IBM

PowerPC 403 Evaluation Board Kit User’s Manual

13H6987 000007

Seventh Edition (June 1997)

Address comments about this publication to:

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

email: ppc400pubs@vnet.ibm.com

Contents

Figures

Tables

About This Book

This book contains the information you need to install and use the IBM® PowerPC™ 403™ Evaluation Board (EVB), a hardware and software development tool for the PowerPC 403GA, 403GC, and 403GCX 32-bit RISC embedded controllers.

Connection of the 403 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 403 EVB hardware module comes with a 403GA, 403GC, or 403GCX controller, an Ethernet controller, 128KB flash memory, two serial ports, 4 MB of DRAM, and expansion and test interfaces. The reference design also includes technical specifications and schematics.

The 403 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 403 embedded controller and use the debugging features of the 403 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 403GA, 403GC, or 403GCX embedded controller
- C and assembler language programming

How This Book is Organized

This book contains the following chapters and appendixes:

- Chapter 1, “Overview of the 403 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 403 EVB.
- Chapter 5, “403 EVB Connectors ,” describes the EVB connectors and the procedures for connecting and configuring the 403 EVB hardware.
- Chapter 6, “403 EVB Hardware,” describes the hardware components and their functions in terms of the overall organization of the 403 EVB .
- Chapter 7, “403 EVB ROM Monitor,” describes the operations of the ROM monitor.
- Chapter 8, “403 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, “403 EVB Function Reference,” lists the OS Open functions for the 403 EVB platform. The function calls are arranged alphabetically by function name.
- Appendix A, “Programmable Logic Equations,” lists the programming for the expansion bus interface and for peripheral addressing.
- Appendix B, “Program Trace Calls,” describes the messages for interfacing a debugger on the host system to the ROM monitor on the 403 EVB.
- Appendix C, “ROM Monitor Load Format,” describes the load format requirements supported by the ROM monitor.
- Appendix D, “403 EVB Bill of Materials,” contains a list of parts used on the 403 EVB.

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

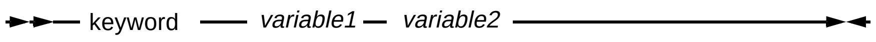
Names of objects known to the software must be entered exactly as shown.

- 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.

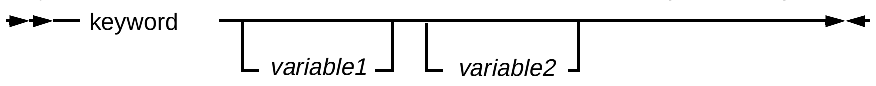
Syntax Diagram Conventions

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

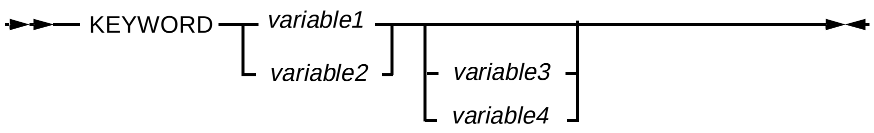
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.
- A ►►─── symbol begins a diagram.
- A ───► symbol indicates continuation of a diagram on the next line.
- A ►─── symbol indicates continuation of a diagram from the previous line.
- A ───►◄ symbol terminates 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.



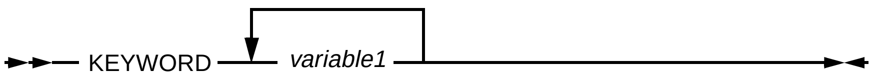
- Keywords or variables shown on branches below the main path are optional.



- 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.



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



Contacting the IBM Embedded Systems Solution Center

For information about the 403 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 400Series User's Manuals**

  PPC403GA Embedded Controller User's Manual, 13H6960

  PPC403GB Embedded Controller User's Manual, 13H6985

  PPC403GC Embedded Controller User's Manual, 13H6986

  PPC403GCX Embedded Controller User's Manual

  PPC401GF Embedded Controller User's Manual, 13H6948

1

Overview of the 403 EVB

This chapter introduces the hardware and software in the 403 EVB kit.

1.1 Introducing the 403 EVB Hardware Components

The 403 EVB kit contains the evaluation board with its power supply, line cord, and serial port cables.

1.1.1 403 Evaluation Board

The 403 EVB is a full featured prototyping board which comes with the PowerPC 403GA, 403GC, or 403GCX embedded controller, 128KB of flash memory (preprogrammed with the ROM Monitor), 4MB of DRAM, two serial ports, an Ethernet controller, and an expansion interface connector. Two DRAM slots are provided to support up to 128MB when both slots are populated with 64MB SIMMs.

Serial port 1 of the EVB attaches to the 403 controller, while serial port 2 connects to a National NS16550 serial communications controller. The Ethernet controller is a National DP83902.

Product documentation for devices other than the 403 can be obtained from the respective manufacturers. Configuration and addressing information for all these devices is included in the subsequent chapter on the 403 EVB hardware.

Header connectors are provided for optional test equipment such as the RISCWatch™ JTAG debugger. This tool allows non-intrusive hardware and software debug through the 403 EVB JTAG port. Connectors are provided for both the RISCWatch JTAG debugger and for its RISCTrace™ feature. For more information on the RISCWatch JTAG tool, call the IBM Embedded Systems Solution Center at (919) 254-1810.

1.1.2 Cables and Power Supply

The 403 EVB kit includes two serial port interface cables for connecting EVB serial ports 1 and 2 to a terminal (or terminal emulator) and to a host system, respectively.
Note: The Sun version of the EVB kit comes equipped with only one serial port cable since the EVB-to-host connection is made over Ethernet and not the second serial port on the EVB. The Sun version also contains a male-to-male adapter to support connectivity between serial port 1 on the EVB and a serial port on the host.

No Ethernet cable is provided in the kit, but a standard 10Base2 (thin coax) connector and an Ethernet controller are provided on the board to support direct Ethernet communication with the host system.

A power supply with line cord is also provided with the 403 EVB kit.

1.2 Introducing the 403 EVB Software Support Package

The 403 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 403 EVB is supplied in the 128KB flash memory module on the board. This code initializes the 403 processor 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 403 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 403 EVB. Appendix B describes the trace calls that support communication between the RISCWatch debugger on the host and the ROM Monitor running on the 403 EVB.

1.2.2 RISCWatch Debugger

The RISCWatch source level debugger provides a window-based debugging environment for application programs running on the 403 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 403 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 and 60x 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 403 EVB.

2

Host System Requirements

This chapter describes the hardware and software requirements of the host system to which the 403 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)

2.1 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 403 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 403 EVB:

- RISCWatch 3.4 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

2.2 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 403 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 10Base2, an Ethernet/IEEE 802.3 10Base2 network transceiver, two BNC "T" type connectors, two terminating resistors, and a thin coaxial cable. At a minimum, a 10Base2 point-to-point connection will require one thin coaxial cable, two BNC "T" connectors, and two BNC terminating resistors.

The following software must be installed on the host PC to run the debugger that communicates with the ROM Monitor on the 403 EVB:

- RISCWatch 3.4 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.

2.3 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 403 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 10Base2, an AUI (or thick Ethernet) adapter cable (or an AUI/Audio Adapter cable depending on your SPARCstation model and options - both are available from Sun), Ethernet/IEEE 802.3 10Base2 network transceiver, two BNC “T” type connectors, two terminating resistors, and a thin coaxial cable. At a minimum, a 10Base2 point-to-point connection will require one thin coaxial cable, two BNC “T” connectors, and two BNC terminating resistors
  - 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 3.4 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)

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.

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

3.1.1 EVB Software Support Package Installation - RS/6000

The software support package is installed from diskettes on an AIX host system using the **s**ystem **m**anagement **i**nterface **t**ool (**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 runnig 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 403GA evaluation board, this directory might be named m403_evb.

- 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.USR 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.

3.1.2 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.

3.2 PC Installation (ELF file format version only)

3.2.1 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
- hcppc.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 downlaods
- 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 403GA evaluation board, this directory might be named m403_evb.

- 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.

3.2.2 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.

3.3 Sun Installation(ELF file format version only)

3.3.1 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 downlaods

- **/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 403GA evaluation board, this directory might be named m403_evb.

- 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.

3.3.2 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.

4

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.

4.1 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.

4.1.1 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 **S**erial **L**ine **I**nternet **P**rotocol (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 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 Interfac**e
- 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 ADRRESS and DESTINATION Adress 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.

4.1.2 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 10Base2 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 403 EVB supports connection via Standard Ethernet, thin coax (10Base2).

A 10Base2 connection requires at least a thin coaxial cable and a BNC “T” connector when the EVB is added to an existing network. If the EVB is at one of the ends of the Ethernet network, a terminating resistor is also required. If the Ethernet network is exclusive between the host and the EVB, a thin coaxial cable, two BNC “T” connectors, and two BNC terminators are required.

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 10Base2 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 403 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.

4.1.3 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.

4.2 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.

4.2.1 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 **S**erial **L**ine **I**nternet **P**rotocol (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.

4.2.2 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 10Base2 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 403 EVB supports connection via Standard Ethernet, thin coax (10Base2). 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.

A 10Base2 connection requires at least a thin coaxial cable and a BNC “T” connector when the EVB is added to an existing network. If the EVB is at one of the ends of the Ethernet network, a terminating resistor is also required. If the Ethernet network is exclusive between the host and the EVB, a thin coaxial cable, two BNC “T” connectors, and two BNC terminators are required.

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.

4.2.2.1 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.

4.2.2.2 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.

4.2.2.3 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.

4.2.3 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.

4.3 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.

4.3.1 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.

4.3.2 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 10Base2 connector on the EVB.

An Ethernet connection requires additional hardware. The 403 EVB supports connection via Standard Ethernet, thin coax (10Base2).

A 10Base2 connection requires at least a thin coaxial cable and a BNC "T" connector when the EVB is added to an existing network. If the EVB is at one of the ends of the Ethernet network, a terminating resistor is also required. An exclusive Ethernet network between the host and the EVB, requires a thin coaxial cable, two BNC "T" connectors, and two BNC terminators. Depending on your SPARCstation model and options, an AUI (or thick ethernet) adapter cable or an AUI/Audio Adapter may also be necessary. Both of these cables are available from Sun. 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.

4.3.3 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
```

5 403 EVB Connectors

This chapter describes the 403 EVB connectors. The 403 EVB can be accessed through two serial ports, an Ethernet 10Base2 connector, and RISCWatch JTAG and RISCTrace connections. An expansion interface is provided for connection to an external customer-supplied prototyping area and a standard five-pin DIN connector is provided for power supply connection.

Positions of the connectors and jumpers on the EVB are indicated on Figure 5-1.

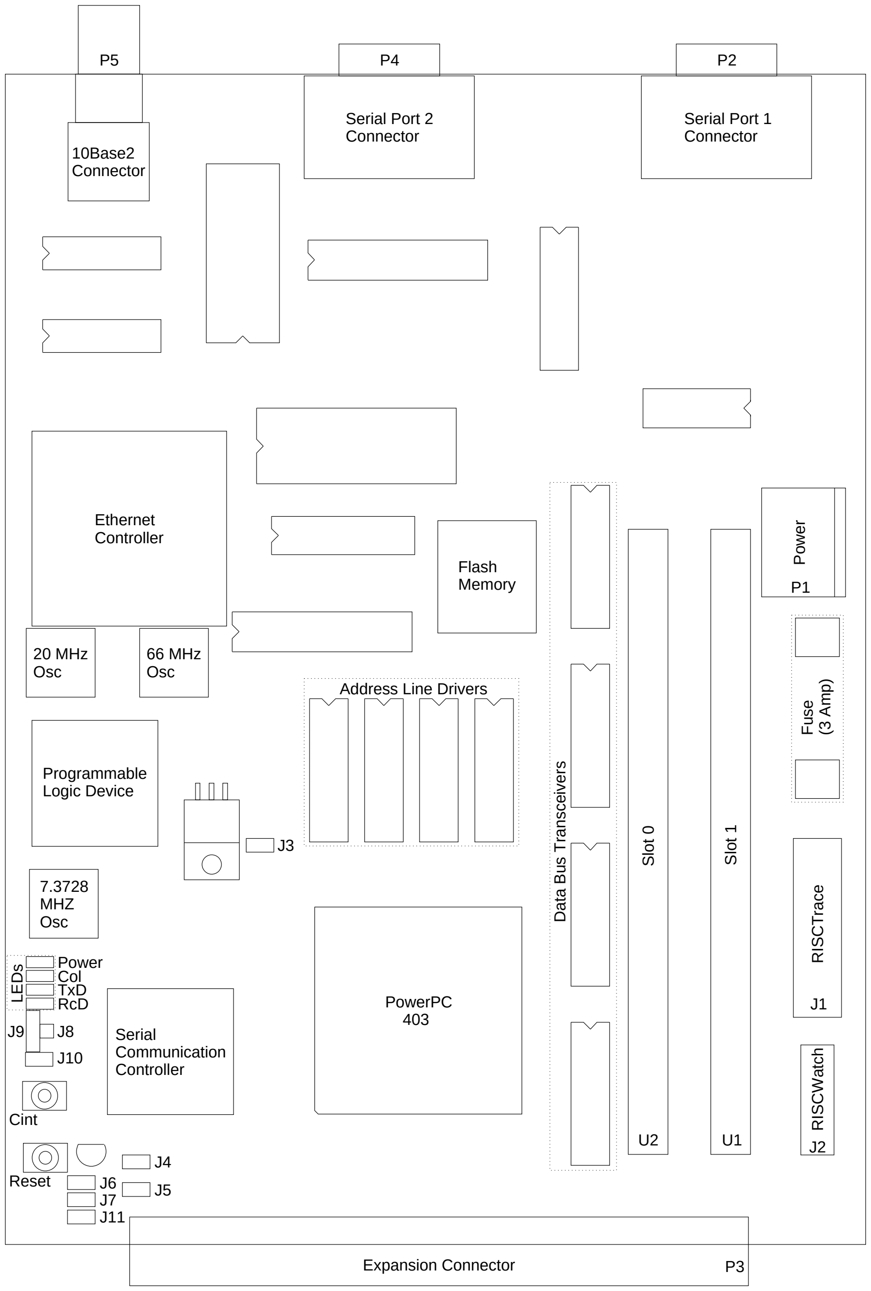


Figure 5-1. 403 EVB Connectors

5.1 Serial Port Connectors

Serial ports 1 and 2 are provided with standard nine-pin male right-angle connectors, as shown in Figure 5-2 below:

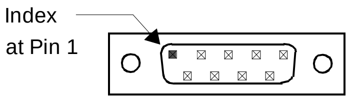


Figure 5-2. Nine-Pin Serial Port Connector

Table 5-1 describes the signal-to-pin assignments for serial ports 1 and 2:

Table 5-1. Serial Port Signal Assignments

Serial Port 1		Serial Port 2	
Pin Number	**Signal Name**	**Pin Number**	**Signal Name**
1	Not connected	1	Not connected
2	RxD	2	RxD
3	TxD	3	TxD
4	$\overline{\text{DTR}}$/$\overline{\text{RTS}}$	4	$\overline{\text{RTS}}$
5	GND	5	GND
6	$\overline{\text{DSR}}$/$\overline{\text{CTS}}$	6	$\overline{\text{CTS}}$
7	Not connected	7	Not connected
8	Not connected	8	Not connected
9	Not connected	9	Not connected

5.2 Ethernet Connector

The 403 EVB is provided with a standard connector for a 10Base2 thin coax Ethernet connector. Table 5-2 describes the connector on the EVB and the recommended mating connectors:

Table 5-2. Ethernet Connector Description

Receptacle Specifications	Mating Connector Specifications
Right-Angle BNC receptacle, AMP 227161-6 or Molex 73137-5005	AMP 221128-1, 227079-5, 2-329082-1 or Molex 73100-5001, 73103-5001, or 73106-5001

5.3 RISCWatch JTAG Debugger and RISCTrace Connectors

The RISCWatch JTAG debugger connects to the 403 EVB through a 2 × 8-pin header. The RISCTrace feature uses a 2 × 10 logic analyzer header. The headers are compatible with connectors for Hewlett-Packard and Tektronix logic analyzers. These headers are shown in Figure 5-3

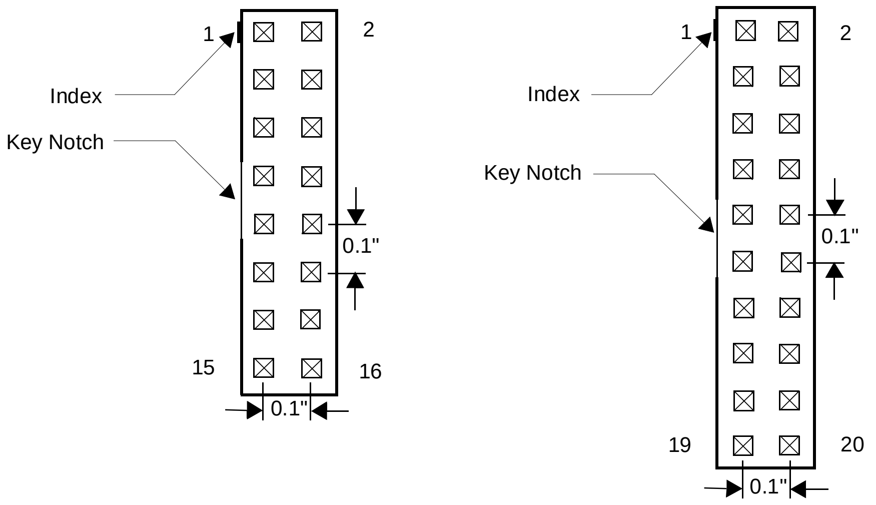


Figure 5-3. RISCWatch JTAG and RISCTrace Headers

Placements of the RISCWatch and logic analyzer headers on the EVB are indicated on the layout drawing in Figure 5-1 above. Signal names and positions on the headers are indicated in the following tables:

Table 5-3. RISCWatch JTAG Header Description

Pin No.	Signal Name	Description
1	TDO	JTAG test data out
2	NC	To be left unconnected
3	TDI	JTAG test data in
4	NC	To be left unconnected
5	NC	To be left unconnected
6	+Power	Power (status signal, not processor V_{DD})
7	TCK	JTAG test clock
8	NC	To be left unconnected
9	TMS	JTAG test mode select
10	NC	To be left unconnected
11	HALT	Processor halt
12	NC	To be left unconnected
13	NC	To be left unconnected
14	Key	Pin in this position should be removed.
15	NC	To be left unconnected
16	GND	Ground

The trace status header is used by the RISCTrace feature of the RISCWatch JTAG debugger:

Table 5-4. RISCTrace Header Description

Pin Number	Signal Name	Pin Number	Signal Name
1	NC	11	NC
2	NC	12	NC
3	SysClk3	13	TS0

Table 5-4. RISCTrace Header Description

Pin Number	Signal Name	Pin Number	Signal Name
4	NC	14	TS1
5	NC	15	TS2
6	NC	16	TS3
7	NC	17	TS4
8	NC	18	TS5
9	NC	19	TS6
10	NC	20	GND

5.4 Expansion Interface Connector

The EVB uses a 120-pin Eurocard type R right-angle receptacle (AMP 650874-4) as the card expansion connector, shown in Figure 5-4 below:

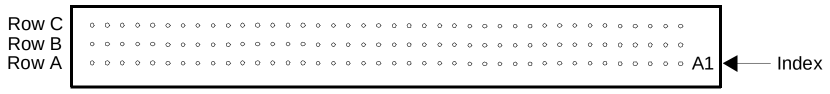


Figure 5-4. 120-Pin Eurocard Type R Header

One of the following 120-pin Eurocard type R right-angle pin assembly should be used as the mating connector: AMP 650949-5 or 650949-9.

A Eurocard type C connector can also be mated to the type R receptacle described above. Note that when connector types C and R are mixed, the connector circuit numbers do not match.

Table 5-5 describes the expansion interface connector on the 403 EVB.

Table 5-5. Expansion Interface Signal Assignments

Pin Number	Signal	Pin Number	Signal	Pin Number	Signal
A1	D31	B1	D30	C1	D29
A2	D28	B2	D27	C2	D26
A3	D25	B3	D24	C3	D23

Table 5-5. Expansion Interface Signal Assignments

Pin Number	Signal	Pin Number	Signal	Pin Number	Signal
A4	D22	B4	D21	C4	D20
A5	D19	B5	D18	C5	D17
A6	D16	B6	D15	C6	D14
A7	D13	B7	D12	C7	D11
A8	D10	B8	D9	C8	D8
A9	D7	B9	D6	C9	D5
A10	D4	B10	D3	C10	D2
A11	D1	B11	D0	C11	$\overline{\text{XCVR\_EN}}$
A12	+5V Power	B12	+5V Power	C12	GND
A13	$\overline{\text{OE}}$	B13	R/$\overline{\text{W}}$	C13	GND
A14	$\overline{\text{CS0}}$	B14	$\overline{\text{CS1}}$	C14	$\overline{\text{CS2}}$
A15	$\overline{\text{CS3}}$	B15	$\overline{\text{CS4}}$	C15	$\overline{\text{CS5}}$
A16	$\overline{\text{CS6}}$	B16	$\overline{\text{CS7}}$	C16	GND
A17	$\overline{\text{DMAR0}}$	B17	$\overline{\text{DMAR1}}$	C17	$\overline{\text{DMAR2}}$
A18	$\overline{\text{DMAR3}}$	B18	+5v Power	C18	GND
A19	$\overline{\text{DMAA0}}$	B19	$\overline{\text{DMAA1}}$	C19	$\overline{\text{DMAA2}}$
A20	$\overline{\text{DMAA3}}$	B20	+5V Power	C20	GND
A21	$\overline{\text{EOT0}}$	B21	$\overline{\text{EOT1}}$	C21	$\overline{\text{EOT2}}$
A22	$\overline{\text{EOT3}}$	B22	+5V Power	C22	GND
A23	INT0	B23	INT1	C23	INT2
A24	INT3	B24	INT4	C24	$\overline{\text{CINT\_EXT}}$
A25	SysClk2	B25	TimerClk	C25	SerClk
A26	$\overline{\text{Reset\_BUFF}}$	B26	Ext_Ready	C26	HoldReq
A27	HoldAck	B27	BusReq	C27	Error
A28	$\overline{\text{BusError}}$	B28	+5V Power	C28	GND

Table 5-5. Expansion Interface Signal Assignments

Pin Number	Signal	Pin Number	Signal	Pin Number	Signal
A29	+5V Power	B29	+5V Power	C29	GND
A30	+5V Power	B30	+5V Power	C30	GND
A31	A29	B31	A28	C31	A27
A32	A26	B32	A25	C32	A24
A33	A23	B33	A22	C33	A21
A34	A20	B34	A19	C34	A18
A35	A17	B35	A16	C35	A15
A36	A14	B36	A13	C36	A12
A37	A11	B37	A10	C37	A9
A38	A8	B38	A7	C38	A6
A39	$\overline{\text{WBE3}}$/A31	B39	$\overline{\text{WBE2}}$/A30	C39	$\overline{\text{WBE1}}$/A5
A40	$\overline{\text{WBE0}}$/A4	B40	+5V Power	C40	GND

5.5 Power Connector

The 403 EVB comes with a standard five-pin DIN connector for quick connect/disconnect to a power supply. A 3 Amp glass fuse is mounted on the board. A power supply and a line cord are provided with the 403 EVB. Power supply tolerances are ± 5% for the 5V supply. Table 5-6 defines the connections for the power supply connector:

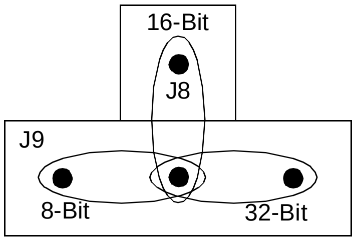
Table 5-6. Power Supply Connections

Pin Number	Function
	(card 42H1843)
1	GND
2	Frame Gnd
3	+5 V
4	No Connect
5	No Connect

Warning: use only the power supply provided in your EVB kit.

5.6 Setting the EVB Jumpers

Eight jumpers are provided on the EVB, as described in Table 5-7. Jumpers J8-9 are laid out as a three-position jumper that selects the width of the flash memory. The EVB comes with the jumper installed to select an eight-bit flash memory width. If the external memory is not eight bits wide, the jumper must be moved to select either 16-bit memory or 32-bit memory. If boot code is to be accessed from external memory, the flash memory must be removed from its socket.



When installed, J11 provides a jumpered connection of the system clock to the timer clock input on the 403. J4 and J5 are used for testing at the factory and must be installed.

Table 5-7. EVB Jumper Settings

Jumper No.	Function	Settings
J3	403 power source	As installed, connects the 403 processor to 3.3 V supply. An ammeter may be connected across these pins to measure ambient power to the 403.
J4	Factory test	Must be installed.
J5	Factory test	Must be installed.
J6	Ext_Ready select	Remove jumper if the signal is to be provided through the expansion connector.
J7	$\overline{\text{XCVR\_EN}}$ select	Remove jumper if the signal is to be provided through the expansion connector.
J8-9	BootWidth select	As installed at factory, selects 8-bit flash memory width.
J10	External reset input	Shorting the two pins together resets the PowerPC 403 processor.
J11	TimerClk source select	When installed, 10MHz is selected as TimerClk source. Optionally, an external clock input may be connected to the input.

5.7 Resetting the EVB

The RST switch on the board is a momentary SPST (Single Pole Single Throw) switch that generates a board hardware reset. A hardware reset simultaneously resets the 403 processor, the National DP83902 Ethernet controller and the National NS16550 serial communications controller.

5.8 Critical Interrupt Switch

The CINT switch on the board is a momentary SPST switch that generates a critical interrupt on the 403 processor chip. The ROM Monitor supports using the critical interrupt as a mechanism for suspending the execution of an application. When debug is not used, the ROM Monitor simply passes the critical interrupt on to the application's first level interrupt handler.

5.9 Connecting the 403 EVB Hardware

In order to establish a working environment, the EVB must be connected to a host system. ROM Monitor access requires a connection between the SP1 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.

Included in the 403 EVB kit are two interface cables, each supporting either 9-pin or 25-pin serial port connections. One is for connecting the SP1 serial port on the board to a terminal (or to a host running a terminal emulator) and the other for connecting the SP2 serial port on the board to a host system for debug and downloading applications. The Sun version of the 403 EVB kit contains only one serial port cable since ethernet is used for host-to-EVB comunications. The hardware necessary for establishing an ethernet connection is not included in the EVB kit.

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 SP1 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 the 25-pin connector or a serial port adapter (not supplied) for connectivity. Sun SPARCstation users may require the 25 pin male-to-male adapter (included in the Sun 403 EVB kit) at the host end. If a SLIP connection is going to be used for host-to-EVB communications, connect the second cable in a similar manner using SP2 on the EVB and the S2 (COM2) serial port on the host.

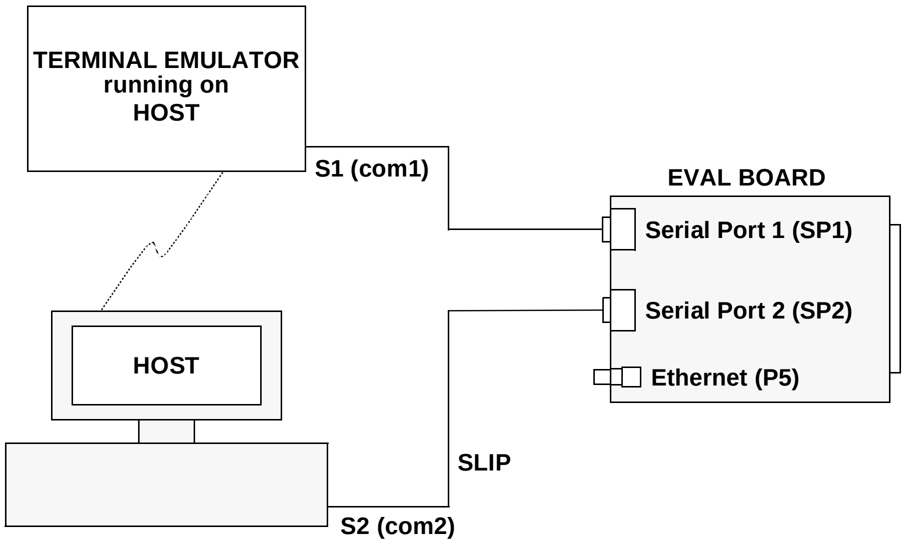


Figure 5-5. Serial Port Connection

If an Ethernet connection is going to be used, make any necessary cable connections between the host and the 10Base2 Ethernet connection on the EVB. If this connection is to be used exclusively for communications between the host and the EVB, each end must have a BNC "T" type connector terminated at one end. If the connection is going to be made to an existing ethernet network, users should consult their Network Adminstrator to insure proper connectivity.

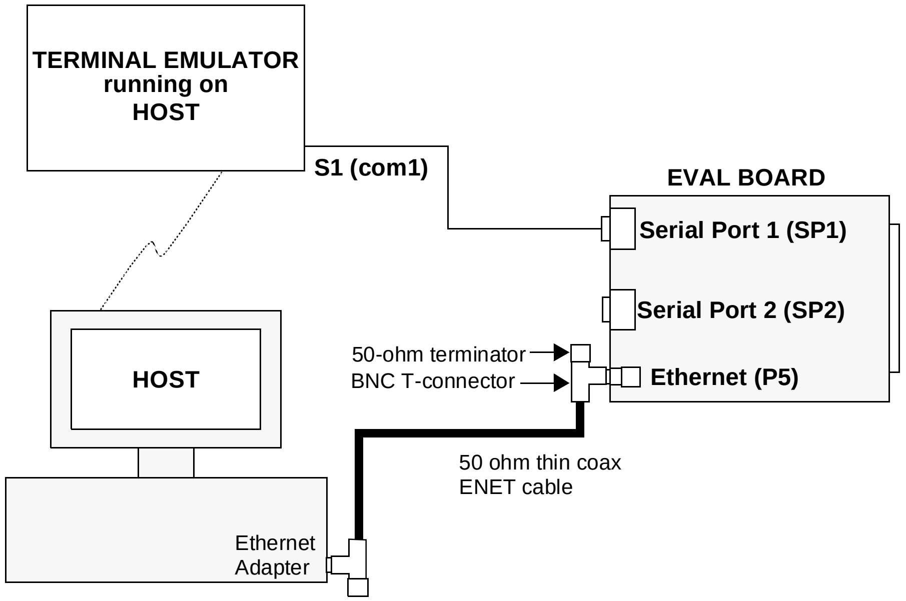


Figure 5-6. Point-to-Point 10Base2 Ethernet Connection

Note: Both a SLIP and Ethernet connection can be used as long as both networks have been configured properly and the proper connections have been made.

Also included in the 403 EVB kit, is a power supply and its power cord. Connect the female end of the power cord to the male connector on the power supply. Also connect the male 5-pin DIN connector from the power supply to the female 5-pin DIN connector on the board.

5.10 Using a Terminal Emulator

The ROM Monitor transmits/receives data through serial port 1 (SP1) on the evaluation board. Access to the ROM Monitor can be achieved by connecting a VT100 (or compatible) teminal directly to SP1 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 SP1 on the EVB and an available serial (or COM) port on the host system.

5.10.1 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.

5.10.2 PC Terminal Emulation

5.10.2.1 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).

5.10.2.2 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 'Hypertrm' 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.

5.10.3 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.

5.11 Booting the PowerPC 403 on the EVB

When the connectors have been installed and power is applied to the 403 EVB, pressing the Reset SPST switch causes the 403 and the communications controllers to reset. After the ROM monitor initializes the EVB, the monitor menu is displayed if a properly configured terminal (or terminal emulator) is attached to serial port 1 of the EVB. Details of ROM Monitor operation are provided in a later chapter.

6

403 EVB Hardware

This chapter describes the PowerPC 403 embedded controllers, memory subsystems, and external interfaces. For more detailed information on the 403 controllers, consult their respective user’s manuals:

- PPC403GA Embedded Controller User’s Manual
- PPC403GC Embedded Controller User’s Manual
- PPC403GCX Embedded Controller User’s Manual

Copies of the user’s manuals can be obtained by calling the IBM PowerPC Literature Center at 1(800)-POWERPC.

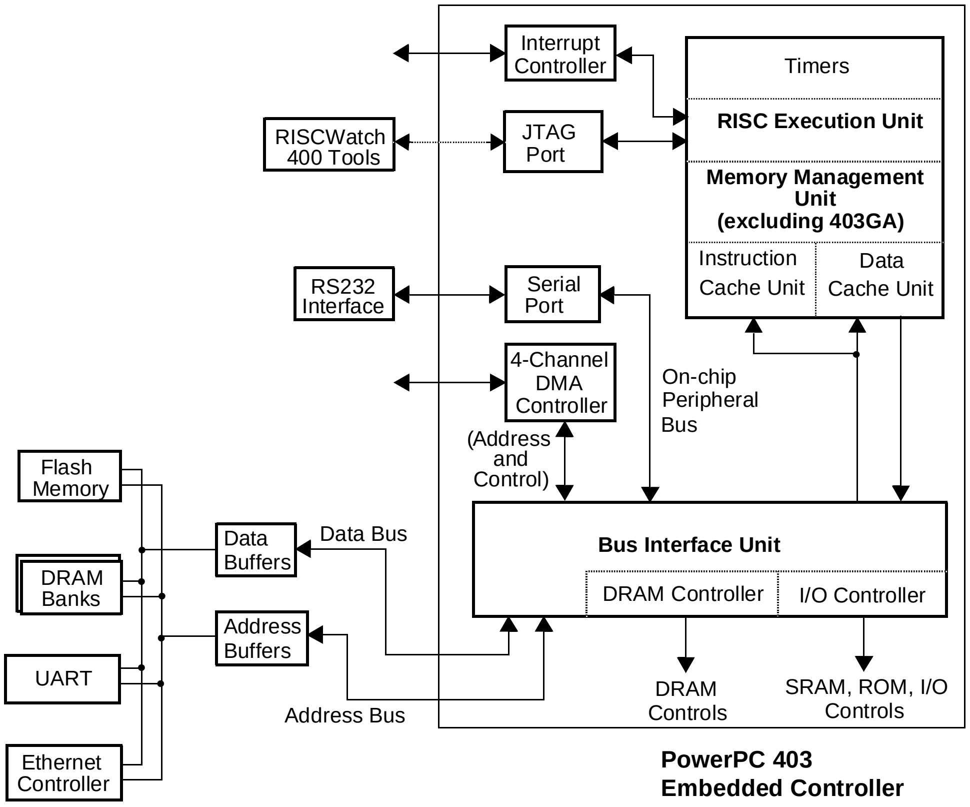


Figure 6-1. 403 EVB Block Diagram

6.1 403 Embedded Controllers

6.1.1 PowerPC 403 Embedded Controller

The 403GA RISC controller consists of a pipelined RISC processor core and several peripheral interface units: bus interface unit, DMA controller, asynchronous interrupt controller, serial port, and JTAG/debug port.

The RISC processor core includes the internal 2KB instruction cache and 1KB data cache, reducing overhead for data transfers to or from external memories. The instruction queue logic manages branch prediction, folding of branch and condition register logical instructions, and instruction prefetching to minimize pipeline stalls.

External I/O devices or SRAM/DRAM memory banks can be directly attached to the 403GA BIU. Interfaces for up to eight memory banks and I/O devices, including a maximum of four DRAM banks, can be configured individually, allowing the BIU to manage devices or memory banks with differing control, timing, or bus width requirements.

6.1.2 403GC Embedded Controller

The 403GC RISC controller provides all of the features of the 403GA plus a Memory Management Unit (MMU). The MMU provides address translation and protection functions for embedded applications. Together with appropriate system level software, the MMU provides the following functions:

- Translation of 4GB logical address space into physical addresses
- Independent enable of Instruction and Data translation/protection
- Page level access control via the translation mechanism
- Software control of page replacement strategy
- Additional control over protection via zones

6.1.3 403GCX Embedded Controller

The 403GCX RISC controller provides all of the features of the 403GC plus larger caches (16KB instruction cache and 8KB data), clock doubling, and EDO memory support.

6.2 Memory Subsystems

The bus interface unit in the 403 processor manages the external memory interfaces and provides services such as wait-state generation and DRAM refresh. The BIU also imposes some addressing restrictions:

- All SRAM (and SRAM look-alikes such as PROM) must have address[1:3]=b’111’.
- All DRAM must have address[1:3]=b’000’.
- Any address for which address[1:3] is neither b’000’ nor b’111’ is reserved, on the On-Chip Peripheral Bus (OPB).

To inform the BIU of the characteristics of the attached external devices, one or more Bank Registers must be programmed. Let BRn refer to the particular Bank Register programmed for an address range. The following must be true:

- If the bank register is any of BR0 - BR3, only SRAM is supported.
  It must be true that address[1:3]=b’111’ and BRn[31]=0.
- If the bank register is any of BR4 - BR7, the following apply:
  - If address[1:3]=b’111’ (the SRAM case), then must have BRn[31]=1.
  - If address[1:3]=b’000’ (the DRAM case), then must have BRn[31]=0.

- If address[1:3] is neither b’000’ nor b’111’, then it is not appropriate to program a Bank Register for this address. This address is reserved for internal use, and Bank Registers are only for external addresses. Neither the SRAM nor the DRAM controller will be activated by this address, and a machine check exception is generated by the timeout error on the OPB.

6.2.1 External Memory Banks

Each Bank Register controls the characteristics of one contiguous block of external memory, where only address bits 1:31 are considered. Address bit 0 can influence cacheability, in conjunction with the cacheability settings in the control registers for the instruction and data caches.

Address bits 1:3 can be set to b’000’ or b’111’ , and (for BR4-BR7) BRn bit 31 reflects this selection. Address bits 4:11 of the starting (lowest) address of the contiguous block are also mapped to BRn bits 0:7.

The contiguous block of memory that is described by the BR must be selected to be one of seven sizes shown in Table 6-1. The starting address in BRn bits 0:7 must be aligned on a boundary that agrees with the selected size. Table 6-1 assists in properly specifying BRn bits 0:7 (‘X’ denotes a don’t care).

Table 6-1. BR Bank Address Select

Size (megabytes)	Size Select BRn Bits 8:10	Bank Addr Sel BRn Bits 0:7
1	b’000’	b’XXXX XXXX’
2	b’001’	b’XXXX XXX0’
4	b’010’	b’XXXX XX00’
8	b’011’	b’XXXX X000’
16	b’100’	b’XXXX 0000’
32	b’101’	b’XXX0 0000’
64	b’110’	b’XX00 0000’

There are eight BRs in the 403. More than one BR is used only when multiple blocks of memory are described. This could occur because the blocks are not contiguous, or because the blocks (even if contiguous) require different memory characteristics (for example, one block is fast memory and another is slow memory). More than one BR should not be used to describe the same address.

6.2.2 Flash Memory Map and Bank Configuration

There is 128KB of flash memory on the 403 EVB. The memory bank is eight bits wide and comprises a single 128Kb $\times$ 8-bit device. The flash memory is accessed using bank register 0 (BR0), and has a read access time of 90 nanoseconds.

The last flash memory address must be located at the top of the 256MB address region, 0xFFFF FFFF, and descend from there. The processor always fetches the first instruction from address 0xFFFF FFFC after power-up or a system reset. To satisfy this requirement, nonvolatile memory addresses are mapped to the highest (uppermost) bank address. The bank address of the highest 1MB region is 0xFF; the addresses in that bank are 0xFFF0 0000 – 0xFFFF FFFF.

Figure 6-2 shows the fields of a bank register configured for SRAM, ROM, or other peripheral devices:

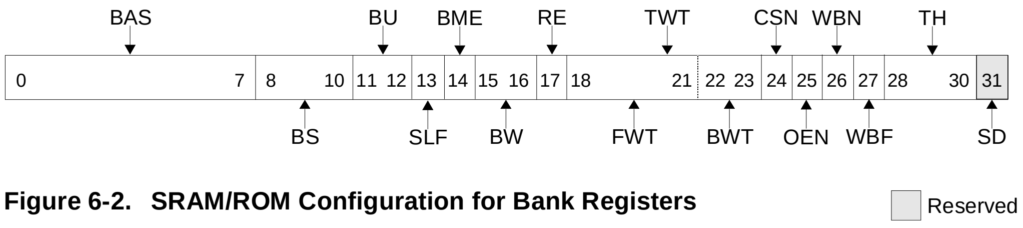


Figure 6-2. SRAM/ROM Configuration for Bank Registers

The ROM monitor writes BR0 with the eight-character hexadecimal word, 0xFF18 0242. This sets the fields in BR0 as follows:

Table 6-2. Bank Register 0 Field Settings

Field	Description	Bit Number	Value
BAS	Bank Address Select	0:7	0xFF
BS	Bank Size	8:10	0x0
BU	Bank Usage	11:12	0x3
SLF	Sequential Line Fills	13	0x0
BME	Burst Mode Enable	14	0x0
BW	Bus Width	15:16	0x0
RE	Ready Enable	17	0x0
TWT	Transfer Wait	18:23	0x02
CSN	Chip Select On Timing	24	0x0
OEN	Output Enable On Timing	25	0x1
WBN	Write Byte Enable On	26	0x0

Table 6-2. Bank Register 0 Field Settings

Field	Description	Bit Number	Value
WBF	Write Byte Enable Off	27	0x0
TH	Transfer Hold	28:30	0x1
	Reserved	31	0x0

6.2.3 DRAM Memory Map and Bank Configuration

The 403 EVB has two vertical-profile 72-pin SIMM slots. Each slot can accept any 72-pin JEDEC-compliant DRAM SIMM having 32 data bits. The board has a maximum DRAM capacity of 128MB when both slots are populated with 64MB SIMMs.

A populated slot contains one or two memory banks, depending on the type of SIMM used. A single sided SIMM provides a single bank of memory and a double-sided SIMM provides two banks. Four memory banks are supported when both slots are populated with double-sided SIMMs.

A 1Mb $\times$ 32-bit 60ns EDO SIMM is factory-installed in the first DRAM slot, slot 0. This single-sided SIMM provides 4MB of fast page-mode (FPM) DRAM for 403GA and 403GC users, or 4MB of EDO DRAM for 403GCX users.

Note - if two SIMMs of different sizes are installed, the larger SIMM must be installed in slot 0.

Like the flash memory bank, the DRAM banks are controlled by the bank registers. Bank registers 4–7 (BR4–7) are used to configure the DRAM banks. During power-up or reset, the ROM Monitor runs a check on the memory installed and configures the bank registers appropriately.

Figure 6-3 shows the fields of BR4 through BR7.

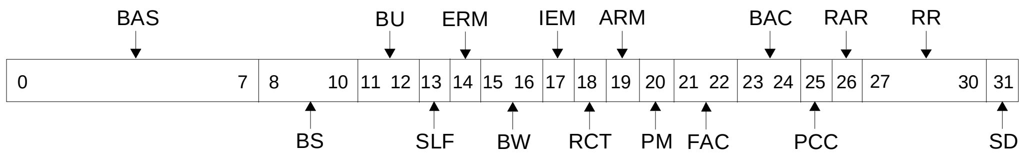


Figure 6-3.   Bank Registers - DRAM Configuration (BR4-BR7)

The installed 4MB memory bank can be accessed as soon as the ROM Monitor writes BR7 with the value 0x0059 0AB0. This value is then optimized based on the system configuration. Possible 403 EVB configurations and their appropriate BR7 values are included in the following table:

Table 6-3. Possible 403 EVB Bank Register 7 Settings

Speed (MHz)	Processor(s)	DRAM Type	BR 7 Value	Notes
25	403GA, 403GC	60ns FPM, EDO	0x00590AAE	IOCR[DRC]=1
33	403GA, 403GC	60ns FPM, EDO	0x00590AB0	IOCR[DRC]=1
25/50	403GCX	60ns FPM	0x00590AAE	IOCR[DRC]=1
33/66	403GCX	60ns FPM	0x00590AB0	IOCR[DRC]=1

25/50	403GCX	60ns EDO	0x0059082E	IOCR[DRC]=0
				IOCR[EDO]=1

33/66	403GCX	60ns EDO	0x00590A30	IOCR[DRC]=0
				IOCR[EDO]=1

Note that these values are not necessarily optimal for use with other board designs.

The ROM Monitor determines the processor speed dynamically by sending a single character through the 403's on-chip serial port in loopback mode, checks the processor version register (PVR), and then sets the bank registers appropriately. Please see the ramcheck.c source file in the openbios/miscLib directory for additional details.

Assuming the 403 EVB has a 403GA running at a speed of 33MHz, upon a reset of the board the ROM Monitor writes BR7 with the value 0x0059 0AB0. This sets the fields in BR7 as follows:

Table 6-4. Bank Register 7 Field Settings

Field	Field Description	Bit Number	Value
BAS	Bank Address Select	0:7	0x00
BS	Bank Size	8:10	0x2
BU	Bank Usage	11:12	0x3
SLF	Sequential Line Fills	13	0x0

Table 6-4. Bank Register 7 Field Settings

Field	Field Description	Bit Number	Value
ERM	Early RAS Mode	14	0x0
BW	Bus Width	15:16	0x2
IEM	Int/Ext Multiplexer	17	0x0
RCT	RAS to CAS Timing	18	0x0
ARM	Alternate Refresh Mode	19	0x0
PM	Page Mode	20	0x1
FAC	First Access Cycles	21:22	0x1
BAC	Burst Access Cycles	23:24	0x1
PCC	Pre-Charge Cycles	25	0x0
RAR	RAS Active During Refresh	26	0x1
RR	Refresh Rate	27:30	0x8
S/D	SRAM/DRAM	31	0x0

If a second 1Mb X 32-bit 60ns SIMM was plugged into slot 1 of the board, upon power-up or reset the ROM Monitor would write BR6 with the eight character hexadecimal word, 0x0459 0AB0.

6.2.4 Bank Configuration (BR1) for the National 16550 Serial Controller

Following power-up or reset, BR1 is disabled. The National NS16550 serial communications controller (which is connected to serial port 2 on the board) can be accessed as soon as the ROM monitor writes BR1 with the eight-character hexadecimal word, 0xE018 0468 This sets the fields in BR1 as follows:

Table 6-5. Bank Register 1 Field Settings

Field	Description	Bit Number	Value
BAS	Bank Address Select	0:7	0xE0
BS	Bank Size	8:10	0x0

Table 6-5. Bank Register 1 Field Settings

Field	Description	Bit Number	Value
BU	Bank Usage	11:12	0x3
SLF	Sequential Line Fills	13	0x0
BME	Burst Mode Enable	14	0x0
BW	Bus Width	15:16	0x0
RE	Ready Enable	17	0x0
TWT	Transfer Wait	18:23	0x04
CSN	Chip Select On Timing	24	0x0
OEN	Output Enable On Timing	25	0x1
WBN	Write Byte Enable On	26	0x1
WBF	Write Byte Enable Off	27	0x0
TH	Transfer Hold	28:30	0x4
	Reserved	31	0x0

6.2.5 Bank Configuration (BR2) for the Ethernet Controller

Following power-up or reset, BR2 is disabled. The National NS83902 Ethernet controller can be accessed as soon as the ROM monitor writes BR2 with the eight-character hexadecimal word, 0x4018 46F4. This sets the fields in BR2 as follows:

Table 6-6. Bank Register 2 Field Settings

Field	Description	Bit Number	Value
BAS	Bank Address Select	0:7	0x40
BS	Bank Size	8:10	0x0
BU	Bank Usage	11:12	0x3
SLF	Sequential Line Fills	13	0x0
BME	Burst Mode Enable	14	0x0
BW	Bus Width	15:16	0x0
RE	Ready Enable	17	0x1
TWT	Transfer Wait	18:23	0x06
CSN	Chip Select On Timing	24	0x1
OEN	Output Enable On Timing	25	0x1
WBN	Write Byte Enable On	26	0x1
WBF	Write Byte Enable Off	27	0x1
TH	Transfer Hold	28:30	0x4
	Reserved	31	0x0

6.3 403 EVB Address Map

The ROM monitor initializes the instruction cache control register (ICCR) and the data cache control register (DCCR) so that the installed flash memory and DRAM banks are cacheable. The peripheral devices configured in bank registers 1-4 are set up in the ICCR and the DCCR as noncacheable.

The addresses for memory banks and peripheral devices are listed in Table 6-7 below:

Table 6-7. 403 EVB Memory Map

Bank Register : Device	Address
BR0: 128KB Flash Memory	Cacheable region: 0xFFF0 0000 - 0xFFFF FFFF
	Noncacheable region: 0x7FF0 0000 - 0x7FFF FFFF
BR1: Serial Communications Controller (Serial Port 2)	Noncacheable region: 0x7E00 0000 - 0x7E00 0007
BR2: Ethernet Controller	Noncacheable region: 0xF400 0000 - 0xF400 003F
BR7: 4MB DRAM	Noncacheable region: 0x8000 0000 - 0x803F FFFF
	Cacheable region: 0x0000 0000 - 0x003F FFFF

6.4 Ethernet and Serial Port Interrupts

The assignments of the Ethernet and serial controller interrupts to the 403 external interrupt inputs are presented in Table 6-8 below:

Table 6-8. Ethernet and Serial Port Interrupts

Interrupt Source	403 Input
Ethernet Controller	INT0, polarity programmed active-high, level sensitive triggering
Serial Port 2	INT1, polarity programmed active-high, level sensitive triggering

These assignments are made via the input/output configuration register (IOCR).

6.5 The Ethernet Controller’s Network Address

The EVB’s Ethernet controller, a National DP83902, 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’s serial port 1 (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 as part of the board_cfg_data structure when a call to its get_board_config() function is made. Sample 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 0xFFFFFE00 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 0xFFFFFE00, scan for “*NA”, verify the next byte is 0x6, and treat the next six bytes as the network address.

6.6 Accessing the Ethernet Controller

The method of attachment of the Ethernet controller to the 403 bus can cause it to be forced off of the bus during a transaction. To avoid this problem a byte register at address 0xF4000020 must be interrogated after every Ethernet chip access. If the value of the register is 0x01, the access was successful. Otherwise, the operation must be retried. See enetLib.c in the openbios/enetLib directory for examples and additional information.

7

403 EVB ROM Monitor

This chapter describes the 403 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 seperate 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
• 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 403 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 403 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 403 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 User's Guide*. To use the facilities of the ROM Monitor to download 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=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 **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 extention. 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 extention) 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 concatonate 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: TFTPD

Command Line: TFTPD

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 "TFTPD.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:
  ```
  TFTPD
  ```

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 connnection 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 10Base2 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:

- Do not alter the EVPR register
- Provide exception vectors for application events as if the EVPR were set to 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 0x0000 A000 and the end of DRAM only, except for application vectors.
- Do not start applications lower than address 0x0000 A000

Figure 7-1 shows the address map of the evaluation board under control of the ROM Monitor. The “folding” characteristics of the high order address bit are not shown.

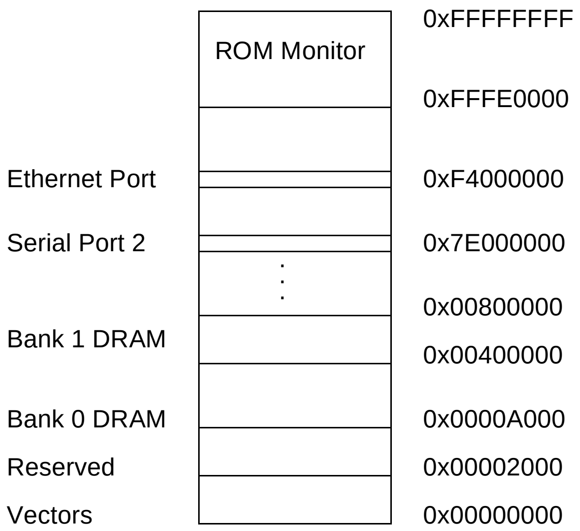


Figure 7-1. ROM Monitor Address Map

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 403 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 flash memory.

7.6.1 Initial ROM Monitor Menu

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

```
403GA 2.1 ROM Monitor (8/2/96)

 ------------- System Info -------------
 Processor speed   =  33 MHz
 Bus speed         =  33 MHz
 Amount of DRAM  =  4 MB
 -------------------------------------------

 --- 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
   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
     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
    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
    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
      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.
- If a local 10Base2 Ethernet network is being used, that is one being used exclusively by the board and the host, insure that **both** ends of the Ethernet cable have BNC “T” type connectors with a terminator at one end.
- 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
     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
      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
     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 flash memory 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 tranfers at upto 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 upto 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
    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
     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 kermit 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
   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 kermit 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.

```
^\c  (Cntrl-\c)
(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 403 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
   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.

8

403 EVB Sample Applications

This chapter describes the steps necessary to build and run the sample programs included in the 403 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 C 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 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 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 bootp 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.

```
$$ Standard ROM Monitor load screen below
403GA 1.2 ROM Monitor (9/5/95)
$$ Version 1.2 already installed corresponds to rom_12.img

 ------------- System Info -------------
 Processor speed   =  33 MHz
 Bus speed         =  33 MHz
 Amount of DRAM  =  4 MB
 ------------------------------------------
 --- 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=ffffffffffff
   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 0x10320 ...

$$ following information is from the ROM Monitor update program
############### IBM 4XX 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, mmu_samp, 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 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 0x10238 ...
## Having loaded an image, the ROM monitor is now transfering 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 0x10180 ...

Hello 403 user!

Your ROM Monitor version is : 7.5

Your 403 Evaluation Board has 8388608 bytes of DRAM installed.

Your Ethernet controller's network address is :  1000abcdef55

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 0x10180 ...

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.

8.3.4 Building and Running the mmu_samp Program

The **mmu_samp.c** program is included as a sample to be built and run on an evaluation board with a 403GC or 403GCX processor. It demonstrates some of the features of the 403GC(X)'s Memory Manager Unit. It uses the MMU to protect read-only and non-executable pages of memory. It also sets a read-only page at the bottom of the stack to immediately detect stack overflow errors. The sample also makes a copy of the ROM code in RAM and translates the addresses to make the copy appear to be at its original location. This allows you to set software breakpoints in the code. For more information on programming the MMU refer to the *403GC Embedded Controller User's Manual* or the *403GCX Embedded Controller User's Manual*

To understand how this sample works, you must understand how the memory is used. The file "mmu_samp.mpf" is used on the link step of this sample to achieve the required alignment of sections. The memory is setup as follows:

- 0x00000000 to 0x00001FFF contains 403GC(X) exception vectors and is mapped as two 4K pages, readable and executable.
- 0x00002000 to 0x00009FFF is used by the ROM monitor and is mapped as eight 4K pages, readable, writable and executable
- 0x0000A000 to 0x0000FFFF is unused.
- 0x00010000 to 0x0001BFFF is the sample's read-only data and text sections. It is mapped as three 16K pages, readable and executable.

- 0x0001C000 to 0x0001C3FF is the sample’s data section and is mapped as one 1K page, readable and writable.
- 0x0001C400 to 0x0001FFFF is unused.
- 0x00020000 to 0x0003FFFF contains the flash contents and is mapped to 0xFFFE0000 through 0xFFFFFFFF as two 64K readable and executable pages.
- 0x00040000 to 0x000403FF contains the sample’s bss section and is mapped as one 1K page, readable and writable.
- 0x00040400 to 0x000407FF is a 1K read-only page that detect stack overflow errors.
- 0x00040800 to 0x00043FFF is the sample’s stack area. It consists of two 1K pages and three 4K pages, readable and writable.
- 0x00044000 to 0x0007FFFFF is heap area and consists of three 16K, three 64K, three 256K, three 1M and one 4M pages, readable and writable. The evaluation kit software will automatically assign stack and heap area following bss to the end of memory.
- 0x40000000 to 0x400003FF is the serial port memory-mapped I/O area. It is a 1K page, readable, writable, cache-inhibited and guarded.

Header file “tlbdef.h” is provided with the sample. It contains useful definitions for managing the TLB.

File “accvaddr.c” contains a sample virtual memory access function. This is used in conjunction with the ROM monitor debugger. If instruction or data translation is active, the debugger calls a helper function to read and write memory. Some applications may overflow the TLB and the software will manage TLB replacements on misses. In this case, a more elaborate function should be written, that understands the TLB replacement algorithms. For applications that fit in the TLB, this sample function will work fine.

To build the mmu_samp program, enter the command “**make mmu_samp**” from the command line while in the **samples** directory. The makefile will compile the mmu_samp.c and accvaddr.c files, link the resulting object files with the support libraries, and produce the ELF format boot file, **mmu_samp**, and the boot image file, **mmu_samp.img**.

If the suggested bootptab was used, then **mmu_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.

For debugger loads use the RISCWatch **load image** command to load the **mmu_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 0x101f0 ...
```

```
Hello 403 user!

Your ROM Monitor version is : 6.2

Your 403 Evaluation Board has 4194304 bytes of DRAM installed.

  Your Ethernet controller’s network address is :  1000abcdef55


Force data storage error...

PANIC!!!================================
failing thread id     =0x  3f7148
fault number           =0x       d
failing address        =0x    1100
```

If you receive the following messages, your evaluation board does not contain a 403GC or 403GCX processor and this sample can not be run.

```
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 0x101f0 ...
This sample cannot be run on this processor
Return from main function. rc: -1
```

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.

9 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	403 EVB
Boot Library(DRAM)	bootlLib.a	403 EVB
Boot Library(ROM)	bootrLib.a	403 EVB

Table 9-1. OS Open Libraries

Library	**File Name**	**Platforms**
ROM Monitor Ethernet Interface Library	benetLib.a	403 EVB
C++ runtime support (High C++™ support) Library	cppLib.a, crti.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	clockLib.a	403 EVB
Debug Support Library	dbLib.a	Common
Device and File Support Library	devLib.a	Common
DOS File System Support Library	fatLib.a	Common
Dynamic Loader Library	ldrLib.a	Common
Ethernet Support Library	enetLib.a	403 EVB
Extended Serial Communication Support Library	esccLib.a	403 EVB
File Transfer Protocol Support Library	ftpLib.a	Common
Floating Point Library	fpeLib.a	403 EVB ELF
Floating Point Emulation Library	fpCSLib.a	Common
Input/output Support Library	ioLib.a	403 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	403 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

Table 9-1. OS Open Libraries

Library	File Name	Platforms
Runtime Library	runlib.a	Common
SCSI Support Library	scsiLib.a	Common
Serial Support Library	asyncLib.a	403 EVB
Socket Services for PCMCIA support	ssLib.a	Common
Symbol Support Library	symLib.a	Common
TCP/IP Protocol Support Library	tcpipLib.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	403 EVB
Trivial File Transfer Protocol	tftp.o	Common
TTY Support Library	ttyLib.a	Common
XL C Compiler Support Library	xlcLib.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 403 EVB are described below:

Table 9-2. OS Open Libraries for the 403 EVB

Library	File Name
Boot Library(RAM)	bootlLib.a
Boot Library for OS Open in ROM	bootrLib.a
Ethernet Device Driver Support Library	enetLib.a
Extended Serial Communication Support Library	esccLib.a
Input/Output Support Library	ioLib.a
PowerPC Low Level Access Support Library	ppcLib.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 native serial port on the 403GA, 403GC, and 403GCX processors. 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 403 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 403 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 PPC403GA’s, PPC403GC’s and the PPC403GCX’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, "403 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 Boot Library

This library contains the OS Open bootstrap program and can be used instead of the boot library. The ROM boot library should be used when OS Open is being burnt into a ROM. The boot library performs initial processing to prepare the completed application program for execution on the platform. The boot library for the 403 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.

9.2.6 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 403 EVB

Device drivers provided with the 403 EVB include:

- Asynchronous
- Ethernet
- ROM Monitor 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 S1 asynchronous communication port found on the 403GA, 403GC and 403GCX processors. 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 7 and 8 bits.
- Support for 1 and 2 stop bits.
- Support for receive and transmit parity.
- Support for odd and even parity.
- Support for transmitting BREAK.
- Programmed I/O (PIO) operation.
- 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, 128,7372800, 2);
```

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

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

Parameter	**Meaning**
Fourth Parameter	Port number to be installed (1)
Fifth Parameter	Size of write buffer
Sixth Parameter	Size of read buffer
Seventh Parameter	Input clock for the divisor
Eighth Parameter	Value for bits 30 and 31 of the IOCR register.

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: asyncParityNone and asyncParityGen_Check
Second Parameter	Parity type. Valid values are asyncParityEven and asyncParityOdd. Because parameters are positional, this parameter must be specified even if parity is not used.
Third Parameter	Number of stop bits. Valid values are asyncStopBits1 and asyncStopBits2.
Fourth Parameter	Data length. Valid values are asyncDataBits5, asyncDataBits6, asyncDataBits7, and asyncDataBits8.
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
ASYNCBREAKSET	None	Starts sending BREAK on port
ASYNCBREAKCLR	None	Stops sending BREAK on port
ASYNCRERRORGET	Pointer to integer	Returns and clears read error conditions. Values are defined in asyncLib.h
ASYNCWERRORGET	Pointer to integer	Returns and clears write error conditions. Values are defined in asyncLib.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
ASYNCDLENGET	Pointer to integer	Returns current data length
ASYNCDLENSET	asyncDataBits7, asyncDataBits8	Sets data length
ASYNCSTOPGET	Pointer to integer	Returns number of stop bits
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

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

Command	Parameters	Explanation
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 serial port:

```
int s1dbprintf(unsigned long uart_clock, int iocr_reg, const char *format, ...)
```

The parameters passed are identical to **printf()** except for *uart_clock* and *iocr_reg*. *uart_clock* specifies the clock speed and *iocr_reg* should be set to the SerClk pin as a clock source(should be set to 0). Because polled I/O transmits characters synchronously, this function 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 Extended Serial Communication Controller Device Driver

The ESCC device driver supports S2 asynchronous communication port found in the standard I/O subsystem on the 403 EVB platform. 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 bytes 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.

9.3.2.1 Device Driver Installation

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

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

escc_init() is declared in the file **<esccLib.h>** as follows:

```
int escc_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*.

9.3.2.2 Device Installation

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

```
rc =device_install(“/dev/s2”,CHRTYPE,devhandle,1,128,128);
```

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

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

Parameter	**Meaning**
First Parameter	Port number to be installed (1 or 2)
Second Parameter	Size of write buffer
Third Parameter	Size of read buffer

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*.

9.3.2.3 Opening ESCC Communication Port

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/s2", 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: asyncParityNone and asyncParityGen_Check
Second Parameter	Parity type. Valid values are asyncParityEven and asyncParityOdd. Because parameters are positional, this parameter must be specified even if parity is not used.
Third Parameter	Number of stop bits. Valid values are asyncStopBits1, asyncStopBits2, and asyncStopBits15. One and a half stop bits are only valid for data length of 5.
Fourth Parameter	Data length. Valid values are asyncDataBits5, asyncDataBits6, asyncDataBits7, and asyncDataBits8.
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*.

9.3.2.4 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.

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

9.3.2.5 I/O Control

An **ioctl()** call issued against ESCC device driver accepts the commands listed in Table 9-4. All parameter constants can be found in **<sys/ioctl.h>.**

Table 9-4. ioctl() Commands for the ESCC Device Driver

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

Table 9-4. ioctl() Commands for the ESCC Device Driver

Command	Parameters	Explanation
ASYNCSTICKONE	None	Parity interpretation tracks even/odd parity
ASYNCRERRORGET	Pointer to integer	Returns and clears read error conditions. Values are defined in asyncLib.h
ASYNCWERRORGET	Pointer to integer	Returns and clears write error conditions. Values are defined in asyncLib.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
ASYNCDLENGET	Pointer to integer	Returns current data length
ASYNCDLENSET	asyncDataBits5, asyncDataBits6, asyncDataBits7, asyncDataBits8	Sets data length
ASYNCSTOPGET	Pointer to integer	Returns number of stop bits
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

Table 9-4. ioctl() Commands for the ESCC Device Driver

Command	**Parameters**	**Explanation**
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.

9.3.2.6 Polled Asynchronous I/O

A function is provided for polled output to s2 serial port:

int s2dbprintf(const char *format, ...)

Parameters passed to this function are identical to those passed to **printf()**. Because polled I/O transmits characters synchronously, this function may be called from first level interrupt handlers (FLIHs) or a user-supplied panic function. Since this function waits until the characters are actually sent before returning, using this with long strings can significantly affect the timing of calling programs.

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.

enet_native_attach() attaches the TCP/IP protocol to the Ethernet device. Once the TCP/IP stack is attached, Ethernet packets can be sent/received using the TCP/IP functions or by using the Ethernet functions provided, such as **enet_send_packet().**

The Ethernet device is attached to the TCP/IP protocol stack after **tcpip_init()** and **net_init()** have been performed. The following is an example of attaching the TCP/IP protocol stack to the Ethernet.

```
#include <enetLib.h>
#define ENETHOST “403_board”
#define ENET_CONFIG “ ent0 403_board netmask 255.255.240.0 up”
#define SRAM_SIZE  8192  /* 8K buffer */
int do_enet()
{
  int rc;
  unsigned long processor_clock_speed;
  processor_clock_speed = processor_speed();
  rc = tcpip_init( ENETHOST, 1, 1000); /* Initialize the TCP/IP library */
  if(rc != 0)
      return -1;
  rc = net_init();  /*initialize netLib */
  if(rc != 0)
      return -1;
      /* attatch the TCP/IP protocol stack */
  rc = oakenet_attach( processor_clock_speed);
  if(rc != 0)
      return -1;
  rc = ifconfig(ENET_CONFIG); /* configure network interface */
  if(rc != 0)
      return -1;
  return 0;
}
```

9.3.4 ROM Monitor Ethernet Device Driver

The ROM Monitor Ethernet device driver provides network access to the applications running on the 403 EVB, while still allowing the ROM Monitor to access the RISCWatch debugger over the ethernet.

This device driver uses code resident in the ROM monitor to send and receive ethernet packets. A different IP address must be specified to distinguish the packets from ROM Monitor and OS Open. I/O initialization should be done by calling **dbg_ioLib_init()** rather than **ioLib_init()**.

9.3.4.1 ROM Monitor Ethernet Installation and Initialization

The ROM Monitor Ethernet device driver is installed by calling **biosenet_attach()**. Following is a prototype of this function:

```
#include <benetLib.h>
int biosenet_attach(unsigned long ipaddr, int init_flag);
```

Upon successful installation, **biosenet_attach()** returns 0; otherwise -1 is returned. The IP address for OS Open is specified in the *ipaddr* parameter. The *init_flag* specifies whether the Ethernet controller needs to be initialized. If *init_flag* is set to 0 then the Ethernet controller is not initialized. If *init_flag* is set to a non-0 value, initialization of the Ethernet controller is performed.

9.4 Utilities

as-emb

This XCOFF assembler is a functional superset of the **as** assembler supplied with AIX for the RS6000 platform only. The XCOFF assembler has been extended with additional machine types and extended mnemonics in support of the PowerPC embedded microcontrollers.

For basic assembler information such as source file requirements, syntax, pseudo ops, etc. consult the AIX Assembler Language Reference SC23-2197. It may also be helpful to have the User’s Manual for the embedded controller being used, such as the User’s Manual.

Extensions to .machine

The .machine pseudo operation has been extended to identify members of the IBM PowerPC embedded controller family. Values that may be used include:

- 403GA - Using this mode specifies PowerPC instructions plus those instructions unique to the 403GA, 403GC and 403GCX.
- PPC-EMB - This mode will permit any instruction valid for the PowerPC Embedded Controller family to be assembled.

Note: This assembler assumes the user is aware of what PowerPC features are not available on a specific processor. For example, the mftb extended mnemonic (move from time base) will assemble without assembler error for .machine “403GA”, but will not execute correctly, as the 403GA does not have a PowerPC timebase.

Additional instructions and extended mnemonics

When one of the previous .machine options is specified, the following instructions and mnemonics are available. Consult the controller User’s Manual for details concerning the operation of these instructions.

New instructions

dccci ra,rb	Data cache congruence class invalidate
dcread rt,ra,rb	Data cache read
icbt ra,rb	Instruction cache block touch
iccci ra,rb	Instruction cache congruence class invalidate
icread ra,rb	Instruction cache read
mfdcr rt,dcr	Move from Device Control Register
mtdcr dcr,rs	Move to Device Control Register
rfci	Return from critical interrupt
wrtee rs	Write External Enable
wrteei ei	Write External Enable Immediate

Instructions for PowerPC registers

mtsprg0 rs	Move to Special General Purpose Register 0
mtsprg1 rs	Move to Special General Purpose Register 1
mtsprg2 rs	Move to Special General Purpose Register 2
mtsprg3 rs	Move to Special General Purpose Register 3
mfsprg0 rt	Move from Special General Purpose Register 0
mfsprg1 rt	Move from Special General Purpose Register 1
mfsprg2 rt	Move from Special General Purpose Register 2
mfsprg3 rt	Move from Special General Purpose Register 3

Extended Mnemonics for mfspr

mfcdbcr rs	Move from Cache Debug Control Register (PPC403GC and PPC403GCX only)
mfdac1 rt	Move from Data Address Compare 1
mfdac2 rt	Move from Data Address Compare 2
mfdbcr rt	Move from Debug Control Register
mfdbsr rt	Move from Debug Status Register
mfdccr rt	Move from Data Cache Cachability Register
mfdcwr rt	Move from Data Cache Write-thru Register (PPC403GC and PPC403GCX only)
mfdear rt	Move from Date Exception Address Register
mfesr rt	Move from Exception Syndrome Register
mfevpr rt	Move from Exception Vector Prefix Register
mfiac1 rt	Move from Instruction Address Compare 1
mfiac2 rt	Move from Instruction Address Compare 2
mficcr rt	Move from Instruction Cache Cachability Register
mticdbdr rt	Move from Instruction Cache Debug Data Register (PPC403GC and PPC403GCX only)
mfpbl1 rt	Move from Protection Bound Lower 1
mfpbl2 rt	Move from Protection Bound Lower 2
mfpbu1 rt	Move from Protection Bound Upper 1
mfpbu2 rt	Move from Protection Bound Upper 2

mfpid rt	Move from Process ID Register (PPC403GC only)
mfpit rt	Move from Programmable Interval Timer
mfpvr rt	Move from Processor Version register
mfsgr rt	Move from Storage Guard Register (PPC403GC only)
mfsmr rt	Move from Srorage Memory-coherent Register (PPC403GC and PPC403CX only)
mfsrr2 rt	Move from Save/Restore Register 2
mfsrr3 rt	Move from Save/Restore Register 3
mftbhi rt	Move from 403 time base high
mftblo rt	Move from 403 time base low
mftcr rt	Move from Timer Control Register
mftsr rt	Move from TImer Status Register
mfzpr rt	Move from Zone Protect Register (PPC403GC and PPC403CX only)

Extended Mnemonics for mtspr

mtcdbcr rs	Move to Cache Debug Control Register (PPC403GC only)
mtdac1 rs	Move to Data Address Compare 1
mtdac2 rs	Move to Data Address Compare 2
mtdbcr rs	Move to Debug Control Register
mtdbsr rs	Move to Debug Status Register
mtdccr rs	Move to Data Cache Cachability Register
mtdear rs	Move to Date Exception Address Register
mtdcwr rs	Move to Data Cache Write-thru Register (PPC403GC and PPC403CX only)
mtesr rs	Move to Exception Syndrome Register
mtevpr rs	Move to Exception Vector Prefix Register
mtiac1 rs	Move to Instruction Address Compare 1
mtiac2 rs	Move to Instruction Address Compare 2
mticcr rs	Move to Instruction Cache Cachability Register

mticdbdr rs	Move to Instruction Cache Debug Data Register (PPC403GC and PPC403CX only)
mtpbl1 rs	Move to Protection Bound Lower 1
mtpbl2 rs	Move to Protection Bound Lower 2
mtpbu1 rs	Move to Protection Bound Upper 1
mtpbu2 rs	Move to Protection Bound Upper 2
mtpid rs	Move to Process ID Register (PPC403GC and PPC403CX only)
mtpit rs	Move to Programmable Interval Timer
mtsgr rs	Move to Storage Guard Register (PPC403GC and PPC403CX only)
mtsmr rs	Move to Srorage Memory-coherent Register (PPC403GC and PPC403CX only)
mtsrr2 rs	Move to Save/Restore Register 2
mtsrr3 rs	Move to Save/Restore Register 3
mttbhi rs	Move to 403 time base high
mttblo rs	Move to 403 time base low
mttcr rs	Move to Timer Control Register
mttsr rs	Move to TImer Status Register
mtzpr rs	Move to Zone Protect Register (PPC403GC and PPC403CX only)

Extended Mnemonics for mfdcr

mfdear rt	Move from Bus Error Address Register
mfbesr rt	Move from Bus Error Syndrome Register
mfbr0 rt	Move from Bank Register 0
mfbr1 rt	Move from Bank Register 1
mfbr2 rt	Move from Bank Register 2
mfbr3 rt	Move from Bank Register 3
mfbr4 rt	Move from Bank Register 4
mfbr5 rt	Move from Bank Register 5
mfbr6 rt	Move from Bank Register 6
mfbr7 rt	Move from Bank Register 7

mfdmacc0 rt	Move from DMA Chained Count Register 0
mfdmacc1 rt	Move from DMA Chained Count Register 1
mfdmacc2 rt	Move from DMA Chained Count Register 2
mfdmacc3 rt	Move from DMA Chained Count Register 3
mfdmacr0 rt	Move from DMA Channel Control Register 0
mfdmacr1 rt	Move from DMA Channel Control Register 1
mfdmacr2 rt	Move from DMA Channel Control Register 2
mfdmacr3 rt	Move from DMA Channel Control Register 3
mfdmact0 rt	Move from DMA Count Register 0
mfdmact1 rt	Move from DMA Count Register 1
mfdmact2 rt	Move from DMA Count Register 2
mfdmact3 rt	Move from DMA Count Register 3
mfdmada0 rt	Move from DMA Destination Address Register 0
mfdmada1 rt	Move from DMA Destination Address Register 1
mfdmada2 rt	Move from DMA Destination Address Register 2
mfdmada3 rt	Move from DMA Destination Address Register 3
mfdamasa0 rt	Move from DMA Source Address Register 0
mfdamasa1 rt	Move from DMA Source Address Register 1
mfdamasa2 rt	Move from DMA Source Address Register 2
mfdamasa3 rt	Move from DMA Source Address Register 3
mfdmasr rt	Move from DMA Status Register
mfexier rt	Move from External Interrupt Enable Register
mfexisr rt	Move from External Interrupt Status Register
mfiocr rt	Move from IO Control Register

Extended Mnemonics for mtdcr

mtdear rs	Move to Bus Error Address Register
mtbesr rs	Move to Bus Error Syndrome Register
mtbr0 rs	Move to Bank Register 0
mtbr1 rs	Move to Bank Register 1
mtbr2 rs	Move to Bank Register 2

mtbr3 rs	Move to Bank Register 3
mtbr4 rs	Move to Bank Register 4
mtbr5 rs	Move to Bank Register 5
mtbr6 rs	Move to Bank Register 6
mtbr7 rs	Move to Bank Register 7
mtdmacc0 rs	Move to DMA Chained Count Register 0
mtdmacc1 rs	Move to DMA Chained Count Register 1
mtdmacc2 rs	Move to DMA Chained Count Register 2
mtdmacc3 rs	Move to DMA Chained Count Register 3
mtdmacr0 rs	Move to DMA Channel Control Register 0
mtdmacr1 rs	Move to DMA Channel Control Register 1
mtdmacr2 rs	Move to DMA Channel Control Register 2
mtdmacr3 rs	Move to DMA Channel Control Register 3
mtdmact0 rs	Move to DMA Count Register 0
mtdmact1 rs	Move to DMA Count Register 1
mtdmact2 rs	Move to DMA Count Register 2
mtdmact3 rs	Move to DMA Count Register 3
mtdmada0 rs	Move to DMA Destination Address Register 0
mtdmada1 rs	Move to DMA Destination Address Register 1
mtdmada2 rs	Move to DMA Destination Address Register 2
mtdmada3 rs	Move to DMA Destination Address Register 3
mtdamasa0 rs	Move to DMA Source Address Register 0
mtdamasa1 rs	Move to DMA Source Address Register 1
mtdamasa2 rs	Move to DMA Source Address Register 2
mtdamasa3 rs	Move to DMA Source Address Register 3
mtdmasr rs	Move to DMA Status Register
mtexier rs	Move to External Interrupt Enable Register
mtexisr rs	Move to External Interrupt Status Register
mtiocr rs	Move to IO Control Register

Extended Mnemonics for tlbwe

tlbwe rs, rs, ws	Write to TLB, ws = 0 for HI, ws = 1 for LO, for entry ra (PPC403GC and PPC403CX only)
tlbwehi rs, ra	Write to TLBHI for entry ra (PPC403GC and PPC403CX only)
tlbwelo rs, ra	Write to TLBLO for entry ra (PPC403GC and PPC403CX only)

Extended Mnemonics for tlbre

tlbwe rt, rs, ws	Read from TLB, ws = 0 for HI, ws = 1 for LO, for entry ra (PPC403GC and PPC403CX only)
tlbrehi rt, ra	Read from TLBHI for entry ra (PPC403GC and PPC403CX only)
tlbrelo rt, ra	Read from TLBLO for entry ra (PPC403GC and PPC403CX only)

Extended Mnemonics for tlbsx

tlbsx rt, ra, rb	Search the TLB for translation(PPC403GC and PPC403CX only)

9.5 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 C for details of the load format). 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 User’s Guide for additional information on loading files.

9.5.1 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.

9.5.2 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 OS Open User's Guide.

9.6 Tools

Several host tools are provided to assist you in using the EVB support package or creating your own applications for the PowerPC 403. The tools can also be used for ROM program development.

9.6.1 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:

```
struct info_block {
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.

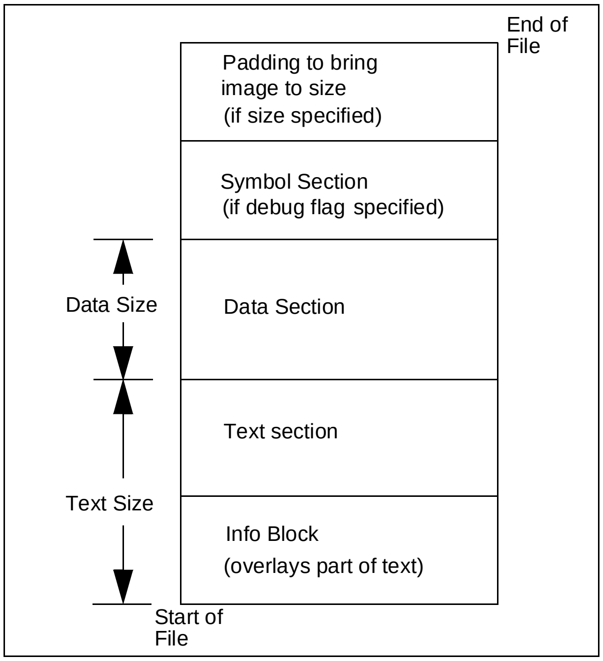

End of File
Padding to bring image to size (if size specified)
Symbol Section (if debug flag specified)
Data Size
Data Section
Text section
Text Size
Info Block (overlays part of text)
Start of File

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**.

9.6.2 hbranch

hbranch places a branch at the end of a ROM image. This simplifies production of ROM images for the PowerPC 403, which executes the instruction at the top location of memory following power-up or reset. **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 *stderr* 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 *patch_file* 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 *stdout*..

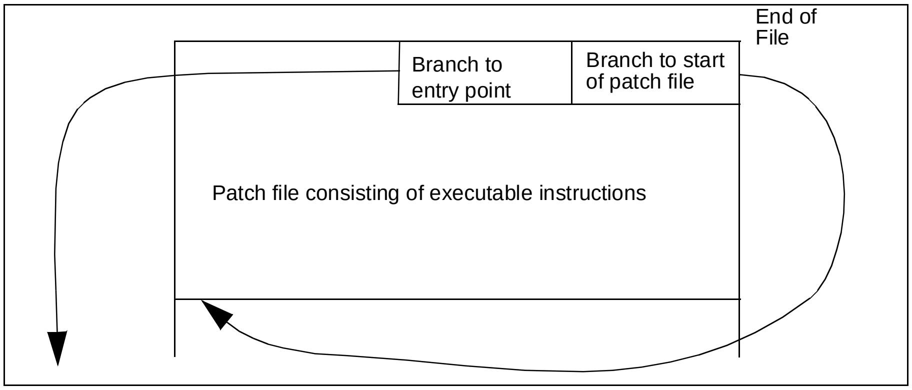


Figure 9-2. Detail of patch file placement

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

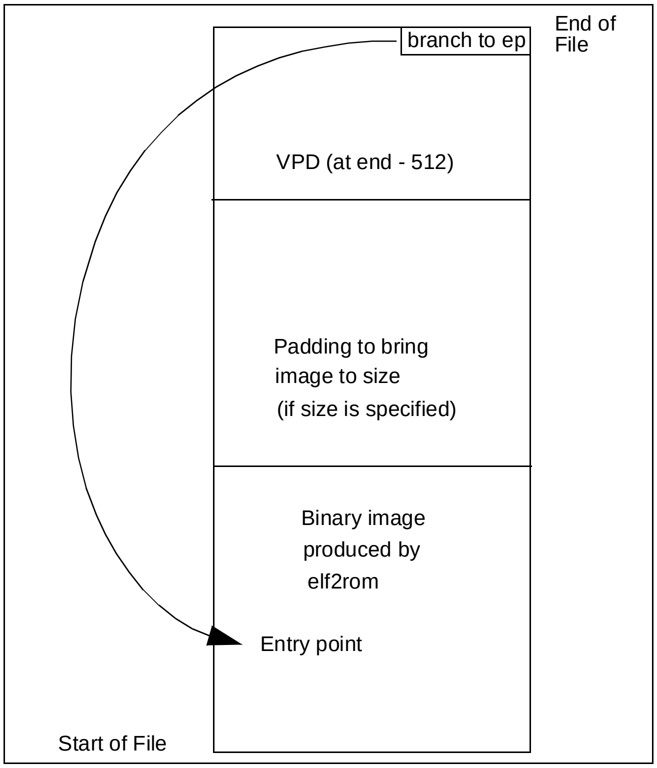


Figure 9-3. hbranch Output Image

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

9.6.3 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 **eimbgld** 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**.

9.6.4 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 eimgbld 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 **nimbgld** 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 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.

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).

10 403 EVB Function Reference

This chapter describes the OS Open functions for the 403 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 PPC403GA or PPC403GC or PPC403GCX Embedded Controller 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.

10.2 403 EVB Functions

Descriptions of the functions provided specifically to support the 403 EVB are listed in Table 10-1:

Table 10-1. Functions Specific to 403 EVB

Function or Macro	Description	Page
async_init()	Installs the asynchronous device driver	10-13
biosenet_attach()	Attaches the Ethernet to an IP address	10-14
clock_set()	Sets the OS Open POSIX clock to the value obtained from the host clock	10-16
dbg_ioLib_init()	Initializes the I/O library	10-17

Table 10-1. Functions Specific to 403 EVB

Function or Macro	Description	Page
dcache_flush()	Flushes cache lines, beginning at the effective address and continuing for a specified number of bytes	10-18
dcache_invalidate()	Invalidates cache lines beginning at the effective address and continuing for a specified number of bytes	10-19
dma_disable()	Inhibits DMA activity on a channel	10-20
dma_setup()	Initializes the DMA channel registers for subsequent DMA slave transfers	10-21
dma_status()	Returns the DMA status for the channel specified by channel	10-23
enet_disable_ipinput	Disables the forwarding of Ethernet packets to the TCP/IP protocol stack.	10-25
enet_enable_ipinput	Enables the forwarding of Ethernet packets to the TCP/IP protocol stack.	10-26
enet_native_attach	.Attaches TCP/IP protocol stack.	10-27
enet_recv_packet()	Returns a pointer to the mbuf chain holding the packet received by the Ethernet device driver.	10-29
enet_send_packet()	Transmits packet over the Ethernet.	10-30
escc_init()	Device driver for S2 communication	10-31
ext_int_config()	Comfigures interrupt type	10-32
ext_int_disable()	Disables the interrupt level specified by an event	10-33
ext_int_enable()	Enables the interrupt level specified by an event	10-34

Table 10-1. Functions Specific to 403 EVB

Function or Macro	Description	Page
ext_int_install()	Installs a first level interrupt handler (FLIH) for an event.	10-35
ext_int_query()	Returns information about the FLIH	10-37
fpemul_init()	Installs floating point interrupt handler.	10-38
ioLib_init()	Initializes I/O library	10-39
oakenet_attach()	Attaches TCP/IP protocol stack to the Ethernet device.	10-27
ppcAbend()	Executes an invalid opcode forcing a program check interrupt	10-41
ppcAndMsr()	ANDs a value with the contents of the MSR	10-42
ppcCntlzw()	Counts consecutive leading zeros in a value	10-43
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	10-44
ppcDcbi()	Invalidates a cache block, discarding any modified contents if the block is valid in cache	10-45
ppcDcbst()	Copies a cache block, discarding any modified contents if the block is valid in cache	10-46
ppcDcbz()	Sets a cache block to 0	10-47
ppcDflush()	Writes 0's into the data cache and then turns data cache off by writing 0's into the data cache cacheability register (DCCR)	10-49

Table 10-1. Functions Specific to 403 EVB

Function or Macro	Description	Page
ppcEieio()	Ensures that all storage references before the call finish before any storage references after the call start	10-50
ppcHalt()	Is a one instruction spin loop, effectively putting the processor in an enabled wait at the point of invocation	10-51
ppcIcbi()	Invalidates an instruction cache block	10-52
ppcIsync()	Causes the processor to discard any instructions that may have been prefetched	10-53
ppdMfbear()	Returns the current value of the bus error address register (BEAR)	10-54
ppcMfbesr()	Returns the current value of the bus error syndrome register (BESR)	10-55
ppcMfbr0() - ppcMfbr7()	Return the value of their respective bank registers (BR0 - BR7)	10-56
ppcMfbrh0() - ppcMfbrh7()	Return the value of their BRH registers (BRH0 - BRH7) (403GCX only)	10-58
ppcMfcdbcr()	Returns the value of the Cache Debug Control Register (CDBCR)	10-59
ppcMfdac1() - ppcMfdac2()	Returns the values of the processor debug address compare registers (DAC1 - DAC2)	10-60
ppcMfdbcr()	Returns the value of the processor debug control register (DBCR)	10-61
ppcMfdbsr()	Returns the value of the processor debug status register (DBSR)	10-62
ppcMfdccr()	Returns the value of the Data Cache Cacheability Register (DCCR)	10-63

Table 10-1. Functions Specific to 403 EVB

Function or Macro	Description	Page
ppcMfdcwr()	Returns the value of the Data Cache Write-thru Register (DCWR)	10-64
ppcMfdear()	Returns the value of the Data Exception Address Register (DEAR)	10-65
ppcMfdmacc0() - ppcMfdmacc3()	Returns the value of the DMA chained count register (DMACC0 - DMACC3)	10-66
ppcMfdmacr0() - ppcMfdmacr3()	Returns the value of the DMA channel control register (DMACR0 - DMACR3)	10-68
ppcMfdmact0() - ppcMfdmact3()	Return the value of the DMA count registers (DMACT0 - DMACT3)	10-69
ppcMfdmada0() - ppcMfdmada3()	Return the value of the DMA destination address registers (DMADA0 - DMADA3)	10-70
ppcMfdmasa0() - ppcMfdmasa3()	Return the value of the DMA source/chained address registers (DMASA0 - DMASA3)	10-71
ppcMfdmasr()	Returns the value of the DMA status register (DMASR)	10-72
ppcMfesr()	Returns the value of the exception syndrom register (ESR)	10-73
ppcMfevpr()	Returns the value of the exception vector prefix register (EVPR)	10-74
ppcMfexier()	Returns the value of the external interrupt enable register (EXIER)	10-75
ppcMfexisr()	Returns the value of the external interrupt status register (EXISR)	10-76
ppcMfgpr1()	Returns the current value of GPR(1)	10-77
ppcMfgpr2()	Returns the current value of GPR(2)	10-78

Table 10-1. Functions Specific to 403 EVB

Function or Macro	Description	Page
ppcMfiac1()	Returns the value of the instruction address compare register 1 (IAC1)	10-79
ppcMfiac2()	Returns the value of the Instruction address compare register 2 (IAC2)	10-80
ppcMficcr()	Returns the value of the instruction cache cacheability register (ICCR)	10-81
ppcMficdbdr()	Returns the value of the Instruction Cache Debug Data Register (ICDBDR)	10-82
ppcMfiocr()	Returns the current value of the Input/Output configuration Register (IOCR)	10-83
ppcMfmsr()	Returns the value of the MSR	10-84
ppcMfpbl1() - ppcMfpbl2()	Returns the value of their respective protection bound lower register (PBL)	10-85
ppcMfpbu1() - ppcMfpbu2()	Returns the value of their respective protection bound upper register (PBU)	10-86
ppcMfpid()	Returns the value of the Process ID Register (PID)	10-87
ppcMfpit()	Returns the value of the Programmable Interval Timer (PIT)	10-88
ppcMfpvr()	Returns the value of the processor version register	10-89
ppcMfsgr()	Returns the value of the Storage Guard Register (SGR)	10-90
ppcMfsprg0()-ppcMfsprg3()	Returns the value of the special purpose register generals (SPRG0 - SPRG3)	10-91
ppcMfsrr0()	Returns the value of SRR0	10-92

Table 10-1. Functions Specific to 403 EVB

Function or Macro	Description	Page
ppcMfsrr1()	Returns the current value of SRR1	10-93
ppcMfsrr2()	Returns the current value of SRR2	10-94
ppcMfsrr3()	Returns the current value of SRR3	10-95
ppcMftb()	Returns the current time base data	10-96
ppcMftcr()	Returns the value of the timer control register	10-97
ppcMftlbhi()	Returns the value of the high section of an Unified TLB (UTLB) entry	10-98
ppcMftlblo()	Returns the value of the lo section of an Unified TLB (UTLB) entry	10-99
ppcMftsr()	Returns the current value of the timer status register (TSR)	10-100
ppcMfutb()	Returns the current value of the User-Mode Time Base (UTB)	10-101
ppcMfzpr()	Returns the value of the Zone Protection Register (ZPR)	10-102
ppcMtbesr()	Sets the current value of the bus error syndrome register (BESR)	10-103
ppcMtbr0() - ppcMtbr7()	Set the specified bank registers (BR0 - BR7)	10-104
ppcMtbrh0() - ppcMtbrh7()	Set the specified BRH registers (BRH0 - BRH7)	10-106
ppcMtcdbcr()	Sets the value of the Cache Debug Control Register (CDBCR)	10-107
ppcMtdac1() - ppcMtdac2()	Sets the values of the processor debug address compare registers (DAC1 - DAC2)	10-108

Table 10-1. Functions Specific to 403 EVB

Function or Macro	Description	Page
ppcMtdbcr()	Sets the value of the debug control register (DBCR)	10-110
ppcMtdbsr()	Sets the value of the debug status register (DBSR)	10-111
ppcMtdccr()	Sets the value of the Data Cache Cache-ability Register (DCCR)	10-112
ppcMtdcwr()	Sets the value of the Data Cache Write-thru Register (DCWR)	10-113
ppcMtdmacc0() - ppcMtdmacc3()	Sets the value of the DMA chained count registers (DMACC0 - DMACC3)	10-114
ppcMtdmacr0() - ppcMtdmacr3()	Sets the value of the DMA chained control registers (DMACR0 DMACR3)	10-115
ppcMtdmact0() - ppcMtdmact3()	Sets the value of the DMA count registers (DMACT0 - DMACT3)	10-116
ppcMtdmada0() - ppcMtdmada3()	Sets the value of the DMA destination address register (DMADA0 - DMADA3)	10-117
ppcMtdmasa0() - ppcMfdmasa3()	Sets the value of the DMA source/chained address registers (DMASA0 - DMASA3)	10-118
ppcMtdmasr()	Sets the value of the DMA status register (DMASR)	10-119
ppcMtesr()	Sets the value of the exception syndrom register (ESR)	10-120
ppcMtevpr()	Sets the value of the exception vector pre-fix register (EVPR)	10-121
ppcMtexier()	Sets the value of the external interrupt enable register (EXIER)	10-122

Table 10-1. Functions Specific to 403 EVB

Function or Macro	Description	Page
ppcMtexisr()	Sets the value of the external interrupt status register (EXISR)	10-123
ppcMtiac1()	Sets the value of the instruction address compare register 1 (IAC1)	10-124
ppcMtiac2()	Sets the value of the Instruction address compare register 2 (IAC2)	10-125
ppcMticcr()	Sets the value of the instruction cache cacheability register (ICCR)	10-126
ppcMtiocr()	Sets the input/output configuration register (IOCR)	10-127
ppcMtmsr()	Sets the MSR	10-128
ppcMtpbl1() - ppcMfpbl2()	Sets the value of their respective protection bound lower register (PBL)	10-129
ppcMtpbu1() - ppcMfpbu2()	Sets the value of their respective protection bound upper register (PBU)	10-130
ppcMtpid()	Sets the value of the Process ID Register (PID)	10-131
ppcMtpit()	Sets the programmable interval timer (PIT)	10-132
ppcMtsgr()	Sets the value of the Storage Guard Register (SGR)	10-133
ppcMtsprg0() - ppcMtsprg3()	Sets the special purpose register generals (SPRG0 - SPRG3)	10-134
ppcMtsrr0()	Sets the SRR0	10-135
ppcMtsrr1()	Sets the SRR1	10-136
ppcMtsrr2()	Sets the SRR2	10-137

Table 10-1. Functions Specific to 403 EVB

Function or Macro	Description	Page
ppcMtsrr3()	Sets the SRR3	10-138
ppcMttb()	Sets the value of the current time base	10-139
ppcMttcr()	Sets the timer control register	10-140
ppcMttlbhi()	Sets the value of the high section of an Unified TLB (UTLB) entry	10-141
ppcMttlblo()	Sets the value of the low section of an Unified TLB (UTLB) entry	10-142
ppcMttsr()	Sets the timer status register	10-143
ppcMtzpr()	Sets the value of the Zone Protection Register (ZPR)	10-144
ppcOrMsr()	Performs the OR of a value and the current MSR, updating the MSR	10-145
ppcSync()	Causes the processor to wait until all data cache lines scheduled to be written to main storage have actually been written	10-146
ppcTlbia()	Invalidates all entries in the TLB	10-147
ppcTlbsx()	Searches for an effective valid address in the TLB	10-148
processor_speed()	Returns the internal clock speed of the 403 processor.	10-149
s1dbprintf()	A version of **printf()** that may be used before I/O has been established	10-150
s1dbprintfapp()	A version of **printf()** that may be used before I/O has been established from an application thread group.	10-151

Table 10-1. Functions Specific to 403 EVB

Function or Macro	Description	Page
s2dbprintf()	A version of **printf()** that may be used before I/O has been established for serial port 2	10-153
s2dbprintfapp()	A version of **printf()** that may be used before I/O has been established for serial port 2 from an application thread group.	10-154
timertick_install()	Installs and starts the timer tick handler	10-156
timertick_remove()	Removes the timer tick handler	10-157
vs1dbprintf()	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	10-158

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 theone asynchronous communication port on the 403 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- driver_install() : *OS Open Programmer's Reference*
- “Device Drivers Supplied with the 403 EVB” on page 9-6

biosenet_attach()

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 device. The IP address should be different from the IP address defined to the 403 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
Cancel Safe	No
Interrupt Handler Safe	No

Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- “Ethernet Device Driver” on page 9-16

clock_set()

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 the host clock.

Errors

[EIO]	Host Time Service not available.

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

Synopsis

#include <ioLib.h>

int dbg_ioLib_init(void);

Library

ioLib.a

Description

dbg_ioLib_init() initializes the I/O library. **dbg_ioLib_init()** is similar to **ioLib_init()** except **dbg_ioLib_init()** does not reset the EVPR register. **dbg_ioLib_init()** should be used if the benetLib.a is used.

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

ioLib_init(), p. 10-39

dcache_flush()

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- dcache_invalidate(), p. 10-19

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- dcache_flush(), p. 10-18

dma_disable()

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- dma_setup(), p. 10-21
- dma_status(), p. 10-23

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.

dmacr is formed from the appropriate combination of DMA control register bits from the file **<ppcLib.h>**. For example, to specify a 32 bit memory to memory DMA transfer, specify DMACR as (DMACR_CE | DMACR_PW_32 | DMACR_DAI |DMACR_TCE). The value in DMACR is loaded into the requested DMA channel's control register to enable a DMA transfer. For a description of the DMA control register bits, see the *PPC403GA or PPC403GC Embedded Controller User's Manuals*.

count is the requested number of DMA transfers. Valid values range from 1 to 65536 (inclusive). This value is loaded into the DMACT register of the appropriate channel.

dst_address is the absolute memory address for the buffered or fly-by mode DMA transfers. This memory address will be either the source or the destination of the next DMA transfer depending on the value of the transfer direction bit in DMACR. For memory to memory mode transfers, *dst_adress* is the absolute destination memory address.

src_address is the absolute source memory address for memory to memory mode transfers. If chaining is enabled, *src_address* is the absolute memory address for the next (chained) transaction. *src_address* is only used if DMACR indicates that the requested DMA is either a memory to memory transaction or chaining is enabled. Otherwise it is ignored.

chained_count is the requested number of DMA transfers for the next (chained) transaction. *chained_count* is only used if DMACR indicates that chaining is enabled. Otherwise it is ignored.

- **Note:** Chaining is valid for DMA channels 1 thru 3 for only 403GA processors with module markings of PPC 403GA-JB.... and beyond and all 403GC processors, **See "ppcMfdmacc0() - ppcMfdmacc3(),' p. 66**.

Note: Although **dma_setup()** is not async safe in general, it can be safely called by concurrently executing threads as long each thread specifies a unique DMA channel.

Errors

[EINVAL]	*channel* does not refer to a valid DMA channel. *type* is not a defined type. *count* exceeds 65535. count is 0 or 65536. DMACR specifies both memory to memory and chaining. Chaining is not supported on the specified channel. DMACR specifies chaining and chained_count is 0 or exceeded 65536.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- dma_disable(), p. 10-20
- dma_status(), p. 10-23

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.*

dstat->current_address contains the current contents of the DMADA register for the specified channel which is the address of the next memory access.

dstat->current_count contains the current contents of the DMACT register for the specified channel which is the number of transfers remaining in the current DMA transaction. *dstat->current_count* will be 0 when a DMA transaction completes. **Note:** *dstat->current_count* will also be 0 until the first DMA transfer occurs on a DMA transaction with 65536 transfers.

d*stat->count_status*, *dstat->transfer_status*, *dstat->error_status*, *dstat->chained_status*, *dstat->internal_req*, *dstat->external_req*, and *dstat->busy* reflect the current values of the status bits in the DMA status register (DMASR) for the specified channel. For a description of the DMASR, see the *PPC403GA and PPC403GC Embedded Controller User's Manuals.*

If successful **dma_status()** returns 0. Otherwise -1.

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- dma_setup(), p. 10-21
- dma_disable(), p. 10-20

enet_disable_ipinput()

Synopsis

#include <enetLib.h>

void enet_disable_ipinput(void);

Library

enetLib.a

Description

enet_disable_ipinput() disables the forwarding of packets to the TCP/IP protocol stack. When forwarding is disabled Ethernet packets received by the Ethernet device driver can be read by the application using **enet_recv_packet()**.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- enet_native_attach(), p. 10-27
- enet_enable_ipinput(), p. 10-26
- enet_send_packet(), p. 10-30
- enet_recv_packet(), p. 10-29

enet_enable_ipinput()

Synopsis

#include <enetLib.h>

void enet_enable_ipinput(void);

Library

enetLib.a

Description

enet_enable_ipinput() enables the forwarding of packets to the TCP/IP protocol stack. When forwarding is enabled all Ethernet packets received by the Ethernet device driver are forwarded to the TCP/IP stack.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- enet_native_attach(), p. 10-27
- enet_disable_ipinput(), p. 10-25
- enet_send_packet(), p. 10-30
- enet_recv_packet(), p. 10-29

Synopsis

#include <enetLib.h>

int enet_native_attach(unsigned long processor_speed, unsigned long sram_size);

Library

enetLib.a

Description

enet_native_attach() attaches the TCP/IP protocol stack to the Ethernet device. The *processor_speed* specifies the CPU speed. The *sram_size* specifies the Ethernet controller's memory size. On the 403 EVB the *sram_size* parameter should be set to 8192. **enet_native_attach()** returns 0 if successful and -1 if it is unsuccessful.

Errors

None.

Example

The following is an example of initializing the TCP/IP protocol stack and attaching the Ethernet device.

```
#include <enet.h>
#define ENETHOST "403_board"
#define ENET_CONFIG " ent0 403_board netmask 255.255.240.0 up"
int do_enet()
{

int rc;
rc = tcpip_init( ENETHOST, 1, 1000); /* Initialize the TCP/IP library */
if(rc != 0)
return -1;

rc = net_init();  /*initialize netLib */
if(rc != 0)
return -1;

rc = enet_native_attach(25000000,8 * 1024); /* attach the tcp/ip proto.
stack */
if(rc != 0)
return -1;

rc = ifconfig(ENET_CONFIG); /* configure network interface */
if(rc != 0)
return -1;
return 0;
}
```

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	No
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- enet_disable_ipinput(), p. 10-25
- enet_enable_ipinput(), p. 10-26
- enet_send_packet(), p. 10-30
- enet_recv_packet(), p. 10-29

Synopsis

#include <enetLib.h>

struct mbuf *enet_recv_packet(struct timespec *timeout);

Library

enetLib.a

Description

enet_recv_packet() returns a pointer to the mbuf chain holding the packet received by the Ethernet device driver. **enet_recv_packet()** will block waiting for packet reception for the maximum of time specified by *timeout*, If successful **enet_recv_packet()** returns a pointer to the mbuf chain containing the Ethernet packet, otherwise NULL is returned.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	No
Interrupt Handler Safe	No
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- enet_native_attach(), p. 10-27
- enet_disable_ipinput(), p. 10-25
- enet_enable_ipinput(), p. 10-26
- enet_send_packet(), p. 10-30

enet_send_packet()

Synopsis

#include <enetLib.h>

int enet_send_packet(struct ether_header *eh, struct mbuf *m, int total);

Library

enetLib.a

Description

enet_send_packet() transmits the packet described by *eh* and *m*. The Ethernet packet header is specified in *eh*. The destination, source hardware address, and packet type must be set in the *eh* structure prior to calling **enet_send_packet()**. *m* points to the mbuf chain that contains the actual packet. *total* must be set to the number of bytes to be transmitted. The value of *total* is set to the size of the Ethernet header plus the size of the packet contained in mbuf. If successful **enet_send_packet()** returns 0 otherwise -1 is returned. **enet_send_packet** will timeout after 3 seconds.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- enet_native_attach(), p. 10-27
- enet_disable_ipinput(), p. 10-25
- enet_enable_ipinput(), p. 10-26
- enet_recv_packet(), p. 10-29

Synopsis

#include <esccLib.h>

int driver_install(int *devhndle, escc_init);

Library

esccLib.a

Description

esccLib.a is the device driver for the S2 asynchronous communication port found in the standard I/O subsystem on the 403 EVB platform. **esccLib.a** is installed by passing **escc_init** to **driver_install()** as the second parameter. The first parameter is the device handle. For more information about **driver_install()** and **escc_init** refer to “Chapter 6: Device Drivers for 403 EVB.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	No
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- “Extended Serial Communication Controller Device Driver” on page 9-11

ext_int_config()

Synopsis

#include <ioLib.h>

int ext_int_config(int event, int flags);

Library

ioLib.a

Description

ext_int_config() configures the interrupt type in the Input/Output Configuration register. ext_int_config() can be used to determine the type of external interrupt that will be processed. File **<ioLib.h>** contains several defines for the external interrupt flags for setting polarity sensitivity.

File **<flih.h>** contains several defines for the external interrupt flags.

The **ext_int_config()** function returns 0 when it is successful.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ext_int_enable(), p. 10-34
- ext_int_install(), p. 10-35
- ext_int_query(), p. 10-37
- ioLib_init(), p. 10-39

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.* DMA interrupt 2The **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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ext_int_enable(), p. 10-34
- ext_int_install(), p. 10-35
- ext_int_query(), p. 10-37
- ioLib_init(), p. 10-39

ext_int_enable()

Synopsis

#include <ioLib.h>

void ext_int_enable(int event);

Library

ioLib.a

Description

ext_int_enable() enables the interrupt level specified by *event.*

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ext_int_config(), p. 10-32
- ext_int_install(), p. 10-35
- ext_int_query(), p. 10-37
- ioLib_init(), p. 10-39

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*.

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:

flih_stack	Pointer to the first stack location; obtained by allocating memory and adding the size of the stack. flih_stack must be 16 byte aligned.
flih_function	Pointer to a function invoked when *event* occurs.
arg	A user-defined (void *) value passed to *flih_function*.

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]	*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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ext_int_config(), p. 10-32
- ext_int_enable(), p. 10-34
- ext_int_query(), p. 10-37
- ioLib_init(), p. 10-39

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*.

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 *event* occurs.
arg	A user-defined (void *) value passed to *flih_function*. 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ext_int_config(), p. 10-32
- ext_int_enable(), p. 10-34
- ext_int_install(), p. 10-35
- ioLib_init(), p. 10-39

fpemul_init()

Synopsis

#include <fpeLib.h>

void fpemul_init(void);

Library

fpCSLib.a

Description

fpemul_init() installs the floating point interrupt handler. **fpemul_init()** is only needed when floating point emulation is required with the XCOFF version of OS Open for PowerPC processors without floating point hardware. **fpemul_init()** is not required for floating point when running with the ELF version of OS Open. **fpemul_init()** returns nothing.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

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 403 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

oakenet_attach()

Synopsis

#include <enetLib.h>

int oakenet_attach(unsigned long processor_speed, unsigned long sram_size);

Library

enetLib.a

Description

oakenet_attach() attaches the TCP/IP protocal stack to the Ethernet device. The *processor_speed* specifies the CPU speed. The *sram_size* specifies the Ethernet controller's memory size. On the 403 EVB the *sram_size* parameter should be set to 8192. **oakenet_attach()** returns 0 if successful and -1 if it is unsuccessful.

Errors

None.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	No
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- enet_disable_ipinput(), p. 10-25
- enet_enable_ipinput(), p. 10-26
- enet_send_packet(), p. 10-30
- enet_recv_packet(), p. 10-29

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ppcOrMsr(), p. 10-145
- ppcMtmsr(), p. 10-128
- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ppcDcbst(), p. 10-46
- ppcDcbi(), p. 10-45
- ppcDcbz(), p. 10-47
- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ppcDcbst(), p. 10-46
- ppcDcbi(), p. 10-45
- ppcDcbz(), p. 10-47
- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ppcDcbf(), p. 10-44
- ppcDcbi(), p. 10-45
- ppcDcbz(), p. 10-47
- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ppcDcbf(), p. 10-44
- ppcDcbi(), p. 10-45
- ppcDcbst(), p. 10-46
- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

```
#include <ppcLib.h>
void ppcDflush(void);
```

Library

ppcLib.a

Description

ppcDflush() will write 0's into the data cache and then turn data cache off by writing 0's into the Data Cache Cacheability Register (DCCR).

Errors

None.

Example

- Force data reads from memory instead of from the data cache:

```
ppcDflush();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcEieio()

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.

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcIcbi()

Synopsis

#include <ppcLib.h>

void ppcIcbi(void *addr);

Library

ppcLib.a

Description

ppcIcbi() 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

Synopsis

#include <ppcLib.h>

void ppcIsync(void);

Library

ppcLib.a

Description

ppcIsync() causes the processor to discard any instructions that may have been prefetched before **ppcIsync()**. This call must be used after modifying instruction storage.

Errors

None.

Example

- Place a trap into a given address:

```
*trap_address = 0x7F000008;
 ppcIsync();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfbear()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfbear(void)

Library

ppcLib.a

Description

ppcMfbear() returns the current value of the Bus Error Address Register.

Errors

None.

Example

- After a machine check, retrieve the BEAR:

```
bear = ppcMfbear();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *ppcMfbesr(), p. 10-55*
- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfbesr(void);

Library

ppcLib.a

Description

ppcMfbesr() returns the current value of the Bus Error Syndrome Register, which identifies the nature of a bus error detected by the processor.

The file **<ppcLib.h>** defines constants for use with the BESR.

Errors

None.

Example

- Retrieve bus error syndrome information.

```
besr = ppcMfbesr()
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ppcMfbear(), p. 10-54
- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfbr0() • ppcMfbr7()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfbr0(void);

unsigned long ppcMfbr1(void);

unsigned long ppcMfbr2(void);

unsigned long ppcMfbr3(void);

unsigned long ppcMfbr4(void);

unsigned long ppcMfbr5(void);

unsigned long ppcMfbr6(void);

unsigned long ppcMfbr7(void);

Library

ppcLib.a

Description

ppcMfbr0() - ppcMfbr7() return the value of their respective bank registers (BR0 - BR7). Four bank registers (BR0 through BR3) control SRAM devices only; four bank registers (BR4 through BR7) control SRAM or DRAM devices. The file **<ppcLib.h>** has several constants defined for use with the BR registers with both the 403GA and 403GC processors.

Errors

None.

Example

- Retrieve the value of BR4. A device driver may use the BR4 value to determine if it is accessing a SRAM or DRAM device. Bit 31 would be 0 for a DRAM device and 1 for a SRAM device.

```
unsigned long current_br4=ppcMfbr4();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfbrh0() • ppcMfbrh7()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfbrh0(void);

unsigned long ppcMfbrh1(void);

unsigned long ppcMfbrh2(void);

unsigned long ppcMfbrh3(void);

unsigned long ppcMfbrh4(void);

unsigned long ppcMfbrh5(void);

unsigned long ppcMfbrh6(void);

unsigned long ppcMfbrh7(void);

Library

ppcLib.a

Description

ppcMfbrh0() - ppcMfbrh7() return the value of their respective BRH register(BRH0 - BRH7).

Errors

None.

Example

- Retrieve the value of BRH4.

```
unsigned long current_brh4=ppcMfbrh4();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	No
PowerPC 403GC	No
PowerPC 403GCX	Yes

References

- *PPC403GCX Embedded Controller User’s Manual*

ppcMfcdbcr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfcdbcr(void);

Library

ppcLib.a

Description

ppcMfcdbcr() returns the value of the Cache Debug Control Register (CDBCR).

<ppcLib.h> has constants defined for use with the CDBCR register.

Errors

None.

Example

- Retrieve the current value of the CDBCR:

```
unsigned long cdbcr_value=ppcMfcdbcr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMfdac1() • ppcMfdac2()

Synopsis

#include <ppcLib.h>

void ppcMfdac1(unsigned long dac1_value);

void ppcMfdac1(unsigned long dac1_value);

Library

ppcLib.a

Description

ppMfdac1() - ppcMfdac2() return the value of the appropriate Data Address Compare register. the DAC1 and DAC2 registers contain addresses for which debug events may be taken, depending on the values set in the DBCR.

Errors

None.

Example

- Get the value of DAC1.

```
unsigned long dac1_value = ppcMfdac1();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfdbcr(void);
```

Library

ppcLib.a

Description

ppcMfdbcr() returns the value of the processor debug control register (DBCR). The DBCR is used to enable debug events, reset the processor, control timer operations during debug events, and set the debug mode of the processor.

WARNING: Enabling bits 0 and 1 can cause unexpected results. Enabling bits 2 and 3 will cause a processor reset to occur. The DBCR is designed to be used by development tools, not applications.

Refer to the **<ppcLib.h>** for defined constants for the DBCR.

Errors

None.

Example

- Retrieve the value of DBCR register. A debugger would require the value of the DBCR:

```
unsigned long current_DBCR=ppcMfdbcr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfdbsr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfdbsr(void);

Library

ppcLib.a

Description

ppcMfdbsr() returns the value of the processor debug status register (DBSR). The DBSR contains the status of debug events, the JTAG serial buffers, and the most recent reset.

The file **<ppcLib.h>** defines constants that can be used when referring to the DBSR.

Errors

None.

Example

- Retrieve the value of DBSR register. A debugger would require the value of the DBSR:

```
unsigned long current_DBSR=ppcMfdbsr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMfdccr()

Synopsis

```
#include <ppcLib.h>
void ppcMfdccr(unsigned long dccr_value);
```

Library

ppcLib.a

Description

ppcMfdccr() returns the value of Data Cache Cacheability Register (DCCR).

Errors

None.

Example

- Set the value of the DCCR:

```
#include<ppcLib.h>
unsigned long dccr_value=ppcMfdccr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMfdcwr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfdcwr(void);

Library

ppcLib.a

Description

ppcMfdcwr() returns the value of the Data Cache Write-thru Register (DCWR).

Errors

None.

Example

- Retrieve the current value of the DCWR:

```
unsigned long dcwr_value=ppcMfdcwr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMfdear(unsigned long dear_value);

Library

ppcLib.a

Description

ppcMfdear() returns the value of Data Exception Address Register (DEAR).

Errors

None.

Example

- Set the value of the DEAR:

```
#include<ppcLib.h>
unsigned long dear_value=ppcMfdear();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMfdmacc0() • ppcMfdmacc3()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfdmacc0(void);

unsigned long ppcMfdmacc1(void);

unsigned long ppcMfdmacc2(void);

unsigned long ppcMfdmacc3(void);

Library

ppcLib.a

Description

ppcMfdmacc0() - ppcMfdmacc3() returns the value of the corresponding DMA chained count register (DMACC0 -DMACC3). When chaining is enabled for the corresponding channel, the corresponding DMACC contains the number of transfers in the next DMA transaction for the corresponding channel 0. When chaining is disabled for the corresponding channel, the corresponding DMACC is not used. Bits 16 to 31 contain the count value.

Note: ppcMfdmacc1() thru **ppcMfdmacc3()** are only valid for 403GA processors with module markings of PPC403GA-JB.... and beyond and all 403GC processors.

Errors

None.

Example

- Retrieve the current value of the DMACC0:

  unsigned long dmacc0_value=ppcMfdmacc0();

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfdmacr0() • ppcMfdmacr3()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfdmacr0(void);

unsigned long ppcMfdmacr1(void);

unsigned long ppcMfdmacr2(void);

unsigned long ppcMfdmacr3(void);

Library

ppcLib.a

Description

ppcMfdmacr0() - ppcMfdmacr3() return the value of the DMA channel control registers (DMACR0 - DMACR3). The DMACRs set up and enables the DMA channels. The file **<ppcLib.h>** contains several constants that can be used when accessing the DMACR's.

Errors

None.

Example

- Retrieve the current value of the DMACR0:

```
unsigned long dmacr0_value=ppcMfdmacr0();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfdmact0(void);
unsigned long ppcMfdmact1(void);
unsigned long ppcMfdmact2(void);
unsigned long ppcMfdmact3(void);
```

Library

ppcLib.a

Description

ppcMfdmact0() - ppcMfdmact3() return the value of the DMA count registers (DMACT0 - DMACT3). The DMACT registers contains the number of transfers left in the DMA transaction for the channel.

Errors

None.

Example

- Retrieve the current value of the DMACT0:

```
unsigned long dmact0_value=ppcMfdmact0();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfdmada0() • ppcMfdmada3()

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfdmada0(void);
unsigned long ppcMfdmada1(void);
unsigned long ppcMfdmada2(void);
unsigned long ppcMfdmada3(void);
```

Library

ppcLib.a

Description

ppcMfdmada0() - ppcMfdmada3() return the value of the DMA destination address registers (DMADA0 - DMADA3)). The DMADA registers contain the memory addresses for transfers between memory and peripheral or the destination addresses for memory to memory transfers.

Errors

None.

Example

- Retrieve an address from DMADA3:

```
unsigned long dmada3_value = ppcMfdmada3();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfdmasa0(void);

unsigned long ppcMfdmasa1(void);

unsigned long ppcMfdmasa2(void);

unsigned long ppcMfdmasa3(void);

Library

ppcLib.a

Description

ppcMfdmasa0() - ppcMfdmasa3() return the value of the DMA source/chained address registers (DMASA0 - DMASA3). The DMASAs are only used in memory to memory move mode for channels 0 through channels 3 or in fly-by mode when chaining has been enabled for channel 0

Errors

None.

Example

- Retrieve the current value of the DMASA0:

```
unsigned long dmasa0_value=ppcMfdmasa0();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfdmasr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfdmasr(void);

Library

ppcLib.a

Description

ppcMfdmasr() returns the value of the DMA status register (DMASR).

The value of the DMASR may be used to determine the status of the DMA channels. The file **<ppcLib.h>** contains several constants that may be used when accessing the DMASR .

TErrors

None.

Example

- Retrieve the current value of the DMASR:

```
unsigned long dmasr_value=ppcMfdmasr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfesr(void);

Library

ppcLib.a

Description

ppcMfesr() returns the value of the Exception Syndrome Register (ESR). Bits 7 to 31 are reserved.

Errors

None.

Example

- Get the ESR value:

```
esr_value= ppcMfesr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfevpr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfevpr(void);

Library

ppcLib.a

Description

ppcMfevpr() returns the value of the exception vector prefix register (EVPR). Bits 0 to 15 contain the prefix of the address of the exception processing routines. Bits 15 to 31 are reserved.

Errors

None.

Example

- Get the EVPR value:

```
evpr_value= ppcMfevpr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfexier(void);

Library

ppcLib.a

Description

ppcMfexier() returns the value of the external interrupt enable register (EXIER).

This register contains enables for six of the external hardware interrupts, the DMA channel interrupts, the JTAG serial port interrupts and the serial port interrupts. The file <**ppcLib.h>** contains several constants that can be used when accessing the EXIER.

Errors

None.

Example

- Retrieve the current value of the EXIER:

```
unsigned long exier_value=ppcMfexier();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfexisr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfexisr(void);

Library

ppcLib.a

Description

ppcMfexisr() returns the value of the external interrupt status register (EXISR). The EXISR contains the status of the five external hardware interrupts, the DMA channel interrupts, the JTAG serial port interrupts and the serial port interrupts.The file <**ppcLib.h>** contains several constants that can be used when accessing the EXISR.

Errors

None.

Example

- Read the value of the EXISR:

```
unsigned long exisr_value=ppcMfexisr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller 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. 10-78.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

Synopsis

#include <ppcLib.h>

unsigned long iac1_value = ppcMfiac1(void);

Library

ppcLib.a

Description

ppcMfiac1() returns the value of the instruction address compare register 1 (IAC1). The IAC1 contains the address of the instruction that the debug event will be based on. The IA1 field of the Debug Control Register (DBCR) controls the instruction address 1 debug event. Bits 30 and 31 of the IAC1 are reserved, since the address must be word aligned.

Errors

None.

Example

- Get the IAC1 register value:

```
iac1_value =ppcMfiac1();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMfiac2()

Synopsis

#include <ppcLib.h>

unsigned long iac2_value = ppcMfiac2(void);

Library

ppcLib.a

Description

ppcMfiac2() returns the value of the instruction address compare register 2 (IAC2). The IAC2 contains the address of the instruction that the debug event will be based on. The IA2 field of the Debug Control Register (DBCR) controls the instruction address 2 debug event. Bits 30 and 31 of the IAC2 are reserved, since the address must be word aligned.

Errors

None.

Example

- Get the IAC2 register value:

```
iac1_value =ppcMfiac2();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMticcr(void);

Library

ppcLib.a

Description

ppcMticcr() returns the value of the Instruction Cache Cacheability Register (ICCR).

Errors

None.

Example

- Get the ICCR value:

```
unsigned long iccr_value=ppcMficcr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMficdbdr()

Synopsis

#include <ppcLib.h>

unsigned long = ppcMficdbdr(void);

Library

ppcLib.a

Description

ppcMficdbdr() returns the current value of the Instruction Cache Debug Data Register (ICDBDR).

<ppcLib.h> has constants defined for use with the ICDBDR register.

Errors

None.

Example

- Retrieve the value of the ICDBDR:

```
unsigned long current_icdbdr = ppcMficdbdr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMiocr(void)

Library

ppcLib.a

Description

ppcMfiocr() returns the current value of the Input/Output Configuration Register (IOCR). The file **<ppcLib.h>** contains several constants that can be used when accessing the IOCR.

Errors

None.

Example

- Retrieve IOCR value:

```
unsigned long iocr_value;
iocr_value=ppcMfiocr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfmsr()

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. 10-128.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfpbl1(void);

unsigned long ppcMfpbl2(void);

Library

ppcLib.a

Description

ppcMfpbl1() and **ppcMfpbl2()** return the values of their respective Protection Bound Lower Register (PBL).

Errors

None.

Example

- Get the current value of PBL2:

```
unsigned long pbl2_value= ppcMfpbl2();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfpbu1() • ppcMfpbu2()

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfpbu1(void);
unsigned long ppcMfpbu2(void);
```

Library

ppcLib.a

Description

ppcMfpbu1() and **ppcMfpbu2()** return the values of their respective Protection Bound Upper Register (PBU).

Errors

None.

Example

- Get the current value of PBU2:

```
unsigned long pbu2_value= ppcMfpbu2();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfpid(void);

Library

ppcLib.a

Description

ppcMfpid() returns the current value of the Process ID register (PID).

Errors

None.

Example

- Retrieve the current value of the PID.

```
#include <ppcLib.h>
unsigned long pid_value = ppcMfpid();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfpit()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfpit(void);

Library

ppcLib.a

Description

ppcMfpit() returns the value of the Programmable Interval Timer (PIT).

Errors

None.

Example

- Get the current PIT value:

```
unsigned long pit_value= ppcMfpit();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfsgr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfsgr(void);

Library

ppcLib.a

Description

ppcMfsgr() returns the value of the Storage Guarded Register (SGR).

Errors

None.

Example

- Retrieve the current value of the SGR.

```
unsigned long current_sgr=ppcMfsgr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC401GF Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfsprg0() - ppcMfsprg3()

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMfsrr0()

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.SRR0 and SRR1 are set for protection, external, alignment, program, PIT, FIT, and syscall interrupts.

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ppcMfsrr1(), p. 10-93
- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

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. The file**<ppcLib.h>** contains several constants that can be used when setting the MSR values in the SRR1 register.

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMfsrr2()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfsrr2(void);

Library

ppcLib.a

Description

ppcMfsrr2() returns the current value of SRR2.

Typically, SRR2 is used in interrupt handlers, as it contains the address of the next instruction which was to be executed next at the time the exception occurred. SRR2 and SRR3 are set for critical, machine check, watchdog, and debug interrupts.

Errors

None.

Example

- Retrieve the current value of SRR2. This register contains the address of the instruction that was to be executed next, which may be needed by an exception handler.

```
unsigned long current_srr2=ppcMfsrr2();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *ppcMfsrr3(), p. 10-95*
- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfsrr3(void);

Library

ppcLib.a

Description

ppcMfsrr3() returns the current value of SRR3.

Typically, SRR3 is used in the critical interrupt handler, as it contains the old MSR value as well as information bits specific to the interrupt. The file **<ppcLib.h>** contains several constants that can be used when setting the MSR values in the SRR3 register.

Errors

None.

Example

- Retrieve the current value of SRR3. This register contains the saved MSR, which may be needed by an exception handler.

```
unsigned long current_srr3=ppcMfsrr3();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMftb()

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMftcr(void);

Library

ppcLib.a

Description

ppcMftcr() returns the value of the Timer Control Register.

File **<ppcLIb.h>** defines several constants for the TCR that can be used as masks. :

Errors

None.

Example

- Retrieve the current value of TCR register:

```
unsigned long tcr_value;
tcr_value = ppcMftcr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMftlbhi()

Synopsis

#include <ppcLib.h>

unsigned long ppcMftlbhi(unsigned long index);

Library

ppcLib.a

Description

ppcMftlbhi() returns the value of the high order bits of the Unified TLB (UTLB) entry specified by the *index* parameter. The TLBHI contains the tag portion of the UTLB. The TID[0:7] value is placed into the PID. A ppcMfpid() may be used to acquire the TID value. **ppcMftlbhi**() returned value contains the following data format:

Errors

None.

Example

- Retrieve the current TLBHI value for entry 1.

```
unsigned long current_tlbhi_1=ppcMftlbhi(1);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMftlblo(unsigned long index);

Library

ppcLib.a

Description

ppcMftlblo() returns the value of the low order bits of the Unified TLB (UTLB) entry specified by the *index* parameter. The TLBLO contains the data entry portion of the UTLB. **ppcMftlblo**() returned value contains the following data format:

Errors

None.

Example

- Retrieve the current TLBLO value for entry 1.

```
unsigned long current_tlblo_1=ppcMtlblo(1);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMftsr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMftsr(void);

Library

ppcLib.a

Description

ppcMftsr(**) returns the current value of the Timer Status Register (TSR). The file **<ppcLib.h> contains several defined constants for the TSR that can be used as masks.

Errors

None.

Example

- Retrieve the current value of the TSR:

```
unsigned long tsr_value;
tsr_value = ppcMftsr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMfutb(tb_t *clock_data);

Library

ppcLib.a

Description

ppcMfutb() returns the current value of the User-Mode Time Base(UTB). The value of the UTB will be placed into the tb_t structure *clock_data*.

Errors

None.

Example

- Retrieve the current value of the UTB.

```
#include <ppcLib.h>
tb_t clock_data;
tb_t * tb_ptr;
tb_ptr=&clock_data;
void ppcMfutb(tb_ptr);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMfzpr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfzpr(void);

Library

ppcLib.a

Description

ppcMfzpr() returns the current value of the ZPR.

Errors

None.

Example

- Set the ZP bits for zone 0 to allow all access if valid.

```
#include <ppcLib.h>
unsigned long zpr_value;
zpr_value=ppcMfzpr();
ppcMtzpr(zpr_value | 0xc0000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtbesr(unsigned long besr_value);

Library

ppcLib.a

Description

ppcMtbesr() sets the value of the Bus Error Syndrome Register, which identifies the nature of a bus error detected by the processor.

The file **<ppcLib.h>** defines constants for use with the BESR.

Errors

None.

Example

- Clear bus error syndrome information.

```
ppcMfbesr(0x0);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- ppcMfbear(), p. 10-54
- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtbr0() • ppcMtbr7()

Synopsis

#include <ppcLib.h>

void ppcMtbr0(unsigned long br0_value);

void ppcMtbr1(unsigned long br1_value);

void ppcMtbr2(unsigned long br2_value);

void ppcMtbr3(unsigned long br3_value);

void ppcMtbr4(unsigned long br4_value);

void ppcMtbr5(unsigned long br5_value);

void ppcMtbr6(unsigned long br6_value);

void ppcMtbr7(unsigned long br7_value);

Library

ppcLib.a

Description

ppcMtbr0() - ppcMtbr7() set the respective bank registers with the specified value. Four bank registers (BR0 through BR3) control SRAM devices only; four bank registers (BR4 through BR7) control SRAM or DRAM devices. The file **<ppcLIb.h>** contains several constants that can be used when modifying the BR registers.:

Errors

None.

Example

- Set the BR4 with a base address of 0x0 a bank size of 4 MB, for read only, and a 16 bit bus width:

```
ppcMtbr4(BR_SIZE_4M | BR_BU_RO | BR_BW_16);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtbrh0() • ppcMtbrh7()

Synopsis

```
#include <ppcLib.h>
void ppcMtbrh0(unsigned long brh0_value);
void ppcMtbrh1(unsigned long brh1_value);
void ppcMtbrh2(unsigned long brh2_value);
void ppcMtbrh3(unsigned long brh3_value);
void ppcMtbrh4(unsigned long brh4_value);
void ppcMtbrh5(unsigned long brh5_value);
void ppcMtbrh6(unsigned long brh6_value);
void ppcMtbrh7(unsigned long brh7_value);
```

Library

ppcLib.a

Description

ppcMtbrh0() - ppcMtbrh7() set the respective BRH (BRH0 - BRH7) with the specified value.

Errors

None.

Example

- Set the BRH4 to 0x80000000:

```
ppcMtbrh4(0x80000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	No
PowerPC 403GCX	Yes

References

- *PPC403GCX Embedded Controller User's Manual*

ppcMtcdbcr()

Synopsis

```
#include <ppcLib.h>
void ppcMtcdbcr(unsigned long cdbcr_value);
```

Library

ppcLib.a

Description

ppcMtcdbcr() sets the CDBCR to the specified value.

<ppcLib.h> has constants defined for use with the Cache Debug Control Register (CDBCR) registers.

Errors

None.

Example

- Set value of the CDBCR:

```
#include<ppcLib.h>
ppcMtcdbcr(CDBCR_CIS);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMtdac1()

Synopsis

#include <ppcLib.h>

void ppcMfdac1(unsigned long dac1_value);

Library

ppcLib.a

Description

ppMfdac1() sets the value of the appropriate Data Address Compare register. the DAC1 register contains addresses for which debug events may be taken, depending on the values set in the DBCR.

Errors

None.

Example

- Set the value of DAC1to address 0x0.

```
ppcMfdac1(0x0);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMfdac2(unsigned long dac1_value);

Library

ppcLib.a

Description

ppMfdac2() sets the value of the appropriate Data Address Compare register. the DAC2 register contains addresses for which debug events may be taken, depending on the values set in the DBCR.

Errors

None.

Example

- Set the value of DAC2 to address 0x0.

```
ppcMfdac2(0x0);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtdbcr()

Synopsis

#include <ppcLib.h>

void ppcMtdbcr(unsigned long dbcr_value);

Library

ppcLib.a

Description

ppcMtdbcr() sets the value of the debug control register (DBCR) to the specified value. The DBCR is used to enable debug events, reset the processor, control timer operations during debug events, and set the debug mode of the processor.

WARNING: Enabling bits 0 and 1 can cause unexpected results. Enabling bits 2 and 3 will cause a processor reset to occur. The DBCR is designed to be used by development tools, not applications.

File **<ppcLib.h>** has several defined constants for the DBCR.

Errors

None.

Example

- Enable external debug mode:

```
ppcMtdbcr(DBCR_EDM);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtdbsr(unsigned long dbsr_value);

Library

ppcLib.a

Description

ppcMtdbsr() sets the value of the debug status register (DBSR) to the specified value. The DBSR contains the status of debug events, the JTAG serial buffers, and the most recent reset. Bits in the DBSR are cleared by writing a 1 to the corresponding bit position.

WARNING: The DBSR is designed to be used by development tools, not application software. It is strongly recommended that this register be treated as a read only register.

The file **<ppcLib.h>** defines constant values that can be used when setting DBSR

Errors

None.

Example

- Set the system reset bits:

```
ppcMtdbsr(DBSR_MRR_SYS);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtdccr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMtdccr(void);

Library

ppcLib.a

Description

ppcMtdccr() sets the value of Data Cache Cacheability Register (DCCR).

Errors

None.

Example

- Set the value of the DCCR so all regions are cacheable:

```
#include<ppcLib.h>
ppcMfdccr(0xffffffff);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

```
#include <ppcLib.h>
void ppcMtdcwr(unsigned long dcwr_value);
```

Library

ppcLib.a

Description

ppcMtdcwr() sets the Data Cache Write-thru Register (DCWR) to the specified value.

Errors

None.

Example

- Set the value of the DCWR:

```
#include<ppcLib.h>
ppcMtdcwr(0x80000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMtdmacc0() • ppcMtdmacc3()

Synopsis

#include <ppcLib.h>

void ppcMtdmacc0(unsigned long dmacc0_value);

void ppcMtdmacc1(unsigned long dmacc1_value);

void ppcMtdmacc2(unsigned long dmacc2_value);

void ppcMtdmacc3(unsigned long dmacc3_value);

Library

ppcLib.a

Description

ppcMtdmacc0() sets the value of the DMA chained count register (DMACC0 - DMACC3). The count value is in bits 16 to 31. Bits 0 to 15 are reserved. The count value is the number of transfer in the next DMA transaction for the corresponding channel. When chaining is disabled for the corresponding channel the DMACC register is not used.

Note: ppcMtdmacc1() thru **ppcMtdmacc3()** are only valid for 403GA processors with module markings of PPC 403GA-JB.... and beyond and all 403GC processors.

Errors

None.

Example

- Set the chain count for the next DMA transaction to 0x00000000:

```
ppcMtdmacc0(0x00000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

Synopsis

#include <ppcLib.h>

void ppcMtdmacr0(unsigned long dmacr0_value);

void ppcMtdmacr1(unsigned long dmacr1_value);

void ppcMtdmacr2(unsigned long dmacr2_value);

void ppcMtdmacr3(unsigned long dmacr3_value);

Library

ppcLib.a

Description

ppcMtdmacr0() - ppcMtdmacr3() set the value of the DMA Channel Control Registers (DMACR0 - DMACR3). Prior to executing DMA transfers, the control register must be initialized and enabled. The file **<ppcLib.h>** contains several constants that may be used when accessing the DMACR's. :

Errors

None.

Example

- Disable channel 2:

```
ppcMtdmacr2(~DMACR_CE);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtdmact0() • ppcMtdmact3()

Synopsis

```
#include <ppcLib.h>
void ppcMtdmact0(unsigned long dmact0_value);
void ppcMtdmact1(unsigned long dmact1_value);
void ppcMtdmact2(unsigned long dmact2_value);
void ppcMtdmact3(unsigned long dmact3_value);
```

Library

ppcLib.a

Description

ppcMtdmact0() - ppcMtdmact3() set the values of the DMA count registers (DMACT0 - DMACT3) to the specified value. The DMACTs contain the number of transfers left in a DMA transaction for the channel. The maximum number of transfers is 64K and each transfer can be 1, 2, or 4 bytes as programmed in the DMA Channel Control

Errors

None.

Example

- Set the DMACT0 for 64K transfers by setting the DMACT0 to 0:

```
ppcMtdmact0(0x00000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtdmada0(unsigned long dmada0_value);

void ppcMtdmada1(unsigned long dmada1_value);

void ppcMtdmada2(unsigned long dmada2_value);

void ppcMtdmada3(unsigned long dmada3_value);

Library

ppcLib.a

Description

ppcMtdmada0() - ppcMtdmada3() set the values of the DMA destination address registers (DMADA0 - DMADA3) to the specified value. The DMAD registers contains the memory address for transfers between memory and peripheral or the destination address for memory to memory transfers.

Errors

None.

Example

- Set the address for a memory -to-memory transfer:

```
ppcMtdmasa0(0x00020000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtdmasa0() • ppcMtdmasa3()

Synopsis

```
#include <ppcLib.h>
void ppcMtdmasa0(unsigned long dmasa0_value);
void ppcMtdmasa1(unsigned long dmasa1_value);
void ppcMtdmasa2(unsigned long dmasa2_value);
void ppcMtdmasa3(unsigned long dmasa3_value);
```

Library

ppcLib.a

Description

ppcMtdmasa0() - ppcMtdmasa3() set the value of the DMA source/chained address registers (DMASA0 - DMASA3). The DMASA registers are only used in memory-to-memory move mode for any channel or when chaining has been enabled in buffered or fly-by mode for channel 0.

Errors

None.

Example

- Set the address for a memory -to-memory transfer:

```
ppcMtdmasa0(0x00020000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

Synopsis

#include <ppcLib.h>

void ppcMtdmasr(unsigned long dec_value);

Library

ppcLib.a

Description

ppcMtdmasr() sets the value of the DMA Status Register (DMASR). Bits in the DMASR may be cleared by writing a 1 to the corresponding bit position. The file **<ppcLib.h>** contains several constants that may be used when accessing the DMASR.

Errors

None.

Example

- Set all status bits for channel 3:

```
ppcMtdmasr(DMASR_ALL3);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtesr()

Synopsis

#include <ppcLib.h>

void ppcMtesr(unsigned long esr_value);

Library

ppcLib.a

Description

ppcMtesr() sets the value of the Exception Syndrome Register (ESR) to the specified value. Bits 7 to 31 are reserved.

Errors

None.

Example

- Set the all exception s off:

```
ppcMtesr(0x0);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

Synopsis

#include <ppcLib.h>

void ppcMtevpr(unsigned long evpr_value);

Library

ppcLib.a

Description

ppcMtevpr() sets the value of the exception vector prefix register (EVPR). Bits 0 to 15 contain the prefix of the address of the exception processing routines. Bits 15 to 31 are reserved.

WARNING: Do not use **ppcMtevpr()** if using OS Open services that use interrupts, ethernet, or SL/IP etc...

Errors

None.

Example

- Set the EVPR to 0x00A00000:

```
ppcMtevpr(0x00A00000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMtexier()

Synopsis

#include <ppcLib.h>

void ppcMtexier(unsigned long exier_value);

Library

ppcLib.a

Description

ppcMtexier() sets the value of the external interrupt enable register (EXIER). The EXIER contains the enable bits for six external interrupts, the DMA channel interrupts, the JTAG serial port interrupts, and the serial port interrupts. Bits in the EXIER may be cleared by writing a 1 to the corresponding bit position.The file <**ppcLib.h>** contains several constants that can be used when accessing the EXIER.

Errors

None.

Example

- Enable DMA channel 0 external interrupts:

```
unsigned long exier_value=ppcMfexier();
(void) ppcMtexier(exier_value|EXIER_D0IE); /* enable DMA chan. 0
interrupts*/
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

Synopsis

#include <ppcLib.h>

void ppcMtexisr(unsigned long exisr_value);

Library

ppcLib.a

Description

ppcMtexisr() sets the value of the external interrupt status register (EXISR). The EXISR contains the status of the five external hardware interrupts, the DMA channel interrupts, the JTAG serial port interrupts and the serial port interrupts.

External hardware interrupts are enabled via the External Interrupt Enable Register (EXIER). The DMA channel interrupts, the JTAG serial port interrupts and the serial port interrupts may be enabled via the EXIER and must be enabled by the interrupt enable bits in their respective control registers. The file <**ppcLib.h>** contains several constants that can be used when accessing the EXISR.

Errors

None.

Example

- Set the external interrupt 0 pending bit on:

```
ppcMtexisr(EXISR_E0IS_PEND);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtiac1()

Synopsis

#include <ppcLib.h>

void ppcMtiac1(unsigned long iac1_value);

Library

ppcLib.a

Description

ppcMtiac1r() sets the value of the instruction address compare register 1 (IAC1). The IAC1 contains the address of the instruction that the debug event will be based on. The IA1 field of the Debug Control Register (DBCR) controls the instruction address 1 debug event. Bits 30 and 31 of the IAC1 are reserved, since the address must be word aligned.

Errors

None.

Example

- Set the IAC1 register to 0x1000:

```
ppcMtiac1(0x00001000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtiac2(unsigned long iac2_value);

Library

ppcLib.a

Description

ppcMtiac2() sets the value of the instruction address compare register 2 (IAC2). The IAC2 contains the address of the instruction that the debug event will be based on. The IA2 field of the Debug Control Register (DBCR) controls the instruction address 2 debug event. Bits 30 and 31 of the IAC2 are reserved, since the address must be word aligned.

Errors

None.

Example

- Set the IAC2 register to 0x1000:

```
ppcMtiac2(0x00001000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMticcr()

Synopsis

#include <ppcLib.h>

void ppcMticcr(unsigned long iccr_value);

Library

ppcLib.a

Description

ppcMticcr() sets the value of the instruction cache cacheability register (ICCR) to the specified value.

Errors

None.

Example

- Set the ICCR register to 0's, making no regions of memory cacheable:

```
ppcMticcr(0x00000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtiocr(unsigned long iocr_value);

Library

ppcLib.a

Description

ppcMtiocr() sets the input/output configuration register (IOCR) to the specified value. **ppcMtiocr()** allows the user to program some of the external multifunctional pins in the PPC403GA and PPC403GC processors. The file **<ppcLib.h>** contains several constants that can be used when accessing the IOCR.

Errors

None.

Example

- Allow external interrupt 0 triggering to be edge triggered:

```
ppcMtiocr(IOCR_E0T_EDGE);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtpbl1(unsigned long pbl1_value);

void ppcMtpbl2(unsigned long pbl2_value);

Library

ppcLib.a

Description

ppcMtpbl1() and **ppcMtpbl2()** sets the values of their respective Protection Bound Lower Register (PBL). Bits 20 to 31 are reserved.

Errors

None.

Example

- Set the current value of PBL2:

```
ppcMtpbl2(0x0000f000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtpbu1() • ppcMtpbu2()

Synopsis

#include <ppcLib.h>

void ppcMtpbu1(unsigned long pbu1_value);

void ppcMtpbu2(unsigned long pbu2_value);

Library

ppcLib.a

Description

ppcMtpbu1() and **ppcMtpbu2()** sets the values of their respective Protection Bound Upper Register (PBU). Bits 20 to 31 are reserved.

Errors

None.

Example

- Set the current value of PBU2:

```
ppcMtpbu2(0x00010000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void pcMtpid(unsigned long pid_value);

Library

ppcLib.a

Description

ppcMtpid() sets the PID to the specified value.

Errors

None.

Example

- Set the value of the PID.

```
#include <ppcLib.h>
unsigned long pid_value = 0x00000020;
void ppcMtpid(pid_value);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtpit()

Synopsis

#include <ppcLib.h>

void ppcMtpit(unsigned long pit_value);

Library

ppcLib.a

Description

ppcMtpit() sets the programmable interval timer (PIT) to the specified value.

Errors

None.

Example

- Set the PIT to a non-0 value, to cause the PIT to start decrementing:

```
ppcMtpit(0x00000001);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtsgr(unsigned long);

Library

ppcLib.a

Description

ppcMtsgr() sets the value of the Storage Guarded Register (SGR) to the specified value.

Errors

None.

Example

- Set the value of the SGR.

```
#include <ppcLib.h>
ppcMtsgr(0x80000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtsprg0() - ppcMtsprg3()

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller 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*. The file **<ppcLib.h>** contains several constants that can be used when accessing the MSR values in the SRR1 register.

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtsrr2(unsigned long srr2_value);

Library

ppcLib.a

Description

ppcMtsrr2() sets the SRR2 to *srr2_value*.

Errors

None.

Example

Set the save/restore register 2 to X'0000BB00':

```
ppcMtsrr2(0x0000BB00);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

ppcMtsrr3()

Synopsis

#include <ppcLib.h>

void ppcMtsrr3(unsigned long srr3_value);

Library

ppcLib.a

Description

ppcMtsrr3() sets the SRR3 to *srr3_value*. The file **<ppcLib.h>** contains several constants that can be used when setting the MSR values in the SRR3 register.

Errors

None.

Example

Set the save/restore register 3 to problem state (ppcMsrPR):

```
ppcMtsrr3(ppcMsrPR);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller 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 403GA	Yes
PowerPC 603GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMttcr()

Synopsis

#include <ppcLib.h>

void ppcMttcr(unsigned long tcr_value);

Library

ppcLib.a

Description

ppcMttcr() sets the timer control register to the specified value.

The WRC bits of the TCR 3 may only be set once, and will be reset by any form of processor reset. File **<ppcLib.h>** defines several constants for the TCR that can be used as masks.

Errors

None.

Example

- Set the TCR to force a system reset:

```
ppcMttcr(TCR_WD_SYS);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMttlbhi(unsigned long data, unsigned long index);

Library

ppcLib.a

Description

ppcMttlbhi() sets the value of the high order bits of the Unified TLB (UTLB) entry specified by the *index* parameter. The TLBHI contains the tag portion of the UTLB. The *data* parameter for **ppcMttlbhi(**) contains the following fields.

EPN[0:21]	Data value[0:21]
SIZE[0:2]	Data value[22:24]
Valid bit	Data value[25]
Reserved	Data value[26:31]

Errors

None.

Example

- Set the valid bit for entry 1 in the UTLB.

```
unsigned long current_tlbhi_1=ppcMftlbhi(1);
ppcMttlbhi(current_tlbhi_1| 0x00000040,1);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMttlblo()

Synopsis

#include <ppcLib.h>

void ppcMttlblo(unsigned long data, unsigned long index);

Library

ppcLib.a

Description

ppcMttlblo() sets the value of the low order bits of the Unified TLB (UTLB) entry specified by the *index* parameter. The TLBLO contains the data entry portion of the UTLB.

PRN[0:21]	Data value[0:21]
EX, WR	Data value[22:23]
ZSEL[0:3]	Data value[24:27]
WIMG	Data value[28:31]

Errors

None.

Example

- Set the TLBLO value of the write-through bit (WT) to a 1 for entry 2.

```
unsigned long current_tlblo_1=ppcMftlblo(2);
ppcMttblo(current_tlblo_1|0x00000100, 2);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMttsr(unsigned long tsr_value);

Library

ppcLib.a

Description

ppcMttsr() sets the timer status register to the specified value. Bits in the TSR may be cleared by writing a 1 to the corresponding bit position.The file **<ppcLib.h>** defines several constants for the TSR that can be used as masks.

Errors

None.

Example

Reset the watchdog interrupt status in the TSR register:

```
    ppcMttsr(TSR_WIS);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcMtzpr()

Synopsis

#include <ppcLib.h>

void ppcMtzpr(unsigned long);

Library

ppcLib.a

Description

ppcMtzpr() sets the Zone Protection Register (ZPR) to the specified value.

Errors

None.

Example

- Set the ZP bits for zone 0 to allow all access if valid.

```
#include <ppcLib.h>
unsigned long zpr_value;
zpr_value=ppcMfzpr();
ppcMtzpr(zpr_value | 0xc0000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

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 debug exceptions:

```
unsigned long old_val = ppcOrMsr(ppcMsrDE);
```

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcSync()

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 **ppcDbci()** 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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

Synopsis

#include <ppcLib.h>

void ppcTlbia(void);

Library

ppcLib.a

Description

ppcTlbia() invalidates all entries in the TLB. All TLB fields in the TLB entries are unmodified, except for the valid(V) bit.

Errors

None.

Example

- Invalidate all entries in the TLB.

```
#include <ppcLib.h>
ppcTlbia();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

ppcTlbsx()

Synopsis

```
#include <ppcLib.h>
unsigned long ppcTlbsx(unsigned long eff_addr);
```

Library

ppcLib.a

Description

ppcTlbsx() searches for a valid TLB entry for the specified effective address. eff_addr is passed in and a search of the TLB is performed in the same fashion as for a normal load/store instruction. If the search is successful **ppcTlbsx()** returns the TLB entry index otherwise **ppcTlbsx()** returns a -1.

Errors

None.

Example

- Search for the TLB entry for an effective address of 0xa000.

```
#include <ppcLib.h>
unsigned long tlb_entry_index=ppcTlbsx(0xa000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	No
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <tickLib.h>

unsigned long processor_speed(void);

Library

tickLib.a

Description

processor_speed() returns the internal clock speed of the 403 processor.

Errors

None.

Example

- Return the the internal processor clock speed.

```
#include <tickLib.h>
unsigned long proc_speed=processor_speed();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

s1dbprintf()

Synopsis

#include <sys/asyncLib.h>

int s1dbprintf(unsigned long uart_clock, int iocr_reg,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. **iocr_reg** is the value of the IOCR register bits. Refer to the IOCR register in the PPC403GA and PPC403GC User's Manuals. For the 403 EVB, *uart_clock* must be 7372800.

Errors

None.

Example

- Print "Hello World" before I/O has been initialized:

```
#include <sys/asyncLib.h>
#define SCLK 7372800

s1dbprintf(SCLK,2,"Hello World\n\r");
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

Synopsis

#include <sys/asyncLib.h>

int s1dbprintf(unsigned long uart_clock, int iocr_reg, 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. **iocr_reg** is the value of the IOCR register bits. Refer to the IOCR register in the PPC403GA and PPC403GC User's Manuals. For the 403 EVB, *uart_clock* must be 7372800. **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 SCLK 7372800
s1dbprintf(SCLK,2,”Hello World\n\r”);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller User’s Manual*

Synopsis

#include <sys/asyncLib.h>

int s2dbprintf(unsigned long uart_clock, int iocr_reg,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.**iocr_reg** is the value of the IOCR register bits. Refer to the IOCR register in the PPC403GA and PPC403GC User's Manuals. For the 403 EVB, *uart_clock* must be 7372800.

Errors

None.

Example

- Print "Hello World" before I/O has been initialized:

```
#include <sys/asyncLib.h>
#define SCLK 7372800

s2dbprintf(SCLK,2,"Hello World\n\r");
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

s2dbprintfapp()

Synopsis

#include <sys/asyncLib.h>

int s2dbprintf(unsigned long uart_clock, int iocr_reg, 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. **iocr_reg** is the value of the IOCR register bits. Refer to the IOCR register in the PPC403GA and PPC403GC User's Manuals. For the 403 EVB, *uart_clock* must be 7372800. **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 SCLK 7372800
s1dbprintf(SCLK,2,”Hello World\n\r”);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User’s Manual*
- *PPC403GC Embedded Controller User’s Manual*
- *PPC403GCX Embedded Controller 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 403GA, 403GC, 403GCX processor.

```
timertick_install();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- timertick_remove(), p. 10-157

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 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- timertick_install(), p. 10-156

vs1dbprintf()

Synopsis

#include <sys/asyncLib.h>

int vs1dbprintf(unsigned long uart_clock, int iocr_reg,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. **Iocr_reg** is the value of the IOCR register bits. Refer to the IOCR register in the PPC403GA and PPC403GC User's Manuals. For the 403 EVB, *uart_clock* must be 7372800.

Errors

None.

Example

- Print "Hello World" before I/O has been initialized:

```
#include <sys/asyncLib.h>
#define SCLK7372800

vs1dbprintf(SCLK,2,"Hello World\n\r");
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

Processors

PowerPC 403GA	Yes
PowerPC 403GC	Yes
PowerPC 403GCX	Yes

References

- *PPC403GA Embedded Controller User's Manual*
- *PPC403GC Embedded Controller User's Manual*
- *PPC403GCX Embedded Controller User's Manual*

A

Programmable Logic Equations

This appendix presents the logic equations programmed into the interface state machines and other control logic.

```
TITLE PPC403 Evaluation Card PAL

CHIP  _403eval  MACH210

;----------------------- PIN Declarations ---------------------

PIN  35  clk2x              ; 2X system clock
PIN  27  sysclk      reg    ; 403 processor clock

PIN  20  rw                 ; r/w*
PIN  39  xcvr_en            ; data transceiver enable input

PIN  41  xcvr_dir    comb   ; data transceiver direction output
PIN  26  xcvr_oe     comb   ; data transceiver output enable

PIN  13  reset_in           ; reset* input
PIN  42  reset       comb   ; active high reset output

PIN  19  ready_in           ; ready input
PIN  21  ready       comb   ; synchronized ready output to 403

PIN  18  oe                 ; 403 oe
PIN  17  wbe0               ; 403 byte wide peripheral write line
PIN  10  ethernet           ; 403 ethernet chip select line
PIN  16  a26                ; 403 address line
PIN  15  a27                ; 403 address line

PIN  14  ack                ; ethernet controller synchronized
PIN  38  prq                ; ethernet host processor request
PIN  33  prd                ; ethernet latched data read

PIN   9  ethernetcs  comb   ; fully decoded ethernet chip select
PIN   3  rack        comb   ; ethernet remote dma read acknowledge
PIN   5  wack        comb  ; ethernet remote dma write acknowledge
PIN   4  g           comb  ; ethernet data transceiver enable
PIN   6  edir        comb  ; ethernet data transceiver direction
PIN   8  sab         comb  ; select      403 -> ethernet
PIN   7  sba         comb  ; select ethernet -> 403

PIN   2  d7          comb  ; 403 data line

PIN  37  rx                ; receive data
PIN  36  tx                ; transmit data
PIN  32  col               ; packet collision

PIN  43  rx_led      comb  ; receive data LED
PIN  24  tx_led      comb  ; transmit data LED
PIN  40  col_led     comb  ; packet collision LED

NODE  ?  a                 ; ready logic buried node (d-ff)
NODE  ?  b                 ; ready logic buried node (d-ff)
NODE  ?  e                 ; ready logic buried node (d-ff)
NODE  ?  f                 ; ready logic buried node (d-ff)
NODE  ?  h                 ; ready logic buried node (d-ff)
NODE  ?  enet_ready        ; ethernet controller ready

PIN  29  clk20             ; 20 MHz clock input
NODE  ?  k                 ; clock divider buried node (d-ff)
NODE  ?  l                 ; clock divider buried node (d-ff)
PIN  28  clk10       comb  ; 10 MHz clock output

NODE  ?  m                 ; success logic buried node (d-ff)
NODE  ?  n                 ; success logic buried node (d-ff)
PIN  30  success     comb  ; ethernet port access succeeded

NODE  ?  p                 ; prq latching logic buried node (d-ff)
NODE  ?  q                 ; prq latching logic buried node (d-ff)
NODE  ?  prqlatched1       ; first stage prq latched
NODE  ?  t                 ; prq latching logic buried node (d-ff)
NODE  ?  u                 ; prq latching logic buried node (d-ff)
NODE  ?  prqlatched2       ; second stage prq latched

PIN  31  prqlatched  comb  ; PRQ latched on CS falling

PIN  11  breq              ; ethernet controller local bus request
NODE  ?  r                 ; back logic buried node (d-ff)
NODE  ?  s                 ; back logic buried node (d-ff)
NODE  ?  breqlatched       ; breq latched on falling edge of sysclk
```

```
PIN  25  back        comb  ; ethernet controller local bus grant

;--------------------- Boolean Equation Segment -------------------

EQUATIONS

     sysclk = /sysclk
sysclk.CLKF = clk2x
sysclk.SETF = gnd
sysclk.RSTF = gnd

   xcvr_dir = /rw
    xcvr_oe = xcvr_en

      reset = /reset_in

     rx_led = /rx
     tx_led = /tx
    col_led = /col

 ethernetcs =  ethernet + a26 +  a27 + back
        sab =  ethernet + a26 +  a27 + back
        sba =  ethernet + a26 +  a27 + back
       rack =  ethernet + a26 + /a27 + back + oe   + /prqlatched
       wack =  ethernet + a26 + /a27 + back + wbe0 + /prqlatched
          g = (ethernet + a26 +  a27 + ack) * rack * prd
       edir =  ethernet +  oe +  back

          e = /a + b * e
          a = /sysclk +  a * /(b * ready_in * enet_ready)
          b = /sysclk + /a +  (b * ready_in * enet_ready)

      ready = /f + h * ready
          f = /sysclk +  f * /(h * e)
          h = /sysclk + /f +  (h * e)

 enet_ready = (ethernet + a26 + a27 + (oe * wbe0) + /ack + back)

    success = /m + n * success
          m = (a26 + ethernet + oe * wbe0) +  m * / ( n *
              (/back * (/a27 + prqlatched * a27)))
          n = (a26 + ethernet + oe * wbe0) + /m +
              ( n * (/back * (/a27 + prqlatched * a27)))

prqlatched1 = /p + q * prqlatched1
          p = sysclk +  p * / ( q * prq )
```

```
               q = sysclk + /p +   ( q * prq )

prqlatched2 = /t + u * prqlatched2
               t = sysclk +  t * / ( u * prqlatched1 )
               u = sysclk + /t +   ( u * prqlatched1 )

 prqlatched = prq * ( ethernet * prqlatched2 + /ethernet *
              prqlatched )

breqlatched = /r + s * breqlatched
          r = sysclk +  r * / ( s * breq )
          s = sysclk + /r +   ( s * breq )
       back = ethernet * breqlatched + /ethernet * back

         d7 = success

    d7.TRST = /ethernet * /oe * a26 * /a27

      clk10 = /k + l * clk10
          k = /clk20 +  k * /(l * /clk10)
               l = /clk20 + /k +  (l * /clk10)
```

B

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 403 EVB kit.

B.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).

B.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.
+----------------------------------------------------------------------*/
#if !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 and 403GCX*/
#define RD_READ_TLB     116     /*ptrace for readingTLB(403GC &
403GCX)         */
#define RD_WRITE_TLB    117     /*ptrace for writing TLB(403GC & 403GCX
)        */ /* 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_NOLOAD_ERR   1000    /*no loader info available             */
#define RD_COM_ERR      1001    /*communication error occured          */
#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
```

B.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.

B.3.1 RD_ATTACH (30)

Attaches debugger to running process in target environment.

B.3.1.1 Request data

Table B-1. RD_ATTACH Request Table

Parameters	Description
msg.request= RD_ATTACH	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.(Any non zero value)
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

B.3.1.2 Response data

Table B-2. RD_ATTACH Response Table

Parameters	Description
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.retcode= EIO (5)	One of the parameters is incorrect.
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_NOTSUPP (1003)	Call not supported for this interface.
msg.retcode= RD_OK (0)	Successful completion.
msg.data_len=0	No additional data

B.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.

B.3.2.1 Request data

Table B-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.

B.3.2.2 Response data

Table B-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	

B.3.3 RD_DETACH (31)

Detaches debugger from running process in target environment. Debugged process is restarted and execution continues without debugger control.

B.3.3.1 Request data

Table B-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.

B.3.3.2 Response data

Table B-6. RD_DETACH Response Table

Parameters	Description
msg.retcode= ESRCH (3)	The *msg.pid* 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.

B.3.4 RD_FILL (105)

Fills memory with zeroes at the location specified by *address* for the number of bytes specified by data.

B.3.4.1 Request data

Table B-7. RD_FILL Request Table

Parameters	Description
msg.request= RD_FILL	Requested API function.
msg.rpid= process_id	Numeric process ID on the target sys-tem.
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.

B.3.4.2 Response data

Table B-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.

B.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.

B.3.5.1 Request data

Table B-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.

B.3.5.2 Response data

Table B-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 *msg.pid* parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

B.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.

B.3.6.1 Request data

Table B-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.

B.3.6.2 Response data

Table B-12. RD_LDINFO Response Table

Parameters	Description
msg.retcode= RD_NOLOAD_ERR (1000)	No loader information is available.
msg.retcode= ESRCH (3)	The *msg.pid* 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 B-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). **X** 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[ldinfo_next]= ldinfo_next	Next loader block (notice "ldinfo_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.
Note: *ldinfo_next=0* indicates that no further loader blocks are present, otherwise ldinfo_next 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 ldinfo_next 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 membername string. Path-name and member-name are strings terminated with a NULL character.	

B.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.

B.3.7.1 Request data

Table B-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.

B.3.7.2 Response data

Table B-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.

B.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.

B.3.8.1 Request data

Table B-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]= user_name	This field is ignored by ROM monitor.
msg.data_len= strlen(host_name)+strlen(user_name)+2	Length of additional data being sent.

B.3.8.2 Response data

Table B-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.

B.3.9 RD_LOGOFF (104)

Performs user LOGOFF function. This is used when the debugger performs normal termination using quit or detach.

B.3.9.1 Request data

Table B-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.

B.3.9.2 Response data

Table B-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.

B.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.

B.3.10.1 Request data

Table B-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.

B.3.10.2 Response data

Table B-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 *msg.pid* 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.

B.3.11 RD_READ_DCR (110)

This request reads data directly from one of the DCRs (not the process's copy). All DCR registers are accessible through this message request. The sender is responsible for supplying valid DCR values, no error checking is performed on this field.

B.3.11.1 Request data

B.3.11.2 Response data

B.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.

B.3.12.1 Request data

Table B-21. 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.

B.3.12.2 Response data

Table B-22. 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 *msg.pid* parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

B.3.13 RD_READ_GPR_MULT(71)

This request returns the contents of general-purpose registers 0 to 18, inclusive, of the debugged process.

B.3.13.1 Request data

Table B-23. 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.

B.3.13.2 Response data

Table B-24. 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 *msg.pid* 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.

B.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.

B.3.14.1 Request data

Table B-25. 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.

B.3.14.2 Response data

Table B-26. 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 *msg.pid* 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.

B.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.

B.3.15.1 Request data

Table B-27. 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.

B.3.15.2 Response data

Table B-28. 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 *msg.pid* 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.

B.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.

B.3.16.1 Request data

Table B-29. 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.

B.3.16.2 Response data

Table B-30. 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.

B.3.17 RD_READ_TLB(116)

This request reads data directly from the Translation Lookaside Buffer (TLB). An index can be passed to read a specific TLB entry or an index of 0xFFFFFFFF can be used to request all of the entries in the TLB. If the value of the index specified is not within the range of valid TLB indices, an appropriate error code is returned. If the index specified is valid, the TLB entry's data word (TLB lo), tag word (TLB hi), and TID (translation id) are returned. If all entries were requested, the TLB lo, TLB hi, and TID values for each of the entries in the TLB are returned in the message buffer. This request is valid only for chips containing a TLB.

B.3.17.1 Request data

Table B-31. RD_READ_TLB Request Table

Parameters	Description
msg.request= RD_READ_TLB	Requested API function.
msg.address= TLB index (use 0xFFFFFFFF to read all entries)	Index of TLB entry to read.
msg.data_len= 0	Length of additional data being sent.

B.3.17.2 Response data

Table B-32. RD_READ_TLB Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= RD_REG_ERR(1004)	TLB index out of range.
msg.buffer[0,1,2,...]= TLB lo, TLB hi, TID, ...	Values read from TLB entry(s).
msg.data_len= 12 (when single entry read) msg.data_len= 12 x number of TLB entries (when all entries read)	Number of bytes in msg.buffer.

B.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.

B.3.18.1 Request data

Table B-33. 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.

B.3.18.2 Response data

Table B-34. 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.

B.3.19 RD_STOP_APPL (113)

This request is used to interrupt program execution.

B.3.19.1 Request data

Table B-35. 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 sys-tem
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

B.3.19.2 Response data

Table B-36. 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.

B.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.

B.3.20.1 Request data

Table B-37. RD_WAIT Request Table

Parameters	Description
msg.request= RD_WAIT	Requested API function.
msg.data_len= 0	Length of data in msg.buffer.

B.3.20.2 Response data

Table B-38. 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).

B.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.

B.3.21.1 Request data

Table B-39. 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.

B.3.21.2 Response data

Table B-40. 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.

B.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.

B.3.22.1 Request data

Table B-41. 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.

B.3.22.2 Response data

Table B-42. 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 *msg.pid* 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.

B.3.23 RD_WRITE_DCR (112)

This request writes data directly to one of the DCRs (not the process's copy). All DCR registers are accessible through this request. The requester is responsible for supplying valid DCR values. No error checking is performed on this field.

B.3.23.1 Request data

Table B-43. RD_WRITE_DCR Request Table

Parameters	Description
msg.request= RD_WRITE_DCR	Requested API function.
msg.address= DCR number	DCR number to be written
msg.data= value	Data to write to register.
msg.data_len= 0	Length of additional data being sent.

B.3.23.2 Response data

Table B-44. RD_WRITE_DCR 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.

B.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.

B.3.24.1 Request data

Table B-45. 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.

B.3.24.2 Response data

Table B-46. 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 *msg.pid* parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

B.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.

B.3.25.1 Request data

Table B-47. 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.

B.3.25.2 Response data

Table B-48. 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 *msg.pid* 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.

B.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.

B.3.26.1 Request data

Table B-49. 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.

B.3.26.2 Response data

Table B-50. 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.

B.3.27 RD_WRITE_TLB(117)

This request writes data directly to the Translation Lookaside Buffer (TLB). An index is passed along with new TLB lo, TLB hi, and TID (translation id) values, to specify the TLB entry and the data to be written. If the value of the index specified is not within the range of valid TLB indices, an appropriate error code is returned. If the index specified is valid, the TLB entry's data word (TLB lo), tag word (TLB hi), and TID are updated appropriately. This request is valid only for chips containing a TLB.

B.3.27.1 Request data

Table B-51. RD_WRITE_TLB Request Table

Parameters	Description
msg.request= RD_WRITE_TLB	Requested API function.
msg.address= TLB index	Index of TLB entry to write.
msg.buffer[0-2]= TLB lo, TLB hi, TID	Data to write to TLB entry.
msg.data_len= 12	Length of additional data being sent.

B.3.27.2 Response data

Table B-52. RD_WRITE_TLB Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= RD_REG_ERR(1004)	TLB index out of range.
msg.data_len= 0	Length of additional data being sent.

B.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.

B.3.28.1 Request data

Table B-53. 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.

B.3.28.2 Response data

Table B-54. 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.

B.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.

B.3.29.1 Request data

Table B-55. 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.

B.3.29.2 Response data

Table B-56. 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.

C

ROM Monitor Load Format

This appendix presents the ROM Monitor load format requirements.

C.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, bootlLib.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.

C.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.

C.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 headr                     */
        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.

C.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.

C.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**.

C.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.

C.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.

D

403 EVB Bill of Materials

This appendix presents the bill of materials for the EVB and its components:

HEADING EXPLANATION:	
QTY	- Quantity Used
IBM PN	- IBM Part Number
DESCRIPTION	- The Component Description
COMP CODE	- System Assigned Component Code (R1, C33, U1, etc)
VENDOR PN	- Supplier Code or Part Number From the Vendor Catalog
VENDOR	- Supplier Code of the Part
W/NW	- Wetable vs Non-Wetable indicator.

QTY	DESCRIPTION/COMP CODE	VENDOR PN	VENDOR	W/NW
1	Raw Card	D13H8618	IBM	W
	(or)	42H1843	IBM	W
1	10 base 2 transformer **T1**	LT6003	VALOR	W
1	open coll hex inverter **U25**	SN7406N	TI	W
8	octal bus transceiver **U3 U4 U5 U6 U8 U9 U12 U16**	SN74ABT245AN	TI	W
1	octal buffer **U24**	SN74ALS244CN	TI	W

1	octal transparent D latch	DM74ALS573BN	NATIONAL	W
	U14			
1	octal bus transcvr & reg	DM74ALS646NT	NATIONAL	W
	U15			
1	flash EPROM 128KX8	AM29F010-70JC	AMD	W
	U7			
1	right angle male BNC conn	227161-2	AMP	W
	P5			
41	.1U,20%,50v	ECQ-V1H104JL	PANASONIC	W
	C1 C2 C3 C4 C5 C6 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C20 C21 C22 C23 C24 C25 C26 C27 C28 C30 C31 C32 C33 C34 C35 C36 C41 C42 C43 C44 C45 C47 C56			
1	10U,20%,25v	ECS-F1EE106K	PANASONIC	W
	C7			
3	22U,20%,25v	ECS-F1EE226	PANASONIC	W
	C29 C37 C40			
1	.001U,20%,1kv	ECK-D3A102KBN	PANASONIC	W
	C38			
1	.01U,20%,100v	ECQ-V1103JM	PANASONIC	W
	C39			
1	120 pin right angle conn	650874-4	AMP	W
	P3			
1	5 pin DIN recpt rt angle	211450-1	AMP	W
	P1			
2	9 pin D-SHELL S&L TAIL conn	747840-4	AMP	W
	P2 P4			
1	serial ntwrk intface cntlr	DP83902AV	NATIONAL	W
	U21			
1	coaxial transcvr interface	DP8392CN	NATIONAL	W
	U20			
1	power on reset gen	DS1233-10	DALLAS	W
	U23			
2	fuse holder clip	F041-ND	DIGIKEY	W
	F1 F2			
1	1x1 pin header	2401-6112-TG	3M	W
	J8			
7	1x2 pin header - male	2402-6112-TG	3M	W
	J3 J4 J5 J6 J7 J10 J11			
1	2x8 pin header - male	2416-6122-TG	3M	W
	J2			
1	1x3 pin header	2403-6112-TG	3M	W
	J9			

1	static ram 8Kx8-100ns	SRM2264LC	SMOS	W
	U13			
2	led display - green	LN342GP	PANASONIC	W
	DS1 DS4			
1	led display - yellow	LN442YP	PANASONIC	W
	DS3			
1	led display - red	LN242RP	PANASONIC	W
	DS2			
1	3.3v reg	LT1086CT	LINEAR TECH	W
	U18			
1	programable logic array	MACH210A-12JC	AMD	W
	U22			
1	quad RS232 transceiver	MAX208CNG	MAXIM	W
	U11			
1	66 mhz osc	MX045HS-66	CTS	W
	Y1			
1	20 mhz osc	MX045HS-20	CTS	W
	Y2			
1	7.3728 mhz osc	MX045HS-7.3728	CTS	W
	Y3			
1	UART	PC16550DV	NATIONAL	W
	U19			
2	push button	A2216	APEM	N
	SW1 SW2			
1	dc to dc converter	PM7102	VALOR	W
	U17			
1	PowerPC RISC	PPC403GA-JA33C1	IBM	W
	(or)	PPC403GC-JA33C1	IBM	W
	(or)	PPC403GCX-JA33C1	IBM	W
	U10			
3	5.1k,1%,1/4w resistor	1/4w 5%-5101X	YAGEO	W
	R2 R10 R12			
1	1m,1%,1/4w resistor	1/4w 5%-1004X	YAGEO	W
	R6			
4	200,1%,1/4w resistor	1/4w 5%-2000X	YAGEO	W
	R3 R4 R5 R11			
6	10K,10%,1/4w resistor	1/4w 5%-1003X	YAGEO	W
	R18 R22 R23 R24 R25 R26			
1	121,1%,1/4w resistor	1/4w 1%-1210X	YAGEO	W
	R7			
1	196,1%,1/4w resistor	1/4w 1%-1960X	YAGEO	W
	R8			
3	24,1%,1/4w resistor	1/4w 5%-24R0X	YAGEO	W
	R19 R20 R21			
3	33,1%,1/4w resistor	1/4w 5%-33R0X	YAGEO	W
	R9 R14 R15			

2	499,1%,1/4w resistor **R16 R17**	1/4w 5%-4990X	YAGEO	W
2	1k,1%,1/4w resistor **R1 R13**	1/4w 5%-1002X	YAGEO	W
1	10k-10 resistor pack **RP1**	770-101-R10K	CTS	W
1	1.5k-5 resistor pack **RP2**	770-61-R1.5K	CTS	W
1	4.7k-5 resistor pack **RP3**	770-61-R4.7K	CTS	W
1	10k-5 resistor pack **RP5**	770-61-R10K	CTS	W
1	330-5 resistor pack **RP4**	770-61-330	CTS	W
2	72 pin simm socket **U1 U2**	822019-4	AMP	W
1	2x10 pin header - male **J1**	2520-6002-UB	3M	W
1	1mx32 60 ns EDO DRAM	V408J32S60	MOSEL	N
1	160 PQFP socket base	822114-4	AMP	W
1	160 PQFP socket cover	822115-4	AMP	N
1	32 pin PLCC socket	821665-1	AMP	W
2	44 pin PLCC socket	821575-1	AMP	W
1	84 pin PLCC socket	821573-1	AMP	W
4	black 2 pin jumper	929950-00-I	3M	N
4	screw lock	207952-3	AMP	N
4	nylon pan head screws (6-32x.5)			N
4	nylon hex spacers (6-32x.5)			N
1	Fuse (3A)	3AG	LITTLEFUSE	N

```
Total number of Components: 133


HIGHEST-USED COMPONENT CODES
--------------------------------------------------
C56, D54, F2, J11, P5, R26, RP5, SW2, T1, U25


UNUSED COMPONENT CODES
---------------------------------------
C46, C48-C55
```

Index