of the processor version register	10-63
ppcMfsda()	Returns the value of the sample data address register (SDA)	10-64
ppcMfsdr1()	Returns the value of storage description register 1	10-65
ppcMfsia	Returns the value of the sample instruction address register (SIA)	10-66
ppcMfsprg0()- ppcMfsprg3()	Returns the value of the special purpose register generals (SPRG0 - SPRG3)	10-67

Table 10-1. Functions Specific to 60x/7xx EVB

Function or Macro	Description	Page
ppcMfsr()	Returns the value of the segment register	10-68
ppcMfsrr0()	Returns the value of SRR0	10-69
ppcMfsrr1()	Returns the current value of SRR1	10-70
ppcMftb()	Returns the current time base data	10-71
ppcMtdabr()	Sets the value of the data address break-point register (DABR)	10-72
ppcMtdar()	Sets the value of the DAR	10-73
ppcMtdbat0l() - ppcMtdbat3l()	Sets the value of their respective lower data BAT register.	10-74
ppcMtdbat0u() - ppcMtdbat3u()	Sets the value of their respective upper data BAT register.	10-75
ppcMtdec()	Sets the value of the decremter	10-76
ppcMtdsisr()	Sets the DSISR register	10-77
ppcMthid0()	Sets the value of the hardware implementation-dependent register 0 (HID0)	10-79
ppcMtiabr()	Sets the value of the instruction address breakpoint register (IABR)	10-80
ppcMtibat0l() - ppcMtibat3l()	Sets the value of their respective lower instruction BAT register.	10-81
ppcMtibat0u() - ppcMtibat3u()	Sets the value of their respective upper instruction BAT register.	10-82
ppcMtmrc0()	Sets the value of the monitor mode control register 0 (MMCR0)	10-83
ppcMtmsr()	Sets the MSR	10-84

Table 10-1. Functions Specific to 60x/7xx EVB

Function or Macro	Description	Page
ppcMtpir()	Sets the value of the processor identification register (PIR)	10-85
ppcMtpmc1()	Sets the value of the performance monitor counter register 1 (PMC1)	10-86
ppcMtpmc2()	Sets the value of the performance monitor counter register 2 (PMC2)	10-87
ppcMtsda()	Sets the value of the sample data address register (SDA)	10-88
ppcMtsdr1()	Sets the SDR1	10-89
ppcMtsia()	Sets the value of the sample instruction address register (SIA)	10-90
ppcMtsprg0() - ppcMtsprg3()	Sets the special purpose register generals (SPRG0 - SPRG3)	10-91
ppcMtsr()	Sets the segment register	10-92
ppcMtsrr0()	Sets the SRR0	10-93
ppcMtsrr1()	Sets the SRR1	10-94
ppcMttb()	Sets the value of the current time base	10-95
ppcOrMsr()	Performs the OR of a value and the current MSR, updating the MSR	10-96
ppcSync()	Causes the processor to wait until all data cache lines scheduled to be written to main storage have actually been written	10-97
setup_bat()	Sets up BAT registers to specified values.	10-98
s1dbprintf()	A version of printf() that may be used before I/O has been established	10-100

Table 10-1. Functions Specific to 60x/7xx EVB

Function or Macro	Description	Page
s1dbprintfapp()	A version of printf() that may be used before I/O has been established from an application thread group.	10-101
s2dbprintf()	A version of printf() that may be used before I/O has been established for serial port 2	10-103
s2dbprintfapp()	A version of printf() that may be used before I/O has been established for serial port 2 from an application thread group.	10-104
timertick_install()	Installs and starts the timer tick handler	10-106
timertick_remove()	Removes the timer tick handler	10-107
vs1dbprintf()	A version of printf() that uses polled writes (no interrupts), and may be used before I/O has been established and accepts a <code>va_list</code> as a parameter instead of a variable number of parameters	10-108
zapatos_enet_setup()	Sets up memory for the Ethernet chip.	10-109

async_init()

Synopsis

```
#include <sys/asyncLib.h>
int driver_install(int *devhandle,async_init);
```

Library

asyncLib.a

Description

asyncLib.a is the asynchronous device driver that supports the asynchronous communication port on the 60x/7xx EVB platform. **asyncLib.a** is installed by calling **driver_install()** with *devhandle* as the first parameter and **async_init** as the second parameter.

Errors

None.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	No
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- driver_install() : *OS Open Programmer's Reference*
- "Device Drivers Supplied with the 60x/7xx EVB" on page 9-7

Synopsis

```
#include <benetLib.h>

int biosenet_attach( unsigned long ipaddr, int init_flag);
```

Library

benetLib.a

Description

biosenet_attach() attaches the TCP/IP protocol stack to the Ethernet . Unlike **enet_init()**, this allows OS Open and the ROM Monitor to share the ethernet device. **biosenet_attach()** should not be used at the same time as the eneLib.a device driver. The IP address should be different from the IP address defined to the 60x/7xx EVB ROM Monitor. *init_flag* determines if **biosenet_attach()** should initialize the Ethernet interface. The Ethernet device should be initialized only if OS Open was loaded through an interface other than Ethernet. A non-0 value will cause **biosenet_attach()** to initialize the Ethernet and a 0 value causes **biosenet_attach()** not to initialize the Ethernet interface. **biosenet_attach()** returns 0 if successful and -1 if it is unsuccessful.

Note: When using **biosenet_attach()** the I/O should be initialized by calling **dbg_ioLib_init()** rather than **ioLib_init()**.

Note: **biosenet_attach()** is unavailable for OS Open with Virtual Memory.

Errors

None.

Example

Initialize TCP/IP and define an IP address to **biosenet_attach()**.

```
#include<sys/tcpipLib.h>

int rc;

rc=tcpip_init("myhostname", 1 , 100);

if (rc!=0) {
return(-1);}

if (net_init() ) return(-1);

return(biosenet_attatch(0x07010104,0)); /* specify the
IP addr. and the init flag*/
```

Attributes

Async Safe

No

biosenet_attach()

Cancel Safe	No
Interrupt Handler Safe	No
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- “Ethernet Device Driver” on page 9-16

Synopsis

```
#include <clockLib.h>
int clock_set(void);
```

Library

clockLib.a

Description

clock_set() sets the OS Open POSIX clock to the value obtained from battery operated real time clock.

Errors

[EIO] Real-time clock not running.

Attributes

Async Safe	Yes/No*
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No
Note: * Not Async Safe in OS Open with Virtual Memory	

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- clockchip_set(), p. 10-14

clockchip_set()

Synopsis

```
#include <clockLib.h>
int clockchip_set( time_t timeval );
```

Library

clockLib.a

Description

clockchip_set() sets the battery-backed real-time clock to *timeval*, which should contain the number of seconds since January 1st, 1970 UTC.

Errors

[EIO]	Real-time clock not running.
[EINVAL]	Library not initialized.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- clock_set(), p. 10-13

Synopsis

```
#include <tickLib.h>

unsigned long cpu_bus_speed(void);
```

Library

tickLib.a

Description

cpu_bus_speed() returns the CPU bus speed. The possible return values are:

- 60000000 (60 MHz)
- 66666666 (66 MHz)

cpu_bus_speed() determines the CPU bus speed by putting a serial port in loopback mode and using it for timing purposes. Since this function can overwrite any previous serial port configuration, it should only be used at system initialization, before the serial ports are initialized. After serial port initialization, results from this function are undefined.

Errors

None

Attributes

Async Safe	No
Cancel Safe	No
Interrupt Handler Safe	No
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

dbg_ioLib_init()

Synopsis

```
#include <ioLib.h>
int dbg_ioLib_init( void );
```

Library

ioLib.a

Description

dbg_ioLib_init() initializes the I/O library. Unlike **ioLib_init()**, this function allows external I/O interrupts to be screened by the ROM monitor, enabling debug to be performed from outside of the OS Open environment. Only external I/O through IRQ's other than those used by the ROM Monitor are available to OS Open.

If successful, **dbg_ioLib_init()** returns 0. Otherwise, **dbg_ioLib_init()** returns -1.

Errors

[ENOMEM]	Insufficient memory to allocate first level interrupt handler control areas.
----------	------------------------------------------------------------------------------

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

ioLib_init(), p. 10-30

Synopsis

```
#include <ioLib.h>

void dcache_flush( void *address, unsigned int count );
```

Library

ioLib.a

Description

dcache_flush() flushes cache lines, beginning at the effective address and continuing for *count* bytes.

A cache line flush forces the current contents of the cache line to main storage (if the line is valid and marked as modified) and then invalidates the line.

Note: Since cache flushes occur on cache line boundaries, the operation can occur outside of the bounds specified by the function call. For example, if *address* is X'216' and *count* is X'12', two cache lines, spanning addresses from X'200' to X'23F', would be flushed.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- dcache_invalidate(), p. 10-18

dcache_invalidate()

Synopsis

```
#include <ioLib.h >

void dcache_invalidate( void *address, unsigned int count );
```

Library

ioLib.a

Description

dcache_invalidate() invalidates cache lines beginning at the effective address given by *address* and continuing for *count* bytes.

Note: Since cache invalidation occurs on cache line boundaries, invalidation can occur outside of the bounds implied by this command. For example, if *address* is X '104' and *count* is 16, the cache line spanning the addresses from X '100' to X '120' would be invalidated.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- dcache_flush(), p. 10-17

Synopsis

```
#include <ioLib.h>

int dma_disable( unsigned int channel );
```

Library

ioLib.a

Description

dma_disable() inhibits DMA activity on the channel specified by *channel*.

Note: Although **dma_disable()** is not async safe in general, it can be safely called by concurrently executing threads as long as each thread specifies a unique DMA channel.

If successful **dma_disable()** returns 0. Otherwise -1.

Errors

[EINVAL] *channel* does not refer to a valid DMA channel.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- **dma_setup()**, p. 10-20
- **dma_status()**, p. 10-21

dma_setup()

Synopsis

```
#include <ioLib.h>

int dma_setup( unsigned int channel, int type, void *address,
               unsigned int count, unsigned long dmacr,
               unsigned long count, void *dst_address, void
               *src_address, unsigned long chained_count );
```

Library

ioLib.a

Description

dma_setup() initializes the DMA channel registers for subsequent DMA slave transfers.

The transfer type is formed from the appropriate combination of DMA transfer types from the file **<ioLib.h>**. For example, to specify a DMA slave transfer from a device to storage, specify *type* as **DMA_IO_TO_MEM**.

address and *count* specify the region in storage that participates in the transfer.

The DMA slave transfer types are:

DMA_MEM_TO_IO	Direction; storage to device
DMA_IO_TO_MEM	Direction; device to storage

Errors

[EINVAL]	<i>channel</i> does not refer to a valid DMA channel.
----------	-------------------------------------------------------

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- [dma_disable\(\)](#), p. 10-19
- [dma_status\(\)](#), p. 10-21

Synopsis

```
#include <ioLib.h>

int dma_status( unsigned int channel, struct dma_stat *dstat );
```

Library

ioLib.a

Description

dma_status() returns the DMA status for the channel specified by *channel*.

If successful **dma_status()** returns 0. Otherwise -1.

dstat->current_addr and *dstat->current_count* change as the transfer continues. *dstat->current_count* begins at one less than the desired transfer count and counts down to X'FFFF' as the transfer completes. Testing for this value in *dstat->current_count* can be used to confirm successful DMA slave transfer completion.

Errors

[EINVAL] *channel* does not refer to a valid DMA channel.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- dma_setup(), p. 10-20
- dma_disable(), p. 10-19

dskt_init()

Synopsis

```
#include <dsktLib.h>
```

```
int driver_install( int devhandle, async_init, int dskt_intlvn, int  
dma_channel);
```

Library

dsktLib.a

Description

dsktLib.a is the device driver that provides the block device driver interface for the diskette drive on the Sandalfort. **dsktLib.a** is installed by calling **driver_install()** with the *devhandle* as the first parameter and **dskt_init** as the second parameter. The third parameter is the interrupt level, *dskt_intlvn*, to be used and the fourth parameter is the DMA channel, *dma_channel*, used for slave DMA operations.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	No
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- driver_install() : *OS Open Programmer's Reference*
- "Diskette Device Driver" on page 9-12

Synopsis

```
#include <enet.h>

int driver_install( int devhandle, enet_init, int enet_intlvl, int
register_base, int pci_mem_addr, int num_blocks, int data_addr, char
*mac_array,);
```

Library

enetLib.a

Description

enetLib.a is the Ethernet device driver supporting packet level read/writes to the integrated ethernet on the 60x/7xx EVB platform. **enetLib.a** is installed by calling **driver_install()** with eight parameters. The first parameter is the device handle, *devhandle*. The second parameter is the device driver initialization function, **enet_init**. The third parameter is the ethernet interrupt level, *enet_intlvl*. The fourth parameter is the base I/O address of the Ethernet controller's register set, *register_base*. The fifth parameter is the pci memory address, *pci_mem_addr*. The sixth parameter is the number of 256 byte buffers allocated for the Ethernet drivers use, *num_blocks*. The seventh parameter is the address of memory to use for buffers, *data_addr*. The eighth parameter is the location of the universal MAC address assigned to the Ethernet controller, *mac_array*.

Please see "Ethernet Device Driver" on page 9-16 for additional information.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- driver_install() : *OS Open Programmer's Reference*
- dskt_init, p. 10-22
- "Ethernet Device Driver" on page 9-16

ext_int_disable()

Synopsis

```
#include <ioLib.h>

void ext_int_disable( int event );
```

Library

ioLib.a

Description

ext_int_disable() disables the interrupt level specified by *event*. **ioLib.h** defines DMA interrupt 2 the interrupt levels that can be disabled.

The **ext_int_disable()** function returns nothing.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- ext_int_enable(), p. 10-25
- ext_int_install(), p. 10-26
- ext_int_query(), p. 10-27
- ioLib_init(), p. 10-30

Synopsis

```
#include <ioLib.h>

void ext_int_enable( int event );
```

Library

ioLib.a

Description

ext_int_enable() enables the interrupt level specified by *event*. **ioLib.h** defines the interrupt levels that can be enabled.

ext_int_enable() returns nothing.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- ext_int_install(), p. 10-26
- ext_int_query(), p. 10-27
- ioLib_init(), p. 10-30

ext_int_install()

Synopsis

```
#include <flih.h>
#include <ioLib.h>
int ext_int_install( int event, flih_t *new_flih, flih_t *old_flih );
```

Library

ioLib.a

Description

ext_int_install() installs a first level interrupt handler (FLIH) for *event*. **ioLib.h** defines the interrupt levels that can be set.

If *new_flih* is NULL, the current interrupt handler is removed for the specified event. If *new_flih* is non-NULL, it points to a **flih_t** structure containing the following fields:

<i>flih_stack</i>	Pointer to the first stack location; obtained by allocating memory and adding the size of the stack. <i>flih_stack</i> must be 16 byte aligned.
<i>flih_function</i>	Pointer to a function invoked when <i>event</i> occurs.
<i>arg</i>	A user-defined (void *) value passed to <i>flih_function</i> .

If *old_flih* is not NULL, the previous values of *flih_function*, *flih_stack*, and *arg* are stored in the structure pointed to by *old_flih*.

If successful, **ext_int_install()** returns 0. Otherwise, **ext_int_install()** returns -1.

Errors

[EINVAL]	<i>event</i> does not refer to a valid event.
----------	-----------------------------------------------

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- [ext_int_enable\(\)](#), p. 10-25
- [ext_int_query\(\)](#), p. 10-27
- [ioLib_init\(\)](#), p. 10-30

Synopsis

```
#include <ioLib.h>
#include <flih.h>
int ext_int_query( int event, flih_t *flih );
```

Library

ioLib.a

Description

ext_int_query() returns information about the first level interrupt handler (FLIH), if any, for *event*.

ioLib.h defines the events for which FLIHs can query.

The *flih* argument points to a **flih_t** structure containing the following fields:

<i>flih_stack</i>	Pointer to the first stack location; obtained by allocating memory and adding the size of the stack.
<i>flih_function</i>	Pointer to a function invoked when <i>event</i> occurs.
<i>arg</i>	A user-defined (void *) value passed to <i>flih_function</i> . If no FLIH is installed for the specified level, each field in the flih_t structure is assigned NULL.

If successful, **ext_int_query()** returns 0. Otherwise, **ext_int_query()** returns -1.

Errors

[EINVAL] *event* does not refer to a valid event.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- ext_int_enable(), p. 10-25
- ext_int_install(), p. 10-26
- ioLib_init(), p. 10-30

int_install()

Synopsis

```
#include <flih.h>
int int_install( int event, flih_t *new_flih, flih_t *old_flih );
```

Library

rtxLib.a

Description

int_install() installs a first level interrupt handler (FLIH) for *event*. **flih.h** defines the interrupt levels that can be set.

If *new_flih* is NULL, the current interrupt handler is removed for the specified event. If *new_flih* is non-NULL, it points to a **flih_t** structure containing the following fields:

<i>flih_stack</i>	Pointer to the first stack location; obtained by allocating memory and adding the size of the stack.
<i>flih_function</i>	Pointer to a function invoked when <i>event</i> occurs.
<i>arg</i>	A user-defined (void *) value passed to <i>flih_function</i> .

If *old_flih* is not NULL, the previous values of *flih_function*, *flih_stack*, and *arg* are stored in the structure pointed to by *old_flih*.

If successful, **int_install()** returns 0. Otherwise, **int_install()** returns -1.

Errors

[EINVAL]	<i>event</i> does not refer to a valid event.
----------	-----------------------------------------------

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- [int_query\(\)](#), p. 10-29
- [ioLib_init\(\)](#), p. 10-30

Synopsis

```
#include <flih.h>
int int_query( int event, flih_t *flih );
```

Library

rtxLib.a

Description

int_query() returns information about the first level interrupt handler (FLIH), if any, for *event*.

flih.h defines the events for which FLIHs can query.

The *flih* argument points to a **flih_t** structure containing the following fields:

flih_stack	Pointer to the first stack location; obtained by allocating memory and adding the size of the stack.
flih_function	Pointer to a function invoked when <i>event</i> occurs.
arg	A user-defined (void *) value passed to <i>flih_function</i> . If no FLIH is installed for the specified level, each field in the flih_t structure is assigned NULL.

If successful, **int_query()** returns 0. Otherwise, **int_query()** returns -1.

Errors

[EINVAL]	<i>event</i> does not refer to a valid event.
----------	-----------------------------------------------

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- int_install(), p. 10-28
- ioLib_init(), p. 10-30

ioLib_init()

Synopsis

```
#include <ioLib.h>
int ioLib_init( void );
```

Library

ioLib.a

Description

ioLib_init() initializes the I/O library.

If successful, **ioLib_init()** returns 0. Otherwise, **ioLib_init()** returns -1.

ioLib_init() should not be used on a 60x/7xx EVB when using the ROM Monitor Ethernet interface or the ROM monitor debugger.

dbg_ioLib_init() should be used instead.

Errors

[ENOMEM]	Insufficient memory to allocate first level interrupt handler control areas.
----------	------------------------------------------------------------------------------

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

Synopsis

```
#include <ppcLib.h>
void ppcAbend(void)
```

Library

ppcLib.a

Description

ppcAbend() executes an invalid opcode forcing a Program Check interrupt.

Errors

None.

Example

- Force an illegal instruction exception.

```
ppcAbend ( )
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

ppcAndMsr()

Synopsis

```
#include <ppcLib.h>
unsigned long ppcAndMsr(unsigned long value);
```

Library

ppcLib.a

Description

ppcAndMsr() ANDs *value* with the contents of the MSR.
The MSR is updated with the result of the AND operation.
ppcAndMsr() returns the previous contents of the MSR.
Refer to the **<ppcLib.h>** file for the defines of the MSR constants.

Errors

None.

Example

```
• Disable external interrupts.
    unsigned long orig_msr = ppcAndMsr(~ppcMsrEE);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- ppcOrMsr(), p. 10-96
- ppcMtmsr(), p. 10-84
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

Synopsis

```
#include <ppcLib.h>
unsigned long ppcCntlzw(unsigned long value);
```

Library

ppcLib.a

Description

ppcCntlzw() counts consecutive leading zeros in *value*.
ppcCntlzw() returns the count, which ranges from 0 through 32, inclusive.

Errors

None.

Example

```
• Return count of leading zeros in variable k.

int k;
unsigned long k = ppcCntlzw(0x0700AA55); /* k = 5 */
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

ppcDcbf()

Synopsis

```
#include <ppcLib.h>
void ppcDcbf(void *addr);
```

Library

ppcLib.a

Description

ppcDcbf() copies the cache block at the effective address specified by *addr* back to main storage (if the block resides in cache and has been modified with respect to main storage) and then invalidates the cache block.

Effectively, this function acts like **ppcDcbst()** followed by **ppcDcbi()**.

Errors

None.

Example

- Flush the cache line at the effective address X'1000' to main storage and then invalidate the cache line. You might do this in preparation for a DMA slave transfer.

```
ppcDcbf((void *)0x1000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- ppcDcbst(), p. 10-36
- ppcDcbi(), p. 10-35
- ppcDcbz(), p. 10-37
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

Synopsis

```
#include <ppcLib.h>
void ppcDcbi(void *addr);
```

Library

ppcLib.a

Description

ppcDcbi() invalidates the cache block containing *addr*, discarding any modified contents if the block is valid in cache.

Errors

None.

Example

- Invalidate the cache line beginning with 0x3000. This might be done before reading an area of storage updated by a DMA transfer.
- ```
ppcDcbi((void *)0x3000);
```

Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | Yes |

Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

References

- ppcDcbst(), p. 10-36
- ppcDcbi(), p. 10-35
- ppcDcbz(), p. 10-37
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcDcbst()

---

## Synopsis

```
#include <ppcLib.h>
void ppcDcbst(void *addr);
```

## Library

ppcLib.a

## Description

**ppcDcbst()** copies the cache block containing *addr* to main storage, if the block is valid in cache and has been modified with respect to main storage.

## Errors

None.

## Example

- Force the cache line beginning with 0x4000 to memory if the block is valid and out of sync with storage. This would be done to synchronize the cache and storage without invalidating the cache line.

```
ppcDcbst((void *)0x4000);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | Yes |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- ppcDcbf(), p. 10-34
- ppcDcbi(), p. 10-35
- ppcDcbz(), p. 10-37
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcDcbz(void *addr);
```

## Library

ppcLib.a

## Description

**ppcDcbz()** sets the cache block containing the byte referenced by *addr* to 0.

The line is established, if necessary, without fetching the line from main storage.

**Note:** If an invalid real address is specified, problems could occur when a subsequent attempt is made by the cache unit to store that line to main storage.

## Errors

None.

## Example

- Assume buffer is 16 cache lines long and cache aligned. To quickly set it to 0, set to first buffer address.

```
char *bpt = buffer;
for(j = 0; j < 16; j++)
{
 ppcDcbz((void *)bpt);
 bpt += cache_line_size;
}
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | Yes |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

# ppcDcbz()

---

## References

- ppcDcbf(), p. 10-34
- ppcDcbi(), p. 10-35
- ppcDcbst(), p. 10-36
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*



## Synopsis

```
#include <ppcLib.h>
void ppcEieio(void);
```

## Library

ppcLib.a

## Description

**ppcEieio()** ensures that all storage references before the call finish before any storage references after the call start.

The PowerPC 60x/7xx may internally reorder operations to storage. In the case of memory mapped I/O, such reordering can be undesirable and can be prevented by appropriate use of **ppcEieio()**.

## Errors

None.

## Example

- Ensure storage references are done in order.

```
char *one_loc = (char *)0x202;
char *two_loc = (char *)0x204;

one_loc = 0xAA; / write a 0xAA to 0x202 */
ppcEieio(); /* insure the store completes before
setting two_loc */
*two_loc = 0x55;
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | Yes |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcHalt()

---

## Synopsis

```
#include <ppcLib.h>
void ppcHalt(void);
```

## Library

ppcLib.a

## Description

**ppcHalt()** is a one instruction spin loop, effectively putting the processor in an enabled wait at the point of invocation.

## Errors

None.

## Example

- Wait at the point of invocation.

```
ppcHalt () ;
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | Yes |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppclcbi(void *addr);
```

## Library

ppcLib.a

## Description

**ppclcbi()** invalidates the Instruction Cache Block pointed to by the address passed. This may be done after updating an instruction.

## Errors

None.

## Example

- Write a trap into location 0x3000.

```
unsigned in * i_addr = (int *) 0x3000;
i_addr = 0x7c800008; / tw instruction */
ppcDbcst((void *) 0x3000);
ppcIcbi((void *) 0x3000);
ppcIsync();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | Yes |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppclsync()

---

## Synopsis

```
#include <ppcLib.h>
void ppclsync(void);
```

## Library

ppcLib.a

## Description

**ppclsync()** causes the processor to discard any instructions that may have been prefetched before **ppclsync()**. This call must be used after modifying instruction storage.

## Errors

None.

## Example

- Place a trap into a given address.

```
*trap_address = 0x7F000008;
ppclsync();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | Yes |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- PowerPC 603 RISC Microprocessor User's Manual*
- PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfdabr(void);
```

## Library

ppcLib.a

## Description

**ppcMfdabr()** returns the value of the Data Address Breakpoint register. The DABR provides a means to detect data access to a specific double word. The data address that is returned from the DABR is on a double word boundary. Bits 0-28 contains the data address (DAB), bit 29 is the breakpoint translation enable (BT) bit, bit 30 is the data write enable (DW) bit, and bit 31 is the data read enable (DR) bit. When a match is detected a DSI exception is created.

## Errors

None.

## Example

- Get the value of the DABR.

```
unsigned long dabr_value=ppcMfdabr();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | No  |
| PowerPC 604 | Yes |

## References

- *PowerPC 604 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The programming Environments*

# ppcMfdar()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfdar(void);
```

## Library

ppcLib.a

## Description

**ppcMfdar()** returns the value of the processor DAR register.

When a Data Storage interrupt occurs, indicating a data storage access failed (this can happen for a variety of reasons) the DAR is set to the data address triggering the failure.

## Errors

None.

## Example

- Retrieve the value of DAR register. An alignment exception handler would require the value of the DAR.

```
unsigned long current_DAR=ppcMfdar();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfdbat0l(void)
unsigned long ppcMfdbat1l(void)
unsigned long ppcMfdbat2l(void)
unsigned long ppcMfdbat3l(void)
```

## Library

ppcLib.a

## Description

**ppcMfdbat0l() - ppcMfdbat3l()** returns the current value of the specified lower Data Block Address Translation Register (DBAT). The DBAT maintains the address translation information for a block of memory. Each lower DBAT is paired with an upper DBAT register.

## Errors

None.

## Example

- Retrieve the current value of the DBAT3L register.

```
unsigned long dbat3l_value= ppcMfdbat3l();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- ppcMfdbat0u() - ppcMfdbat3u()*, p. 10-46
- PowerPC 603 RISC Microprocessor User's Manual*
- PowerPC 604 RISC Microprocessor User's Manual*

# ppcMfdbat0u() - ppcMfdbat3u()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfdbat0u(void)
unsigned long ppcMfdbat1u(void)
unsigned long ppcMfdbat2u(void)
unsigned long ppcMfdbat3u(void)
```

## Library

ppcLib.a

## Description

**ppcMfdbat0u() - ppcMfdbat3u()** returns the current value of the specified upper Data Block Address Translation Register (DBAT). The DBAT maintains the address translation information for a block of memory. Each upper DBAT is paired with a lower DBAT register.

## Errors

None.

## Example

- Retrieve the current value of the DBAT3U register.

```
unsigned long dbat3ur_value= ppcMfdbat3u();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *ppcMfdbat0l() - ppcMfdbat3l()*, p. 10-45
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*



## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfdec(void);
```

## Library

ppcLib.a

## Description

**ppcMfdec()** returns the value of the decrementer.

The decrementer is a continuously running 32 bit binary down counter that can assert an interrupt request whenever bit 0 (MSB) changes from 0 to 1. Although not all of the least significant bits may be implemented, bit 31 effectively changes every nanosecond.

## Errors

None.

## Example

- Retrieve the current value of the decrementer. An exception handler for the decrementer may require the value to adjust the value of the next “tick”.

```
unsigned long current_dec=ppcMfdec();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 602 | Yes |
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcMfdsisr()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfdsisr(void);
```

## Library

ppcLib.a

## Description

**ppcMfdsisr()** returns the value of the Data Storage Interrupt Status Register (DSISR) register.

This register indicates the reason for a Data Storage Interrupt.

## Errors

None.

## Example

- Retrieve the current value of the data storage interrupt status register. An alignment exception handler could take advantage of the partial instruction decoding provided by this register.

```
unsigned long current_dsir=ppcMfdsisr();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfear(void);
```

## Library

ppcLib.a

## Description

**ppcMfear()** returns the value of the External Access Register (EAR). This register identifies the target device for external control operations.

## Errors

None.

## Example

- Retrieve the current value of the EAR.

```
unsigned long current_ear=ppcMfear();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 602 | Yes |
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The Programming Environments*

# ppcMfesasrr()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfesasrr(void);
```

## Library

ppcLib.a

## Description

**ppcMfesasrr()** returns the value of the ESA Save/Restore Register (ESASRR).

## Errors

None.

## Example

- Retrieve the current value of the ESASRR.

```
unsigned long current_esasrr=ppcMfesasrr();
```

## Attributes

|                        |     |
|------------------------|-----|
| Async Safe             | Yes |
| Cancel Safe            | Yes |
| Interrupt Handler Safe | Yes |

## Processors

|             |    |
|-------------|----|
| PowerPC 603 | No |
|-------------|----|

## References

- *PowerPC 603 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfgpr1(void);
```

## Library

ppcLib.a

## Description

**ppcMfgpr1()** returns the current value of GPR(1).  
Typically, this is the value of the current stack frame.

## Errors

None.

## Example

See **ppcMfgpr2()**, p. 10-52.

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | Yes |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcMfgpr2()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfgpr2(void)
```

## Library

ppcLib.a

## Description

**ppcMfgpr2()** returns the current value of GPR(2).

For XCOFF-based OS Open this is typically the value of the table of contents (TOC) pointer for the current execution context.

## Errors

None.

## Example

- Retrieve TOC and stack frame base from current context.

```
toc = ppcMfgpr2();
unsigned long stack_base = ppcMfgpr1();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | Yes |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfhid0(void)
```

## Library

ppcLib.a

## Description

**ppcMfhid0()** returns the current value of the Hardware Implementation Dependent Register 0 (HID0).

## Errors

None.

## Example

- Retrieve HID0 value.

```
unsigned long hid0_value= ppcMfhid0();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcMfhid1()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfhid1(void)
```

## Library

ppcLib.a

## Description

**ppcMfhid1()** returns the current value of the Hardware Implementation Dependent Register 1(HID1).

## Errors

None.

## Example

- Retrieve HID1 value.

```
unsigned long hid1_value= ppcMfhid1();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
|-------------|-----|

## References

- *PowerPC 603 RISC Microprocessor User's Manual*



## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfiabr(void)
```

## Library

ppcLib.a

## Description

**ppcMfiabr()** returns the current value of the Instruction Address Breakpoint Register(IABR). When the IABR is enabled, the instruction fetch address will be compared with the effective address that is stored in the IABR. The compares will be done on a word boundary. If the IABR and the instruction fetch address compare the instruction breakpoint handler will be invoked.

## Errors

None.

## Example

- Retrieve IABR value.

```
unsigned long iabr_value= ppcMfiabr();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcMfibat0l() - ppcMfibat3l()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfibat0l(void)
unsigned long ppcMfibat1l(void)
unsigned long ppcMfibat2l(void)
unsigned long ppcMfibat3l(void)
```

## Library

ppcLib.a

## Description

**ppcMfibat0l() - ppcMfibat3l()** returns the current value of the specified lower Instruction Block Address Translation Register (IBAT). The IBAT maintains the address translation information for a block of memory. Each lower IBAT is paired with an upper IBAT register.

## Errors

None.

## Example

- Retrieve the current value of the IBAT3L register.

```
unsigned long ibat3l_value= ppcMfibat3l();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *ppcMfibat0u() - ppcMfibat3u()*, p. 10-57
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The Programming Environments*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfibat0u(void)
unsigned long ppcMfibat1u(void)
unsigned long ppcMfibat2u(void)
unsigned long ppcMfibat3u(void)
```

## Library

ppcLib.a

## Description

**ppcMfibat0u()** - **ppcMfibat3u()** returns the current value of the specified upper Instruction Block Address Translation Register (IBAT). The IBAT maintains the address translation information for a block of memory. Each upper IBAT is paired with a lower IBAT register.

## Errors

None.

## Example

- Retrieve the current value of the IBAT3U register.

```
unsigned long ibat3u_value= ppcMfibat3u();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *ppcMfibat0l()* - *ppcMfibat3l()*, p. 10-56
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The Programming Environments*

# ppcMfmmcr0()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfmmcr0(void);
```

## Library

ppcLib.a

## Description

**ppcMfmmcr0()** returns the value of the Monitor Mode Control Register 0 (MMCR0). The MMCR0 is partitioned into bit fields. The bit fields determine the events to be counted and recorded.

## Errors

None.

## Example

- Get the value of the MMCR0.

```
unsigned long mmcr0_value = ppcMfmmcr0();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 604 | Yes |
|-------------|-----|

## References

- *ppcMfpmc1()*, p. 10-61
- *ppcMfpmc2()*, p. 10-62
- *PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfmsr(void);
```

## Library

ppcLib.a

## Description

**ppcMfmsr()** returns the value of the Machine State Register(MSR).  
Refer to the **<ppc\_arch.h>** file for the defines of constants that can be used as masks with the MSR value.

## Errors

None.

## Example

See **ppcMtmsr()**, p. 10-84.

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcMfpir()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfpir(void);
```

## Library

ppcLib.a

## Description

**ppcMfpir()** returns the value of the Processor Identification Register(PIR), which contains a processor identification tag in the four least significant bits(28-31). This register value is useful in processor differentiation in multiprocessor system designs. This register is also used for several direct-store bus operations in the form of a “bus transaction from” tag.

## Errors

None.

## Example

- Retrieve the current value of the processor identification register and print it.

```
printf("This is processor %x\n", ppcMfpir());
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 604 | Yes |
|-------------|-----|

## References

- *PowerPC 604 RISC Microprocessor User's Manual*

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfpmc1(void);
```

Library

ppcLib.a

Description

**ppcMfpmc1()** returns the value of the Performance Monitor Counter Register 1 (PMC1). The PMC1 is a 32 bit counter that can be programmed to generate interrupt signals when it becomes negative. An interrupt signal is not generated unless both the MMCR0(PMC1INTCONTROL) and MMCR0(ENINT) bits are also set.

Errors

None.

Example

- Retrieve the current value of the PMC1 register and print it.  

```
printf("This current count is %x\n", ppcMfpmc1());
```

Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

Processors

|             |     |
|-------------|-----|
| PowerPC 604 | Yes |
|-------------|-----|

References

- *ppcMfmmcr0()*, p. 10-58
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcMfpmc2()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfpmc2(void);
```

## Library

ppcLib.a

## Description

**ppcMfpmc2()** returns the value of the Performance Monitor Counter Register 2 (PMC2). The PMC2 is a 32 bit counter that can be programmed to generate interrupt signals when it becomes negative. An interrupt signal is not generated unless both the MMCR0(PMC2INTCONTROL) and MMCR0(ENINT) bits are also set.

## Errors

None.

## Example

- Retrieve the current value of the PMC2 register and print it.

```
printf("This current count is %x\n", ppcMfpmc2());
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 604 | Yes |
|-------------|-----|

## References

- *ppcMfmmcr0()*, p. 10-58
- *PowerPC 604 RISC Microprocessor User's Manual*



## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfpvr(void);
```

## Library

ppcLib.a

## Description

**ppcMfpvr()** returns the value of the processor version register, which indicates the version and revision of the PowerPC processor.

## Errors

None.

## Example

- Retrieve the current value of the processor version register. Processor version-specific code may require this value.
- ```
printf("This is processor version %x\n", ppcMfpvr());
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- PowerPC 603 RISC Microprocessor User's Manual*
- PowerPC 604 RISC Microprocessor User's Manual*

ppcMfsda()

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfsda(void);
```

Library

ppcLib.a

Description

ppcMfsda() returns the value of the Sample Data Address register (SDA). The SDA contains the effective address of an operand of an instruction executing around the time a performance monitor interrupt condition was signaled by the processor.

Errors

None.

Example

- Get the value of the SDA

```
unsigned long sda_value=ppcMfsda();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	No
PowerPC 604	Yes

References

- *PowerPC 604 RISC Microprocessor User's Manual*

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfsdr1(void);
```

Library

ppcLib.a

Description

ppcMfsdr1() returns the value of Storage Description Register 1, which contains high order address bits of the Page Frame Table used in address translation and the hash mask.

Errors

None.

Example

- Retrieve the value of the storage description register 1. This register provides the anchor for structures used to manage virtual storage mapping for the PowerPC 6xx processors.

```
unsigned long current_sdr1=ppcMfsdr1();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- PowerPC 603 RISC Microprocessor User's Manual*
- PowerPC 604 RISC Microprocessor User's Manual*

ppcMfsia()

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfsia(void);
```

Library

ppcLib.a

Description

ppcMfsia() returns the value of the Sample Instruction Address register (SIA). The SIA contains the effective address of an instruction executing around the time a performance monitor interrupt condition was signaled by the processor.

Errors

None.

Example

- Get the value of the SIA.

```
unsigned long sia_value=ppcMfsia();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	No
PowerPC 604	Yes

References

- *PowerPC 604 RISC Microprocessor User's Manual*

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfsprg0(void);
unsigned long ppcMfsprg1(void);
unsigned long ppcMfsprg2(void);
unsigned long ppcMfsprg3(void);
```

Library

ppcLib.a

Description

ppcMfsprg0() - ppcMfsprg3() returns the current value of the special purpose register generals (SPRG0 - SPRG3).

Typically, the SPRGs provide temporary storage at the operating system level.

NOTE: OS Open reserves these registers for its own use.

Errors

None.

Example

- Read value of SPRG0.

```
unsigned long sprg0_value = ppcMfsprg0();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

ppcMfsr()

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfsr(unsigned long seg_offset);
```

Library

ppcLib.a

Description

ppcMfsr() returns the value of the segment register given by *seg_offset*.

Valid values for *seg_offset* are 0-15 inclusive.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 602	Yes
PowerPC 603	Yes
PowerPC 604	Yes

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfsrr0(void);
```

Library

ppcLib.a

Description

ppcMfsrr0() returns the value of SRR0.

Typically, SRR0 is used in interrupt handlers, as it usually contains the address of the next instruction to be executed at the time of the interrupt.

Errors

None.

Example

- Retrieve the current value of the SRR0. An exception handler may use this value to determine the point of exception.

```
unsigned long current_srr0=ppcMfsrr0();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- ppcMfsrr1(), p. 10-70
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

ppcMfsrr1()

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfsrr1(void);
```

Library

ppcLib.a

Description

ppcMfsrr1() returns the current value of SRR1.

Typically, SRR1 is used in interrupt handlers, as it contains the old MSR value as well as information bits specific to the interrupt.

Errors

None.

Example

- Retrieve the current value of SRR1. This register contains the saved MSR, which may be needed by an exception handler.

```
unsigned long current_srr1=ppcMfsrr1();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

Synopsis

```
#include <ppcLib.h>
void ppcMftb(tb_t *clock_data);
```

Library

ppcLib.a

Description

ppcMftb() returns the current time base data.
Typically, the time base registers are used to determine the number of clock cycles that have passed.

Errors

None.

Example

```
• Retrieve the current value of time base high and low registers.

tb_t clock_data;
ppcMftb(&clock_data);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

ppcMtdabr()

Synopsis

```
#include <ppcLib.h>
void ppcMtdabr(unsigned long);
```

Library

ppcLib.a

Description

ppcMtdabr() sets the value of the Data Address Breakpoint register. The DABR provides a means to detect data access to a specific double word. The data address that is returned from the DABR is on a double word boundary. Bits 0-28 contains the data address (DAB), bit 29 is the breakpoint translation enable (BT) bit, bit 30 is the data write enable (DW) bit, and bit 31 is the data read enable (DR) bit. When a match is detected a DSI exception is created.

Errors

None.

Example

- Set the DABR so that a DSI exception occurs only when data is written to the real 0x0000a000 location.

```
ppcMtdabr(0x0000a002);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 604	Yes
-------------	-----

References

- *PowerPC 604 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The programming Environments*

Synopsis

```
#include <ppcLib.h>
void ppcMtdar(unsigned long dar_value);
```

Library

ppcLib.a

Description

ppcMtdar() sets the value of the DAR to *dar_value*.

Errors

None.

Example

- Set the data address register to 0x1234. It may be necessary to simulate an exception condition to subsequent software.

```
ppcMtdar(0x1234);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 602	Yes
PowerPC 603	Yes
PowerPC 604	Yes

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

ppcMtdbat0l() - ppcMtdbat3l()

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMtdbat0l(void)
unsigned long ppcMtdbat1l(void)
unsigned long ppcMtdbat2l(void)
unsigned long ppcMtdbat3l(void)
```

Library

ppcLib.a

Description

ppcMtdbat0l() - ppcMtdbat3l() sets the current value of the specified lower Data Block Address Translation Register (DBAT). The DBAT maintains the address translation information for a block of memory. Each lower DBAT is paired with an upper DBAT register.

Errors

None.

Example

- Retrieve the current value of the DBAT3L register and set to inhibit caching.

```
unsigned long dbat3l_value= ppcMfdbat3l();
ppcMtdbat3l(dbat3l_value | 0x00000020);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 602	Yes
PowerPC 603	Yes
PowerPC 604	Yes

References

- *ppcMtdbat0u() - ppcMtdbat3u()*, p. 10-75
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The Programming Environments*

Synopsis

```
#include <ppcLib.h>

unsigned long ppcMtdbat0u(void)
unsigned long ppcMtdbat1u(void)
unsigned long ppcMtdbat2u(void)
unsigned long ppcMtdbat3u(void)
```

Library

ppcLib.a

Description

ppcMtdbat0u() - ppcMtdbat3u() sets the value of the specified upper Data Block Address Translation Register (DBAT). The DBAT maintains the address translation information for a block of memory. Each upper DBAT is paired with a lower DBAT register.

Errors

None.

Example

- Retrieve the current value of the DBAT3U register and update the block length to 1 Mbytes.

```
unsigned long dbat3u_value= ppcMfdbat3u();
ppcMtdbat3u(dbat3u_value & 0xfffffe01f);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 602	Yes
PowerPC 603	Yes
PowerPC 604	Yes

References

- *ppcMtdbat0l() - ppcMtdbat3l()*, p. 10-74
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The programming Environments*

ppcMtdec()

Synopsis

```
#include <ppcLib.h>
void ppcMtdec(unsigned long dec_value);
```

Library

ppcLib.a

Description

ppcMtdec() sets the value of the decrementer to *dec_value*.

The decrementer is a continuously running 32-bit binary down counter that can assert an interrupt request whenever bit 0 (MSB) changes from 0 to 1. Although not all of the least significant bits may be implemented, bit 31 effectively changes every nanosecond.

Errors

None.

Example

- Set the decrementer register to X'0'. A decrementer exception handler may need to reset the value of the decrementer to a specific value.

```
ppcMtdec(0x0);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

Synopsis

```
#include <ppcLib.h>

void ppcMtdsisr(unsigned long dsisr_value);
```

Library

ppcLib.a

Description

ppcMtdsisr() sets the DSISR register to *dsisr_value*.

Errors

None.

Example

- Set the data storage interrupt status register to X'00004321'.
- ```
ppcMtdsisr(0x00004321);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 602 | Yes |
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcMtear()

---

## Synopsis

```
#include <ppcLib.h>
void ppcMtear(unsigned long ear_value);
```

## Library

ppcLib.a

## Description

**ppcMtear()** set the External Access Register (EAR) to the specified value. This register identifies the target device for external control operations. Bit 0 is the enable bit for **eciw**x and **ecow**x instructions. Bits 1-25 are reserved. Bits 26-31 contain the resource ID.

## Errors

None.

## Example

- Set the value of the EAR to enable **eciw**x and **ecow**x instructions for target device 0xa.

```
ppcMtear(0x1000000a);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 602 | Yes |
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The Programming Environments*



Synopsis

```
#include <ppcLib.h>
void ppcMthid0(unsigned long hid0_value)
```

Library

ppcLib.a

Description

**ppcMthid0()** set the value of the Hardware Implementation Dependent Register 0 (HID0).

Errors

None.

Example

```
• Set the Instruction Cache Enable bit in HID0.
 unsigned long hid0_value= ppcMfhid0();
 ppcMthid0(hid0_value|0x00008000);
```

Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcMtiabr()

---

## Synopsis

```
#include <ppcLib.h>
void ppcMfiabr(iunsigned long)
```

## Library

ppcLib.a

## Description

**ppcMtiabr()** sets the specified value of the Instruction Address Breakpoint Register(IABR). When the IABR is enabled, the instruction fetch address will be compared with the effective address that is stored in the IABR. The compare will be done on a word boundary. If the IABR and the instruction fetch address match, the instruction address breakspoint exception will be taken. Bits 0-29 is the address of the requested instruction. Bit 30 is the breakpoint enable bit. Bit 31 is reserved for other processors.

## Errors

None.

## Example

- Set the IABR to invoke the instruction breakpoint handler before the instruction at the real address of 0x0000a004 is executed.

```
ppcMtiabr(0x0000a006);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMtibat0l(void)
unsigned long ppcMtibat1l(void)
unsigned long ppcMtibat2l(void)
unsigned long ppcMtibat3l(void)
```

## Library

ppcLib.a

## Description

**ppcMtibat0l() - ppcMtibat3l()** sets the current value of the specified lower Instruction Block Address Translation Register (IBAT). The IBAT maintains the address translation information for a block of memory. Each lower IBAT is paired with an upper IBAT register.

## Errors

None.

## Example

- Retrieve the current value of the IBAT3L register and set to inhibit caching.

```
unsigned long ibat3l_value= ppcMfibat3l();
ppcMtibat3l(ibat3l_value | 0x00000020);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- ppcMtdbat0u() - ppcMtdbat3u()*, p. 10-75
- PowerPC 603 RISC Microprocessor User's Manual*
- PowerPC 604 RISC Microprocessor User's Manual*
- PowerPC Microprocessor Family: The programming Environments*

# ppcMtibat0u() - ppcMtibat3u()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcMtibat0u(void)
unsigned long ppcMtibat1u(void)
unsigned long ppcMtibat2u(void)
unsigned long ppcMtibat3u(void)
```

## Library

ppcLib.a

## Description

**ppcMtibat0u()** - **ppcMtibat3u()** sets the value of the specified upper Instruction Block Address Translation Register (IBAT). The IBAT maintains the address translation information for a block of memory. Each upper IBAT is paired with a lower IBAT register.

## Errors

None.

## Example

- Retrieve the current value of the IBAT3U register and update the block length to 1 Mbytes.

```
unsigned long ibat3u_value= ppcMtibat3u();
ppcMtibat3u(ibat3u_value & 0xfffffe01f);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *ppcMtibat0l()* - *ppcMtibat3l()*, p. 10-81
- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*
- *PowerPC Microprocessor Family: The programming Environments*

## Synopsis

```
#include <ppcLib.h>
void ppcMtmmcr0(unsigned long);
```

## Library

ppcLib.a

## Description

**ppcMtmmcr0()** sets the value of the Monitor Mode Control Register 0 (MMCR0). The MMCR0 is partitioned into bit fields. The bit fields determine the events to be counted and recorded.

## Errors

None.

## Example

- Set the value of the MMCR0 to allow performance monitoring interrupt signaling.

```
ppcMtmmcr0 (0x04000000) ;
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 604 | Yes |
|-------------|-----|

## References

- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcMtmsr()

---

## Synopsis

```
#include <ppcLib.h>
void ppcMtmsr(unsigned long msr_value);
```

## Library

ppcLib.a

## Description

**ppcMtmsr()** sets the MSR to *msr\_value*.

The file **<ppc\_arch.h>** defines constants that can be use with the MSR.

## Errors

None.

## Example

- Enable external interrupts.

```
unsigned long msr = ppcMfmsr();
ppcMtmsr(msr | ppcMsrEE);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcMtpir(unsigned long pir_value);
```

## Library

ppcLib.a

## Description

**ppcMtpir()** sets the value of the Processor Identification Register(PIR), which contains a processor identification tag in the four least significant bits(28-31). This register value is useful in processor differentiation in multiprocessor system designs. This register is also used for several direct-store bus operations in the form of a “bus transaction from” tag.

## Errors

None.

## Example

- Set the value of the processor identification register to 0x1.

```
ppcMtpir(0x00000001);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 604 | Yes |
|-------------|-----|

## References

- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcMtpmc1()

---

## Synopsis

```
#include <ppcLib.h>
void ppcMtpmc1(unsigned long);
```

## Library

ppcLib.a

## Description

**ppcMtpmc1()** sets the value of the Performance Monitor Counter Register 1 (PMC1). The PMC1 is a 32 bit counter that can be programmed to generate interrupt signals when it becomes negative. An interrupt signal is not generated unless both the MMCR0(PMC1INTCONTROL) and MMCR0(ENINT) bits are also set.

## Errors

None.

## Example

- Set the current value of the PMC1 register to a negative value so that an interrupt signal is generated.

```
ppcMtpmc1(0x80000000);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 604 | Yes |
|-------------|-----|

## References

- *ppcMtmocr0()*, p. 10-83
- *PowerPC 604 RISC Microprocessor User's Manual*



Synopsis

```
#include <ppcLib.h>
void ppcMtpmc1(unsigned long);
```

Library

ppcLib.a

Description

**ppcMtpmc2()** sets the value of the Performance Monitor Counter Register2 (PMC2). The PMC2 is a 32 bit counter that can be programmed to generate interrupt signals when it becomes negative. An interrupt signal is not generated unless both the MMCR0(PMC2INTCONTROL) and MMCR0(ENINT) bits are also set.

Errors

None.

Example

- Set the current value of the PMC2 register to a negative value so that an interrupt signal is generated.
- ```
ppcMtpmc2( 0x80000000 ) ;
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 604	Yes
-------------	-----

References

- *ppcMtmocr0()*, p. 10-83
- *PowerPC 604 RISC Microprocessor User's Manual*

ppcMtsda()

Synopsis

```
#include <ppcLib.h>
void ppcMtsda(unsigned long);
```

Library

ppcLib.a

Description

ppcMtsda() sets the value of the Sample Data Address register (SDA). The SDA contains the effective address of an operand of an instruction executing around the time a performance monitor interrupt condition was signaled by the processor.

Errors

None.

Example

- Set the value of the SDA to 0x0000a000.

```
ppcMtsda(0x0000a000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 604	Yes
-------------	-----

References

- *PowerPC 604 RISC Microprocessor User's Manual*

Synopsis

```
#include <ppcLib.h>
void ppcMtsdr1(unsigned long sdr1_value);
```

Library

ppcLib.a

Description

ppcMtsdr1() sets the SDR1 to *sdr1_value*.

Errors

None.

Example

- Set the storage description register 1 to X'A0000'. Setting the SDR1 register would usually be done by software managing virtual translation.

```
ppcMtsdr1( 0xA0000 );
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

ppcMtsia()

Synopsis

```
#include <ppcLib.h>
void ppcMtsia(unsigned long);
```

Library

ppcLib.a

Description

ppcMtsia() sets the value of the Sample Instruction Address register (SIA). The SIA contains the effective address of an instruction executing around the time a performance monitor interrupt condition was signaled by the processor.

Errors

None.

Example

- Set the value of the SIA to 0x0000a000.

```
ppcMtsia(0x0000a000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 604	Yes
-------------	-----

References

- *PowerPC 604 RISC Microprocessor User's Manual*

Synopsis

```
#include <ppcLib.h>
void ppcMtsprg0(unsigned long data);
void ppcMtsprg1(unsigned long data);
void ppcMtsprg2(unsigned long data);
void ppcMtsprg3(unsigned long data);
```

Library

ppcLib.a

Description

ppcMtsprg0() - ppcMtsprg3() set the special purpose register generals (SPRG0 - SPRG3) to the specified values.

Typically, the SPRGs provide temporary storage at the operating system level.

NOTE: OS Open reserves these registers for its own use.

Errors

None.

Example

- Set SPRG0 to 0xA0000000.

```
ppcMtsprg0 ( 0xA0000000 ) ;
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

ppcMtsr()

Synopsis

```
#include <ppcLib.h>
void ppcMtsr(unsigned long seg_offset, unsigned long value);
```

Library

ppcLib.a

Description

ppcMtsr() sets the segment register given by *seg_offset* to *value*.

Errors

None.

Example

- Set segment register 7 to memory forced I/O.

```
ppcMtsr(7, 0x87F00007);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 603	Yes
PowerPC 604	Yes

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

Synopsis

```
#include <ppcLib.h>
void ppcMtsrr0(unsigned long srr0_value);
```

Library

ppcLib.a

Description

ppcMtsrr0() sets the SRR0 to *srr0_value*.

Errors

None.

Example

- Set the save/restore register 0 to X'DF000000'.
- ```
ppcMtsrr0(0xDF000000);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- PowerPC 603 RISC Microprocessor User's Manual*
- PowerPC 604 RISC Microprocessor User's Manual*

# ppcMtsrr1()

---

## Synopsis

```
#include <ppcLib.h>
void ppcMtsrr1(unsigned long srr1_value);
```

## Library

ppcLib.a

## Description

**ppcMtsrr1()** sets the SRR1 to *srr1\_value*.

## Errors

None.

## Example

- Set the save/restore register 1 to X'0000BB00'.

```
ppcMtsrr1(0x0000BB00);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*



Synopsis

```
#include <ppcLib.h>
void ppcMttb(tb_t *clock_data);
```

Library

ppcLib.a

Description

**ppcMttb()** sets the current time base data.  
Typically, the time base registers are used to determine the number of clock cycles that have passed.

Errors

None.

Example

```
• Set the current value of time base high and low registers.

tb_t clock_data;
ppcMttb(0x00000000);
```

Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# ppcOrMsr()

---

## Synopsis

```
#include <ppcLib.h>
unsigned long ppcOrMsr(unsigned long value);
```

## Library

ppcLib.a

## Description

**ppcOrMsr()** performs the OR of *value* and the current MSR, updating the MSR.

The previous value of the MSR is returned.

The file **<ppcLib.h>** defines several constants for the MSR that can be used as masks.

## Errors

None.

## Example

- Enable instruction address translation.

```
unsigned long old_val = ppcOrMsr(ppcMsrIR);
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | No  |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <ppcLib.h>
void ppcSync(void);
```

## Library

ppcLib.a

## Description

**ppcSync()** causes the processor to wait until all data cache lines scheduled to be written to main storage have actually been written.

## Errors

None.

## Example

- Ensure a **ppcDcbi()** completes before using the values.

```
char *memptr = (char *)0x2000;
char new_value;
ppcDcbi((void *)memptr)
ppcSync();
new_value = *memptr;
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# setup\_bat()

---

## Synopsis

```
#include <ioLib.h>

void setup_bat(bat_struct *bat_reg);
```

## Library

ioLib.a

## Description

**setup\_bat()** sets up the BAT registers to the values specified in the *bat\_reg* structure, turns on instruction and data translation in the MSR. With OS Open without Virtual Memory **setup\_bat()** also sets up the Segment Registers. The Segment Registers 0 through 15 are assigned to the following values.

| Segment Register | Value      |
|------------------|------------|
| SR0              | 0x0        |
| SR1              | 0x80000001 |
| SR2              | 0x80000002 |
| SR3              | 0x80000003 |
| SR4              | 0x80000004 |
| SR5              | 0x80000005 |
| SR6              | 0x80000006 |
| SR7              | 0x80000007 |
| SR8              | 0x80000008 |
| SR9              | 0x80000009 |
| SR10             | 0x8000000A |
| SR11             | 0x8000000B |
| SR12             | 0x8000000C |
| SR13             | 0x8000000D |
| SR14             | 0x8000000E |
| SR15             | 0x8000000F |

**Setup\_bat()** function must be called before any other I/O related function. For the PowerPC 604 superscalar execution bit in the HID0 register is also turned on by **setup\_bat()**. BAT register memory mapping must not overlap.

## Errors

None.

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- dbg\_ioLib\_init(), p. 10-16
- ioLib\_init(), p. 10-30

# s1dbprintf()

---

## Synopsis

```
#include <sys/asyncLib.h>
```

```
int s1dbprintf(unsigned long uart_clock, unsigned char *base_reg, int
reg_delta, event_t event, const char *format,...);
```

## Library

asyncLib.a

## Description

**s1dbprintf()** is a version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established. **s1dbprintf()** may be called before the async device driver is installed. **uart\_clock** is the clock frequency of the serial port. **base\_reg** specifies the address of the base UART register. **reg\_delta** specifies the space between UART registers. **event** specifies the external interrupt level. For the 60x/7xx EVB, **uart\_clock** must be 1843200, **base\_reg** must be 0x800003F8, **reg\_delta** should be 1.

## Errors

None.

## Example

- Print “Hello World” before I/O has been initialized.

```
#include <sys/asyncLib.h>
#define S1DB_PARAMS 1843200, (unsigned char *)
0x800003F8, 1, EXT_IRQ_COM1
s1dbprintf(S1DB_PARAMS, "Hello World\n\r");
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <sys/asyncLib.h>

int s1dbprintf(unsigned long uart_clock, unsigned char *base_reg, int
reg_delta, event_t event, const char *format,...);
```

## Library

asyncLib.a

## Description

**s1dbprintfapp()** is a version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established. **s1dbprintfapp()** may be called before the async device driver is installed. **uart\_clock** is the clock frequency of the serial port. **base\_reg** specifies the address of the base UART register. **reg\_delta** specifies the space between UART registers. **event** specifies the external interrupt level. For 60x/7xx EVB, **uart\_clock** must be 1843200, **base\_reg** must be 0x800003F8, **reg\_delta** should be 1. **s1dbprintfapp()** may be called from an application thread group.

**Note:** **s1dbprintfapp()** is only available with OS Open with Virtual Memory.

## Errors

None.

## Example

- Print "Hello World" using polled mode.

```
#include <sys/asyncLib.h>
#define S1DB_PARAMS 1843200, (unsigned char *)
0x800003F8, 1, EXT_IRQ_COM1
s1dbprintfapp(S1DB_PARAMS, "Hello World\n\r");
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | Yes |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

# s1dbprintfapp()

---

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*



## Synopsis

```
#include <sys/asyncLib.h>

int s2dbprintf(unsigned long uart_clock, unsigned char *base_reg, int
reg_delta, event_t event, const char *format,...);
```

## Library

asyncLib.a

## Description

**s2dbprintf()** is a version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established. **s2dbprintf()** may be called before the async device driver is installed. **uart\_clock** is the clock frequency of the serial port. **base\_reg** specifies the address of the base UART register. **reg\_delta** specifies the space between UART registers. **event** specifies the external interrupt level. For the 60x/7xx EVB, **uart\_clock** must be 1843200, **base\_reg** must be 0x800002F8, **reg\_delta** should be 1.

## Errors

None.

## Example

- Print "Hello World" before I/O has been initialized.

```
#include <sys/asyncLib.h>
#define S2DB_PARAMS 1843200, (unsigned char *)
0x800002F8, 1, EXT_IRQ_COM2
s2dbprintf(S2DB_PARAMS, "Hello World\n\r");
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# s2dbprintfapp()

---

## Synopsis

```
#include <sys/asyncLib.h>

int s2dbprintf(unsigned long uart_clock, unsigned char *base_reg, int
reg_delta, event_t event, const char *format,...);
```

## Library

asyncLib.a

## Description

**s2dbprintfapp()** is a version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established. **s2dbprintfapp()** may be called before the async device driver is installed. **uart\_clock** is the clock frequency of the serial port. **base\_reg** specifies the address of the base UART register. **reg\_delta** specifies the space between UART registers. **event** specifies the external interrupt level. For 60x/7xx EVB, **uart\_clock** must be 1843200, **base\_reg** must be 0x800002F8, **reg\_delta** should be 1. **s2dbprintfapp()** may be called from an application thread group.

**Note:** **s2dbprintfapp()** is only available with OS Open with Virtual Memory.

## Errors

None.

## Example

Print "Hello World" using polled mode.

```
#include <sys/asyncLib.h>
#define S2DB_PARMS 1843200, (unsigned char *)
0x800002F8, 1, EXT_IRQ_COM2
s2dbprintfapp(S2DB_PARMS, "Hello World\n\r");
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | Yes |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

### References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

# timertick\_install()

---

## Synopsis

```
#include <tickLib.h>
int timertick_install(void);
```

## Library

tickLib.a

## Description

**timertick\_install()** installs and starts the timer tick handler to maintain time-of-day in the OS Open real-time executive.

## Errors

[ENOMEM]                      Insufficient memory to install the timer tick handler.

## Example

- Do a **timertick\_install()** for a 603, 604, 7XX processor.

```
timertick_install();
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- **timertick\_remove()**, p. 10-107

Synopsis

```
#include <tickLib.h>
int timertick_remove(void);
```

Library

tickLib.a

Description

**timertick\_remove()** removes the timer tick handler installed by **timertick\_install()**.

Errors

[EINVAL]                      Internal error involving tick handler level.

Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

References

- timertick\_install(), p. 10-106

# vs1dbprintf()

---

## Synopsis

```
#include <sys/asyncLib.h>

int vs1dbprintf(unsigned long uart_clock, unsigned char *base_reg,
int reg_delta, event_t event, const char *format, va_list arg_list);
```

## Library

asyncLib.a

## Description

**vs1dbprintf()** is a version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established and accepts a *va\_list* as a parameter instead of a variable number of parameters. **vs1dbprintf()** may be called before the async device driver is installed. *uart\_clock* is the clock frequency of the serial port. *base\_reg* specifies the address of the base UART register. *reg\_delta* specifies the space between UART registers. *event* specifies the external interrupt level. For the 60x/7xx EVB, *uart\_clock* must be 1843200, *base\_reg* must be 0x800003F8, *reg\_delta* should be 1.

## Errors

None.

## Example

- Print “Hello World” before I/O has been initialized.

```
#include <sys/asyncLib.h>
#define S1DB_PARMS 1843200, (unsigned char *)
0x800003F8, 1, EXT_IRQ_COM1

vs1dbprintf(S1DB_PARMS, "Hello World\n\r");
```

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- *PowerPC 603 RISC Microprocessor User's Manual*
- *PowerPC 604 RISC Microprocessor User's Manual*

## Synopsis

```
#include <enetLib.h>
int zapatos_enet_setup(void);
```

## Library

enetLib.a

## Description

**zapatos\_enet\_setup()** performs the PCI configuration cycles to map the 60x/7xx EVB Ethernet chip to system address 0X80100000.

## Errors

[EINVAL] Internal error involving tick handler level.

## Attributes

|                                        |     |
|----------------------------------------|-----|
| Async Safe                             | Yes |
| Cancel Safe                            | Yes |
| Interrupt Handler Safe                 | Yes |
| Callable from Application Thread Group | No  |

## Processors

|             |     |
|-------------|-----|
| PowerPC 603 | Yes |
| PowerPC 604 | Yes |

## References

- PowerPC 604 SMP Reference Design Technical Specification, Release 3.0, MPRZAPRSU-03

## **zapatos\_enet\_setup()**

---





# Program Trace Calls

This appendix describes the remote debugging interface provided by the ROM monitor. These calls may be used by remote debuggers other than the **RISCWatch** debugger provided with the 60x/7xx EVB kit.

## A.1 Overview

The following section describes the message (ptrace) protocol that has been implemented in the ROM monitor to support debug. If you want to interface your own debugger to the ROM monitor or modify the ROM monitor to interface with your debugger, you will need to understand the existing message protocol associated with the various debugging functions.

The ptrace interface to the ROM monitor can best be understood by reviewing the information below along with the debug-specific ROM monitor source code (dbLib/ptrace.c).

## A.2 MSGDATA Structure

In the interface descriptions shown below, several references are made to a "process id." The concept of process ids does not apply to the ROM monitor, so any nonzero value can be used. The ROM monitor uses the value "42".

Data structure "MSGDATA" is defined in dbg.h. New register definitions and new error messages are also defined in dbg.h file.

### dbg.h File

```
/* @(#)dbg.h 4.3 5/9/95 09:12:14 */
/*-----+
| COPYRIGHT I B M CORPORATION 1994
| LICENSED MATERIAL - PROGRAM PROPERTY OF I B M
| REFER TO COPYRIGHT INSTRUCTIONS: FORM G120-2083
| US Government Users Restricted Rights - Use, duplication or
| disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
|-----*/
#ifndef !defined(DBG_H)
#define DBG_H
#define BREAKPT 0x7D821008
```

```

#ifndef MIN
#define MIN(X,Y) ((X) < (Y) ? (X) : (Y))
#endif

/*ptrace definitions based on AIX ptrace */
#define RD_TRACE_ME 0 /* used ONLY by target task to be traced*/
#define RD_READ_I 1 /* read target instruction addr space */
#define RD_READ_D 2 /* read target data address space */
#define RD_READ_U 3 /* read offset from the user structure */
#define RD_WRITE_I 4 /* write target instruction addr space */
#define RD_WRITE_D 5 /* write target data address space */
#define RD_WRITE_U 6 /* write offset to the user structure */
#define RD_CONTINUE 7 /* continue execution */
#define RD_KILL 8 /* terminate execution */
#define RD_STEP 9 /**execute one or more instructions** */
#define RD_READ_GPR 11 /* read general purpose register */
#define RD_READ_FPR 12 /* read floating point register */
#define RD_WRITE_GPR 14 /* write general purpose register */
#define RD_WRITE_FPR 15 /* write floating point register */
#define RD_READ_BLOCK 17 /* read block of data */
#define RD_WRITE_BLOCK 19 /* write block of data */
#define RD_ATTACH 30 /* attach to a process */
#define RD_DETACH 31 /* detach a proc to let it keep running */
#define RD_REGSET 32 /* return entire register set to caller */
#define RD_REATT 33 /* reattach debugger to proc */
#define RD_LDINFO 34 /* return loaded program info */
#define RD_MULTI 35 /* set/clear multi-processing */
#define RD_READ_I_MULT 70 /* Read multiple inst words */
#define RD_READ_GPR_MULT 71 /* Read multiple registers */
#define RD_SINGLE_STEP 100 /**source line single step***** */
#define RD_LOAD 101 /* load a task */
#define RD_LOGIN 103 /*ptrace for login */
#define RD_LOGON 103 /*ptrace for logon */
#define RD_LOGOFF 104 /*ptrace for logoff */
#define RD_FILL 105 /*ptrace for fill memory */
#define RD_PASS 106 /*ptrace for pass */
#define RD_SEARCH 107 /*ptrace for search memory */
#define RD_WAIT 108 /*ptrace for wait status information */
/* Added to support ADEPT */
#define RD_READ_DCR 110 /*ptrace for reading DCR's */
#define RD_WRITE_SPR 111 /*ptrace for writing SPR's */
#define RD_WRITE_DCR 112 /*ptrace for writing DCR's */
#define RD_STOP_APPL 113 /*ptrace for stopping the application */
#define RD_STATUS 114 /*ptrace for getting run status */
#define RD_READ_SPR 115 /*ptrace for reading SPR's */
/* Added to support 403GC */
#define RD_READ_TLB 116 /*ptrace for readingTLB(403GC) */
#define RD_WRITE_TLB 117 /*ptrace for writing TLB(403GC) */
/* Added to support 602 */
#define RD_READ_SR 118 /*ptrace for reading SR's */
#define RD_WRITE_SR 119 /*ptrace for writing SR's */

```

```

#define MAX_PTRACE 119 /*last ptrace number */
#define RL_LOAD_REQ 180 /* Remote Loader - Load Request */
#define RL_LDINFO 181 /* Remote Loader - Load Information */
/*TCP/IP services for all sorts of remote debug */
#define OSOPEN_SERVNAME "osopen-dbg" /* OS/Open debug service */
#define OSOPEN_MON_SERVNAME "osopen-mon" /* OS/Open debug monitor svc */
/*new register definition
#define DAR 137 /* Data Address Register ($dar) */
#define DSISR 138 /* Data St Int Status Reg ($dsisr) */
#define SRR0 139 /* Save and Restore Register 0 ($srr0) */
#define SRR1 140 /* Save and Restore Register 0 ($srr1) */
#define SR0 141 /* Segment Register ($sr0) */
#define SR1 142 /* Segment Register ($sr1) */
#define SR2 143 /* Segment Register ($sr2) */
#define SR3 144 /* Segment Register ($sr3) */
#define SR4 145 /* Segment Register ($sr4) */
#define SR5 146 /* Segment Register ($sr5) */
#define SR6 147 /* Segment Register ($sr6) */
#define SR7 148 /* Segment Register ($sr7) */
#define SR8 149 /* Segment Register ($sr8) */
#define SR9 150 /* Segment Register ($sr9) */
#define SR10 151 /* Segment Register ($sr10) */
#define SR11 152 /* Segment Register ($sr11) */
#define SR12 153 /* Segment Register ($sr12) */
#define SR13 154 /* Segment Register ($sr13) */
#define SR14 155 /* Segment Register ($sr14) */
#define SR15 156 /* Segment Register ($sr15) */
#define DEC 157 /* Decrementer ($dec) */
#define RTCU 158 /* Real Time Clock Upper ($rtcu) */
#define RTCL 159 /* Real Time Clock Lower ($rtcl) */
#define SDR0 160 /* Storage Description Reg ($sdr0) */
#define SDR1 161 /* Storage Description Reg ($sdr1) */
#define EIS0 162 /* External Int Summary Reg1($eis1) */
#define EIS1 163 /* External Int Summary Reg2($eis2) */
#define EIM0 164 /* External Int Mask Reg1($eim1) */
#define EIM1 165 /* External Int Mask Reg2($eim2) */
#define SRR2 166 /* Save and Restore Register 2 ($srr2) */
#define SRR3 167 /* Save and Restore Register 3 ($srr3) */
/*other definitions needed for remote debug
#define RD_MAXDATA 1800 /* Total no of DWORDS in a MSGDATA */
#define RD_MINLENGTH 6 /* Min no of dwords in msg */
#define RD_MINBYTES (RD_MINLENGTH*sizeof(unsigned long))
#define RD_MAXBUFFER (RD_MAXDATA - RD_MINLENGTH)
#define RD_MAXPACKET 1000000 /* Max bytes in TCP/IP packet */
#define RD_REGBYTES (32+8)*4 /* No of bytes for all registers */
#define NO_KILL 1 /*do not kill any users processes */
#define KILL_PROC 0 /*kill user process upon logoff */
#define MAX_ERROR 1014 /*last error for rptrace */
#define MIN_ERROR 1000 /*first error for rptrace */
#define MIN_PACKET_SIZE 24

```

```

#define DBG_SPORT 20044
#define DBG_DPORT 20050
/*new error codes */
#define RD_NLOAD_ERR 1000 /*no loader info available */
#define RD_COM_ERR 1001 /*communication error occurred */
#define RD_SIZE_ERR 1002 /*not enough room to pass all info */
#define RD_NOTSUPP 1003 /*call not supported */
#define RD_REG_ERR 1004 /*invalid register number requested */
#define RD_NOTAVAIL 1005 /*call not implemented at this time */
#define RD_NOFILE_ERR 1006 /*file could not be loaded, no file */
#define RD_NOSCAN_ERR 1008 /*could not locate scan string file */
#define RD_NOPERM 1010 /*no permission to log on */
#define RD_INVALID_SEQ 1011 /*invalid rptrace sequence */
#define RD_BUSY_ERR 1012 /*some users is already logged on */
#define RD_PTRACE_ERR 1014 /*internal ptrace error */
#define RD_OK 0 /*rptrace completed ok */
#define ARCH_403 0x34000000 /* 403 architecture */
#define ARCH_601 0x36000000 /* 601 architecture */
#define ARCH_602 0x36303200 /* 602 architecture */
#define ARCH_603 0x36303300 /* 603 architecture */
#define ARCH_604 0x36303400 /* 604 architecture */
typedef struct msgdata /* message data structure */
{
 unsigned long data_len; /* optional data length */
 unsigned long retcode; /* return code */MIN
 unsigned long request; /* request type */PART
 unsigned long address; /* function parameter */=
 unsigned long data /* function parameter */6*DWORD */
 struct {
 unsigned f1:1;
 unsigned f2:1;
 unsigned f3:1;
 unsigned padd:21;
 unsigned f25:8;
 } flags;
#define printmsg flags.f1
#define breakpt flags.f2
#define dbg_seqno flags.f25
 union {
 unsigned long trace_buffer[RD_MAXBUFFER];
 unsigned long processid;
 } parameter;
#define buffer parameter.trace_buffer /* buffer for data, in any */
#define rpid parameter.processid /* process id */
} MSGDATA;
#endif

```

### A.3 Ptrace Definitions

The following section presents the application programming interface (API) for rptrace messages. One field that is not shown here, because it is common to every call, is the *msg.printmsg* flag. This may be set in an rptrace response where *msg.retcode* does not equal

RD\_OK. When the *msg.printmsg* flag is set it indicates that a text string is contained in *msg.buffer* and that this message should be displayed to the user. Typically this is an error message that provides more detail as to why the *rptrace* call failed to return RD\_OK.

Another field that is not shown is the *dbg\_seqno* field. The field provides a mechanism for recovering from lost requests and responses. If a request has the *dbg\_seqno* field as not zero, it is compared with the value from the previous request. If it matches, the action is not performed and instead, the previous response is sent. This allows the debugger to time-out and re-try requests without danger of performing the same function twice.

### A.3.1 RD\_ATTACH (30)

Attaches debugger to running process in target environment.

#### A.3.1.1 Request data

**Table A-1. RD\_ATTACH Request Table**

| Parameters                                    | Description                                                  |
|-----------------------------------------------|--------------------------------------------------------------|
| <i>msg.request</i> = RD_ATTACH                | Requested API function.                                      |
| <i>msg.rpid</i> = <i>process_id</i>           | Numeric process ID on the target system.(Any non zero value) |
| <i>msg.data_len</i> = <i>sizeof(msg.rpid)</i> | Length of additional data being sent.                        |

#### A.3.1.2 Response data

**Table A-2. RD\_ATTACH Response Table**

| Parameters                             | Description                                                            |
|----------------------------------------|------------------------------------------------------------------------|
| <i>msg.retcode</i> = ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist. |
| <i>msg.retcode</i> = EIO (5)           | One of the parameters is incorrect.                                    |
| <i>msg.retcode</i> = RD_COM_ERR (1001) | Communication error occurred.                                          |
| <i>msg.retcode</i> = RD_NOTSUPP (1003) | Call not supported for this interface.                                 |
| <i>msg.retcode</i> = RD_OK (0)         | Successful completion.                                                 |
| <i>msg.data_len</i> =0                 | No additional data                                                     |

### A.3.2 RD\_CONTINUE (7)

This request causes the process to resume execution. If the *dbg\_seqno* field of the request is zero, the response is not returned until the process stops due to a breakpoint or error. Otherwise, an immediate response is sent from the **RD\_CONTINUE** request and the debugger should send the **RD\_STATUS** request to see if the process has stopped.

#### A.3.2.1 Request data

Table A-3. RD\_CONTINUE Request Table

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_CONTINUE       | Requested API function.                  |
| msg.address= address           | This field is ignored by ROM monitor.    |
| msg.data= signal               | 0                                        |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.2.2 Response data

Table A-4. RD\_CONTINUE Response Table

| Parameters                     | Description                   |
|--------------------------------|-------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred. |
| msg.retcode= RD_OK (0)         | Successful completion.        |
| msg.data= 0                    |                               |

### A.3.3 RD\_DETACH (31)

Detaches debugger from running process in target environment. Debugged process is restarted and execution continues without debugger control.

#### A.3.3.1 Request data

Table A-5. RD\_DETACH Request Table

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_DETACH         | Requested API function.                  |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.data= 0                    | Ignored by ROM monitor.                  |
| msg.address=1                  | Ignored by ROM monitor.                  |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.3.2 Response data

Table A-6. RD\_DETACH Response Table

| Parameters                     | Description                                                                                                                |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| msg.retcode= ESRCH (3)         | The <i>msg.rpid</i> parameter identifies a process that does not exist, or a process that is currently not being debugged. |
| msg.retcode= RD_COM_ERR (1001) | Communications error occurred.                                                                                             |
| msg.retcode= RD_NOTSUPP (1003) | Call not supported for this interface.                                                                                     |
| msg.retcode= RD_OK (0)         | Successful completion.                                                                                                     |
| msg.retcode= EIO (5)           | One of the parameters is incorrect.                                                                                        |
| msg.data_len= 0                | No additional data is being sent.                                                                                          |

### A.3.4 RD\_FILL (105)

Fills memory with zeroes at the location specified by *address* for the number of bytes specified by *data*.

#### A.3.4.1 Request data

Table A-7. RD\_FILL Request Table

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_FILL           | Requested API function.                  |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.address= address           | Address of memory to fill with zeroes    |
| msg.data= count                | Number of bytes to fill with zeroes      |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.4.2 Response data

Table A-8. RD\_FILL Response Table

| Parameters                     | Description                            |
|--------------------------------|----------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communications error occurred.         |
| msg.retcode= RD_NOTSUPP (1003) | Call not supported for this interface. |
| msg.retcode= RD_OK (0)         | Successful completion.                 |
| msg.retcode= EIO (5)           | One of the parameters is incorrect.    |
| msg.data_len= 0                | No additional data is being sent.      |



### A.3.5 RD\_KILL (8)

This request causes the process to terminate the same way it would with an exit routine. The ROM monitor does not implement this function but simply returns an **RD\_OK** response for compatibility with older debuggers.

#### A.3.5.1 Request data

Table A-9. RD\_KILL Request Table

| Parameters                     | Description                             |
|--------------------------------|-----------------------------------------|
| msg.request= RD_KILL           | Requested API function.                 |
| msg.rpid= process_id           | Process ID of the process to be killed. |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.   |

#### A.3.5.2 Response data

Table A-10. RD\_KILL Response Table

| Parameters                     | Description                                                            |
|--------------------------------|------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                          |
| msg.retcode= RD_OK (0)         | Successful completion.                                                 |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist. |
| msg.data_len= 0                | Length of additional data being sent.                                  |

### A.3.6 RD\_LDINFO (34)

Request loader information from target environment. This information is provided to the ROM monitor in the boot header or by the **RL\_LDINFO** request. Refer to **ROM Monitor Load Format** section for more information.

#### A.3.6.1 Request data

**Table A-11. RD\_LDINFO Request Table**

| Parameters                     | Description                                                |
|--------------------------------|------------------------------------------------------------|
| msg.request= RD_LDINFO         | Requested API function.                                    |
| msg.rpid= process_id           | Process ID from which the loader information is requested. |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.                      |

#### A.3.6.2 Response data

**Table A-12. RD\_LDINFO Response Table**

| Parameters                        | Description                                                                                                 |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------|
| msg.retcode= RD_NOLOAD_ERR (1000) | No loader information is available.                                                                         |
| msg.retcode= ESRCH (3)            | The <i>msg.pid</i> parameter identifies a process that does not exist.                                      |
| msg.retcode= RD_COM_ERR (1001)    | Communication error occurred.                                                                               |
| msg.retcode= RD_SIZE_ERR (1002)   | Not enough room in the buffer to fit all load information.                                                  |
| msg.retcode= RD_OK (0)            | Successful completion.                                                                                      |
| msg.retcode= EIO (5)              | One of the parameters is incorrect.                                                                         |
| msg.buffer[0]= ldinfo_next        | Offset to next loader information segment. See note below.                                                  |
| msg.buffer[1]= fd                 | File descriptor for loaded object. In remote debug 0xFFFF FFFF should be returned (this is a space filler). |

**Table A-12. RD\_LDINFO Response Table**

| Parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Description                                                                                                                                                                                     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| msg.buffer[2]= textorig                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Starting text address.                                                                                                                                                                          |
| msg.buffer[3]= textsize                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Size of text.                                                                                                                                                                                   |
| msg.buffer[4]= dataorig                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Starting data address.                                                                                                                                                                          |
| msg.buffer[5]= datasize                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Size of data.                                                                                                                                                                                   |
| msg.buffer[6]= (char *)pathname                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Fully qualified filename of the object file.                                                                                                                                                    |
| msg.buffer[X]= (char *)membername                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Member name (used for shared library objects). <b>X</b> does not represent position on word boundary. A NULL has to be returned for the membername even if the debugged file has no membername. |
| msg.buffer[linfo_next]= linfo_next                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Next loader block (notice "linfo_next").                                                                                                                                                        |
| msg.data_len= "variable"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Set to length of data sent in msg.buffer. Data length will vary depending on the amount of information passed. Remember to count all the NULL characters.                                       |
| <p>Note: <i>linfo_next</i>=0 indicates that no further loader blocks are present, otherwise <i>linfo_next</i> contains the offset of the next loader block in the buffer. This is actually the length of the current block. For example, if the buffer contains three blocks of lengths 38, 40 and 41 bytes, the <i>linfo_next</i> fields would be 38, 40 and 0, respectively. Note also that the blocks do not have to be contiguous - it is possible that the end of one block may not directly abut the following block. This may occur if additional information or word-aligning padding is placed after the end of the member-name string. Path-name and member-name are strings terminated with a NULL character.</p> |                                                                                                                                                                                                 |

### A.3.7 RD\_LOAD (101)

Loads executable program. Full path name of the file to be loaded is passed in this message. The ROM monitor will respond by sending an **RL\_LOAD\_REQ** to the remote loader daemon port.

#### A.3.7.1 Request data

**Table A-13. RD\_LOAD Request Table**

| Parameters                       | Description                                                                                                      |
|----------------------------------|------------------------------------------------------------------------------------------------------------------|
| msg.request= RD_LOAD             | Requested API function.                                                                                          |
| msg.buffer= filename             | Name of file to load. A NULL character terminates filename. Filename contains fully qualified path to that file. |
| msg.data_len= strlen(filename)+1 | String length of filename plus NULL character.                                                                   |

#### A.3.7.2 Response data

**Table A-14. RD\_LOAD Response Table**

| Parameters                        | Description                                                                                           |
|-----------------------------------|-------------------------------------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001)    | Communication error occurred.                                                                         |
| msg.retcode= RD_OK (0)            | Successful completion.                                                                                |
| msg.retcode= RD_NOFILE_ERR (1006) | Could not locate/load the file.                                                                       |
| msg.rpid= process_id              | Process_id of the newly loaded file. This number (integer) can not be equal to -1 (0xFFFF FFFF) or 0. |
| msg.data_len= sizeof(msg.rpid)    | Length of additional data being sent.                                                                 |

### A.3.8 RD\_LOGIN (103)

Initializes users LOGIN. This request must be the first rptrace request issued by the debugger or results will be unpredictable.

#### A.3.8.1 Request data

**Table A-15. RD\_LOGIN Request Table**

| Parameters                                             | Description                           |
|--------------------------------------------------------|---------------------------------------|
| msg.request= RD_LOGIN                                  | Requested API function.               |
| msg.buffer[0]= host_name                               | This field is ignored by ROM monitor. |
| msg.buffer[strlen(host_name)+1]=<br>user_name          | This field is ignored by ROM monitor. |
| msg.data_len=<br>strlen(host_name)+strlen(user_name)+2 | Length of additional data being sent. |

#### A.3.8.2 Response data

**Table A-16. RD\_LOGIN Response Table**

| Parameters                     | Description                           |
|--------------------------------|---------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.         |
| msg.retcode= RD_OK (0)         | Successful completion.                |
| msg.data_len= 0                | Length of additional data being sent. |

### A.3.9 RD\_LOGOFF (104)

Performs user LOGOFF function. This is used when the debugger performs normal termination using quit or detach.

#### A.3.9.1 Request data

**Table A-17. RD\_LOGOFF Request Table**

| Parameters             | Description                           |
|------------------------|---------------------------------------|
| msg.request= RD_LOGOFF | Requested API function.               |
| msg.data= NO_KILL      | This field is ignored by ROM monitor. |
| msg.data_len= 0        | Length of additional data being sent. |

#### A.3.9.2 Response data

**Table A-18. RD\_LOGOFF Response Table**

| Parameters                         | Description                           |
|------------------------------------|---------------------------------------|
| msg.retcode= RD_COM_ERR (1001)     | Communication error occurred.         |
| msg.retcode= RD_OK (0)             | Successful completion.                |
| msg.retcode= RD_INVALID_SEQ (1011) | Not logged on.                        |
| msg.data_len= 0                    | Length of additional data being sent. |

### A.3.10 RD\_READ\_D (2)

This request returns the integer in the debugged process address space at the location pointed to by the *address* parameter. If the value of *address* is not in a valid address space, unpredictable results will occur.

#### A.3.10.1 Request data

**Table A-19. RD\_READ\_D Request Table**

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_READ_D         | Requested API function.                  |
| msg.address= address           | Address of memory to read data from.     |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.10.2 Response data

**Table A-20. RD\_READ\_D Response Table**

| Parameters                     | Description                                                            |
|--------------------------------|------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                          |
| msg.retcode= RD_OK (0)         | Successful completion.                                                 |
| msg.retcode= EIO (5)           | Debugged process can not access given address.                         |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist. |
| msg.data= data                 | Data read at location pointed to by address. -1 if error.              |
| msg.data_len= 0                | Length of additional data being sent.                                  |

### A.3.11 RD\_READ\_FPR (12)

This request returns the content of one of the floating-point registers.

#### A.3.11.1 Request data

**Table A-21. RD\_READ\_FPR Request Table**

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_READ_FPR       | Requested API function.                  |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.address= register          | Name of the register to be read.         |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.11.2 Response data

**Table A-22. RD\_READ\_FPR Response Table**

| Parameters                     | Description                                                            |
|--------------------------------|------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                          |
| msg.retcode= RD_OK (0)         | Successful completion.                                                 |
| msg.retcode= EIO (5)           | Register is not defined.                                               |
| msg.retcode= RD_REG_ERR (1004) | Unable to access given register.                                       |
| msg.data= value                | Value read from register. 0xFFFFFFFF if error occurred.                |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist. |
| msg.data_len= 0                | Length of additional data being sent.                                  |



### A.3.12 RD\_READ\_GPR (11)

This request returns the content of one of the general-purpose or special-purpose registers of the debugged process. Valid registers are defined in "dbg.h" and "sys/reg.h". Not all defined registers are supported for all environments.

#### A.3.12.1 Request data

**Table A-23. RD\_READ\_GPR Request Table**

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_READ_GPR       | Requested API function.                  |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.address= register          | Name of the register to be read.         |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.12.2 Response data

**Table A-24. RD\_READ\_GPR Response Table**

| Parameters                     | Description                                                            |
|--------------------------------|------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                          |
| msg.retcode= RD_OK (0)         | Successful completion.                                                 |
| msg.retcode= EIO (5)           | Register is not defined.                                               |
| msg.retcode= RD_REG_ERR (1004) | Unable to access given register.                                       |
| msg.data= value                | Value read from register. 0xFFFFFFFF if error occurred.                |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist. |
| msg.data_len= 0                | Length of additional data being sent.                                  |

### A.3.13 RD\_READ\_GPR\_MULT(71)

This request returns the contents of general-purpose registers 0 to 18, inclusive, of the debugged process.

#### A.3.13.1 Request data

**Table A-25. RD\_READ\_GPR\_MULT Request Table**

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_READ_GPR_MULT  | Requested API function.                  |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.13.2 Response data

**Table A-26. RD\_READ\_GPR\_MULT Response Table**

| Parameters                     | Description                                                            |
|--------------------------------|------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                          |
| msg.retcode= RD_OK (0)         | Successful completion.                                                 |
| msg.retcode= RD_NOTSUPP (1003) | Call not supported by this interface.                                  |
| msg.retcode= RD_REG_ERR (1004) | Unable to access given register.                                       |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist. |
| msg.data_len= 76 (0x4C)        | Length of additional data being sent.                                  |
| msg.buffer[0-18]               | Values read from GPR0 to GPR18. Undefined if error.                    |

### A.3.14 RD\_READ\_I (1)

This request returns the integer in the debugged process address space at the location pointed to by the *address* parameter. If the value of *address* is not in a valid address space, unpredictable results will occur.

#### A.3.14.1 Request data

Table A-27. RD\_READ\_I Request Table

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_READ_I         | Requested API function.                  |
| msg.address= address           | Address of memory to read data from.     |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.14.2 Response data

Table A-28. RD\_READ\_I Response Table

| Parameters                     | Description                                                                                   |
|--------------------------------|-----------------------------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                                                 |
| msg.retcode= RD_OK (0)         | Successful completion.                                                                        |
| msg.retcode= EIO (5)           | Debugged process can not access given address.                                                |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist.                        |
| msg.data= data                 | Data read at location pointed to by address. -1 if error (retcode should also be set to EIO). |
| msg.data_len= 0                | Length of additional data being sent.                                                         |

### A.3.15 RD\_READ\_I\_MULT (71)

This request returns the 32 integers in the debugged process address space at the location pointed to by the *address* parameter. If the value of *address* is not in a valid address space, unpredictable results will occur.

#### A.3.15.1 Request data

**Table A-29. RD\_READ\_I\_MULT Request Table**

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_READ_I_MULT    | Requested API function.                  |
| msg.address= address           | Address of memory to read data from.     |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.15.2 Response data

**Table A-30. RD\_READ\_I\_MULT Response Table**

| Parameters                     | Description                                                                  |
|--------------------------------|------------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                                |
| msg.retcode= RD_OK (0)         | Successful completion.                                                       |
| msg.retcode= EIO (5)           | Debugged process can not access given address.                               |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist.       |
| msg.retcode= RD_NOTSUPP (1003) | Call not supported by this interface.                                        |
| msg.buffer[0-0x1F]             | Contents of addresses from location pointed to by address to address + 0x1F. |
| msg.data_len= 128 (0x80)       | Length of additional data being sent.                                        |

### A.3.16 RD\_READ\_SPR (115)

This request reads data directly from one of the SPRs (not the process's copy). All SPR registers are accessible through this message request. The sender is responsible for supplying valid SPR values, no error checking is performed on this field.

#### A.3.16.1 Request data

**Table A-31. RD\_READ\_SPR Request Table**

| Parameters               | Description                           |
|--------------------------|---------------------------------------|
| msg.request= RD_READ_SPR | Requested API function.               |
| msg.address= SPR number  | SPR number to read.                   |
| msg.data_len= 0          | Length of additional data being sent. |

#### A.3.16.2 Response data

**Table A-32. RD\_READ\_SPR Response Table**

| Parameters                     | Description                           |
|--------------------------------|---------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.         |
| msg.retcode= RD_OK (0)         | Successful completion.                |
| msg.data= value                | Value read from register.             |
| msg.data_len= 0                | Length of additional data being sent. |

### A.3.17 RD\_READ\_SR (118)

This request returns the content of one of the segment registers.

#### A.3.17.1 Request data

**Table A-33. RD\_READ\_SR Request Table**

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_READ_SR        | Requested API function.                  |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.address= register          | Name of the register to be read.         |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.17.2 Response data

**Table A-34. RD\_READ\_SR Response Table**

| Parameters                     | Description                                                            |
|--------------------------------|------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                          |
| msg.retcode= RD_OK (0)         | Successful completion.                                                 |
| msg.retcode= EIO (5)           | Register is not defined.                                               |
| msg.retcode= RD_REG_ERR (1004) | Unable to access given register.                                       |
| msg.data= value                | Value read from register. 0xFFFFFFFF if error occurred.                |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist. |
| msg.data_len= 0                | Length of additional data being sent.                                  |

### A.3.18 RD\_STATUS (114)

This request is used to get program execution status and to determine if a previous **RD\_CONTINUE** request was received.

#### A.3.18.1 Request data

**Table A-35. RD\_STATUS Request Table**

| Parameters                     | Description                             |
|--------------------------------|-----------------------------------------|
| msg.request= RD_STATUS         | Requested API function.                 |
| msg.rpid= process_id           | Numeric process ID on the target system |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.   |

#### A.3.18.2 Response data

**Table A-36. RD\_STATUS Response Table**

| Parameters                     | Description                                                                                                      |
|--------------------------------|------------------------------------------------------------------------------------------------------------------|
| msg.address= execution status  | Status is 1 if program is running and 0 if stopped. In the case of an error, this field will be -1 (0xFFFFFFFF). |
| msg.data= sequence number      | Sequence number of the last RD_CONTINUE request that was received.                                               |
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                                                                    |
| msg.retcode= RD_OK (0)         | Successful completion.                                                                                           |
| msg.retcode= RD_ESRCH (3)      | The msg.pid field identifies a process that does not exist.                                                      |

### A.3.19 RD\_STOP\_APPL (113)

This request is used to interrupt program execution.

#### A.3.19.1 Request data

**Table A-37. RD\_STOP\_APPL Request Table**

| Parameters                     | Description                             |
|--------------------------------|-----------------------------------------|
| msg.request= RD_STOP_APPL      | Requested API function.                 |
| msg.rpid= process_id           | Numeric process ID on the target system |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.   |

#### A.3.19.2 Response data

**Table A-38. RD\_STOP\_APPL Response Table**

| Parameters                     | Description                                                 |
|--------------------------------|-------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                               |
| msg.retcode= RD_OK (0)         | Successful completion.                                      |
| msg.retcode= RD_ESRCH (3)      | The msg.pid field identifies a process that does not exist. |



### A.3.20 RD\_WAIT (108)

This call allows the debugger to determine the current status of the debugged process after it is stopped. The first (least significant) byte of the process status indicates the reason for stoppage: this is always 0x7f. The second byte contains the signal number that caused the stop. Valid signals are:

- AIX\_SIGILL (4) - illegal instruction
- AIX\_SIGTRAP (5) - hit a trap instruction (breakpoint)
- AIX\_SIGFPE (8) - floating point error
- AIX\_SIGSEGV (11) - storage violation

For example after hitting a breakpoint, the status of 0x57f is returned to the debugger. After the program terminates, the first byte contains 0x00 and the rest of the status holds the program exit code. After RD\_KILL call wait status of 0x57f should be returned.

#### A.3.20.1 Request data

Table A-39. RD\_WAIT Request Table

| Parameters           | Description                   |
|----------------------|-------------------------------|
| msg.request= RD_WAIT | Requested API function.       |
| msg.data_len= 0      | Length of data in msg.buffer. |

#### A.3.20.2 Response data

Table A-40. RD\_WAIT Response Table

| Parameters                           | Description                                      |
|--------------------------------------|--------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001)       | Communication error occurred.                    |
| msg.retcode= RD_OK (0)               | Successful completion.                           |
| msg.data= status                     | Process status.                                  |
| msg.address= pid                     | Process id.                                      |
| msg.data_len= strlen(message_string) | The ROM monitor always returns 0 in this field.  |
| msg.buffer= message_string           | Formatted message string text (NULL terminated). |

### A.3.21 RD\_WRITE\_BLOCK (19)

This request writes a block of data into the address space of the debugged process at the address pointed to by the *msg.address* field. The number of bytes to write is contained in the *msg.data* field and the data is in the *msg.buffer* field. Unpredictable results occur if the *msg.address* parameter points to a location that can not be accessed by the debugged process.

#### A.3.21.1 Request data

Table A-41. RD\_WRITE\_BLOCK Request Table

| Parameters                  | Description                                  |
|-----------------------------|----------------------------------------------|
| msg.request= RD_WRITE_BLOCK | Requested API function.                      |
| msg.address= address        | Address of memory to write data to.          |
| msg.data= count             | Number of bytes of buffer area to be written |
| msg.buffer                  | Data to be written.                          |
| msg.data_len= count         | Length of additional data being sent.        |

#### A.3.21.2 Response data

Table A-42. RD\_WRITE\_BLOCK Response Table

| Parameters                     | Description                                    |
|--------------------------------|------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                  |
| msg.retcode= RD_OK (0)         | Successful completion.                         |
| msg.retcode= EIO (5)           | Debugged process can not access given address. |
| msg.data_len= 0                | Length of additional data being sent.          |

### A.3.22 RD\_WRITE\_D (5)

This request writes the value of the *msg.data* parameter into the address space of the debugged process at the address pointed to by the *msg.address* parameter. Unpredictable results occur if the *msg.address* parameter points to a location that can not be accessed by the debugged process.

#### A.3.22.1 Request data

Table A-43. RD\_WRITE\_D Request Table

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_WRITE_D        | Requested API function.                  |
| msg.address= address           | Address of memory to write data to.      |
| msg.data= data                 | Data to write to memory.                 |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.22.2 Response data

Table A-44. RD\_WRITE\_D Response Table

| Parameters                     | Description                                                                                               |
|--------------------------------|-----------------------------------------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                                                             |
| msg.retcode= RD_OK (0)         | Successful completion.                                                                                    |
| msg.retcode= EIO (5)           | Debugged process can not access given address.                                                            |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist.                                    |
| msg.data= data                 | Data written at location pointed to by address. -1 if error (retcode should also be set to EIO or ESRCH). |
| msg.data_len= 0                | Length of additional data being sent.                                                                     |

### A.3.23 RD\_WRITE\_FPR (15)

This request writes data to one of the floating-point registers.

#### A.3.23.1 Request data

**Table A-45. RD\_WRITE\_FPR Request Table**

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_WRITE_FPR      | Requested API function.                  |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.address= register          | Name of the register to be written.      |
| msg.data= value                | Value to be written to the register.     |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.23.2 Response data

**Table A-46. RD\_WRITE\_FPR Response Table**

| Parameters                     | Description                                                            |
|--------------------------------|------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                          |
| msg.retcode= RD_OK (0)         | Successful completion.                                                 |
| msg.retcode= EIO (5)           | Register is not defined.                                               |
| msg.retcode= RD_REG_ERR (1004) | Unable to access given register.                                       |
| msg.data= value                | Value written to register. 0xFFFFFFFF if error occurred.               |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist. |
| msg.data_len= 0                | Length of additional data being sent.                                  |

### A.3.24 RD\_WRITE\_GPR (14)

This request writes data to one of the general-purpose or special-purpose registers of the debugged process. Valid registers are defined in "dbg.h" and "sys/reg.h". Not all defined registers are supported for all environments.

#### A.3.24.1 Request data

Table A-47. RD\_WRITE\_GPR Request Table

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_WRITE_GPR      | Requested API function.                  |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.address= register          | Name of the register to be written.      |
| msg.data= value                | Value to be written to the register.     |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.24.2 Response data

Table A-48. RD\_WRITE\_GPR Response Table

| Parameters                     | Description                                                             |
|--------------------------------|-------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                           |
| msg.retcode= RD_OK (0)         | Successful completion.                                                  |
| msg.retcode= EIO (5)           | Register is not defined.                                                |
| msg.retcode= RD_REG_ERR (1004) | Unable to access given register.                                        |
| msg.data= value                | Value written to register. 0xFFFFFFFF if error occurred.                |
| msg.retcode= ESRCH (3)         | The <i>msg.rpid</i> parameter identifies a process that does not exist. |
| msg.data_len= 0                | Length of additional data being sent.                                   |

### A.3.25 RD\_WRITE\_I (4)

This request writes the value of the *msg.data* parameter into the address space of the debugged process at the address pointed to by the *msg.address* parameter. This request fails if the *msg.address* parameter points to a location that can not be accessed by debugged process. This call sets break points in the debugged process by writing TRAP (0x7D821008) instructions.

#### A.3.25.1 Request data

Table A-49. RD\_WRITE\_I Request Table

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_WRITE_I        | Requested API function.                  |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.address= address           | Address of memory to write data to.      |
| msg.data= data                 | Data to write to memory.                 |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.25.2 Response data

Table A-50. RD\_WRITE\_I Response Table

| Parameters                     | Description                                                                                               |
|--------------------------------|-----------------------------------------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                                                             |
| msg.retcode= RD_OK (0)         | Successful completion.                                                                                    |
| msg.retcode= EIO (5)           | Debugged process can not access given address.                                                            |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist.                                    |
| msg.data= data                 | Data written at location pointed to by address. -1 if error (retcode should also be set to EIO or ESRCH). |
| msg.data_len= 0                | Length of additional data being sent.                                                                     |

### A.3.26 RD\_WRITE\_SPR (112)

This request writes data directly to one of the SPRs (not the process's copy). All SPR registers are accessible through this request. The requester is responsible for supplying valid SPR values. No error checking is performed on this field.

#### A.3.26.1 Request data

**Table A-51. RD\_WRITE\_SPR Request Table**

| Parameters                | Description                           |
|---------------------------|---------------------------------------|
| msg.request= RD_WRITE_SPR | Requested API function.               |
| msg.address= SPR number   | SPR number to be written              |
| msg.data= value           | Data to write to register.            |
| msg.data_len= 0           | Length of additional data being sent. |

#### A.3.26.2 Response data

**Table A-52. RD\_WRITE\_SPR Response Table**

| Parameters                     | Description                           |
|--------------------------------|---------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.         |
| msg.retcode= RD_OK (0)         | Successful completion.                |
| msg.data_len= 0                | Length of additional data being sent. |

### A.3.27 RD\_WRITE\_SR (119)

This request writes data to one of the segment registers.

#### A.3.27.1 Request data

**Table A-53. RD\_WRITE\_SR Request Table**

| Parameters                     | Description                              |
|--------------------------------|------------------------------------------|
| msg.request= RD_WRITE_SR       | Requested API function.                  |
| msg.rpid= process_id           | Numeric process ID on the target system. |
| msg.address= register          | Name of the register to be written.      |
| msg.data= value                | Value to be written to the register.     |
| msg.data_len= sizeof(msg.rpid) | Length of additional data being sent.    |

#### A.3.27.2 Response data

**Table A-54. RD\_WRITE\_SR Response Table**

| Parameters                     | Description                                                            |
|--------------------------------|------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.                                          |
| msg.retcode= RD_OK (0)         | Successful completion.                                                 |
| msg.retcode= EIO (5)           | Register is not defined.                                               |
| msg.retcode= RD_REG_ERR (1004) | Unable to access given register.                                       |
| msg.data= value                | Value written to register. 0xFFFFFFFF if error occurred.               |
| msg.retcode= ESRCH (3)         | The <i>msg.pid</i> parameter identifies a process that does not exist. |
| msg.data_len= 0                | Length of additional data being sent.                                  |



### A.3.28 RL\_LDINFO (181)

This request provides load information from the host to the ROM monitor. This request is used when the target is loaded by a process other than the debugger. The information specified on the this request will be returned on subsequent **RD\_LDINFO** requests.

#### A.3.28.1 Request data

**Table A-55. RL\_LDINFO Request Table**

| Parameters                                             | Description                           |
|--------------------------------------------------------|---------------------------------------|
| msg.request= RL_LDINFO                                 | Requested API function.               |
| msg.data_len= sizeof(struct ldinfo) + strlen(pathname) | Length of additional data being sent. |
| msg.buffer= load information                           | See description of RD_LDINFO request. |

#### A.3.28.2 Response data

**Table A-56. RL\_LDINFO Response Table**

| Parameters                     | Description                           |
|--------------------------------|---------------------------------------|
| msg.retcode= RD_COM_ERR (1001) | Communication error occurred.         |
| msg.retcode= RD_OK (0)         | Successful completion.                |
| msg.data_len= 0                | Length of additional data being sent. |

### A.3.29 RL\_LOAD\_REQ(180)

This request flows from the ROM monitor to the host when a RD\_LOAD request is received. The port of the request is for the remote loader daemon (20050) to accommodate loading by a process independent from the debugger.

#### A.3.29.1 Request data

**Table A-57. RL\_LOAD\_REQ Request Table**

| Parameters                     | Description                                                                  |
|--------------------------------|------------------------------------------------------------------------------|
| msg.request= RL_LOAD_REQ       | Requested API function.                                                      |
| msg.buffer= filename           | NULL terminated string containing fully qualified name of file to be loaded. |
| msg.data_len= strlen(filename) | Length of additional data being sent.                                        |

#### A.3.29.2 Response data

**Table A-58. RL\_LOAD\_REQ Response Table**

| Parameters                        | Description                                                                                       |
|-----------------------------------|---------------------------------------------------------------------------------------------------|
| msg.retcode= RD_COM_ERR (1001)    | Communication error occurred.                                                                     |
| msg.retcode= RD_OK (0)            | Successful completion.                                                                            |
| msg.retcode= RD_NOFILE_ERR (1006) | Can't open file or file is incorrect format.                                                      |
| msg.retcode= RD_PTRACE_ERR (1014) | Error reading file.                                                                               |
| msg.rpid= process_id              | Process ID of newly loaded file. This number (integer) can not be equal to -1 (0xFFFF FFFF) or 0. |
| msg.data_len= sizeof(msg.rpid)    | Length of additional data being sent.                                                             |



# ROM Monitor Load Format

---

This appendix presents the ROM Monitor load format requirements.

## B.1 Overview

The ROM Monitor load format is designed to permit the specification of multiple text and data sections. The format consists of a linked list of sections of specified types prefixed by a small boot header, *boot\_block*, that specifies the initial target of the image and the entry point. The *boot\_block* header is placed at the front of the image by **eimgbld** or **nimgbld**. The ROM Monitor does no relocation. It is assumed that the destination addresses for the individual sections are the same ones specified during the application's linkage. The *info\_block* structure is reserved in the bootstrap program, bootLib.s. **nimgbld** or **eimgbld** patch in the values within the *info\_block* structure for bootLib to use at run time. The bootstrap program processes the sections back to front, that is, from the end of the image to the beginning. This is to avoid destructive overlap during the processing of typical images.

The sections are preceded by header blocks which identify the section types. The headers are linked together in a doubly linked list.

## B.2 Section Types

There are three basic section types. Generally, they can occur in the image in any order, but are usually arranged in ascending address order. The section header block has the following format:

```
/*-----+
| Relocation block structure.
+-----*/
typedef struct rel_block {
 unsigned long type;
 unsigned long dest_addr;
 unsigned long size;
 union {
 struct data_info {
 unsigned long size_to_fill;
 unsigned long char_to_fill;
 };
 };
};
```

```

 } data_info_str;
 struct text_info {
 unsigned long toc_pointer; /* used for XCOFF; not used for ELF */
 unsigned long entry_pt;
 } text_info_str;
 unsigned long number_symbols;
} section_info;
struct rel_block *next;
struct rel_block *bptr;
} rel_block_t;

```

The **type** field is one of the following manifest constants:

```

#define TEXT_SECT 0x00000001
#define DATA_SECT 0x00000002
#define SYMB_SECT 0x00000004

```

The **dest\_addr** specifies the target for the block, while **size** is the extent of the block, not counting the header. The bootstrap program uses this information to move the block to the destination specified at link time. **next** and **bptr** are the section header forward and backward pointers, respectively.

## B.2.1 First Section

The first section is a text section. The ROM loader places the entire image at the address specified in the *boot\_block* header. The entry point specified in the *boot\_block* header is assumed to be a branch, followed by the first section header, *info\_block*. This is to allow the bootstrap to easily gain immediate addressability to the first section block.

The format of the first section block is shown below:

```

/*-----+
| First section header
+-----*/
struct info_block {
 long magic_num; /* magic number */
 long text_start; /* addr of text section from section header */
 long text_size; /* size of text section from section header */
 long data_start; /* addr of data section from section header */
 long data_size; /* size of data section from section header */
 long elf_hdr_size; /* size of ELF header */
 long sym_start; /* addr of symbol table */
 long num_syms; /* number of symbols */
 long toc_ptr; /* used for XCOFF; not used for ELF */
 struct rel_block * next; /* pointer to next boot section header */
};

```

- **magic\_num** is used for verification purposes and must be X'004D 5054'.
- **text\_start** is the physical address value from the object text header.
- **text\_size** is the size in bytes from the object text header.
- **data\_start** is the physical address from the object data header.

- **data\_size** is the size in bytes from the object data header.
- **elf\_hdr\_size** is the size of the object header. The debugger requires this information.
- **sym\_start** is the address of the symbol table in storage.
- **num\_syms** is the number of symbol entries.
- **next** points to the next section header.

### B.2.2 Text Section

For a text section, the union **section\_info** contains the structure **text\_info**, specifying the entry point of the text section.

### B.2.3 Data Section

For a data section, the union **section\_info** contain the structure **data\_info**, specifying **size\_to\_fill** and **char\_to\_fill**. These parameters are used to optionally fill a region past the size extent specified in the base rel\_block with a character. It is most often used to zero bss by specifying the size of the bss in **size\_to\_fill** and 0x0 for **char\_to\_fill**.

### B.2.4 Symbol Section

For symbols, the union **section\_info** contains the number of symbols in the section. The data in this section consists of the symbol table from the original object file.

## B.3 Boot Header

The entire image is preceded by the boot header that was added by **nimgbld** or **eimgbld**. The ROM loader uses this information to verify that it is a ROM Monitor load image, determine where to place the image, and whether to invoke the ROM Monitor debugger before transferring control to the entry point. The boot header is stripped off by the ROM Monitor loader and does not appear at the load address.

The boot header has the following format:

```
/*-----+
| Boot header.
+-----*/
typedef struct boot_block {
 unsigned long magic;
 unsigned long dest;
 unsigned long num_512blocks;
 unsigned long debug_flag;
 unsigned long entry_point;
 unsigned long reserved[3];
} boot_block_t;
```

- **magic** identifies this image as a legitimate ROM Monitor image and must have the value X'0052 504F'.
- **dest** is the target address for the image (after the boot header is stripped off).
- **num\_512blocks** - Boot images are padded to a multiple of 512 byte blocks. This field specifies the number of blocks.
- **debug\_flag** controls whether the ROM Monitor debugger gets control before the loaded image starts. If the value is 0x0, the image runs immediately. If 0x01, the debugger gains control as soon as the load is complete.
- **entry\_point** specifies the address where the image will receive control.

# Index

---

## A

- Alignment Exception Support Library 9-1
- ANSI C I/O Library 9-1
- ANSI C Library 9-1
- ANSI C Math Library 9-1
- async safe 10-1
- async\_init() function 10-10
- asyncLib.a library 9-4

## B

- biosenet\_attach() function 10-11
- BLKMEDIA\_CHANGE command 9-14
- Block Buffer Library 9-1
- book
  - conventions used xxi
  - highlighting xxi
  - numeric xxi
  - syntax diagrams xxii
  - how organized xx
  - who should use this book xx
- Boot Library 9-1, 9-4
- booting the processor 5-7

## C

- C++ runtime support library 9-2
- cancel safe 10-2
- Clock Support Library 9-2
- clock\_set() function 10-13
- clockchip\_set() function 10-14
- clockLib.a library 9-5
- connecting the EVB hardware 5-1
- conventions used xxi
  - highlighting xxi
  - numeric xxi
  - syntax diagrams xxii
- Creating a loadable diskette 9-30

## D

- dbg\_ioLib\_init() function 10-16
- dcache\_flush() function 10-17
- dcache\_invalidate() function 10-18
- Debug Support Library 9-2
- Device and File Support Library 9-2

## device drivers

- asynchronous 9-7
- diskette, device installation 9-12
- diskette, driver installation 9-12
- diskette, general information 9-12
- diskette, I/O control 9-13
- diskette, opening files 9-13
- diskette, reading 9-13
- diskette, writing 9-13
- Ethernet 9-16
- Diskette Block Device Driver Support Library 9-5
- dma\_disable() function 10-19
- dma\_setup() function 10-20
- dma\_status() function 10-21
- DOS File System Support Library 9-2
- driver\_install
  - async\_init 9-7
  - dskt\_init 9-12
  - enet\_init 9-16
- dskt\_init function 10-22
- dsktLib.a library 9-5
- Dynamic Loader Library 9-2

## E

- enet\_init function 10-23
- Ethernet 9-16
- Ethernet Device Driver Installation 9-16
- ext\_int\_disable() function 10-24
- ext\_int\_enable() function 10-25
- ext\_int\_install() function 10-26
- ext\_int\_query() function 10-27

## F

- Federal Communications Commission (FCC) Statement xxiii
- File Transfer Protocol Support Library 9-2
- Flash update utility 7-31
- Floating Point Emulation Library 9-2
- functions
  - async\_init() 10-10
  - biosenet\_attach() 10-11
  - clock\_set() 10-13
  - clockchip\_set() 10-14
  - cpu\_bus\_speed() 10-15
  - dbg\_ioLib\_init() 10-16

|                               |              |                               |        |
|-------------------------------|--------------|-------------------------------|--------|
| dcache_flush()                | 10-17        | ppcMfsda()                    | 10-64  |
| dcache_invalidate()           | 10-18        | ppcMfsdr1()                   | 10-65  |
| dma_disable()                 | 10-19        | ppcMfsia()                    | 10-66  |
| dma_setup()                   | 10-20        | ppcMfsprg1() • ppcMfsprg3()   | 10-67  |
| dma_status()                  | 10-21        | ppcMfsprg1() • ppcMtsprg3()   | 10-91  |
| dskt_init                     | 10-22        | ppcMfsr()                     | 10-68  |
| enet_init                     | 10-23        | ppcMfsrr0()                   | 10-69  |
| ext_int_disable()             | 10-24        | ppcMfsrr1()                   | 10-70  |
| ext_int_enable()              | 10-25        | ppcMftb()                     | 10-71  |
| ext_int_install()             | 10-26        | ppcMtdabr()                   | 10-72  |
| ext_int_query()               | 10-27        | ppcMtdar()                    | 10-73  |
| int_install()                 | 10-28        | ppcMtdbat0l() - ppcMtdbat3l() | 10-74  |
| int_query()                   | 10-29        | ppcMtdbat0u() - ppcMtdbat3u() | 10-75  |
| ioLib_init()                  | 10-30        | ppcMtdec()                    | 10-76  |
| ppcAbend()                    | 10-31        | ppcMtdsisr()                  | 10-77  |
| ppcAndMsr()                   | 10-32        | ppcMtear()                    | 10-78  |
| ppcCntlzw()                   | 10-33        | ppcMthod0()                   | 10-79  |
| ppcDcbf()                     | 10-34        | ppcMtiabr()                   | 10-80  |
| ppcDcbi()                     | 10-35        | ppcMtibat0l() - ppcMtibat3l() | 10-81  |
| ppcDcbst()                    | 10-36        | ppcMtibat0u() - ppcMtibat3u() | 10-82  |
| ppcDcbz()                     | 10-37        | ppcMtmucr0()                  | 10-83  |
| ppcEieio()                    | 10-39        | ppcMtmsr()                    | 10-84  |
| ppcHalt()                     | 10-40        | ppcMtpir()                    | 10-85  |
| ppclcbi()                     | 10-41        | ppcMtpmc1()                   | 10-86  |
| ppclsync()                    | 10-42        | ppcMtpmc2()                   | 10-87  |
| ppcMfdabr()                   | 10-43        | ppcMtsda()                    | 10-88  |
| ppcMfdar()                    | 10-44        | ppcMtsdr1()                   | 10-89  |
| ppcMfdbat0l() - ppcMfdbat3l() | 10-45        | ppcMtsia()                    | 10-90  |
| ppcMfdbat0u() - ppcMfdbat3u() | 10-46        | ppcMtsr()                     | 10-92  |
| ppcMfdec()                    | 10-47        | ppcMtsrr1()                   | 10-94  |
| ppcMfdsisr()                  | 10-48        | ppcMttb()                     | 10-95  |
| ppcMfear()                    | 10-49        | ppcOrMsr()                    | 10-96  |
| ppcMfesasrr()                 | 10-50        | ppcSync()                     | 10-97  |
| ppcMfgpr1()                   | 10-51        | s1dbprintf()                  | 10-100 |
| ppcMfgpr2()                   | 10-52        | s1dbprintfapp()               | 10-101 |
| ppcMfhod0()                   | 10-53, 10-54 | s2dbprintf()                  | 10-103 |
| ppcMfiabr()                   | 10-55        | s2dbprintfapp()               | 10-104 |
| ppcMfibat0l() - ppcMfibat3l() | 10-56        | setup_bat()                   | 10-98  |
| ppcMfibat0u() - ppcMfibat3u() | 10-57        | timertick_install()           | 10-106 |
| ppcMfmmucr0()                 | 10-58        | timertick_remove()            | 10-107 |
| ppcMfmsr()                    | 10-59        | vs1dbprintf()                 | 10-108 |
| ppcMfpir()                    | 10-60        | zapatos_enet_setup()          | 10-109 |
| ppcMfpmc1()                   | 10-61        | zapatos_enet_setup() function | 10-109 |
| ppcMfpmc2()                   | 10-62        |                               |        |
| ppcMfpvr()                    | 10-63        |                               |        |



## H

- hardware components 1-1
  - cables and power supply 1-1
  - evaluation board 1-1
- host system requirements
  - PC 2-2
  - RS/6000 2-1
  - Sun 2-3

## I

- I/O control 9-10
- IBM Embedded Systems Solution Center xxiii
- initialization 9-22
  - board bootstrap 9-22
- Input/output Support Library 9-2
- installing
- int\_install() function 10-28
- ioLib.a library 9-4
- ioLib\_init() function 10-30

## K

- Kernel Abstract Data Types Library 9-2

## L

- library description
  - asyncLib.a 9-4
  - clockLib.a 9-5
  - dsktLib.a 9-5
  - ioLib.a 9-4
  - ppcLib.a 9-5
  - rtx.o 9-3
  - rtxLib.a 9-3
  - tickLib.a 9-6

## N

- Network Support Library 9-2
- NFS Support Library 9-2

## O

- Opening and Closing Ethernet Files 9-18
- opening asynchronous communication ports 9-9
- OpenShell 9-2
- OS Open kernel extensions 9-3
- OS Open minimal kernel 9-3

## P

- PC host configuration 4-7
  - ethernet setup 4-14
  - ethernet setup for Windows 3.1 4-10
  - ethernet setup for Windows 95 4-12
  - ethernet setup for Windows NT 3.51 4-13
  - serial port setup 4-8
- PC installation 3-4
  - RISCWatch debugger 3-7
  - software support package 3-4
- PCMCIA ATA/IDE 9-2
- PCMCIA card services/enabler 9-2
- PCMCIA socket services 9-3
- polled asynchronous I/O 9-12
- PowerPC Low Level Access Support Library 9-2
- PowerPC Low-Level Processor Access Support Library 9-5
  - ppcAbend() function 10-31
  - ppcAndMsr() function 10-32
  - ppcCntlzw() function 10-33
  - ppcDcbf() function 10-34
  - ppcDcbi() function 10-35
  - ppcDcbst() function 10-36
  - ppcDcbz() function 10-37
  - ppcEieio() function 10-39
  - ppcHalt() function 10-40
  - ppcIcbi() function 10-41
  - ppcLib.a library 9-5
  - ppcIsync() function 10-42
  - ppcMfdabr() function 10-43
  - ppcMfdar() function 10-44
  - ppcMfdbat0l() - ppcMfdbat3l() function 10-45
  - ppcMfdbat0u() - ppcMfdbat3u() function 10-46
  - ppcMfdec() function 10-47
  - ppcMfdisr() function 10-48
  - ppcMfear() function 10-49, 10-50
  - ppcMfgpr1() function 10-51
  - ppcMfgpr2() function 10-52
  - ppcMfhid0() function 10-53
  - ppcMfhid1() function 10-54
  - ppcMfiabr() function 10-55
  - ppcMfibat0l() - ppcMfibat3l() function 10-56
  - ppcMfibat0u() - ppcMfibat3u() function 10-57
  - ppcMfmmcr0() function 10-58

- ppcMfmsr() function 10-59
- ppcMfpir() function 10-60
- ppcMfpmc1() function 10-61
- ppcMfpmc2() function 10-62
- ppcMfpvr() function 10-63
- ppcMfsda() function 10-64, 10-88
- ppcMfsdr1() function 10-65
- ppcMfsia() function 10-66
- ppcMfsprg0() • ppcMfsprg3() function 10-67
- ppcMfsr() function 10-68
- ppcMfsrr0() function 10-69
- ppcMfsrr1() function 10-70
- ppcMftb() function 10-71
- ppcMtdabr() function 10-72
- ppcMtdar() function 10-73
- ppcMtdbat0l() - ppcMtdbat3l() function 10-74
- ppcMtdbat0u() - ppcMtdbat3u() function 10-75
- ppcMtdec() function 10-76
- ppcMtdsisr() function 10-77
- ppcMtear() function 10-78
- ppcMthid0() function 10-79
- ppcMtiabr() function 10-80
- ppcMtibat0l() - ppcMtibat3l() function 10-81
- ppcMtibat0u() - ppcMtibat3u() function 10-82
- ppcMtmucr0() function 10-83
- ppcMtmsr() function 10-84
- ppcMtpir() function 10-85
- ppcMtpmc1() function 10-86
- ppcMtpmc2() function 10-87
- ppcMtsdr1() function 10-89
- ppcMtsia() function 10-90
- ppcMtsprg0() • ppcMtsprg3() function 10-91
- ppcMtsr() function 10-92
- ppcMtsrr1() function 10-94
- ppcMttb() function 10-95
- ppcOrMsr() function 10-96
- ppcSync() function 10-97
- ptrace
  - definitions A-4
  - RD\_ATTACH A-5
  - RD\_CONTINUE A-6
  - RD\_DETACH A-7
  - RD\_FILL A-8
  - RD\_KILL A-9
  - RD\_LDINFO A-10

- RD\_LOAD A-12
- RD\_LOGIN A-13
- RD\_LOGOFF A-14
- RD\_READ\_D A-15
- RD\_READ\_GPR A-16, A-17, A-22
- RD\_READ\_GPR\_MULT A-18
- RD\_READ\_I A-19
- RD\_READ\_I\_MULT A-20
- RD\_READ\_SPR A-21
- RD\_STATUS A-23
- RD\_STOP\_APPL A-24
- RD\_WAIT A-25
- RD\_WRITE\_BLOCK A-26
- RD\_WRITE\_D A-27
- RD\_WRITE\_GPR A-28, A-29, A-32
- RD\_WRITE\_I A-30
- RD\_WRITE\_SPR A-31
- RL\_LDINFO A-33
- RL\_LOAD\_REQ A-34
- MSGDATA structure A-1
- overview A-1

## Q

- QDEVATTR command 9-13, 9-14
- QDSKTATTR command 9-14
- Queue Library 9-2

## R

- RAM Disk Library 9-2
- Rate Monotonic Scheduling (RMS) Library 9-2
- Real\_time Executive 9-3
- Real-time Clock Interface Support Library 9-5
- related publications xxiii
- Remote Source Level Debug Library 9-2
- Ring Buffer Library 9-2
- ROM monitor
  - accessing 7-8
  - bootp and tftp configuration for loads 7-2
  - PC 7-4, 7-6
  - RS/6000 7-2
- communication features 7-2
- menus 7-9
  - changing IP addresses 7-15
  - disabling the automatic display 7-22
  - displaying the current configuration 7-23
  - entering the debugger 7-19

- exiting the main menu 7-29
  - initial ROM monitor menu 7-9
  - saving the current configuration 7-24
  - selecting boot devices 7-13
  - selecting power-on tests 7-11
  - using the ping test 7-17
- source code 7-1
- user functions 7-31
- ROM monitor load format
  - boot header B-3
  - overview B-1
  - section types B-1
    - data section B-3
    - first section B-2
    - symbol section B-3
  - sections types
    - text section B-3
- RPC Support Library 9-2
- RS/6000 host configuration 4-1
  - ethernet setup 4-5
  - serial port setup 4-1
- RS/6000 installation 3-1
  - RISCWatch debugger 3-4
  - software support package 3-1
- rtx.o library 9-3
- rtxLib.a library 9-3
- Runtime Library 9-2

## S

- s1dbprintf() function 10-100
- s1dbprintfapp() function 10-101
- s2dbprintf() function 10-103
- s2dbprintfapp() function 10-104
- sample applications
  - overview 8-1
  - resolving problems 8-9
    - bootp and tftp servers 8-10
    - using the ping test 8-9
  - ROM monitor flash image 8-2
  - using 8-4
    - Dhrystone benchmark 8-6
    - timesamp program 8-8
    - usr\_samp program 8-7
- SCSI Support Library 9-2
- Serial Port Support Library 9-4

- Serial Support Library 9-3
- setup\_bat() function 10-98
- software support package 1-1
  - application libraries and tools 1-3
  - Dhrystone benchmark 1-3
  - High C/C++ compiler 1-2
  - RISCWatch 400 debugger 1-2
  - ROM monitor 1-2
- Software Timer Tick Support Library 9-6
- Sun host configuration 4-13
- Sun installation
  - RISCWatch debugger 3-10
  - software support package 3-7
- Symbol Support Library 9-3

## T

- TCP/IP Protocol Support Library 9-3
- Telnet Client Support Library 9-3
- Telnet Daemon Support Library 9-3
- terminal emulator 5-4
  - PC terminal emulation 5-5
    - Windows 3.1 & NT 5-5
    - Windows 95 5-5
  - RS/6000 terminal emulation 5-4
  - Sun terminal emulation 5-6
- tickLib.a library 9-6
- Timer Tick Support 9-3
- timertick\_install() function 10-106
- timertick\_remove() function 10-107
- tools 9-24
  - dd(AIX) 9-30
  - dd(Sun OS) 9-30
  - dsktimg 9-31
  - eimgbld 9-28, 9-29
  - elf2rom 9-24
  - hbranch 9-26
- Trivial File Transfer Protocol Library 9-3
- TTY Support Library 9-3

## V

- vs1dbprintf() function 10-108

## W

- writing calls on asynchronous ports 9-9

**X**

XL C Compiler Support Library 9-3