PowerPC™ Microprocessor Common Hardware Reference Platform: A System Architecture

Personal Use Copy - Not for Reproduction

LICENSE INFORMATION

To the extent that Apple Computer, Inc., International Business Machines Corporation, and Motorola, Inc. (referred to as "the creators") own licensable copyrights in the *PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture* (including accompanying source code samples), the creators grant you a copyright license to copy and distribute portions of this document (including accompanying source code samples) in any form, without payment to the creators, for the purpose of developing original documents, code, or equipment (except integrated circuit processors) which conform to the requirements in this document and for the purpose of using, reproducing, marketing, and distributing such code or equipment. This authorization applies to the content of this specification only and not to the referenced material. This authorization does not give you the right to copy and distribute this document in its entirety.

In consideration you agree to include for each reproduction of any portion of these documents or any derivative works the copyright notice as displayed below.

You are responsible for payment of any taxes, including personal property taxes, resulting from this authorization. If you fail to comply with the above terms, your authorization terminates.

The creators and others may have patents or pending patent applications, or other intellectual property rights covering the subject matter described herein. This document neither grants or implies a license or immunity under any of the creators or third party patents, patent applications or other intellectual property rights other than as expressly provided in the above copyright license. The creators assume no responsibility for any infringement of third party rights resulting from your use of the subject matter disclosed in, or from the manufacturing, use, lease, or sale of products described in, this document.

Licenses under utility patents of IBM® in the field of information handling systems are available on reasonable and non-discriminatory terms. IBM does not grant licenses to its appearance design patents. Direct your licensing inquiries in writing to the IBM Director of Licensing, International Business Machines Corporation, 500 Columbus Avenue, Thornwood, NY 10594.

Licenses under utility patents of Apple Computer, Inc., that are necessary to implement the specification set forth in this document are available on reasonable and non-discriminatory terms. Apple Computer, Inc. does not grant licenses to its appearance design patents. Direct your licensing inquires in writing to Mac OS Licensing Department, Apple Computer, Inc., 1 Infinite Loop, MS 305-1DS, Cupertino, CA 95014.

© Copyright Apple Computer, Inc., International Business Machines Corporation, Motorola, Inc. 1995. All rights reserved.

Note to U.S. Government Users—Documentation related to restricted rights— Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Personal Use Copy - Not for Reproduction

NOTICES

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law. In such countries, the minimum country warranties will apply.

THE CREATORS PROVIDE THIS DOCUMENT (INCLUDING ACCOMPANYING SOURCE CODE EXAMPLES) “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THE DISCLAIMER OF WARRANTY APPLIES NOT ONLY TO THE DOCUMENT (INCLUDING ACCOMPANYING SOURCE CODE EXAMPLES) BUT ALSO TO ANY COMBINATIONS, INCORPORATIONS, OR OTHER USES OF THE DOCUMENT (INCLUDING ACCOMPANYING SOURCE CODE EXAMPLES) UPON WHICH A CLAIM COULD BE BASED.

Some states do not allow disclaimers of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

These materials could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the document. The creators may make improvements and/or changes in the product(s) and/or the program(s) described in, or accompanying, this document at any time.

It is possible that this document may contain reference to, or information about, products (machines and programs), programming, or services of the creators that are not announced in your country. Such reference or information must not be construed to mean that the creators intend to announce such products, programming, or services in your country.

Requests for copies of this document or for technical information about products described herein should be directed to the creators. Refer to “Obtaining Additional Information” on page 300 for a description of information available and telephone numbers.

Version 1.0 (November 1995)

TRADEMARKS AND SERVICE MARKS

Trademarks or service marks in the United States or other countries are denoted by a registered symbol (®) or a trademark symbol(™) on their first occurrence in this document. See the section entitled “Trademark Information” on page 295 for a complete listing of all referenced trademarks and the companies that own them.

Personal Use Copy - Not for Reproduction

PowerPC™ Microprocessor Common Hardware Reference Platform: A System Architecture

Developed by Apple Computer, Inc., International Business Machines Corporation, and Motorola, Inc.

MORGAN KAUFMANN PUBLISHERS, INC. SAN FRANCISCO, CALIFORNIA

To order copies of this book, please contact the publisher at (800)745-7323 or your Apple, IBM, or Motorola contacts given in Section , “Sources for Documents,” on page 299

Sponsoring Editor: Jennifer Mann
Production Manager: Yonie Overton
Production Editor: Julie Pabst
Editorial Assitant: Jane Elliott
Cover Design: Carron Design
Printer: Courier Corporation

Morgan Kaufmann Publishers, Inc.
Editorial and Sales Office:
340 Pine Street, Sixth Floor
San Francisco, CA 94104-3205
USA
Telephone: 415/392-2665
Fasimile: 415/982-2665
Internet: mkp@mkp.com
Order toll free: 800/745-7323

This book was typeset by the authors, using FrameMaker cross-platform.

Printed in the United States of America

00 99 98 97 96 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data is available for this book.

ISBN 1-55860-394-8

Personal Use Copy - Not for Reproduction

Foreword

Personal, business, and technical computing environments continue to demand ever-increasing levels of function and performance. The PowerPC™ microprocessor has been jointly developed by IBM®, Motorola®, and Apple® to meet the computer performance growth requirements of the 90s and enable a new generation of computer platform and applications with industry-leading capabilities. A prerequisite for these new computer systems to gain wide acceptance by the application and operating system development communities is a “system architecture” that provides consistent interfaces between hardware and software. The architecture maintains application software compatibility across platforms manufactured by different system vendors, and enables scalability to meet future demands. This specification satisfies these prerequisite by documenting a PowerPC-based *open system architecture* that can be used by system and software companies for the development of compatible PowerPC computer systems, subsystems, and software. The architecture is intended to support a range of PowerPC system implementations including portable, desktop, and server computer systems.

Our engineering teams, in conjunction with other industry participants, have endeavored to develop a leading-edge computer architecture that can satisfy manufacturer and customer needs into the next century. They have also expended significant effort to accommodate legacy IBM PC and Apple Macintosh® hardware and software issues, while supporting future system evolution. This focus on the future, with an eye on the past, is a key attribute of this architecture. Another key attribute is the ability, through a combination of hardware and software, for system manufacturers to differentiate their systems while maintaining application compatibility. This attribute creates greater opportunity for value to be added by system manufacturers.

We want to thank our teams and those in the computer industry who have contributed to development of this specification. Each of our companies supports making this architecture a success, and we invite each of you to join in the opportunity that this open industry architecture creates. For us, this is the beginning of an exciting new era in personal computing.

David Nagel
Senior Vice President, Apple Computer, Inc.

Robert M. Stephenson
IBM Senior Vice President and Group Executive, Personal Systems Group

Joe Guglielmi
Corporate Vice President and General Manager of Motorola Computer Group

Personal Use Copy - Not for Reproduction

Contents

Personal Use Copy - Not for Reproduction

Personal Use Copy - Not for Reproduction

Personal Use Copy - Not for Reproduction

Figures

Tables

Personal Use Copy - Not for Reproduction

About this Document

The purpose of this document is to define an open architecture and minimum system requirements that enable the development and manufacture of industry-standard computer systems based on PowerPC microprocessors. These requirements are intended to be precise enough to assure application software compatibility for several operating system environments, broad enough to cover portables through server platforms in single or multiprocessor configurations, and forward-looking enough to allow evolution, including 64-bit addressing.

These requirements were developed by Apple Computer, Inc., International Business Machines Corporation, and Motorola, Inc. to define a system which is intended to become the pervasive open industry standard for single-user portable through multi-user server configurations. Systems built to these requirements will have PowerPC microprocessor(s) and will share components with the Apple Macintosh family and IBM compatible personal computers. These systems will be capable of running various native operating systems. The set of operating systems anticipated to be available includes Apple Mac™ OS, IBM AIX™, and PowerPC™ editions of IBM OS/2™ Warp Connect, Microsoft Windows NT™ Workstation, Novell NetWare™, and SunSoft Solaris™. Most applications for these operating environments may be run on these systems either in native mode or through some emulation capability. With the appropriate operating system and x86 emulation support, these systems will be capable of running Windows™ and DOS applications.

Within the context of this document, “architecture” is defined as the specification of the interface between the hardware platform and the operating systems and applications. Device drivers also use this architecture, but require additional definition of the device interfaces to the hardware and operating system interfaces within the software.

To the extent that firmware abstracts the hardware interface, it becomes part of the hardware. The firmware to operating system interface is defined in this architecture. Two types of firmware are discussed here. Open Firmware is the initialization or boot code that controls the platform prior to the transfer of control to the operating system. Run-Time Abstraction Services (RTAS) is the run-time firmware which provides abstractions to the executing operating system. Interfaces within the software or within the hardware are not defined in this document. Where necessary, reference will be made to documents where those definitions can be found.

Within the context of this document the term “the architecture” or “the CHRP architecture” is used to refer to the requirements contained in this document, *PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture*.

Goals of the Specification

The specific goals of this specification are as follows:

- To create an open industry standard to be used for the implementation of PowerPC based systems. The architecture document is available to the industry and can be used by any hardware or software vendor to develop compliant products.
- To allow compatible differentiation through the use of abstracted hardware interfaces, defined minimum hardware, and extension mechanisms.
- To leverage existing and future industry-standard buses and interfaces. Existing bus architectures have a proven level of performance and functionality. Established industry-standard interfaces—for example SCSI, IDE, LocalTalk®, Ethernet™, etc.— and newer bus architectures, interfaces and protocols —for example PCI, PC Card, IrDA, etc.— provide higher levels of performance or utility not achievable by the older standards. The architecture allows platform and system designers to determine which buses, interfaces, and protocols best suit their target environment.
- To provide a flexible address map. Another key attribute of this specification is the relocatability of devices and subsystems within the PowerPC address space. Subsystem address information, which defines where I/O devices reside, is detected by the Open Firmware and passed to the operating systems in the device tree. The architecture accommodates the use of multiple identical buses and adapters in the same platform without address conflicts.

Personal Use Copy - Not for Reproduction

- To build upon the Open Firmware boot environment defined in IEEE 1275, *IEEE Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices* [9]. Currently the abstraction approach for some operating systems uses platform description information discovered by a legacy boot process and passed to the operating system in data structures. With these systems, operating systems and platforms will migrate to the Open Firmware boot process and device tree.
- To architect the control of power management by different operating system. It is important that the combination of hardware and software be allowed to minimize power consumption through automatic or programmed power-saving methods. Power management of systems will reduce the operational cost for the user and reduce the impact of the system on the environment.
- To provide an architecture which can evolve as technology changes. The creators of the architecture invite industry participation in evolving future versions of it.
- To minimize the support cost for multiple operating systems through the definition of common platform abstraction techniques. Common and compatible approaches to the abstraction of hardware will reduce the burden on hardware vendors who produce differentiated machines.
- To architect a mechanism for error handling, error reporting, and fault isolation. The architecture provides for the implementation of more robust systems if desired by the system developers.

Audience for this Document

This document defines the Common Hardware Reference Platform (CHRP™) architecture and system requirements for building standard systems. This document is the primary source of information that a hardware platform, operating system, or hardware component developer would need to create compatible products. Additional requirements are defined by the industry standards referenced in this document.

This document must be used by those building CHRP hardware platforms. The document describes the hardware to operating system interface which must be provided in these platforms. Platform designers must assemble components and firmware which match this interface. Also, the document defines minimum system configuration requirements. Platform designers must meet or exceed these minimums to build a standard platform. The operating systems which are expected to support this architecture are listed in Appendix A,

Personal Use Copy - Not for Reproduction

"Operating System Information," on page 223. Each operating system has configuration requirements which are described in documents maintained by the owners of the operating systems. Using these requirements, a hardware platform designer may target a configuration to support multiple operating systems.

This document must be used by those building compatible software including operating systems, boot software, or firmware. If a function is supported, software developers must provide support for the interfaces described in this document. This software must provide the mandatory functions and capabilities as described in the requirements in this document. However, this document does not limit this software from going beyond the specification through software tailored to specific hardware. For example, an operating system must implement the interface to the required abstracted interfaces to hardware, but the operating system may also implement fast paths to specific hardware that it recognizes.

Sample implementations of CHRP platforms will be developed and their specifications will be published by several companies. Platform designers may use these descriptions as a reference for developing clone designs, or may use these descriptions in conjunction with this document to develop unique compliant platform designs.

Organization of this Document

The following chapters provide the detailed requirements for system developers to build compliant systems. The chapters cover all needed topics from minimum system requirements to power management. Where appropriate, references are made to other industry standards or readily available reference documentation for the required architecture information.

Following is a summary and brief description of the chapters and appendices of this document:

- Chapter 1, "Introduction," on page 1 describes the advantages of the architecture and shows typical platform topologies.
- Chapter 2, "System Requirements," on page 7 describes the system operation and 64 Bit Addressing support. Firmware, Bi-Endian, and minimum platform requirements are documented. The minimum platform requirements are given for three types of platforms. The Bi-Endian capability of these platforms is one key feature which allows the multitude of operating systems.

Personal Use Copy - Not for Reproduction

- Chapter 3, “System Address Map,” on page 23 describes the address map and the requirements for the address map. The chapter describes some address map extensions to support PC emulation.
- Chapter 4, “Processor and Memory,” on page 49 defines the required PowerPC microprocessor attributes for processors included in these platforms and provides specific requirements for the memory subsystem.
- Chapter 5, “I/O Bridges,” on page 69 defines the requirements for host bridges and I/O bridges. “Host Bridges” are the interface to the processor bus and the I/O bus and “I/O bridges” are any additional interfaces to other I/O buses.
- Chapter 6, “Interrupt Controller,” on page 85 defines the required interrupt controller for standard platforms.
- Chapter 7, “Run-Time Abstraction Services,” on page 91 defines the services which provide an abstract interface to some hardware components. These services provide to the operating systems a platform-independent interface for hardware components.
- Chapter 8, “Non-Volatile Memory,” on page 141 defines the NVRAM content and structure to be used on these platforms. NVRAM contains information such as the platform configuration and error logs that must persist across platform power cycles.
- Chapter 9, “I/O Devices,” on page 151 defines the requirements for I/O devices and references another document that defines the specific registers used to address these devices.
- Chapter 10, “Error and Event Notification,” on page 157 defines platform error reporting requirements and describes the use of RTAS services for obtaining error and event information on these platforms.
- Chapter 11, “Power Management,” on page 185 describes the anticipated power management approach. For systems which implement power management, this chapter defines the requirements that software and hardware must meet.
- Chapter 12, “The Symmetric Multiprocessor Option,” on page 215 gives those requirements specifically for symmetric multiprocessor platforms and describes the boot process for symmetric multiprocessors.
- Appendix A, “Operating System Information,” on page 223 lists sources of additional information for targeted operating systems.

Personal Use Copy - Not for Reproduction

- Appendix B, “Requirements Summary,” on page 225 is a complete list of all requirements imbedded in a chapter level table of contents. This appendix is a useful summary or check list for those implementing standard platforms.
- Appendix C, “Bi-Endian Designs,” on page 265 provides some explanation of endianess and the design approaches for implementing Bi-Endian platforms.
- Appendix D, “Architecture Migration Notes,” on page 281 provides information describing creation of this architecture from the legacy systems.

Suggested Reading

The “Bibliography” on page 297 provides a full list of references and ordering information for these references. Within this document, the number of the reference in the bibliography is placed after the citation in brackets, “[nn]”.

This document assumes the reader has an understanding of computer architecture, the PowerPC microprocessor architecture, the current PowerPC processors, Apple and IBM compatible personal computers, Peripheral Component Interconnect (PCI) local bus, and Open Firmware. Some understanding of the current personal computer and workstation architectures is also useful. A list of suggested background reading includes:

- *Computer Architecture: A Quantitative Approach* [2]
- *The PowerPC Architecture: A Specification for a New Family of RISC Processors* [1]. Note that this specification is referred to as *The PowerPC Architecture* in the body of this document.
- *PowerPC 603 RISC Microprocessor User’s Manual* [4]
- *PowerPC 603 RISC Microprocessor Technical Summary* [5]
- *PowerPC 604 RISC Microprocessor User’s Manual* [6]
- *PowerPC 604 RISC Microprocessor Technical Summary* [7]
- *Technical Introduction to the Macintosh Family* [26]
- *PowerPC Reference Platform Specification,* Version 1.1 [28]
- *PCI Local Bus Specification*, Revision 2.1 [14]
- IEEE 1275, *IEEE Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices* [9] and relevant bindings

Personal Use Copy - Not for Reproduction

Conventions Used in this Document

Within the body of this document lists of requirements are clearly defined. The convention used is to head each requirements list with the word "**Requirements:**" in bold face type. Following this heading are one or more requirements. These requirements may point to other standards documents, or figures or tables which conveniently show the requirement. The referenced material becomes part of the requirements in this document. Users of this document must comply with these requirements to build a standard platform. Other material in this document is supportive description of these requirements, architecture notes, or implementation notes. Architecture or implementation notes are flagged with a descriptive phrase—for example, "Hardware Implementation Note"— and followed by indented paragraphs. The descriptive material and notes provide no additional requirements and may be used for their information content.

Big-Endian numbering of bytes and bits is used in this document, unless indicated otherwise.

Typographical conventions used in this document are described in Table on page xxiii.

Table 1. Typographical Conventions

Text Element	Description of Use
Italics	Used for emphasis such as the first time a new term is used. Indicates a book title. Indicates PowerPC instruction mnemonics. Indicates Open Firmware properties, methods, and configuration variables. Indicates RTAS function and parameter names
0xnnnn	Prefix to denote hexadecimal numbers.
0bnnnn	Prefix to denote binary numbers.
nnnnn	Numbers without a prefix are decimal numbers.
0xF. . . FFF100	This hexadecimal notation represents a replication of the hexadecimal character to the right of the ellipsis to fill out the field width. For example, the address 0xF . . . FFF100 would be 0xFFFFF100 on a processor with a 32-bit address bus or 0xFFFFFFFFFFFFF100 on a processor with a 64-bit address bus.
0:9	Ranges of bits are specified by two numbers separated by a colon. The range includes the first number, all numbers in between, and the last number.
0xm-0xn	A range of addresses or values within the document is always inclusive, from m up to and including n.

Personal Use Copy - Not for Reproduction

Acknowledgments

This document has come together through the work of many people in several companies. The primary responsibility for writing sections fell on employees of Apple, IBM, and Motorola. Industry review was solicited and comments were received from such companies as Advanced Micro Devices, Canon, FirePower, National Semiconductor, and Toshiba.

Not everyone who worked on this document can be mentioned here due to space limitations. The contributions of some key individuals are worth mentioning. The document was brought together by a team of engineers who merged existing material with new material and spent countless hours reviewing and discussing this document. This team included Art Adkins, Richard Arndt, Stanford Au, Don Banks, Jerry Barrett, Rob Baxter, Rich Bealkowski, Mike Bell, Doug Bossen, Jim Brennan, John Bruner, Steve Bunch, Pat Carr, Leo Clark, Ron Clark, Bob Coffin, Clayton Cole, Scott Comer, Gary De Angelis, Sanjay Deshpande, Brad Frey, Jim Gable, Bill Galcher, Brian Hansche, Ron Hochsprung, Kohichi Kii, John Kingman, Steve MacKenzie, Don McCauley, Andy McLaughlin, Todd Moore, Dan Neal, Luan Nguyen, Jim Nicholson, John O'Quin, Mike Paczan, Ray Pedersen, Charlie Perkins, Patrick Perrino, Dave Peterson, Greg Pfister, Craig Prouse, Andy Rawson, Paul Resch, Joe St. Clair, Kanti Shah, Fred Strietelmeier, Howard Tanner, M. Teodorovich, Don Thorson, Steve Thurber, Dave Tjon, Abraham Torres, George Towner, Ted Toyokawa, Tom Tyson, and Lee Wilson.

Comments on this Document

Comments on this document are welcome. Because of time and resource constraints, we will not always be able to provide responses to general industry comments. We will review your comments for possible incorporation in future versions of the specification. All comments become the property of Apple, IBM, and Motorola and may be used for any purpose whatsoever. For this reason, comments must not contain any proprietary data. Please preface your comments with the statement "These comments do not contain confidential or proprietary information and may be used for any purpose". Comments may be addressed to:

info-hrp@austin.ibm.com

Personal Use Copy - Not for Reproduction

Introduction 1

This architecture specification provides a comprehensive computer system hardware-to-software interface definition, combined with minimum system requirements, that enables the development of and software porting to a range of compatible industry-standard computer systems from portables through servers. These systems are based on the PowerPC microprocessor, as defined in *The PowerPC Architecture* [1]. The definition supports the development of both uniprocessor and multiprocessor system implementations.

A key attribute and benefit of the architecture is the ability of hardware platform developers to have degrees of freedom of implementation below the level of architected interfaces and therefore have the opportunity for adding unique value. This flexibility is achieved through architecture facilities including: (1) device drivers; (2) Open Firmware (OF); (3) Run-Time Abstraction Services (RTAS); and (4) hardware abstraction layers. The role of items 1 and 4 are described in separate operating system documentation. The role that items 2 and 3 play in the architecture will be described in subsequent paragraphs and chapters.

Though the PowerPC microprocessor is the most widely used RISC processor, substantial legacy software exists and a mechanism for running the bulk of this legacy software is a requirement. The system address map has been defined with a specific objective of assisting efficient x86 emulation. Additionally, the PowerPC microprocessors support Bi-Endian (see Section 2.3, “Bi-Endian Support,” on page 14 and Appendix C, “Bi-Endian Designs,” on page 265) operation which is a key attribute important to running the supported operating systems and applications. Bi-Endian capability is not available in the current IBM PC compatible x86-based system architecture.

The architecture combines leading-edge IBM PC and Apple Macintosh technologies to create a superior personal computing platform. By design, it

supports a wide range of computing needs including personal productivity, engineering design, data management, information analysis, education, desktop publishing, multimedia, entertainment, and database, file, and application servers. The architecture effectively leverages industry-standard I/O through the PCI bus while accommodating legacy I/O from both the IBM PC compatible and the Apple Macintosh domains. This approach provides several key benefits for system manufacturers and end customers: (1) systems can be designed and manufactured to enable the customer a choice of operating system support which could include AIX, Mac OS, NetWare, OS/2, Solaris, or Windows NT and (2) smooth application, operating system, and customer system transitions are enabled by accommodation for legacy software, I/O devices, and peripherals. This architecture helps protect the customer's investment while moving to more advanced portable, desktop, and server computing platforms. Systems based on this architecture are expected to offer price/performance advantages and to address the expected growth in computing performance and functionality.

CHRP systems must be able to meet certain minimum requirements to be considered compliant with the architecture specified in this document. Apple, IBM, and Motorola are reviewing these requirements and intend to make them available at a later date.

1.1 Platform Topology

To the experienced computer designer and system manufacturer, much of the content of the architecture will be familiar. A typical desktop topology is shown in Figure 1 on page 4. This topology consists of a single PowerPC microprocessor, volatile System Memory, and a single Host Bridge providing a PCI Bus. The PCI Bus and ISA Bridge provide for connection of I/O devices.

A more complex general topology is shown in Figure 2 on page 5. All platforms consist of one or more PowerPC microprocessors, a volatile System Memory separate from other subsystems, and a number of I/O devices, which may initiate transactions to System Memory. The processors are linked over the primary processor bus/switch to each other, to the System Memory, and to one or more Host Bridges (Host Bridge 0 must be a PCI Host Bridge). In general, I/O devices do not connect to the primary processor bus/switch. The Host Bridges connect to secondary buses which have I/O devices connected to them. In turn, one or more bus bridges may be employed to tertiary buses (for instance ISA or PCI) with additional I/O devices connected to them. Typically, the bus speeds and throughput decrease and the number of supportable loads increases as one progresses from the primary processor bus to more remote buses. Multiprocessor platforms have a symmetric (at least from the software

Personal Use Copy - Not for Reproduction

point of view) shared memory model; refer to Section 12.1, “SMP System Organization,” on page 216.

There are variations on these topologies, which are likely to occur and are therefore worth describing below. The architecture describes interfaces, not implementation. The logical software model must remain the same, even if the physical topology is different.

- In a smaller platform, the Host Bridge and/or the memory and/or an I/O device may be integrated into a single chip. In this case, the topology would not look like Figure 1 from a chip point of view, but instead would be integrated onto the single chip.
- In a larger platform, secondary buses may be implemented, with two or more Host Bridges, as two or more parallel expansion buses for performance reasons. Similarly, tertiary buses may be two or more parallel expansion buses off each secondary bus. This is indicated by the ellipses near the Host Bridge and the Bus Bridge.
- In a high performance platform, with multiple processors and multiple memories, a switch may be employed to allow multiple parallel accesses by the processors to memory. The path through the switches would be decided by the addressing of the memory.

Each of the following chapters provides information necessary to successfully implement compliant systems. It is recommended that the reader become thoroughly familiar with the contents of these chapters and their references prior to beginning system or software development. It is anticipated that standard chip sets will simplify the development of compliant implementations consistent with the topologies shown below and will be available from third-party industry sources.

Personal Use Copy - Not for Reproduction

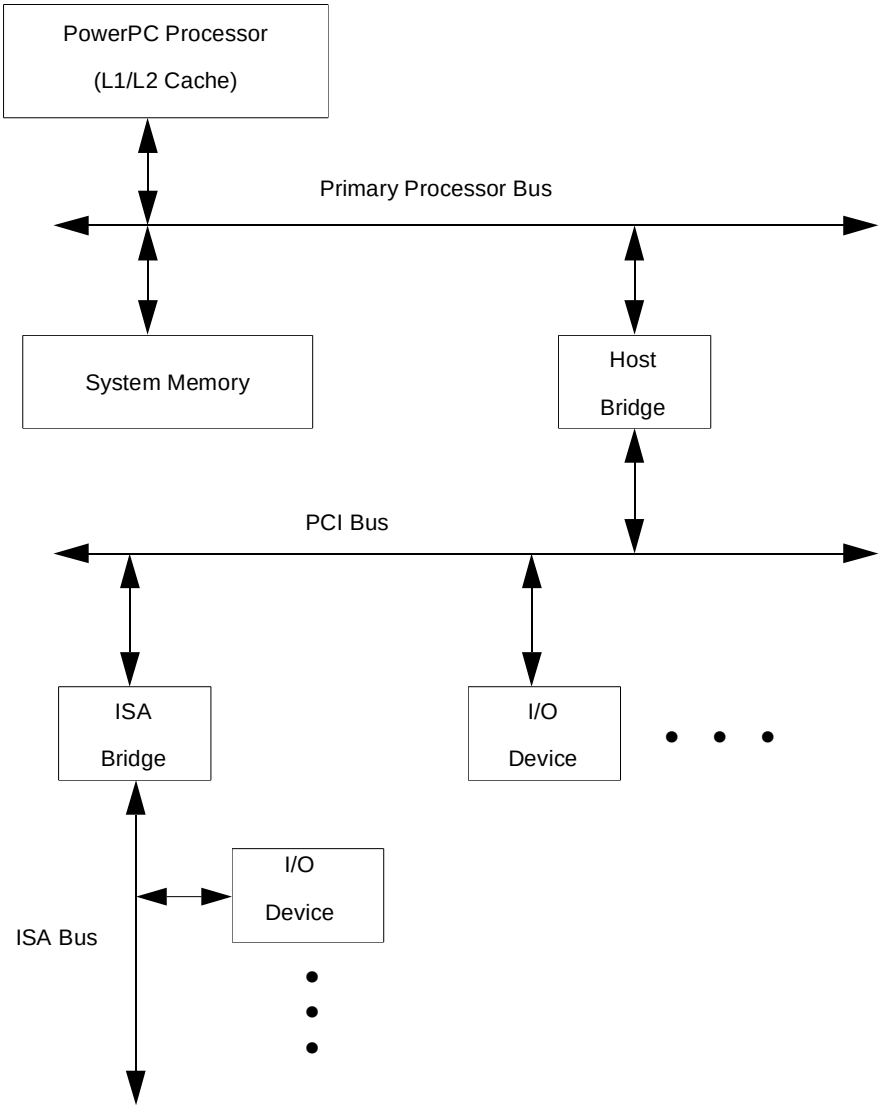


Figure 1. Typical Desktop Topology

Personal Use Copy - Not for Reproduction

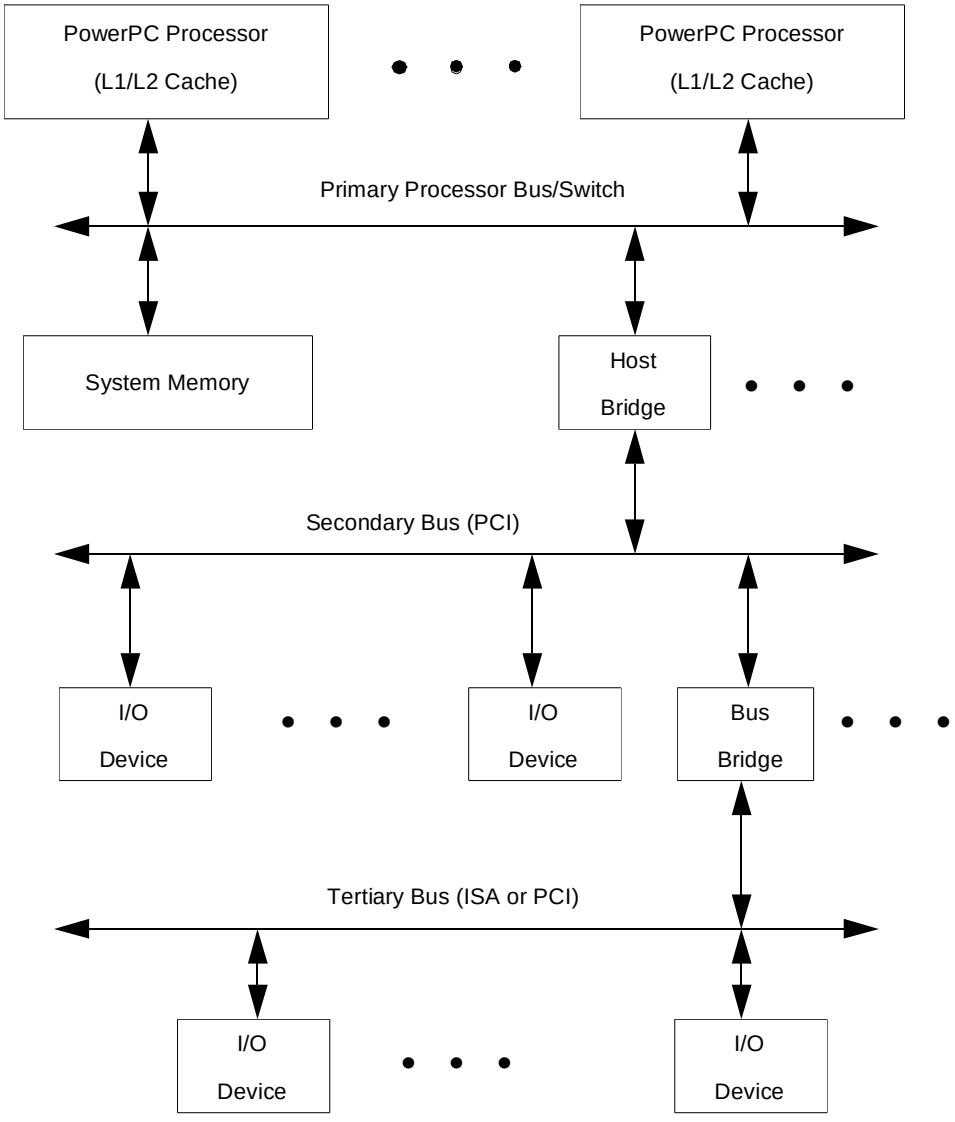


Figure 2. General Platform Topology

Personal Use Copy - Not for Reproduction

Blank page when cut - 6

System Requirements

2

This chapter gives an operational overview of Common Hardware Reference Platform systems and introduces platform specific software components that are required for operating system support. The chapter also addresses some system level requirements that are broad in nature and are fundamental to the architecture described in later chapters. Lastly, a table of requirements is presented as a guide for platform providers.

2.1 System Operation

2.1.1 Control Flow

Figure 3 on page 8 is an example of typical phases of operation from power-on to full system operation to termination. This section gives an overview of the processes involved in moving through these phases of operation. This section will introduce concepts and terms that will be explained in more detail in the following chapters. Most requirements relating to these processes will also appear in later chapters.

The discussion in this chapter will be restricted to systems with a single processor. Refer to Chapter 12, “The Symmetric Multiprocessor Option,” on page 215 for the unique requirements relating to multiprocessor systems.

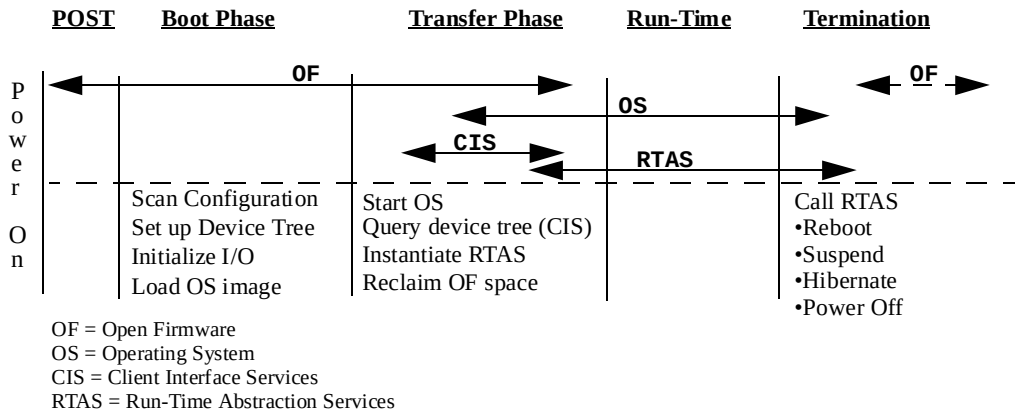


Figure 3. Phases of Operation (example)

2.1.2 POST

Power On Self Test (POST) is the process by which the firmware tests those areas of the hardware that are critical to its ability to carry out the boot process. It is not intended to be all-inclusive or to be sophisticated in how it relates to the user. Diagnostics with these characteristics will generally be provided as a service aid.

2.1.3 Boot Phase

The following sections describe the boot phase of operation. The fundamental requirements of the boot phase are:

1. Identify and configure system components.
2. Generate a Device Tree.
3. Initialize/reset system components.
4. Locate an operating system boot image.
5. Load the boot image into memory.

Personal Use Copy - Not for Reproduction

2.1.3.1 Identify and Configure System Components

The firmware must, by various means, become aware of every component in the system and configure or reset those components into a known state. Components include all bridges and device controllers, but may exclude devices that are not involved in the boot process.

Firmware is generally written with a hardware platform in mind, so that some components and their configuration data can be hardcoded. Examples of these components are: number and type of processors, cache characteristics, and the use of components on the planar. This hardcoding is not a requirement, only a practical approach to a part of this task.

Certain system information must come from “walking” the I/O buses. This is a technique that will yield identification of I/O device controllers and bridges that reside on modern, well-behaved buses such as PCI. In general, it is not possible to walk the ISA bus.

ISA configuration requires either Plug-n-Play (PnP) protocols and adapters or special handling by the system firmware or software. This architecture requires a portion of non-volatile memory (name = *isa-config*) for storage of ISA configuration data which allows firmware and operating systems to record and retrieve this data as required (see Section 8.4.4, “Configuration (0x71),” on page 145).

2.1.3.2 Generate a Device Tree

The firmware will build a device tree. The operating system will gain access to the device tree through Client Interface Services (CIS).

Certain configuration information (configuration variables) may be stored in non-volatile memory. They will be stored under the partition names *of-config* or *common*, depending on the nature of the information (see Chapter 8, “Non-Volatile Memory,” on page 141).

2.1.3.3 Initialize/Reset System Components

Operating Systems require devices to be in a known state at the time control is transferred from the firmware. Firmware may gain control with the hardware in various states depending on what has initiated the boot process.

- Normal boot: Initiated by a power-on sequence; all devices and registers begin in a hardware reset state.
- Wakeup from Hibernation: Should be indistinguishable from a normal boot.

Personal Use Copy - Not for Reproduction

- Resume from Suspend: Devices that were left powered on during suspend may be in the state left by software or in a hardware reset state. Firmware must be able to deal with this and alter the state as necessary to conform to the conditions below.
- Reboot: Device state is unpredictable. Firmware must be able to deal with this and alter the state as necessary to conform to the conditions below.

The hardware reset state for a device is an *inactive* state. An *inactive* state is defined as a state that allows no system level activity; there can be no bus activity, interrupt requests, or DMA requests possible from a device that is in a reset state. Since operating systems may configure devices in a manner that requires very specific control over these functions to avoid transitory resource conflicts, these functions should be disabled at the *device* and not at a central controlling agent (for example, the interrupt controller). Devices that do not share any resources may have these resources disabled at a system level (for example, keyboard interrupts may be disabled at the interrupt controller in standard configurations).

Requirements:

2–1. I/O devices must adhere to the reset states given in Table 1 on page 10 when control of the system is passed from firmware to an operating system.

Table 1. I/O Device Reset States

Bus	Devices Left Open by Open Firmware	Other Devices
PCI	Interrupts not active No outstanding I/O operations Device is configured	The device is *inactive*: •I/O access response disabled •Memory access response disabled •PCI master access disabled •Interrupts not active Device is reset (see note)
ISA	Configured per NVRAM •Tri-state interrupts (inactive) •Tri-state DMA (inactive)	Hardware reset state (see note) •Tri-state interrupts (inactive) •Tri-state DMA (inactive)
PCMCIA	Configured per OF device tree	Powered off (see note)

Note: May optionally be *configured*, but must be *inactive*.

2–2. Prior to passing control to the operating system, firmware or hardware must initialize all registers not visible to the operating system to a state

Personal Use Copy - Not for Reproduction

that is consistent with the system view represented by the OF device tree.

2–3. Hardware must provide a mechanism, callable by software, to hard reset all processors and I/O subsystems in order to facilitate the implementation of the RTAS *system-reboot* function.

2–4. **For the Power Management option:** Hardware must provide a software-controllable mechanism to reset the I/O subsystems without affecting the state of the processors or memory to facilitate implementation of the RTAS *hibernate* function.

Hardware Implementation Note: Requirement 2–4 provides RTAS with a means to reset I/O that can only be reset by hardware independent of the device.

2.1.3.4 Locate an OS Boot Image

The operating system boot image is located as described in the *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10]. A device and filename can be specified directly from the command interpreter (the *boot* command) or OF will locate the image through an automatic boot process controlled by configuration variables. Once a boot image is located, the device path is set in the device tree as the *bootpath* property of the */chosen* node.

If multi-boot (multiple bootable operating systems residing on the same hardware platform) is supported, a configuration variable instructs the firmware to display a multi-boot menu from which the OS and bootpath are selected.

2.1.3.5 Load the Boot Image into Memory

After locating the image, it is loaded into memory at the location given by a configuration variable or as specified by the OS load image format.

2.1.4 Transfer Phase

The image is prepared for execution by checking it against certain configuration variables; this may result in a reboot (for example, *little-endian?*).

Personal Use Copy - Not for Reproduction

Once the operating system gains control, it may use the CIS interface to learn about the hardware platform contents and configuration. The OS will generally build its own version of this configuration data and may discard the OF code and device tree in order to reclaim the space used by Open Firmware. A set of platform-specific functions are provided by Run-Time Abstraction Services (RTAS) which is instantiated by the OS invoking the *instantiate-rtas* method of the RTAS Open Firmware device tree node.

2.1.5 Run-Time

During run-time, the operating system has control of the system and will have RTAS instantiated to provide low-level hardware-specific functions.

2.1.6 Termination

Termination is the phase during which the operating system yields control of the system and may return control to the firmware depending on the nature of the terminating condition.

2.1.6.1 Power Off

If the user activates the system power switch, power may be removed from the hardware immediately (switch directly controls the power supply) or software may be given an opportunity to bring the system down in an orderly manner (power management control of the power switch).

If power is removed from the hardware immediately, the operating system will lose control of the system in an undetermined state. Any I/O underway will be involuntarily aborted and there is potential for data loss or system damage. A shut-down process prior to power removal is highly recommended and has become standard in most modern operating systems.

In most power managed systems, power switch activation is fielded as a power management interrupt and the operating system (through RTAS) is able to quiesce the system before removing power. The operating system may turn off system power using the RTAS *power-off* function.

2.1.6.2 Suspend

The suspend state is entered upon the occurrence of various events in certain power managed systems. During suspend, only the contents of main memory

Personal Use Copy - Not for Reproduction

are preserved. Since the processor and all non-essential I/O may be turned off, a reboot-like process is required to restore the system.

To enter a suspend state, the operating system (OS) must call the RTAS *suspend* function. When system operation resumes, the firmware returns to the caller of the *suspend* function.

2.1.6.3 Hibernate

The hibernation state is entered upon the occurrence of various events in certain power managed systems. During hibernation, the system is effectively powered down. Memory contents are stored (normally to disk) and a reboot is required to restore the system.

RTAS provides a *hibernate* function to assist operating systems with saving the storage image. When the system reboots, the OS determines that it is waking up from hibernation and takes the appropriate actions to restore the system.

2.1.6.4 Reboot

The operating system may cause the system to reset and reboot by calling the RTAS *system-reboot* function.

2.2 Firmware

Requirements:

2–5. Platforms must implement Open Firmware as defined in *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10].

2–6. Platforms must implement the Run-Time Abstraction Services (RTAS) as described in Chapter 7, “Run-Time Abstraction Services,” on page 91.

2–7. Operating systems must use Open Firmware and the RTAS functions to be compatible with all platforms.

Hardware Implementation Note: Legacy Firmware may be implemented in addition to the above requirements. Legacy Firmware is described in the *PowerPC Reference Platform Specification,* Version 1.1 [28] including any addenda on the PowerPC forum on CompuServe.

Personal Use Copy - Not for Reproduction

Legacy firmware may be required by some operating systems until new versions are released which interface to Open Firmware.

Software Implementation Note: Operating systems may interface directly with the hardware without calling the RTAS functions until new versions of those operating systems are released which interface to RTAS.

Software Implementation Note: In addition to providing the RTAS interface, operating systems may provide direct hardware interfaces. An operating system which utilizes a platform specific interface instead of an RTAS function is called a “platform aware” operating system.

2.3 Bi-Endian Support

In Little-Endian mode, the address of a word in memory is the address of the least significant byte (the “little” end) of the word. Increasing memory addresses will approach the most significant byte of the word. In Big-Endian systems, the address of a word in memory is the address of the most significant byte (the “big” end) of the word. Increasing memory addresses will approach the least significant byte of the word.

Operating systems written to work in either Little-Endian or Big-Endian mode can operate on Common Hardware Reference Platform implementations. Firmware will initialize the platform to either Little-Endian (MSR_{LE}=1) or Big-Endian (MSR_{LE}=0) mode at boot time, depending on the need of the operating system.

This subject is covered in more detail in tutorials on endianess in *The PowerPC Architecture* [1], in *How to Create Endian-Neutral Software for Portability* [3], and in Appendix C, “Bi-Endian Designs,” on page 265.

Requirements:

2–8. Platforms must support operation in Big-Endian mode.

2–9. Platforms must support operation in Little-Endian mode.

2.4 64-Bit Addressing Support

A *32-bit-addressing platform* is designed to operate with a 32-bit real address space (4 GB) only. Such a platform may contain either of the two processor (32-bit or 64-bit) implementations defined by the PowerPC microprocessor architecture, subject to the requirement 4–8 that a 64-bit processor implementa-

Personal Use Copy - Not for Reproduction

tion also implement the 32-bit-compatibility bridge architecture. Given this last requirement, the only difference in programming model between a 32-bit and 64-bit processor in a 32-bit-addressing platform is in the format of page table entries. Operating systems are required to support both formats (reference requirement 4–7).

A *64-bit-addressing-capable platform* is defined as one capable of supporting memory configured above 4 GB (greater than 32 bits of real addressing). This means that all hardware elements in the topology down to the Host Bridges are capable of dealing with a real address range greater than 32 bits, and all Host Bridges are capable of providing a translation mechanism for 32-bit I/O bus addresses. The initial mode of operation (when the operating system receives control from firmware) is 32-bit execution mode, including Host Bridges. All of System Memory is configured and reported by OF. (Note that this architecture allows only System Memory above 4 GB in the real address space.)

A *32-bit operating system* will utilize memory in the range up to 4 GB, and will operate a 64-bit-addressing-capable platform as a 32-bit-addressing platform with a 64-bit processor in it, regardless of how much memory is present. Memory above 4 GB will be unusable.

A *64-bit-addressing-aware operating system* is an OS that can deal with a real address space larger than 4GB. It must handle the 64-bit processor page table format (required of all OSs), and must understand Host Bridge mechanisms and Host Bridge Open Firmware methods for supporting System Memory greater than 4 GB.

On a 64-bit-addressing-capable platform with no memory configured above 4 GB, it is expected (although not required) that a 64-bit-addressing-aware OS would not enable 64-bit addressing in Host Bridges, and would operate the platform as if it were a 32-bit-addressing platform with a 64-bit processor. On a 64-bit-addressing-capable platform with memory configured above 4 GB, it is expected that a 64-bit-addressing-aware OS would recognize and use the memory above 4 GB (implying the enablement of 64-bit addressing in the Host Bridge(s)).

Software Implementation Note: The 64-bit addressing elements of the configuration are defined in the OF device tree as follows:

1. The *64-bit* property of the *cpu* nodes. This property is used to define the presence of a 64-bit PowerPC processor and the associated page table format (independent of the real addressing capability of the entire platform).

2. The *64-bit-addressing* property of the Host Bridge node(s). This property is required to be present on all HB nodes when the memory configuration has memory above 4 GB. It is reported so that

Personal Use Copy - Not for Reproduction

64-bit-addressing-aware OSs may recognize a 64-bit-addressing-capable platform and may choose to enable 64-bit addressing in Host Bridges even when no memory is configured above 4 GB. Open Firmware supplies a method on 64-bit-addressing-capable platforms (*set-64-bit-addressing*) which an OS uses to enable HB addressing of the full real address space above 4 GB.

3. Memory being reported above 4 GB in the *memory* node(s). This implies that the platform is 64-bit-addressing-capable.

2.5 Minimum System Requirements

This section summarizes the minimum hardware and functionality required for Common Hardware Reference Platform compliance.

The term *portable* is used to describe that class of systems that is primarily battery powered and is easily carried by its user.

The term *personal* is used to describe that class of systems that is bound to a specific work area due to its size or power source, and whose use is generally restricted to a single direct user or a small set of users.

The term *server* is used to describe that class of systems that supports a multi-user environment, providing a particular service such as file storage, software repository, or remote processing capability.

Each of these classes have unique requirements due to the way they are used or what operating systems they generally employ and, for this reason, the requirements that follow are based on the type of system being developed.

Requirements:

2–10. Platforms must contain the minimum required components given in Table 2 on page 19.

2–11. Portable and personal CHRP operating systems must support all the following:

a. PS/2™ *and* ADB™ keyboard/mouse interfaces.

b. 16550-compatible *and* SCC serial ports.

c. SCSI *and* IDE hard disk interfaces.

Software Implementation Note: Server-targeted CHRP operating systems are not subject to requirement 2–11 and may limit this I/O support to the intended target platform. Server-targeted CHRP operating

Personal Use Copy - Not for Reproduction

systems may, however, run on appropriately configured portable and/or personal platforms.

Hardware Implementation Note: Platform providers should consult OS providers for specific requirements of individual server-targeted operating systems.

2.5.1 Table Description

Minimum requirements for Common Hardware Reference Platforms are summarized in Table 2 on page 19.

2.5.1.1 Subsystem

Major subsystems are listed. This enumeration does not preclude the addition of new subsystems or the enhancement of listed subsystems; however, these must meet the definition and requirements given in Section 2.6, “Options and Extensions,” on page 20.

2.5.1.2 Specification

This column lists the *specification* of subsystems in a standard platform.

2.5.1.3 Portable, Personal, Server

These columns categorize the specifications according to the following legend.
Legend:

- R: Required.
- R*: Entries categorized as R* denote specifications from which platform vendors may *choose* one or more to satisfy the requirement. All R* specifications are supported by portable and personal CHRP operating systems.
- O: Optional (refer to the definition of optional given in Section 2.6, “Options and Extensions,” on page 20).
- M: Required for systems designed to run Apple Mac OS.

Personal Use Copy - Not for Reproduction

The categorization of the subsystem is given in the first line of each table entry; each individual specification is categorized separately in subsequent lines. If a subsystem is listed as optional, there may be specifications that are required *if* the particular subsystem is provided. This would be represented by an “O” in the first row of the entry with “R”s listed for the required specifications.If a subsystem is listed as required, a list of optional specifications means that a choice may be made from the list to fulfill the requirement.

Hardware Implementation Note: Platform providers are not required to provide all the I/O mechanisms labeled with R*. However, some application security and licensing measures will depend on them and platform limitations may result from their omission.

Hardware and Software Implementation Note: Server-targeted CHRP operating systems are not required to support all of the I/O mechanisms labeled with R* and thus platform providers should consult OS providers for specific requirements of individual server-targeted CHRP operating systems.

2.5.1.4 Description

The *description* column gives additional information about the requirements. Two documents are referenced extensively:

1. [20]*PowerPC Microprocessor Hardware Reference Platform: I/O Device Reference.* Use this document for additional information on the annotated specifications.
2. [23] *Macintosh Technology in the Common Hardware Reference Platform.* Use this document for specific physical, timing, and electrical requirements of the annotated specifications.

Personal Use Copy - Not for Reproduction

Table 2. Summary of Minimum Platform Requirements

Subsystem	Specification	Portable	Personal	Server	Description
Processor	PowerPC microprocessor	R	R	R	Further defined in Chapter 4, “Processor and Memory,” on page 49. Contact OS vendor for specific OS dependencies.
Minimum System Memory	8 MB expandable to at least 32 MB	R	R	R	Operating systems have specific requirements; contact OS vendor. All operating systems will run with 32 MB installed.
OS ROM or Socket	4 MB	R	R	M	[23]
Firmware Storage	Sized as needed	R	R	R	Most firmware implementations will fit in 256 KB. The preferred implementation is FLASH memory and should be processor-writable.
Non-Volatile Memory	8 KB	R	R	R	8 KB is sufficient for a system with a single operating system installed. Multi-boot systems must evaluate the need for more space. Each operating system may require up to 1 KB. See Chapter 8, “Non-Volatile Memory,” on page 141 for more information on Non-Volatile Memory.
External Cache		O	O	O	An external cache is recommended for performance. See Section 4.2.4, “Cache Memory,” on page 65 for more information.
Hard Disk	Sized as needed SCSI IDE PC Card	R O O O	R O O O	R O O O	[20], [23] “Medialess” systems are not covered by this architecture.
Floppy	 3.5” 1.44 MB MFM Media sense Auto eject Manual eject	O R R R R	O R R R R	O R R M R	[20] Not required, but a means to attach for software installation must be provided. This may be through a provided connector or over a network. Media sense: Implementations must allow polling of the drive up to 100x per second to determine the presence of media in the drive. A method for manual ejection of floppies is required.
CD-ROM	 4X speed ISO9660 Multi-session	O R R R	O R R R	O R R R	[20] Not required, but a means to attach for software installation must be provided. This may be through a provided connector or over a network
Alphanumeric Input Device	 PS/2 Keyboard interface ADB Terminal	R R* R* O	R R* R* O	O O O O	[20], [23] Servers do not require a keyboard for normal operation, however a means to attach an A/N Input Device must be provided; an ASCII terminal is a example of such a device. Keyboards must be capable of generating at least 101 scan codes.
Pointing Device	 2 buttons PS/2 interface (see note 1) ADB (see note 1)	R R R* R*	R R R* R*	O O O O	[20], [23] If a platform includes a keyboard, it must also include a pointing device with the functionality of at least a 2-button mouse

Personal Use Copy - Not for Reproduction

Table 2. Summary of Minimum Platform Requirements ***(Continued)***

Subsystem	Specification	Portable	Personal	Server	Description
Audio	16bit Stereo, 22.05 and 44.1 KHz, full duplex.	R	R	O	Servers do not require high quality audio. The programming model for this basic audio capability is specified in [20].
Tone		R	R	R	Must be implemented with separate hardware from system audio; must be capable of concurrent operation with system audio. May share speaker with system audio. See Table 29 on page 120 for the RTAS interface for tone generation.
Graphics		R	R	O	Servers do not require graphics during normal operation and need not support a graphics subsystem.
	1024x768	O	R	O	Portables may provide screen resolution in accordance
	Bi-Endian	R	R	O	with current state-of-the-art LCD technology.
	640x480x8 LFB	R	R	O	
	VGA	O	O	O	Some operating environments will prefer platforms that provide hardware support of VGA for performance considerations.
Real Time Clock		R	R	R	The RTC must be non-volatile and run continuously with a resolution of at least one second. See Section 7.3.3, “Time of Day,” on page 106 for further information.
VIA		R	R	M	[23]
Serial Port		O	R	O	
	16550	O	R*	O	[20]
	SCC	O	R*	O	[23]
Parallel Port	P1284 +ECP Mode	O	R	O	[20]
Network		O	O	O	
Interrupt Controller	Open PIC	R	R	R	[20]
	8259 tree	R	R	R	8259 required for ISA compatibility.
Direct Memory Access (DMA)	ISA	R	R	R	[20] ISA DMA must be 82378ZB compatible.
Power Management	States as defined.	R	O	O	Refer to Chapter 11, “Power Management,” on page 185
Infrared	IrDA	O	O	O	

2.6 Options and Extensions

An *option* is a system resource that is covered by this architecture, but is not required to be implemented. Platforms that implement options are required to conform to the definitions in this architecture, so that an aware OS environment can recognize and support them.

An *extension* is a resource that is outside the scope of, but does not contradict, this architecture. It is a resource which cannot be depended on by software to be present on a platform; operating systems are not required to be aware of, or capable of supporting, these resources.

Personal Use Copy - Not for Reproduction

Options and extensions must be dormant or invisible in the presence of a non-aware OS environment. In general this means they are initialized to an inactive state, and can be activated by an aware OS.

Requirements:

2–12. An option, if implemented, must operate as specified in this architecture.

2–13. Extensions, if implemented, must come up passively, such that an operating system which does not use the extension will not be affected.

2–14. Options, if implemented, must come up passively or as otherwise specified in this architecture.

2–15. An extension, if implemented, must not contradict this architecture.

Options provided by this architecture are listed below. Open Firmware configuration variable and properties associated with these options are given in *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10].

1. Power Management
2. Symmetrical Multiprocessing
3. 64 bit support
4. PC Emulation
5. Multi-boot

Personal Use Copy - Not for Reproduction

Blank page when cut - 22

System Address Map

3

The address map of a Common Hardware Reference Platform is made up of several distinct areas. These areas are one of five basic types. Each of these types has its own general characteristics such as coherency, alignment, size restrictions, variability of starting address and size, the system action on access of the area, and so on. This chapter will give details on some of those characteristics, and other chapters will define the other characteristics. The variable characteristics of these areas will be reported to the operating system via properties in the OF device tree.

3.1 Address Areas

The following is a definition of the five areas and some of their characteristics:

- *System Memory* refers to memory which forms a coherency domain with respect to the PowerPC processor(s) that execute application software on a system. See Section 4.2.2.1, “Memory Coherence,” on page 57 for details on aspects of coherence. *System Memory Spaces* refer to one or more pieces that together form the System Memory.
- *Peripheral Memory Space* refers to a range of real addresses which are assigned to the Memory Space of a Host Bridge (HB) and which are sufficient to contain all of the *Load* and *Store* address space requirements of the devices in the Memory Space of the I/O bus that is generated by the HB. The frame buffer of a graphics adapter is an example of a device which may reside in the Peripheral Memory Space. In addition to a Memory Space, many

types of I/O buses have a separate address space called the I/O Space. An HB which generates such I/O buses must decode another address range, the Peripheral I/O Space.[1]

- *Peripheral I/O Space* refers to a range of real addresses which are assigned to the I/O Space of an HB and which are sufficient to contain all of the *Load* and *Store* address space requirements of all the devices in the I/O Space of the I/O bus that is generated by the HB, including the expansion of those addresses due to the discontiguous I/O address mode. A keyboard controller is an example of a device which may require Peripheral I/O Space addresses.
- *System Control Area* refers to a range of addresses which contains the system ROM(s) (firmware ROM and, if implemented, OS-ROM) and an unarchitected, reserved, platform-dependent area used by firmware and Run-Time Abstraction Services for control of the platform. The ROM areas are defined by the OF properties in the *openprom* and *os-rom* nodes of the OF device tree. The Mac OS-ROM definition, for implementations which require it, can be found in *Macintosh Technology in the Common Hardware Reference Platform* [23].
- *Undefined* refers to areas that are not one of the above four areas. The result of accessing one of these areas is defined in Chapter 10, "Error and Event Notification," on page 157 as an "invalid address" error.

In addition to these five major areas, there are several subsets of these areas for the first HB (HB0) of a platform. Note that HB0 must be a PCI Host Bridge (PHB) so HB0 will be indicated by PHB0 from here on. These subsets may be optional and may modify the general characteristics of the area. They include:

- *Compatibility holes* are address decodes on PHB0 of a platform which are defined for use by software which needs compatibility with existing PC systems relative to the requirement for specific holes in the System Memory and I/O bus Memory Spaces. These holes may also impact the design of the System Memory controller. These holes can be enabled independently, but the platform need not provide the capability to enable the processor-hole while the io-hole is disabled.
  - *io-hole* is an address range on the I/O bus side of PHB0. Platforms which provide support for Video Graphics Array (VGA) compatibility will support the io-hole. When enabled, this creates a hole in the first System Memory Space (as viewed by the I/O) from 640 KB to (1 MB - 1), and

[1] A peripheral space may also include a "configuration" address space. The configuration space will be abstracted by a Run-Time Abstraction Service (for example, see Section 7.3.5, "PCI Configuration Space," on page 114).

Personal Use Copy - Not for Reproduction

accesses by the I/O to this address range will remain on the I/O bus (allowing I/O device to device transfers in this address range). The existence of this hole is indicated by the existence of the *io-hole* node property in the PHB0 node of the OF device tree. The hole is activated or deactivated by the *set-io-hole* OF method.

— *processor-hole* is an optional address range on the processor side of PHB0. When enabled, this creates a hole in the first System Memory Space (as viewed by the processor) from 640 KB to (768 KB - 1), and causes accesses in this address space to go to the Memory Space of the I/O bus of PHB0, in the 640 KB to (768 KB - 1) range (thus giving the Peripheral Memory Space characteristics to this address range in System Memory). The existence of this optional hole is indicated by the existence of the *processor-hole* node property in the PHB0 node of the OF device tree. The hole is activated or deactivated by the *set-processor-hole* OF method.

■ *Initial memory alias spaces* are areas in the Peripheral Memory Space of PHB0 which allow accessing the first (initial) 16 MB address range (0 to (16 MB - 1)) on one side of PHB0 from a different address range on the other side of PHB0. These spaces are required to be enabled when either of the compatibility holes are enabled. It is necessary to enable these alias spaces on PHB0 in order to support devices which must be configured in the 0 to (16 MB - 1) range. The peripheral-memory-alias and system-memory-alias spaces are enabled or disabled as a pair, with one exception, via the *set-initial-aliases* OF method for PHB0. The exception is when the PC Emulation option is enabled (see Section 3.3, “PC Emulation Option,” on page 44). The state of the alias spaces (enabled or disabled) is indicated by the *initial-memory-alias* property of PHB0 node. The initial memory alias spaces are entirely contained within the Peripheral Memory Space.

— *peripheral-memory-alias space* is an address space on the processor side of PHB0. When enabled, this address range allocates the 16 MB of address space at the high end of the Peripheral Memory Space for PHB0, and this is used to address the first 16 MB of the Memory Space on the I/O side of PHB0. This address range is used to access I/O devices which cannot have their addresses set in the Peripheral Memory Space. For example, ISA devices or ISA-compatible devices must be setup to have addresses below 16 MB.

— *system-memory-alias space* is an address space on the I/O side of PHB0. When enabled, this address range, which is at the same address in the I/O bus Memory Space as the peripheral-memory-alias on the processor side, can be used to address the first 16 MB of System Memory. This ad-

Personal Use Copy - Not for Reproduction

dress range is used, for example, to access System Memory in the 640 KB to (1 MB - 1) range when the io-hole is enabled (see also Table 10 on page 79 for other uses).

Table 3 on page 26 shows the valid combinations of these alias and hole areas for normal system operations. Of these combinations, the rows which indicate the valid states when the OF passes control to the operating system are States 1 and 2, depending on whether the initial memory alias spaces are enabled or not.

Table 3. Valid Hardware Combinations of Compatibility Holes and Initial Memory Alias Spaces

State Number	PC Emulation Option	Peripheral-memory-alias	System-memory-alias	Processor-hole	Io-hole
1	disabled	disabled	disabled	disabled	disabled
2	disabled	enabled	enabled	disabled	disabled
3	disabled	enabled	enabled	disabled	enabled
4	disabled	enabled	enabled	enabled	enabled
5	enabled	enabled	disabled	disabled	enabled
6	enabled	enabled	disabled	enabled	enabled

Software Implementation Note: Software should be careful to assure that no I/O operations using the alias spaces or holes are occurring during the transition period from one state in the table to another. Software may pass through states which do not appear in the table during the transition from one state in the table to another.

In describing the characteristics of these various areas, it will be convenient to have a nomenclature for the various boundary addresses. Table 4 on page 26 defines the labels which will be used when describing the various address ranges. Note that “bottom” refers to the smallest address of the range and “top” refers to the largest address.

Table 4. Map Legend

Label	Description
BIOn	Bottom of Peripheral I/O Space for HBn (n=0, 1, 2,...). The OF property *ranges* in the device tree for HBn will contain the value of BIOn.

Personal Use Copy - Not for Reproduction

Table 4. Map Legend ***(Continued)***

Label	Description
TIOn	Top of Peripheral I/O Space for HBn (n=0, 1, 2,...). The value of TIOn can be determined by adding the size of the area as found in the OF property *ranges* in the OF device tree for HBn to the value of BIOn found in that same property.
BPMn	Bottom of Peripheral Memory Space for HBn (n=0, 1, 2,...). The OF property *ranges* in the device tree for HBn will contain the value of BPMn.
TPMn	Top of Peripheral Memory Space for HBn (n=0, 1, 2,...). For PHB0, when the initial memory alias spaces are enabled, the Peripheral Memory Space is split into two pieces in the OF device tree, as indicated by the *ranges* property in the OF device tree for the PHB0 node; BPM0 to (BIM - 1) and BIM to TPM0. When the initial memory alias spaces are disabled or for other HBs other than PHB0, the Peripheral Memory Space is in one piece in the OF device tree, as indicated by the *ranges* property in the OF device tree for HBn node; BPMn to TPMn. The value of TPMn can be determined by either adding the size of the area as found in the OF property *ranges* in the OF device tree for HBn to the value of BPMn found in that same property or by adding the size of the area as found in the OF property *ranges* in the OF device tree for HBn to the value of BIM found in that same property, depending on whether the initial memory alias spaces are disabled or enabled, respectively.
BIM	Bottom of initial memory alias space for PHB0. Corresponding top of initial memory alias space is equal to TPM0. The value of BIM is (TPM0 - 16 MB + 1). If the initial memory alias spaces are enabled, then the value of BIM is indicated by the *ranges* property in the OF device tree for PHB0 node, otherwise it can be calculated by (TPM0 - 16 MB + 1) (see explanation under TPMn, above).
BSCA	Bottom of System Control Area. Corresponding top of the System Control Area is (4 GB - 1). The OF property *reg* in the OF device tree for the System Control Area node contains the value of BSCA.
BSMn	Bottom of System Memory Space n (n=0, 1, 2,...); BSM0 = 0. The OF property *reg* in the OF device tree for the Memory Controller node contains the value of BSMn.
TSMn	Top of System Memory Space n (n=0, 1, 2,...). The value of TSMn can be determined by adding the value of BSMn as found in the Memory Controller node of the OF device tree to the value of the size of that area, found in the same property.

Operating systems and other software should not use fixed addresses for these various areas. A given platform may, however, make certain of these addresses unchangeable. Each of these areas will be defined in the OF device tree in the node of the appropriate controller. This will give platforms maximum flexibility in implementing the System Address Map to meet their market requirements. Some of the values set up for these areas by the OF may be changeable by an OF method. Refer to the chapters describing the various controller architectures for more information.

Personal Use Copy - Not for Reproduction

Requirements:

3–1. All unavailable addresses in the Peripheral Memory and Peripheral I/O Spaces must be conveyed in the OF device tree.

a. A device type of *reserved* must be used to specify areas which are not to be used by software and not otherwise reported by OF.

b. Shadow aliases must be communicated as specified by the appropriate OF bus binding.

3–2. There must not be any address generated by the system which causes the system to hang.

Hardware Implementation Note: The reason for requirement 3–1 is to reserve address space for registers used only by the firmware or addresses which are used only by the hardware.

3.2 Address Decoding and Translation

In general, different components in the hardware are going to decode the address ranges for the various areas. In some cases the component may be required to translate the address to a new address as it passes through the component. The requirements, below, describe the various system address decodes and, where appropriate, what address transforms take place outside of the processor. The details are clarified by the example system address maps which can be found in Figure 4 on page 33, Figure 5 on page 34, and Figure 6 on page 35. Note that Direct Memory Access (DMA) operations can either be controlled directly by the device doing the I/O operation (that is, an I/O bus master operation), or may be controlled by a controller which is separate from the device (that is, a third party DMA operation).

The HB requirements in this section refer to HBs which are defined by the CHRP architecture. Currently, there is only one HB defined by the CHRP architecture, and that is the PHB. HBs which implement I/O buses other than those defined by the CHRP architecture are encouraged to use this addressing model. However, HBs which are implemented as extensions, are in the disabled (passive) state when control is passed to the operating system, and therefore need not implement the same programming model.

Requirements:

3–3. Processor *Load* and *Store* operations must be routed and translated as shown in Table 5 on page 32.

Personal Use Copy - Not for Reproduction

3–4. DMA operations in the Memory Space of an I/O bus must be routed and translated as shown in Table 6 on page 32.

3–5. In addition to Table 6 on page 32, if the platform is designed to support System Memory configured at an address of 4 GB or above, then the Translation Control Entry (TCE) translation mechanism, described in Section 3.2.2, “Translation of 32-Bit DMA Addresses in 64-Bit Addressing Systems,” on page 38 must be implemented on all HBs. If the operating system enables the platform to access System Memory at or above 4 GB, then TCEs must be used to translate all DMA operations in the Memory Space of the I/O bus of the HB which use a 32-bit address.

3–6. An HB must not act as a target for operations in the I/O Space of an I/O bus.

3–7. The following are the System Control Area requirements:

a. Each platform must have exactly one System Control Area.

b. The System Control Area must not overlap with the System Memory Space(s), Peripheral Memory Space(s), or the Peripheral I/O Space(s) in the platform.

3–8. The following are the System Memory Space requirements:

a. Each platform must have at least one System Memory Space.

b. The System Memory Space(s) must not overlap with the Peripheral I/O Space(s), Peripheral Memory Space(s), the System Control Area, or other System Memory Space(s) in the platform.

c. The first System Memory Space must start at address 0 (BSM0 = 0), must be at least 16 MB before a second System Memory Space is added, and must be contiguous except that if the processor-hole is enabled, then there will be a hole from 640 KB to (768 KB - 1).

d. Each of the additional (optional) System Memory Space(s) must start on a 4 KB boundary.

e. Each of the additional (optional) System Memory Space(s) must be contiguous within itself.

f. There must be at most eight System Memory Spaces below BSCA and at most eight at or above 4 GB.

Personal Use Copy - Not for Reproduction

g. If multiple System Memory Spaces exist below BSCA, then they must not have any Peripheral Memory or Peripheral I/O Spaces interspersed between them.

3–9. The following are the Peripheral Memory Space requirements:

a. The Peripheral Memory Space(s) must not overlap with the System Memory Space(s), Peripheral I/O Space(s), the System Control Area, or other Peripheral Memory Space(s) in the platform.

b. When the OF passes control to the operating system, there must be no I/O device configured in the address range (TPM0 - 16 MB + 1) to TPM0 in the Peripheral Memory Space for PHB0, in order to assure that 16 MB of space is available for the initial memory alias spaces.

c. The size of each Peripheral Memory Space (TPMn - BPMn + 1) must be a power of two for sizes up to and including 256 MB, with the minimum size being 16 MB, and an integer multiple of 256 MB plus a power of two which is greater than or equal to 16 MB for sizes greater than 256 MB (for example, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB, 256 + 16 MB, 256 + 32 MB,..., 512 + 16 MB,...).

d. The boundary alignment for each Peripheral Memory Space must be an integer multiple of the size of the space up to and including 256 MB and must be an integer multiple of 256 MB for sizes greater than 256 MB.

e. There must be exactly one Peripheral Memory Space per HB.

f. The Peripheral Memory Space for every HB defined by the CHRP architecture must reside below BSCA.

3–10. The following are the Peripheral I/O Space requirements:

a. The Peripheral I/O Space(s) must not overlap with the System Memory Space(s), Peripheral Memory Space(s), the System Control Area, or other Peripheral I/O Space(s) in the platform.

b. The size of each Peripheral I/O Space (TIOn - BIOn + 1) must be a power of two with the minimum size being 8 MB (that is, sizes of 8 MB, 16 MB, 32 MB, 64 MB, and so on, are acceptable).

c. The boundary alignment for each Peripheral I/O Spaces must be an integer multiple of the size of the space.

d. There must be at most one Peripheral I/O Space per HB.

Personal Use Copy - Not for Reproduction

e. Peripheral I/O Spaces for all HBs defined by the CHRP architecture must reside below BSCA.

3–11. The following are the initial memory alias space requirements:

a. The peripheral-memory-alias and system-memory-alias spaces must be implemented on PHB0, and these initial memory alias spaces must have the capability to be enabled or disabled by the *set-initial-aliases* OF method for PHB0 node.

b. The initial state of the peripheral-memory-alias and system-memory-alias spaces, when control passes from OF to the operating system, must be enabled if there is any device on the I/O side of PHB0 which is configured in the 0 to (16 MB - 1) address range of the Memory Space of the I/O bus, and must be disabled otherwise.

3–12. The following are the compatibility hole requirements:

a. The initial state of the io-hole and the processor-hole, when control passes from OF to the operating system, must be disabled (if implemented).

b. If a platform implements the PC Emulation option, then the io-hole must be implemented. See Section 3.3, “PC Emulation Option,” on page 44 for more information.

c. Platforms for which VGA support is provided or which have an ISA bus must also implement the io-hole (see Table 2 on page 19 for the platform VGA requirements).

3–13. I/O devices which cannot be configured in the Peripheral Memory Space address range must be located on the I/O bus of PHB0, or on another I/O bus which is generated by a bridge attached to this bus, and must be configured in the 0 to (16 MB - 1) address range.

Hardware Implementation Note: OF may wish to display an error message to the user if it finds a device which cannot be configured due to requirement 3–13.

After a DMA accesses passes through an HB it is routed and translated as per Table 5 on page 32. Platforms do not need to support I/O device DMA access to the ROM areas.

The figures beginning with Figure 4 on page 33 show examples of system address maps.

Personal Use Copy - Not for Reproduction

Table 5. Processor Bus Address Space Decoding and Translation

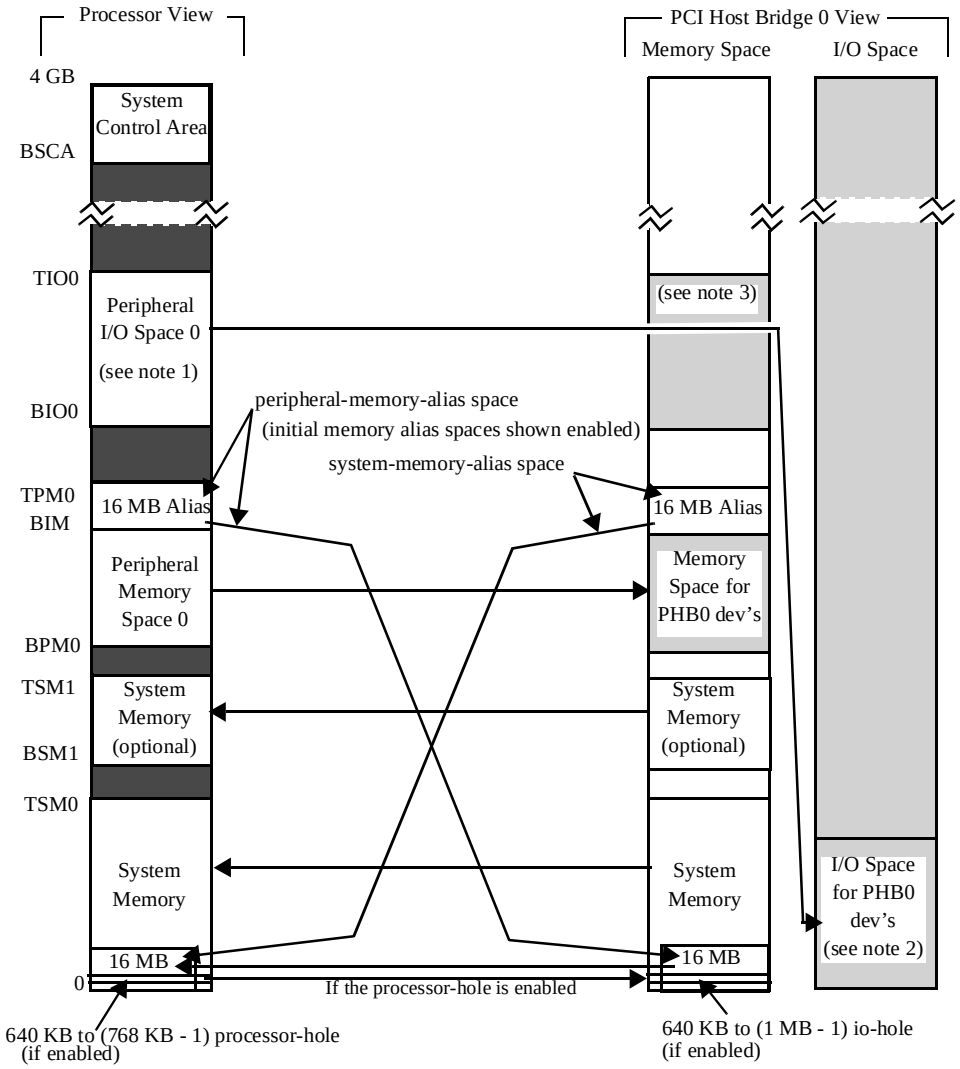
Address Range at Processor Bus	Route and Translation Requirements	Other Requirements and Comments
BSCA to (4 GB - 1)	To ROM controller or to a platform dependent area. Translation dependent on implementation.	Areas other than ROM are reserved for firmware use, or will have their address passed by the OF device tree.
BIOn to TIOn (n=0,1,...)	HBn responds. For requirements, see Section 3.2.1, “Peripheral I/O Address Translation,” on page 37.	
BPMn to TPMn (n=0, 1,...)	Send through HBn to the Memory Space of the I/O bus. For PHB0, if the peripheral-memory-alias is enabled, then translate an address in the peripheral-memory-alias space (BIM to TPM0; 16 MB range) so that this address range becomes 0 to (16 MB - 1), otherwise do not translate.	
BSMm to TSMm (m>0)	To System Memory Space m, no translation.	Can be at or above 4 GB, or below BSCA.
0 to TSM0	No translation. If the processor-hole is disabled or not implemented, then send entire range to System Memory. If processor-hole is enabled, then send 640 KB to (768 KB - 1) through PHB0 to the Memory Space of the I/O bus and send the remainder of the range to System Memory.	
All other addresses	See Chapter 10, “Error and Event Notification,” on page 157.	Access is to undefined space.

Table 6. DMA Address Decoding and Translation (I/O Bus Memory Space)

Address Range at I/O Side of HBn	Route and Translation Requirements	Other Requirements and Comments
BIOn to TIOn (note 1)	HBn either does not respond or responds and signals an error to the device (see Chapter 10, “Error and Event Notification,” on page 157).	This is an error.
BPMn to TPMn (note 1)	For PHB0, if the system-memory-alias space is enabled and the access is to the system-memory-alias space (BIM to TPM0), then translate so that this address range becomes 0 to (16 MB - 1) and send through PHB0 (note 2), otherwise the HBn does not respond.	If not an access to the system-memory-alias space, this is an I/O device to device transfer.
640 KB to 1 MB - 1 (PHB0 only)	If the io-hole is disabled or not implemented, then pass through PHB0 untranslated (note 2). If the io-hole is enabled, PHB0 does not respond.	
BSCA to (4 GB - 1)	May pass through HB untranslated, or HB may not respond.	Implementation dependent.
All other Memory Space addresses	Pass through HB. If the address is greater than or equal to 4 GB, see requirements 3–22 and 3–23 for translation requirement. If the address is less than 4 GB, then do not translate unless note 2 is applicable.	Valid accesses are to System Memory or possibly to other HB Peripheral Memory Spaces.

Note 1: n = # of HB Viewing or Receiving the Operation.
Note 2: If the HB 64-bit-addressing option is enabled, translate via the TCE before passing through the HB.

Personal Use Copy - Not for Reproduction



Note 1: BIOn + 64 KB to BIOn + (8 MB - 1) are not accessible when the discontiguous mode is disabled.

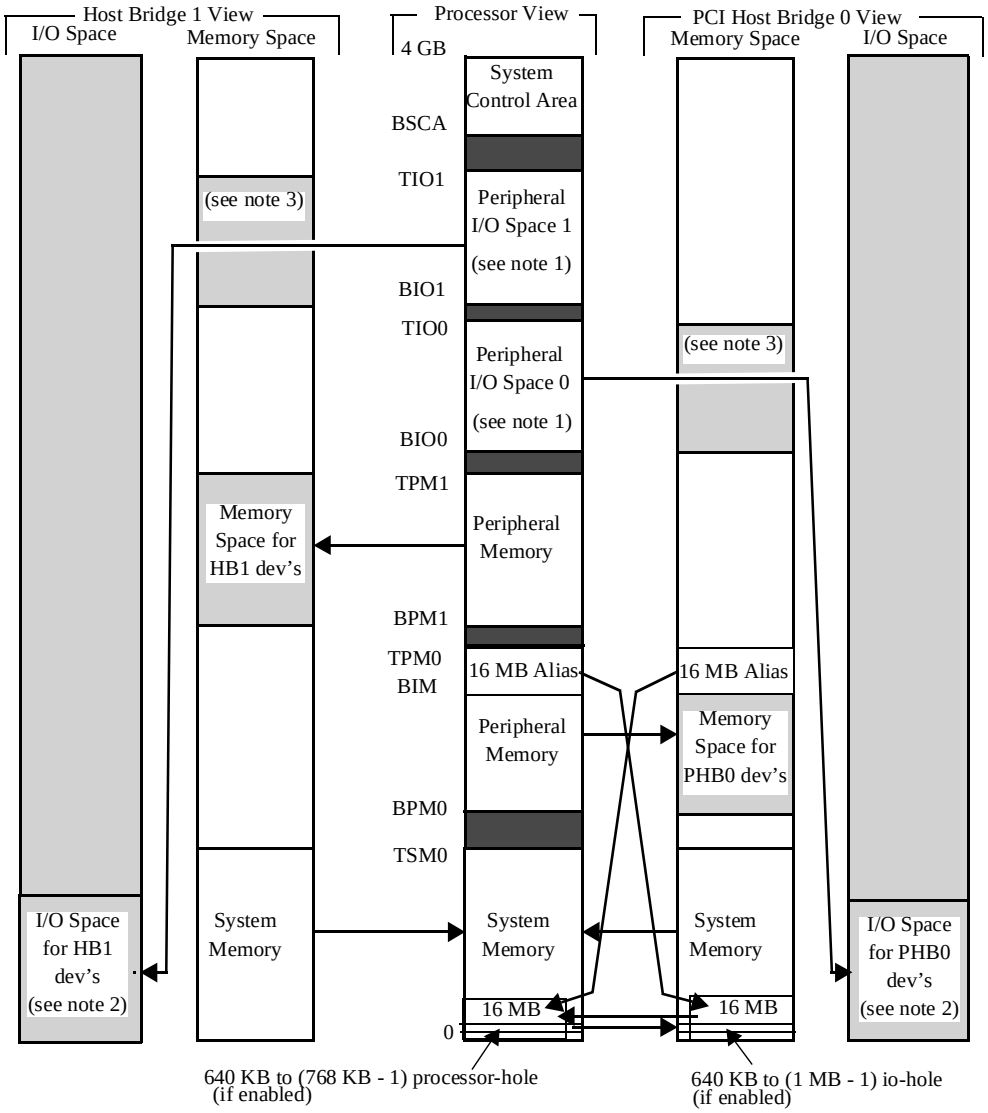
Note 2: Addresses from 64 KB to (8 MB - 1) are not accessible.

Note 3: HB may optionally respond to addresses from BIOn to TIOn and signal an error to the device.

= Addresses which are invalid for *Load* and *Store* = PHB does not respond

Figure 4. Example of an Address Map for a 32-Bit Addressing System with One PHB

Personal Use Copy - Not for Reproduction



Notes 1, 2, and 3 refer to the same notes as in Figure 4 on page 33.

= Addresses which are invalid for *Load* and *Store* = HB does not respond

Figure 5. Example of an Address Map for a 32-Bit Addressing System with Two HBs

Personal Use Copy - Not for Reproduction

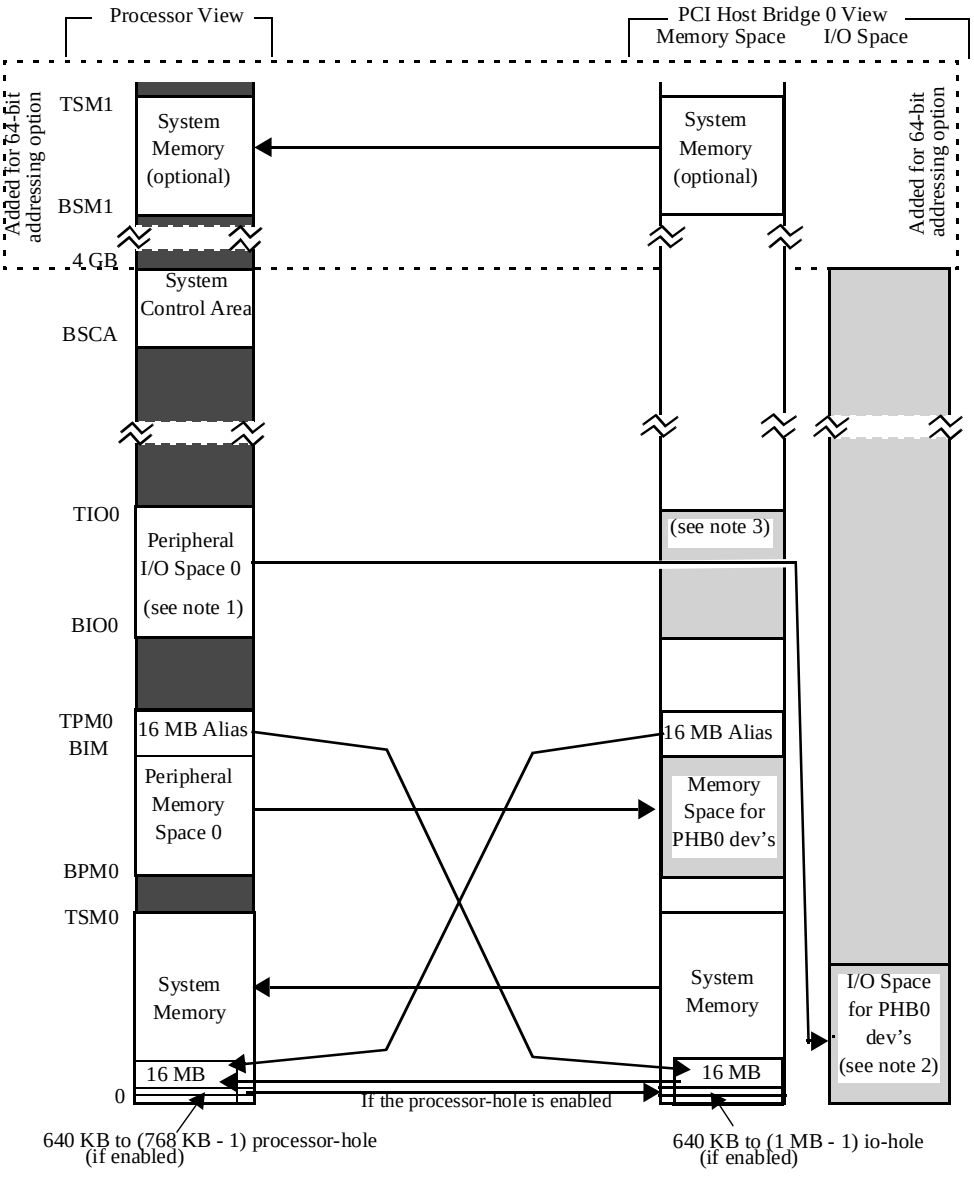


Figure 6. Example of an Address Map for a 64-Bit Addressing System with One PHB

Personal Use Copy - Not for Reproduction

The beginning addresses and sizes of the Peripheral I/O Space(s) and Peripheral Memory Space(s), as set up by OF, may have values which are fixed by the platform or may be changeable by the *change-address-map* OF method. The *change-address-map* OF method will return a failure indication if it was not able to change the map (for example, if the address area is unchangeable in the platform, or if an invalid or unsupported size or alignment was specified). The OF node property corresponding to the boundary and size of these areas is the *ranges* property of the HB node. When a platform allows these areas to be changed, they can be changed to a size and alignment according to requirements 3–9c, 3–9d, 3–10b, and 3–10c.

Hardware and Software Implementation Notes:

1. The *change-address-map* OF method does not need to support all possible sizes for the areas to which it allows changing; it is permissible to return a failure indication on some combination of sizes and alignments allowed by requirements 3–9 and 3–10, but to allow others.

2. If software receives a failure indication from *change-address-map* it may try other valid size and alignment combinations, in case the platform does not support all valid size and alignments but does support some (see note 1, above).

3. A platform which supports changing of the address map via the *change-address-map* OF method should support all sizes via that method which can be reported by OF during boot time. This allows the operating system to set the map after a wakeup back to what it was before a hibernate, in the case that the configuration and address map have changed during the time between hibernate and wakeup.

Support for the io-hole may also be needed to support certain peer to peer operations (see Table 10 on page 79).

Certain System Memory addresses must be reserved in all systems for specific uses (see Section 4.1.2, “PowerPC Microprocessor Differences,” on page 51 for more information).

Before leaving this section, it is important to note (because it is somewhat buried in various requirements, above) that all I/O devices which cannot be configured at an address in the Peripheral Spaces of a HB (for example, VGA or ISA devices which have to be configured below 16 MB), must reside in the Peripheral Space of PHB0 and be on the PCI bus generated by PHB0 or on an ISA bus or another I/O bus which is generated from that PCI bus.

Personal Use Copy - Not for Reproduction

3.2.1 Peripheral I/O Address Translation

The CHRP architecture defines a discontiguous I/O address mode which allows for the expansion of low 64 KB of the I/O Space of the I/O bus to an 8 MB address space in the processor's Peripheral I/O Space. This mode is enabled by the *set-discontiguous-io* OF method. When enabled, this expansion puts each 32 bytes of the lowest 64 KB of the I/O Space of the I/O bus in its own 4 KB page in the processor's Peripheral I/O Space. Enabling this mode allows operating systems to add protection for accesses to an I/O Space of the I/O bus on a 32 byte granularity (using the normal processor page protection mechanisms). This is illustrated in Figure 7 on page 38.

Requirements:

3–14. When the discontiguous I/O mode is enabled, Processor *Load* and *Store* addresses in the first 8 MB of Peripheral I/O Space experience the following translation. The high order seven bits of each 4 KB page offset are ignored. Thus, all page offsets wrap to the same 32 bytes within that page. Successive page numbers, starting at BIO, reference successive 32-byte blocks of Peripheral I/O Space, starting at address 0 (see Figure 7 on page 38).

3–15. When the discontiguous I/O mode is disabled (that is, in the contiguous mode), addresses are translated so that BIO to (BIO + 64 KB - 1) is sent through to the I/O Space of the I/O bus starting at address 0, and (BIO + 8 MB) to TIO is sent through to the I/O Space of the I/O bus starting at 8 MB (see Figure 7 on page 38).

3–16. The discontiguous I/O mode must be disabled when control is passed to the operating system.

Addresses in the Peripheral I/O Space from BIO + 64 KB to BIO + (8 MB - 1) must be reserved for hardware use.Addresses in the Peripheral I/O Space from (BIO + 64 KB) to (BIO + (8 MB - 1)) are reserved for hardware use. HBs may treat the areas as reserved or may pass them through to the I/O Space of the I/O bus at addresses 64 KB to (8 MB - 1). I/O devices should not be configured in the 64 KB to (8 MB - 1) range for two reasons. First, hardware may not pass this address range through the HB when the discontiguous mode is disabled. Secondly, if the platform is switched from contiguous to discontiguous mode, these devices would no longer be accessible if configured in this range.

Personal Use Copy - Not for Reproduction

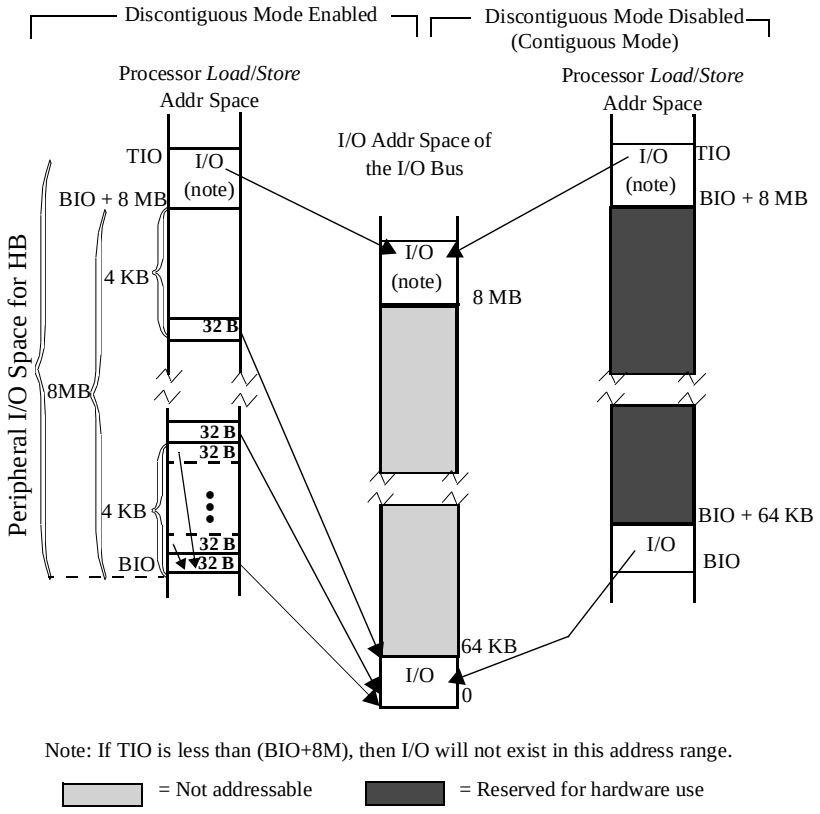


Figure 7. Models for *Load* and *Store* Instructions to Peripheral I/O Space

3.2.2 Translation of 32-Bit DMA Addresses in 64-Bit Addressing Systems

I/O devices which are only capable of accessing up to 4 GB via DMA need a way to access above that limit when used in 64-bit addressing systems and the addressing requirements go beyond 4 GB. This could be accomplished by the device driver transferring the data to a DMA buffer in System Memory in the first 4 GB of address space, and then transferring the data at or above 4 GB by doing a software move of the data. This, however, will not perform well, and it is in these large systems where performance is generally the most critical.

Personal Use Copy - Not for Reproduction

Therefore, the CHRP architecture defines a mechanism for translating DMA addresses for I/O devices which are only capable of addressing a 4 GB I/O Memory Space, to the larger address space required in some 64-bit addressing systems. This capability in an HB will be called the 64-bit addressing option, and is required to be implemented if the platform is designed to support System Memory configured at an address of 4 GB or above.

The 64-bit addressing option is disabled when the operating system is given control. After the operating system has determined that the platform contains System Memory at or above 4 GB, it may assume that all HBs support the 64-bit addressing option and enable the option by using the *set-64-bit-addressing* OF method in each HB node of the OF device tree (each HB has to be enabled separately in order to set up the TCE table address and size for each). The operating system can determine the existence of HB support of 64-bit addressing option by looking for the existence of the *64-bit-addressing* property in the HB node(s) of the OF device tree and if all HBs support the 64-bit addressing option, the operating system may enable the HBs' 64-bit addressing option, even if there is no System Memory configured at an address of 4 GB or above (that is, 64-bit capable systems can be enabled to support the Translation Control Entry (TCE) translation mechanism to translate DMA operations, even if they don't have any System Memory located at or above 4 GB).

Requirements:

3–17. **For 64-bit addressing option in HBs:** In platforms which are designed to support System Memory configured at an address of 4 GB or above, the TCE mechanism, described in Section 3.2.2, "Translation of 32-Bit DMA Addresses in 64-Bit Addressing Systems," on page 38, must be implemented on all HBs. After a DMA operation is accepted by the HB and pre-translated as per Table 6 on page 32, if the 64-bit addressing option of the HB is enabled, the HB must use the TCE mechanism to translate the address when the I/O device presents a 32-bit address to the Memory Space of the HB.

3–18. **For 64-bit addressing option in HBs:** The bits of the TCE must be implemented as defined in Table 5 on page 32.

3–19. **For 64-bit addressing option in HBs:** Enough bits must be implemented in the TCE so that I/O DMA devices are able to access all System Memory addresses.

3–20. **For 64-bit addressing option in HBs:** TCEs must be stored as Big-Endian entities.

Personal Use Copy - Not for Reproduction

3–21. For 64-bit addressing option in HBs: When the 64-bit addressing option is enabled, an HB must not accept 32-bit accesses unless they would also be accepted under the requirements in Table 6 on page 32.

3–22. For 64-bit addressing option in HBs: If an HB accepts 64-bit addresses on DMA accesses (as reported by the existence of the *64-bit-dma* property in the HB node of the OF device tree for that specific HB) and if the 64-bit addressing option of the HB is enabled, then that HB must not use TCEs to translate I/O bus Memory Space DMA addresses which support a full 64-bit address (for example, Dual Address Cycle (DAC) accesses on the PCI bus do not use the TCE translation mechanism in the PCI HB (PHB)).

3–23. For 64-bit addressing option in HBs: If an HB accepts 64-bit addresses on DMA accesses and if the 64-bit addressing option of the HB is enabled, that HB must translate 64-bit I/O bus Memory Space DMA accesses (for example, DAC PCI accesses for the PHB) in the upper 4 GB of the 64-bit Memory Space of the I/O bus to the 0 to (4 GB - 1) address range before passing the access to the host side of that HB (for example, for a PHB, if AD[63:32] (PCI notation) are equal to 0xFFFF FFFF, then the PHB must set AD[63:32] to 0), otherwise the HB must not translate 64-bit accesses (see Figure 8 on page 43).

3–24. For 64-bit addressing option in HBs: After translation of the address via requirements 3–17 or 3–23, above, an HB must use the translated address to access the system, unless that address would re-access the same HB (for example, is in the Peripheral Memory Space or Peripheral I/O Space of that HB), in which case the HB should generate an invalid address error. (See Chapter 10, “Error and Event Notification,” on page 157)

3–25. For 64-bit addressing option in HBs: TCEs must be located in System Memory or appear to software as though they are in System Memory, the memory must be a contiguous real address range, and the memory must be coherent.

3–26. For 64-bit addressing option in HBs: Each HB must provide the capability of having its own independent TCE table.

3–27. For 64-bit addressing option in HBs: Any non-recoverable error while an HB is accessing its TCE table must result in a TCE access error; the action to be taken by the HB being defined under the TCE access error in Chapter 10, “Error and Event Notification,” on page 157.

Personal Use Copy - Not for Reproduction

3–28. For 64-bit addressing option in HBs: In implementations which cache TCEs, if software issues a *Store* instruction to a TCE, then the hardware must perform the following steps: First, if any data associated with the page represented by that TCE is in an I/O bridge cache or buffer, the hardware must write the data, if modified, to System Memory. Secondly, it must invalidate the data in the cache. Finally, it must invalidate the TCE in the cache.

3–29. For 64-bit addressing option in HBs: Neither an I/O device nor an HB must ever modify a TCE.

3–30. For 64-bit addressing option in HBs: If the page mapping and control bits in the TCE are set to 0b00, the hardware must not change its state based on the values of the remaining bits of the TCE.

Software Implementation Note: Software must be careful when attempting DMA operations through an HB to that HB's own TCE table, because HB implementations which cache TCEs are not required to detect changes to the TCEs that they have cached while doing a DMA operation to their own TCE table.

The translation in requirement 3–23 is so that an I/O device which is capable of accessing a full 64-bit address space can get access to the first 4 GB of the address space without going through a TCE translation. Normally, one would expect that this would be possible by the 64-bit device just issuing a DMA operation with an address of less than 4 GB. However, some I/O buses (for example, PCI) do not have a protocol where the I/O device tells the HB that they are capable of greater than 32 bits of addressing when the address is less than 4 GB. In such a case, the HB will not be able to tell the difference between a 32-bit device and a 64-bit device for addresses below 4 GB, and therefore has to assume a 32-bit device and use TCEs. Note that none of this precludes having 32-bit and 64-bit I/O devices on the same bus at the same time.

Figure 8 on page 43 is a representation of how the 32-bit to 64-bit TCE and 64-bit to 64-bit translations work. The information for the translation from 32 bits to the number of address bits required by the platform is contained in the TCE entries in a TCE table of an HB. The I/O bus address is first checked to see if the access is targeted to the address space on the other side of the HB. Since there is not a way for the device to indicate that the operation is headed to the other side of the HB, the selection process is by the same method as for the 32-bit addressing system case; that is, an HB will accept a 32-bit address for translation via a TCE if the access would be accepted under the requirements in Table 6 on page 32. If an HB accepts the access and responds, the I/O bus address is first pre-translated according to Table 6 on page 32. If the address

Personal Use Copy - Not for Reproduction

accesses the area from BIM to TPM for PHB0 (the system-memory-alias space), and if the system-memory-alias space is enabled, then the address is pre-translated by changing the address to be in the range of 0 to (16 MB - 1). The resulting address is then used to select the appropriate TCE in the TCE table of an HB. It does this as follows. The most significant 20 bits of the address (for example, AD[31:12], for PCI) is used as an offset into the TCE table for an HB to select the TCE. Thus, the first TCE maps the addresses 0x00000000 to 0x00000FFF of the Memory Space of the I/O bus; the second entry controls translation of addresses 0x00001000 to 0x00001FFF, and so on. The translated real system address is generated as follows. The Real Page Number (RPN) from the TCE replaces the 20 most significant bits of the address from the I/O bus. The least significant 12 bits from the I/O bus address are used as-is for the least significant 12 bits of the new address.

Thus, the HB's TCE table entries have a one-to-one correspondence with the first *n* pages of the Memory Space of the I/O bus. Each HB has its own TCE table starting address, but software may elect to overlay the TCE tables from different HBs. The size of the Memory Space of the I/O bus that can be mapped to System Memory for a particular HB depends on how much System Memory is allocated to the TCE table for that HB and on how much mappable I/O bus Memory Space is unavailable due to I/O devices which are mapped there (that is, the Peripheral memory and Peripheral I/O spaces are unavailable). The size and location of the HB's TCE table is set up and changed by the *set-64-bit-addressing* OF method for the HB. At the time that OF passes control to the operating system, the 64-bit addressing option of the HB will not be enabled and the TCE table will not have any default size or location. The *set-64-bit-addressing* OF method allows the operating system to set the size of the TCE table to zero while enabling the 64-bit-addressing option of the HB. This would only be used in the case where all I/O devices are 64-bit capable and will always put out an address greater than or equal to 4 GB.

Software Implementation Note: All DMA activity for the I/O devices serviced by an HB must be stopped before setting or changing the size of the TCE table for that HB, or changing the table's location.

Hardware and Software Implementation Note: It may be desirable to have the maximum size of contiguous I/O Memory Space possible that gets translated by TCEs without having to first go through the pre-translation for the system-memory-alias space, and which does not have holes due to the Peripheral Memory or Peripheral I/O Space interference. In order to accomplish this, the hardware or OF can place the Peripheral I/O and Peripheral Memory Spaces for the HB at the largest address possible in the lower 4 GB of address space, and make those areas as small as possible. For example, if the Peripheral

Personal Use Copy - Not for Reproduction

I/O and Peripheral Memory Spaces for an HB are in the upper 1 GB of address space, and if the io-hole is disabled, then there will be 3 GB of contiguous I/O bus Memory Space which can be mapped via TCEs.

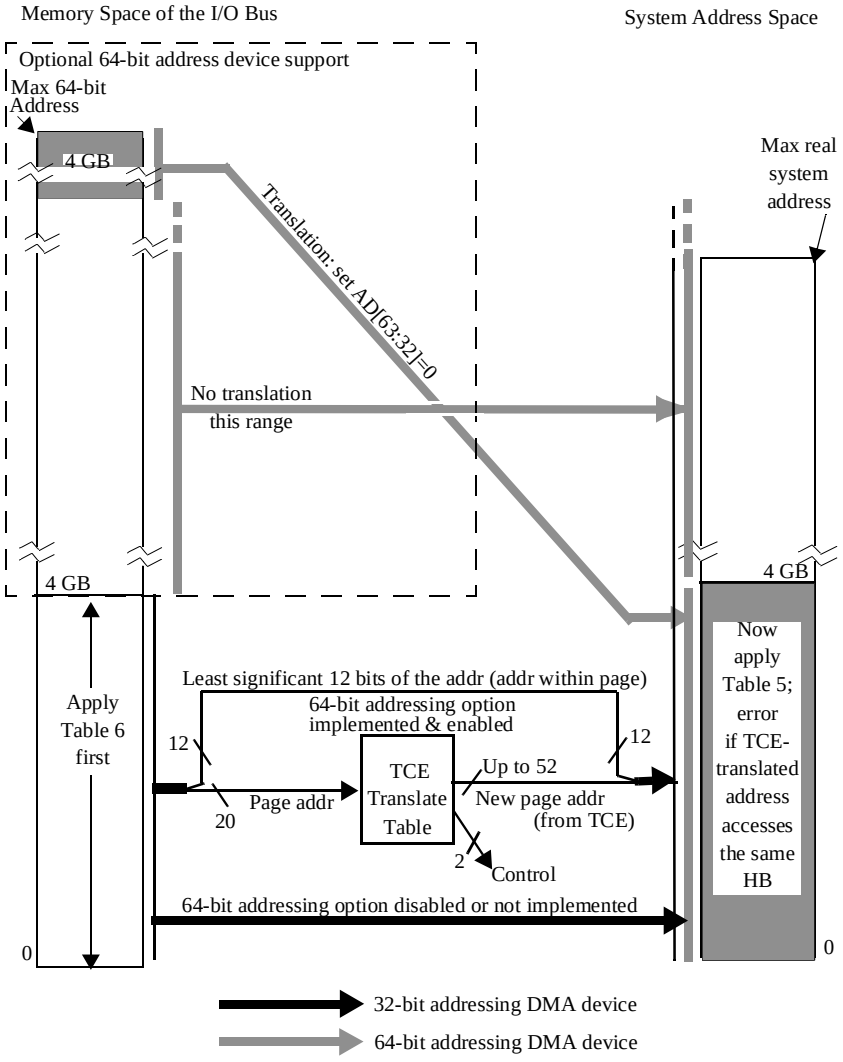


Figure 8. I/O Device DMA Address Translation

Personal Use Copy - Not for Reproduction

Each TCE also contains two control bits. These are used to identify whether that page is mapped to the system address space, and if the page is mapped, whether it is mapped read/write, read only, or write only. See the Table 7 on page 44 for a definition of these control bits.

The HB's TCE table is the analogue of the system translation tables. However, unlike the system translation tables, the dynamic page faulting of memory during an I/O operation is not required (the page fault code point in the TCE control field is used to indicate an error condition and is to be used for error detection).

Table 7. TCE Definition

Bits	Description
0 to 51	RPN: If the page mapping and control field of the TCE indicate anything other than page fault, then these bits contain the Real Page Number (RPN) to which the bus address is mapped in System Memory. In certain HB implementations, all of these bits may not be required, however enough bits must be implemented to match the largest real address in the platform.
52 to 60	Reserved for future use.
61	Reserved for future use (assigned for IBM use).
62 to 63	Page Mapping and Control: These bits define page mapping and read-write authority. They are coded as follows: 00 Page fault (no access) 01 System Memory (read only) 10 System Memory (write only) 11 System Memory (read/write) Code point 0b00 signifies that the page is not mapped. It must be used to indicate a page fault error. Hardware must not change its state based on the value in the remaining bits of a TCE when code point 0b00 is set in this field of the TCE. For accesses to System Memory with an invalid operation (write to a read-only page or read to a write-only page), the HB will generate an error. See Chapter 10, "Error and Event Notification," on page 157 for more information about error handling.

3.3 PC Emulation Option

This section describes an optional set of changes to the system architecture that improves the performance and/or compatibility of a stand-alone emulation environment for PC software. The PC Emulation option consists of minor modifications to the system address map. To support the PC Emulation option, platforms must implement the io-hole and the complete set of additional features described in this section.

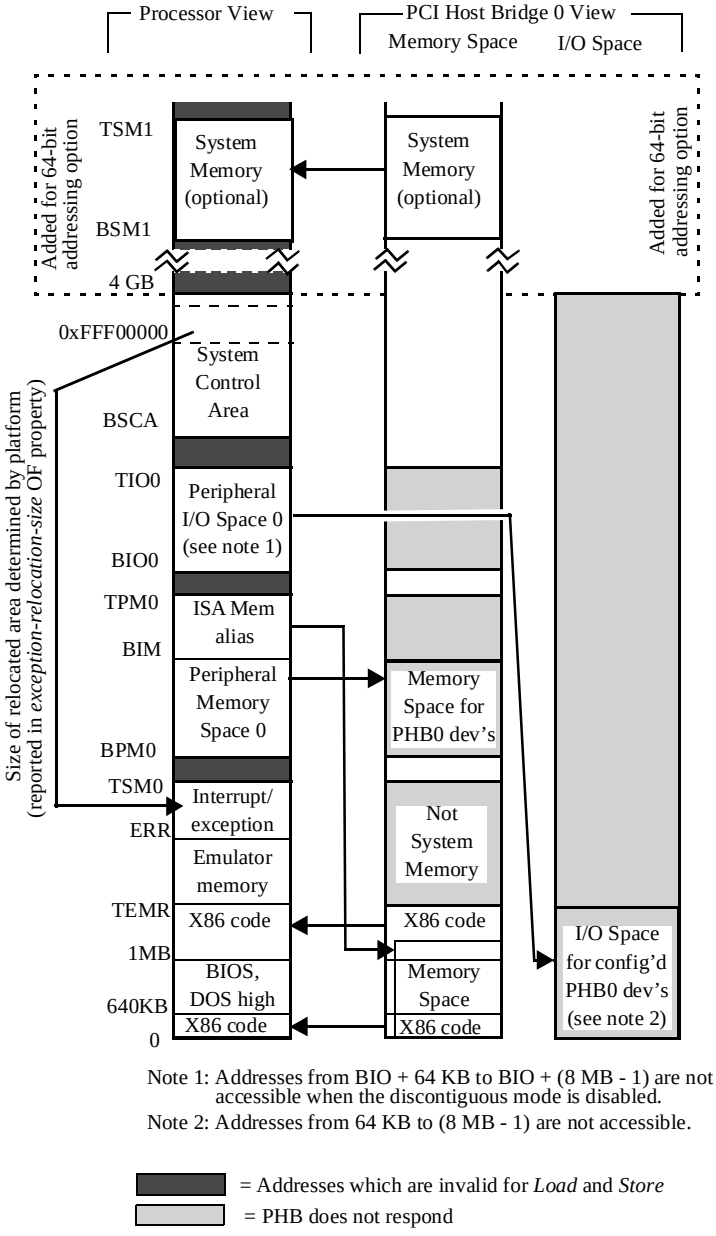
Personal Use Copy - Not for Reproduction

Using the system address map described in this chapter as a base for discussion, the environment foreseen by the PC Emulation option has all of the following attributes:

- From the points of view of the processor and I/O bus masters (including third party DMA controllers), an image in memory, from 0 through a "top of emulated memory," which exactly matches that of a PC.
- The io-hole is enabled, to allow I/O bus master and third party DMA access to legacy devices such as VGA.
- From the point of view of the processor, memory above the "top of emulated memory" in which an emulator may reside, including areas for unique interrupt and exception handling code.
- From the points of view of I/O bus masters (including ISA bus masters, if below 16 MB), the address space above the "top of emulated memory" is prevented from mapping to System Memory, and may instead reference PCI/ISA memory or be intercepted by the host bridge as an error.
- From the points of view of I/O bus masters, disabling of the alias for the low 16 MB of System Memory (the system-memory-alias space) so that address range would address PCI memory or be intercepted by the host bridge as an error.

If a platform implements the PC Emulation option, this will be indicated by the existence of the *pc-emulation* property in the root node of the OF device tree, and the option is enabled by the *set-pc-emulation* method in the system node. In addition, the *set-pc-emulation* method initializes the Top of Emulated Memory Register (TEMR) and the Exception Relocation Register (ERR), and assures the requirement 3–34 is met. Figure 9 on page 46 shows an example of an address map with the PC Emulation option enabled.

Personal Use Copy - Not for Reproduction



Note 1: Addresses from BIO + 64 KB to BIO + (8 MB - 1) are not accessible when the discontiguous mode is disabled.

Note 2: Addresses from 64 KB to (8 MB - 1) are not accessible.

= Addresses which are invalid for *Load* and *Store*

= PHB does not respond

Figure 9. Example Address Map with the PC Emulation Option Enabled

Personal Use Copy - Not for Reproduction

To create the environment described above, one must define two pointer registers and associate some new behaviors with various regions of the system address map. The following requirements describe the functionality that is needed to provide the appropriate system address map.

Requirements:

3–31. For PC Emulation option: The platform must implement the TEMR, and all of the following must be true:

a. The TEMR must contain the largest address in System Memory which is set aside for the image of PC memory.

b. The granularity of the contents of TEMR must be 1 MB.

c. The HB must not respond to I/O bus Memory Space DMA operations in the address range of (TEMR + 1) to the top of System Memory, or must respond and signal an invalid address error.

3–32. For PC Emulation option: PCI devices must not be configured between (TEMR + 1) and the top of System Memory which is below BSCA.

3–33. For PC Emulation option: The platform must implement the ERR, and all of the following must be true:

a. The value of *exception-relocation-size* OF property must be the amount of address space above 0xFFF00000 that is relocated down into System Memory and is a fixed value, as determined by the platform.

b. The granularity of *exception-relocation-size* must be 4 KB and the minimum size must be 12 KB.

c. The contents of the ERR must be the address of the base of the region in System Memory to which references to the interrupt/exception handling area (normally at the top of the 32-bit address space) are relocated.

d. The granularity of the ERR must be 1 MB.

3–34. For PC Emulation option: When the PC Emulation option is enabled, the peripheral-memory-alias space must be enabled, the system-memory-alias space must be disabled, and the io-hole must be enabled.

Hardware Implementation Note: Since the software only has access to the ERR and the TEMR through the *set-pc-emulation* OF method, the

Personal Use Copy - Not for Reproduction

actual implementation of these registers can be abstracted from the software, and the value that gets put into the actual hardware registers can be transformed by *set-pc-emulation* to be something more directly usable by the hardware.

Software Implementation Notes:

- It is assumed that the processor is programmed with Machine State Register (MSR) Interrupt Prefix (IP) bit set to a 1 (MSR_{IP}=1; that is, exception/interrupt handlers at the top of the 32-bit address space).
- Software must set up the interrupt/exception area at the new location before relocating that area via the ERR (that is, before calling the *set-pc-emulation* method which sets the ERR).
- The *set-pc-emulation* method, when used to disable the PC Emulation option, does not have to put the system-memory-alias space back into the state that it was in before the enabling of the PC Emulation option.
- The *set-pc-emulation* method may use the *set-initial-aliases* OF method of the PHB0 node to assure that the peripheral-memory-alias is enabled and may use the *set-io-hole* OF method of the PHB- to assure that the io-hole is enabled.
- The *set-initial-aliases* method should not be called while the PC Emulation option is enabled, or else the system-memory-alias may get re-enabled.

Personal Use Copy - Not for Reproduction

Processor and Memory

4

The purpose of this chapter is to specify the processor and memory related requirements of the CHRP architecture. The processor architecture section addresses differences between the processors in the PowerPC family as well as their interface variations and features of note. The memory architecture section addresses coherency, minimum system memory requirements, memory controller requirements, and cache requirements.

4.1 Processor Architecture

The PowerPC architecture governs software compatibility at an instruction set and environment level. However, each processor implementation has unique characteristics which are described in its user's manual. To facilitate shrink-wrapped software, the CHRP architecture places some limitations on the variability in processor implementations. Nonetheless, it is possible for these differences to affect platform-aware software. Further, it is possible for system support chips (for example memory controller, L2 cache, PCI bridge) to affect platform-aware software. Platform-aware software is defined as any software which attempts to manipulate the platform external to the processor itself, including optimization of external data transfers and synchronization, for example. Lastly, evolution of the PowerPC architecture and implementations creates a need for both software and hardware developers to stay current with its progress. The following material highlights areas deserving special attention and provides pointers to the latest information.

4.1.1 Processor Architecture Compliance

The PowerPC architecture is defined in *The PowerPC Architecture* [1]. This architecture description is broken into three parts as defined below:

- Book I, *PowerPC User Instruction Set Architecture*
- Book II, *PowerPC Virtual Environment Architecture*
- Book III, *PowerPC Operating Environment Architecture*

The latest updates to the PowerPC architecture can be obtained by accessing the following URL: *http://www.austin.ibm.com/tech/ppc-chg.html*

Requirements:

4–1. Platforms must incorporate only processors which comply fully with the PowerPC architecture.

4–2. **For the Symmetric Multiprocessor option:** Multiprocessing platforms must use only processors which implement the processor identification register. See *PowerPC 604 RISC Microprocessor User's Manual* [6] for a definition of this register.

4–3. Platforms must incorporate only processors which implement *tlbie* and *tlbsync*, and *slbie* and *slbia* for 64-bit implementations.

4–4. Except where specifically noted otherwise in Section 4.1.4, "PowerPC Architecture Features Deserving Comment," on page 53, platforms must support all functions specified by the PowerPC architecture.

Hardware Implementation Note: Requirements 4–1, 4–2, and 4–3 may restrict the platform developer's choice of processor. For example, an embedded processor might have a different operating environment architecture. All announced PowerPC microprocessors in the 600-series (for example, 603, 603e, and 604) conform to these requirements, with the exception that the 603 family does not support requirement 4–2.

Hardware and Software Implementation Note: The PowerPC architecture and the CHRP architecture view *tlbia* as an optional performance enhancement. Processors need not implement *tlbia*. Software that needs to purge the TLB should provide a sequence of instructions that is functionally equivalent to *tlbia* and use the content of the OF device tree to choose the software implementation or the hardware

Personal Use Copy - Not for Reproduction

instruction. See Section 4.1.2, “PowerPC Microprocessor Differences,” on page 51 for details.

Hardware Implementation Note: The timebase freeze/thaw functionality described in requirement 12–8 may also be useful to a uniprocessor system in achieving B2 and better security ratings.

4.1.2 PowerPC Microprocessor Differences

There are a few significant differences among the programming models provided by PowerPC microprocessors in the 600-series. The 603 family implements a set of power management modes which are not described in the PowerPC architecture, and which are different from the minimal support provided by the 604 family. The 64-bit implementations use a different page table format which enables a 64-bit address space, and adds a 64-bit execution mode. A more complete understanding of these differences may be obtained by studying *The PowerPC Architecture* [1] and the user’s manuals for the various processors. The 601 is not supported by the CHRP architecture.

The CHRP architecture creators cooperate with processor designers to maintain a list of supported differences, to be used by operating systems instead of the processor version number (PVN), enabling execution on future processors. Open Firmware communicates these differences via properties of the *cpu* node of the OF device tree. The currently supported differences and their associated properties in the OF device tree include the following: *64-bit*, *32-64-bridge*, *603-translation*, *603-power-management*, *general-purpose*, *graphics*, *tlbia*, *performance-monitor*, *emulation-assists*, and *external-control*. See *PowerPC processor binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [11] for more details.

In addition to the differences described above, the PowerPC architecture allows for implementation-specific, software-transparent differences. The second and third pages above *Base* (the address specified for interrupt vector location by MSR_{IP}) are reserved for implementation-specific routines provided by the processor designer which make details of the processor implementation transparent to operating system and application software; see Table 8 on page 52 and also the description of the ERR in requirement 3–33. The software-assisted translation mechanism implemented by the 603 is an example of how the second and third pages can be used.

Software Implementation Note: A 64-bit operating system need not require 64-bit client interface services in order to boot. Because of the problems that might be introduced by dynamically switching between 32-bit and 64-bit modes in Open Firmware, the configuration variable

Personal Use Copy - Not for Reproduction

64-bit-mode? is provided so that Open Firmware can statically configure itself to the needs of the operating system.

Software Implementation Note: Only those portions of the range from 0xFFF00000 to 0xFFF02FFF which are actually used prior to setting MSR_{IP}=0 must be set aside in ROM as described in Table 8 on page 52.

.

Table 8. Fixed Real Storage Locations Having Defined Uses

Real Address (hex) when MSR_{IP}=0	Real Address (hex) when MSR_{IP}=1	Use
0000 - 00FF	FFF00000-FFF000FF	Reserved for OS use.
0100 - 0FFF	FFF00100-FFF00FFF	These locations are for the interrupt vectors.
1000 - 2FFF	FFF01000-FFF02FFF	Reserved for microprocessor-dependent firmware use. OF must initialize this area and OS software must preserve this area.

Requirements:

4–5. Operating systems must use the properties of the *cpu* node of the OF device tree to determine the programming model of the processor implementation.

4–6. Operating systems must provide an execution path which uses the properties of the *cpu* node of the OF device. The PVN is available to the platform aware operating system for exceptional cases such as performance optimization and errata handling.

4–7. Operating systems must support both of the page table formats (32-bit and 64-bit) defined by the PowerPC architecture.

4–8. Processors which exhibit the *64-bit* property of the *cpu* node of the OF device tree must also implement the “bridge architecture,” an option in the PowerPC architecture. See the updates on the Internet associated with *The PowerPC Architecture* [1].

4–9. Platforms must restrict their choice of processors to those whose programming models may be described by modifications to that of the 604 by the properties defined for the *cpu* node of the OF device tree in *PowerPC processor binding to: IEEE Std 1275-1994 Standard for Boot*

Personal Use Copy - Not for Reproduction

(Initialization, Configuration) Firmware [11] (for example, *64-bit* and *603-translation).*

4–10. Platform firmware must initialize the second and third pages above *Base* correctly for the processor in the platform, and in the correct endian mode, prior to giving control to the operating system.

4–11. Operating system and application software must not alter the state of the second and third pages above *Base*.

4.1.3 Processor Interface Variations

The processor interface for the 32-bit processors (for example, 603 and 604) is described by the *PowerPC 60x Microprocessor Interface Specification* [18]. The processor interface for the 64-bit processors is described by the *PowerPC 6xx Bus Definition* [19]. Individual implementations may subset these specifications, provided these variations are described in their respective user's manuals. The processor interface for the 603 family has been optimized for portable applications by keeping address-only cycles internal to the processor. This optimization leads to functional restrictions which must be understood by the platform and system software providers.

Requirements:

4–12. **For the Symmetric Multiprocessor option:** The 603 family of processors must not be used in a symmetric multiprocessing complex.

4–13. The 603 family of processors must be used only with system logic that does not reorder data transfers.

4.1.4 PowerPC Architecture Features Deserving Comment

Some PowerPC architecture features are optional, and so need not be implemented in a platform. Usage of others may be discouraged due to their potential for poor performance. The following sections elaborate on the disposition of these features in regard to compliance with the PowerPC architecture.

4.1.4.1 Unaligned Little-Endian Scalar Operations

The initial set of PowerPC processor designs did not support unaligned Little-Endian scalar operations in hardware, but interrupted to the system alignment

Personal Use Copy - Not for Reproduction

handler to provide the opportunity for the operating system to emulate these operations. Later implementations of the architecture have added hardware support for these operations.

Requirements:

4–14. Operating systems must provide an alignment interrupt handler which correctly emulates the execution of any unaligned LE accesses that are not implemented in hardware, except those which may be generated by *lwarx, ldarx, stwcx., stdcx., eciwx, ecowx,* and the multiple scalar operations (see 4.1.4.2, "Little-Endian Multiple Scalar Operations," on page 54).

Software Implementation Note: Because of performance impacts, software is strongly discouraged from using unaligned Little-Endian scalar operations.

Hardware Implementation Note: Because legacy software generates unaligned Little-Endian scalar operations, hardware is not exempted from supporting these operations.

4.1.4.2 Little-Endian Multiple Scalar Operations

The PowerPC architecture does not support multiple scalar operations in Little-Endian mode. The multiple scalar operations are *Load and Store String* and *Load and Store Multiple*.

Requirements:

4–15. Software must not use multiple scalar operations in LE mode. Results of multiple scalar operations in LE mode are undefined.

Architecture Note: Multiple scalar operations in LE mode actually generate an alignment interrupt instead of a program interrupt. They are, nonetheless, intended to be unsupported.

4.1.4.3 Direct-Store Segment Support

The CHRP architecture does not require platforms to support direct-store segments. However, direct-store segments may be required for some I/O bus extensions to the CHRP architecture.

Personal Use Copy - Not for Reproduction

Requirements:

4–16. Platforms must not use direct-store segments to implement interfaces defined by the CHRP architecture.

4–17. Operating systems must not depend on direct-store segment support when using interfaces specified by the CHRP architecture.

4.1.4.4 Big-Endian Multiple Scalar Operations

The PowerPC architecture supports multiple scalar operations in Big-Endian mode. The multiple scalar operations are *Load and Store String* and *Load and Store Multiple*. In future PowerPC processor implementations, these instructions are likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual *Load* or *Store* instructions.

Software Implementation Note: Because of the long-term performance disadvantage associated with multiple scalar operations, their use by software is not recommended. However, there is a window of system implementations in which store multiples will provide the highest bandwidth from the processor to I/O. Therefore, with the knowledge of imminent need to rewrite some code, certain developers (especially graphics device driver developers) may choose to use store multiples. Note that the PowerPC architecture does not guarantee order among data stored by a *Store Multiple* instruction.

4.1.4.5 External Control Instructions (Optional)

The external control instructions (*eciwx* and *ecowx*) provide a mechanism for non-privileged code to interact with external devices which require a knowledge of real, rather than effective, addresses. Together with monitoring of translation activity by the external hardware, they provide a means for external access to system memory without pinning that memory.

Software Implementation Note: Operating systems should isolate applications from the external control instructions by providing them in service libraries, and may also use them in device drivers.

Hardware Implementation Note: Platforms are not required to support the external control instructions.

Personal Use Copy - Not for Reproduction

Requirements:

4–18. If the external control facility defined by the PowerPC architecture is supported, that support will be described as properties of the appropriate nodes (for example, memory controller and host bridge) of the OF device tree. See *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10] for more details.

4–19. Platforms must not use external control instructions as the sole interface to functions specified by the CHRP architecture.

4–20. Operating systems must not require the external control instructions when using interfaces specified by the CHRP architecture.

4.2 Memory Architecture

The Memory Architecture of a Common Hardware Reference Platform implementation is defined by *The PowerPC Architecture* [1] and the System Memory, Storage Ordering Models, Memory Controller, and Cache Memory sections which follow, as well as Chapter 3, "System Address Map," on page 23, which defines what platform elements are accessed by each real (physical) system address.

4.2.1 System Memory

System Memory normally consists of dynamic read/write random access memory which is used for the temporary storage of programs and data being operated on by the processor(s). A platform usually provides for the expansion of System Memory via plug-in memory modules and/or memory boards.

Requirements:

4–21. Platforms must provide at least 8 MB of System Memory. (Also see Chapter 3, "System Address Map," on page 23 for other requirements which apply to memory within the first 16 MB of System Memory.)

4–22. Platforms must support the expansion of System Memory to 32 MB or more.

Personal Use Copy - Not for Reproduction

Hardware Implementation Note: These requirements are minimum requirements. Each operating system has its own recommended configuration which may be greater.

Software Implementation Note: System Memory will be described by the properties of the *memory* node(s) of the Open Firmware (OF) device tree.

4.2.2 Storage Ordering Models

For "correct" execution of a program, correct ordering of its storage accesses is essential. Most programs are written with a "sequential" ordering (also called "serial," or "strong" ordering) assumption or model. In this type of ordering, *Load*s and *Store*s are performed in the order specified by the program. The satisfaction of this assumption is guaranteed by the hardware.

The sequential execution model in the PowerPC architecture requires that *Load*s and *Store*s issued by a program appear to the program to be strongly ordered. (See requirement 4–27.) However, it is not necessary that those accesses complete in storage, or are viewed by other processors or mechanisms, in the same order. This is called a *weakly ordered* storage model. Weak ordering can lead to performance improvements in complex systems by allowing a processor to consider its *Load* and *Store* instructions complete before the corresponding storage operations are performed with respect to other processors and mechanisms.

4.2.2.1 Memory Coherence

Coherence refers to the ordering of writes to a single location. Specifically, a location is said to be coherent if no two processors or I/O observe any subset of a series of *Store*s to that location to be occurring in mutually conflicting orders. The *PowerPC Architecture* [1] provides specifications for implementing coherence among caches and physical memory for System Memory locations within a symmetric multiprocessor.

The PowerPC architecture specifies accesses (*Load*s and *Store*s) to bytes, half-words aligned on half-word boundaries, words aligned on word boundaries, and, for 64-bit implementations of the architecture, also, double-words aligned on a double-word boundaries, as being "single-copy atomic." A single-copy atomic operation is performed without permitting other processors or mechanisms to access the target locations from the beginning to the end of the operation. Accesses to data of all other sizes can be non-single copy atomic.

Single copy atomic *Store*s to a given location are coherent if they are serialized in some order, and no processor is able to observe any subset of those

Personal Use Copy - Not for Reproduction

*Store*s in a conflicting order. This serialization results in a sequence of values assigned to the location. For a location that is cachable, the unique sequence is defined over values assumed not only by the physical memory location, but also by copies of the location in caches. It is not necessary that the physical memory location assume each value in the sequence. For example, if a processor has a write-back cache, it may update a location several times before the value is written to the physical memory. Not every value written is visible to another processor. However, if the location is coherent, the order of values seen by another processor is compatible with the order in which *Store*s are made.[1] Thus, for example, a processor can never *Load* a "newer" value first and then, later, *Load* an "older" value.

A platform may maintain coherence on blocks of data, referred to as *coherency blocks*, that are larger than data-sizes prescribed for single-copy atomic accesses. Typically, the coherency block is of the size of a cache line, but it can be smaller. The coherency block is treated as a whole, or as a single location, with respect to values it assumes, and these values form a unique sequence of which all the processors and I/O observe a subset.

The PowerPC architecture allows assignment of the hardware coherence property on a per-page (4 KB) basis. If a given page of physical memory is declared as Memory Coherence Required[2] (M-bit equals 1), all coherency blocks within that page are kept coherent. It is permitted by the PowerPC architecture, however, that a page in physical memory may be accessed via different virtual addresses that have different coherence property specifications. This is called "M-bit aliasing." Similar mismatches are also permitted by the PowerPC architecture among Caching Inhibited (I-bit), Write Through (W-bit), and Guarded Storage (G-bit) properties. Under certain circumstances, the PowerPC architecture disallows the assumption of hardware-maintained coherence when a location is accessed with aliased W, I, or M bits.[3]

Requirements:

4–23. Platforms must provide the ability to maintain System Memory Coherence.

4–24. I/O transactions to System Memory through a Host Bridge must be made with coherence required.

---

[1] See the section entitled "Memory Coherence" in Book II, *The PowerPC Architecture* [1].

[2] See the section entitled "Storage Access Modes" in Book III, *The PowerPC Architecture* [1].

[3] *ibid*.

Personal Use Copy - Not for Reproduction

Hardware Implementation Note: For some system buses, enabling a transaction for snooping is a method for invoking the coherence mechanism. For these buses, requirement 4–24 translates into a requirement that all I/O transactions to system memory be enabled for snooping.

Hardware Implementation Note: Some processors may ignore the M=1 setting for a line when it is accessed as a result of an instruction-fetch miss. However, if the line contains both instructions and data, there may be subsequent data accesses to that line which are required to be coherent. In this case, the non-coherent copy of the line must be discarded and the line must be refetched coherently. I/O writes to the line must also be performed coherently with respect to copies of the line contained in data caches or in combined caches.

Memory, other than System Memory, is not required to be coherent. Such memory may include memory in I/O devices.

4.2.2.2 Consistency Model

Consistency refers to the ordering of accesses to more than one location. In systems which support a *sequentially consistent* memory model, *Load*s and *Store*s performed by a program not only appear to be strongly ordered to the program executing them, but are also observed by concurrent programs on other processors as being performed in the same order. This is not true of a system that employs a *weakly consistent* memory model. In a weakly consistent memory model, weak ordering is the default for storage accesses.

Weakly ordered storage accesses can lead to a temporarily inconsistent view of memory among processors. For example, suppose a processor performs a *Store* to location A, followed by a *Store* to location B, then for some finite period of time, it is possible that if another processor or mechanism reads the new value of B, it can subsequently perform a read on location A and yet receive the old value of A — a view of memory that is inconsistent with that of the program which performed the two store operations.

However, with mutually inconsistent views of memory, cooperating programs cannot communicate correctly and, therefore, cannot compute correctly. Such programs must ensure that a consistent view of memory is attained before they start to communicate. An ability to enforce a sequential consistency on demand is therefore essential.[1]

[1] Note that cooperating concurrent programs running on a system that enforces sequential consistency for all storage accesses always execute correctly.

Personal Use Copy - Not for Reproduction

The PowerPC architecture supports a weakly consistent memory model. To enforce sequential consistency on demand, it provides ordering instructions: *eieio*, *sync*, and *isync*. *Eieio* guarantees that qualifying storage accesses separated by it get performed in the order they are issued by the program. *Sync* and *isync* instructions guarantee that when they complete, all coherent storage accesses issued prior to them have been performed with respect to all processors and mechanisms that accesses those locations coherently. Thus, using these instructions, it is possible for communicating programs to guarantee that accesses before and after these instructions are observed to have performed in a mutually consistent order.

Most platforms based on the CHRP architecture are expected to incorporate the weakly consistent memory model as specified by the PowerPC architecture.

Requirements:

4–25. Software must assume only the Weakly Consistent storage model.

4–26. Platforms must guarantee that a processor's accesses to the same location are kept strongly ordered, unless the location is accessed by the processor with WIM-bit aliasing which prohibits the assumption of hardware-maintained coherence.

As specified in the PowerPC architecture, software must use ordering instructions to impose strong ordering between accesses to different physical memory locations, coherent or otherwise, when it has a dependency on the order of accesses being strong. Self modifying code, if used, must be written in such a fashion as to ensure that the instruction fetch pipeline and the cache do not contain the original code once the modification has been made.

Software Implementation Note: Explicit ordering must be employed if self modifying code is used. The following sequence of instructions is one way to accomplish this: *dcbst, sync, icbi, sync, isync*.

4.2.2.3 Storage Ordering and I/O

Looking out from a processor, there are interfaces and bridge devices beyond which the ordering operations defined by the PowerPC architecture cannot be propagated. Such interfaces or bridges form the boundary of the system which encloses the PowerPC *coherency domain* which satisfies the PowerPC coherency architecture.

Figure 10 on page 61 shows an example system. The shaded portion is the PowerPC coherency domain. Buses 1 through 3 lie outside this domain. The figure shows two I/O subsystems, each interfacing with the host system via a

Personal Use Copy - Not for Reproduction

Host Bridge. Notice that the boundary is shown to traverse through the Host Bridges. This marks the usually identifiable boundary inside the Host Bridge hardware. On one side (the shaded portion) of this boundary PowerPC architectural semantics are in force, and on the other side lies the logic that interfaces with protocols of the I/O interconnect.

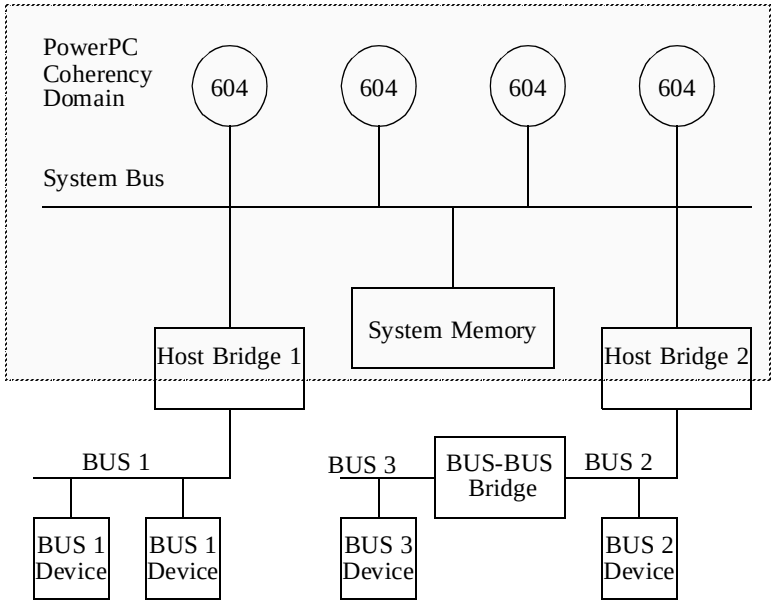


Figure 10. Example System Diagram Showing the PowerPC Ordering Domain

The fundamental ordering rule for the PowerPC sequential execution model is that if a processor performs a *Store* operation to a location followed by a *Load* to the same location, and if it is the only processor storing to that location, then the *Load* must return the value deposited by the *Store* operation. This ordering rule might be violated outside the coherency domain if I/O devices within the same I/O subsystem are allowed to make concurrent accesses to the location. The burden of guaranteeing satisfaction of this sequential ordering

Personal Use Copy - Not for Reproduction

rule, therefore, is on both system hardware and software. For a more detailed discussion see Section 5.1.2.3, "PCI Delayed Read Interaction," on page 74.

Accesses outside the coherency domain are assumed to be made to I/O devices. These accesses are considered performed (or complete) when they complete at the device's I/O bus interface.

It is not sufficient to ensure strong ordering on a processor's accesses to a single location outside the coherency domain. It is also important to able to ensure that *Store*s to a number of different locations outside the coherency domain are completed in a strongly ordered manner. For example, suppose some data has to be deposited in an I/O device before the device's control register is written to start its operation. It is essential, in this case, that the *Store*s that deposit the data complete before the *Store* that writes into the control register. For this reason, semantics of *eieio* must be extended beyond the coherency domain. Because the PowerPC ordering operations do not travel beyond the PowerPC coherency domain, system hardware must take appropriate action to enforce the ordering semantics of *eieio* on accesses leaving the domain. See Chapter 5, "I/O Bridges," on page 69.

Requirements:

4–27. Platforms must guarantee that, for a particular processor, accesses to the same location beyond the PowerPC coherency domain are performed in a strongly ordered manner, given that no I/O device is allowed to make concurrent accesses to the location. Note that it is not sufficient to merely produce an appearance of strong ordering with respect to the processor performing the accesses.

4–28. Platforms must guarantee that the storage ordering semantics of *eieio* are preserved for accesses leaving the PowerPC coherency domain at any given host bridge, all the way to destination I/O devices. That is, *Load* and *Store* accesses (in any combination) by a processor to an I/O device which are separated by an *eieio*, must complete in the same order that the processor issued those accesses.

Apart from the ordering disciplines stated in requirements 4–27 and 4–28, no other ordering discipline is guaranteed by the system hardware for *Load*s and *Store*s performed by a processor to locations outside the PowerPC coherency domain. Any other ordering discipline, if necessary, must be enforced by software via programming means.

Software Implementation Note: For example, if software wants to initiate a peer-to-peer operation only after a number of *Store* operations it issued to the device have completed, it can issue a sequence of

Personal Use Copy - Not for Reproduction

instructions such as follows: *Store* <Location A>, *Store* <Location B>, *Store* <Location C>, *eieio*, *Load* <Location C>, *eieio*, where locations A, B, and C are in the device and are assumed to have a "cache-inhibited and guarded" mapping. After issuing this sequence the software can start the peer-to-peer operation. The first *eieio* guarantees that all the *Store*s are sent before the *Load* is sent. The second *eieio* guarantees that all subsequent storage operations are not sent until the *Load* completes at the processor, at which point the program can be certain that all the previous *Store*s to device are also complete.

The elements of a system outside its coherency domain are not expected to issue explicit PowerPC ordering operations. System hardware must therefore take appropriate action to impose ordering disciplines on storage accesses entering the coherency domain. A strong-ordering rule is enforced on an I/O device's accesses to the same location. Write operations from the same source are completed in a sequentially consistent manner.

Requirements:

4–29. Platforms must guarantee that accesses entering the PowerPC coherency domain that are from the same I/O device and to the same location are completed in a sequentially consistent manner.

4–30. Platforms must guarantee that multiple write operations entering the PowerPC coherency domain that are issued by the same I/O device are completed in a sequentially consistent manner.

It is the responsibility of the host bridges to guarantee that requirements 4–27 through 4–30 are satisfied. A host bridge must take into account the particular I/O interconnection network that it interfaces to and the protocol characteristics and interfaces of the I/O buses involved, which will influence the actual set of ordering rules that the above requirements translate into and that the host bridge implements. For example, Section 5.1.2, "Data Buffering and Instruction Queuing," on page 70 specifies ordering rules for a PCI host bridge that interfaces to a PCI-based I/O tree.

4.2.2.3.1 Storage Ordering and I/O Interrupts

The conclusion of I/O operations is often communicated to processors via interrupts. For example, at the end of a DMA operation that deposits data in the system memory, the device performing the operation might send an interrupt to the processor. Arrival of the interrupt, however, is no guarantee that all the data has actually been deposited; some might be on its way. The receiving program

Personal Use Copy - Not for Reproduction

must not attempt to read the data from the memory before ensuring that all the data has indeed been deposited. There may be system and I/O subsystem specific method for guaranteeing this. See Section 5.1.2.2, “DMA Ordering,” on page 73.

4.2.2.4 Atomic Update Model

An update of a memory location by a processor, involving a *Load* followed by a *Store*, can be considered “atomic” if there are no intervening *Store*s to that location from another processor or mechanism. The PowerPC architecture provides primitives in the form of *Load And Reserve* and *Store Conditional* instructions which can be used to determine if the update was indeed atomic. These primitives can be used to emulate operations such as “atomic read-modify-write” and “atomic fetch-and-add.” Operation of the atomic update primitives is based on the concept of “Reservation,”[1] which is supported in a CHRP system via the coherence mechanism.

Requirements:

4–31. The *Load And Reserve* and *Store Conditional* instructions must not be assumed to be supported for Write-Through storage.

Software Implementation Note: To emulate an atomic read-modify-write operation, the instruction pair must access the same storage location, and the location must have the Memory Coherence Required attribute.

Hardware Implementation Note: The reservation protocol is defined in Book II of the *PowerPC Architecture* [1] for atomic updates to locations in the same coherency domain.

4.2.3 Memory Controllers

A *Memory Controller* responds to the real (physical) addresses produced by a processor or a host bridge for accesses to System Memory. It is responsible for handling the translation from these addresses to the physical memory modules within its configured domain of control.

[1] See the section entitled “Synchronization” in Appendix E of Book I, *The PowerPC Architecture* [1], and the section entitled “Atomic Update Primitives” in Book II, *The PowerPC Architecture* [1]

Personal Use Copy - Not for Reproduction

Requirements:

4–32. Memory controller(s) must support the accessing of System Memory as defined in Chapter 3, “System Address Map,” on page 23.

4–33. Memory controller(s) must be fully initialized and set to full power mode prior to the transfer of control to the operating system. This requirement applies to normal boot and wakeup. (See Chapter 11, “Power Management,” on page 185 for an explanation of these terms.)

4–34. All allocations of System Memory space among memory controllers must have been done prior to the transfer of control to the operating system.

4–35. Memory controller(s) must maintain System Memory in Big-Endian byte order when the system is operating in Big-Endian mode (MSR_{LE}=0) and in Little-Endian byte order when the system is operating in Little-Endian mode (MSR_{LE}=1). Whether System Memory is maintained in “True Little-Endian” or “PowerPC Little-Endian”[1] form while in Little-Endian mode is platform dependent, but must be transparent to software. See Section 2.3, “Bi-Endian Support,” on page 14 and Appendix C, “Bi-Endian Designs,” on page 265 for more information on Bi-Endian designs.

Software Implementation Note: Memory controller(s) are described by properties of the *memory-controller* node(s) of the OF device tree.

4.2.4 Cache Memory

All of the PowerPC microprocessors include some amount of on-chip or *internal cache* memory. The CHRP architecture allows for cache memory which is external to the processor chip, and this *external cache* memory forms an extension to internal cache memory.

Requirements:

4–36. All caches must meet the requirements of the CHRP architecture coherency model, as stated in “Storage Ordering Models” on page 57.

4–37. **For the Symmetric Multiprocessor or Power Management option:** Each cache must be able to be completely flushed and invalidated via

[1] See the section entitled “PowerPC Little-Endian Byte Ordering” in Appendix D of Book I of *The PowerPC Architecture* [1], for an explanation of PowerPC Little-Endian byte ordering.

Personal Use Copy - Not for Reproduction

the RTAS *cache-control* function. (See Section 7.3.10.1, “Cache-control,” on page 136 for more information.)

4–38. For the Power Management option: External caches must be able to be placed in low power or disabled states via the RTAS *cache-control* function.

4–39. For the Symmetric Multiprocessor or Power Management option: To ensure compatibility with all CHRP implementations, operating systems must use the RTAS *cache-control* function to flush an entire cache rather than code directly to any specific system or processor implementation.

4–40. If a platform implementation elects not to cache portions of the address map in all external levels of the cache hierarchy, the result of not doing so must be transparent to the operation of the software, other than a difference in performance.

4–41. All caches must be fully initialized and enabled, and they must have accurate state bits prior to the transfer of control to the operating system.

4–42. If an in-line external cache is used, it must support one reservation as defined for the *Load And Reserve* and *Store Conditional* instructions.

4–43. For the Symmetric Multiprocessor or Power Management option: Platforms must implement their cache hierarchy such that all caches at a given level in the cache hierarchy can be flushed and disabled before any caches at the next level which may cache the same data are flushed and disabled (that is, L1 first, then L2, and so on).

4–44. For the Symmetric Multiprocessor or Power Management option: If a cache implements snarfing, then the cache must be capable of disabling the snarfing during flushing in order to implement the RTAS *cache-control* function in an atomic way.

4–45. Software must not depend on being able to change a cache from copy-back to write-through.

Software Implementation Note: Each first level cache will be defined via properties of the *cpu* node(s) of the OF device tree. Each higher level cache will be defined via properties of the *l2-cache* node(s) of the OF device tree. See the *PowerPC processor binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [11] for more details.

Personal Use Copy - Not for Reproduction

Software Implementation Note: To ensure proper operation, cache(s) at the same level in the cache hierarchy should be flushed and disabled before cache(s) at the next level (that is, L1 first, then L2, and so on).

Hardware Implementation Note: The PowerPC architecture would seem to imply that all caches can be controlled via the cache control and prefetching instructions and the page table attribute bits. In fact, the vast majority of external caches are not affected by these attributes of the PowerPC architecture and, in general, depend solely on a fixed address boundary check to determine cachability. Internal caches are expected to comply with the PowerPC processor architecture. It is the responsibility of the platform designer to verify that the allowed flexibility in external cache function affects only the performance and not the programming model (coherency, for example) of the platform.

Hardware Implementation Note: There may be a significant performance impact to some operating systems if parts of an OS's System ROM and/or a local bus video frame buffer are not actually cached outside the processor, for example, in an external Level 2 cache. The CHRP architecture permits this cost vs. performance trade-off to be made by hardware vendors, since software compatibility is, by definition, not affected.

Personal Use Copy - Not for Reproduction

Blank page when cut - 68

I/O Bridges 5

There is at least one I/O bridge in a Common Hardware Reference Platform. That is, there will be at least one bridge which interfaces to the system bus on the processor side, and interfaces to the Peripheral Component Interface (PCI) bus on the other. This bridge is called the PCI Host Bridge (PHB). The architecture for PHBs is defined in this chapter. In addition, there may be other bridge components in the platform to bridge from one I/O bus to some other or to the same I/O bus. For example, to bridge from the PCI bus to an ISA bus. Requirements for these other bridges will be discussed, as appropriate, in this chapter.

5.1 PCI Host Bridge (PHB) Architecture

The PHB architecture places certain requirements on PHBs. There should be no conflict between this document and the *PCI Local Bus Specification*, revision level 2.1 [14] document, but if there is, the PCI documentation takes precedence. The intent of this architecture is to provide a base architectural level which supports the PCI architecture and to provide optional constructs which allow for use of 32-bit PCI devices in platforms with greater than 4 GB of system addressability.

Requirements:

5–1. All PHB implementations must be compliant with the *PCI Local Bus Specification*, revision level 2.1 [14].

5–2. All requirements defined in Chapter 3, “System Address Map,” on page 23 for HBs must be implemented by all PHBs in the platform.

5–3. HB0 must be a PHB (PHB0).

5.1.1 PHB Implementation Options

There are a few implementation options when it comes to implementing a PHB. Some of these become requirements, depending on the characteristics of the system for which the PHB is being designed. Requirement 5–2, which states that all requirements defined in Chapter 3, "System Address Map," on page 23 for HBs must be implemented by all PHBs, defines the requirements for the following:

- The 64-bit addressing option requires that PHBs designed for use in platforms which are designed to support System Memory configured at an address of 4 GB or above must provide the TCE address translation mechanism in order to give 32-bit PCI devices in such platforms the capability to address all of System Memory directly. The existence of this 64-bit addressing option support is reported by the existence of the *64-bit-addressing* property in the PHB node(s) of the OF device tree, and the 64-bit addressing option can be enabled by the operating system by the *set-64-bit-addressing* OF method.
- Platforms which implement the 64-bit addressing option may optionally support the PCI Dual Address Cycle (DAC) capability during DMA operations. The existence of this DAC option support is reported by the existence of the *64-bit-dma* property in the PHB node(s) of the OF device tree, and this option is enabled along with the 64-bit addressing option by the operating system by the *set-64-bit-addressing* OF method.
- PHB implementations (for 32-bit or 64-bit addressing systems) may choose to implement one or both of the compatibility holes, as defined in Chapter 3, "System Address Map," on page 23.

5.1.2 Data Buffering and Instruction Queuing

Some PHB implementations may include buffers or queues for DMA, *Load*, and *Store* operations. The PowerPC architecture provides several instructions, including *eieio* and *sync*, which allow programs to order and synchronize operations to System Memory or I/O. These instructions may or may not propagate (depending on the processor and platform implementation) down to the PHB(s), but will definitely not propagate beyond a PHB. As far as *eieio* is concerned, where the propagation stops is immaterial, and the platform will keep the programming model the same (see requirement 5–4, below). The *sync*

Personal Use Copy - Not for Reproduction

instruction, however, cannot be propagated out to the end I/O device because the PCI bus does not lend itself to this capability. What this means is that for the cases where the programming model is dependent on completion of a write to an I/O device before continuing, software must take care of the synchronization. For example, the PCI architecture specifies that a read operation issued to a PCI device will flush the PCI and PHB data buffers. This can be used, then, to assure completion of a write to a device by doing a read of something on the device after writing to the device. This will force the write data to arrive at the device before the read data is returned (within the restrictions detailed in section 5.1.2.3, "PCI Delayed Read Interaction," on page 74). In addition, interrupts from a PCI device can bypass data written by the PCI device, and it may be necessary to use a similar technique to guarantee that data written by a device is forced out of data buffers on the I/O side of the PHB prior to using the data.

Most processor accesses to System Memory go through the processor data cache. When sharing System Memory with I/O devices, hardware must maintain consistency with the processor data cache and the System Memory, as defined by the requirements in Section 4.2.2.1, "Memory Coherence," on page 57.

Requirements:

5–4. PHB implementations which include buffers or queues for DMA, *Load*, and *Store* operations must make sure that these are transparent to the software, with a few exceptions which are allowed by the PCI architecture, by the PowerPC architecture, and in Section 4.2.2.1, "Memory Coherence," on page 57.

Since the implications of combining the PowerPC ordering rules and the PCI ordering rules and the exceptions that they allow are not obvious, the following sections will describe the combination of these rules and the resulting requirements on the PHB.

5.1.2.1 *Load* and *Store* Ordering

Although it is possible to create a PHB which would emulate an *eieio* instruction, such a PHB would be unnecessarily complex. By combining the PowerPC and PCI ordering rules and implementing those rules, a PHB can maintain the *eieio* programming model without having to put an *eieio* instruction in its instruction queue. The following are the requirements on the PHB in order to meet these other architectures' ordering rules.

Personal Use Copy - Not for Reproduction

Requirements:

5–5. A *Load* or *Store* to either the Peripheral Memory Space or the Peripheral I/O Space of a PHB must never be passed to the I/O bus before a previous *Store* to either the Peripheral Memory Space or the Peripheral I/O Space of that same PHB (that is, multiple *Stores* to the I/O bus generated by one PHB must be kept in order and a *Load* must not pass a *Store*).

5–6. A *Load* to either the Peripheral Memory Space or the Peripheral I/O Space of a PHB must never be passed to the I/O bus before a previous *Load* to either the Peripheral Memory Space or the Peripheral I/O Space of that same PHB when both of those *Loads* go to the exact same address.

With the exception of the one case stated in requirement 5–6 for two *Load* operations to the same address, a *Store* or *Load* to either the Peripheral Memory or Peripheral I/O Spaces is allowed to pass a previous *Load*, but is not required to do so. This works because when the software is concerned about the ordering of these, it will place an *eieio* after the *Load*, and the processor will not issue the following *Load* or *Store* until the data is returned from the previous *Load* (that is, the processor handles the ordering in this case when the software cares about the ordering). The reason for requirement 5–6 is that the read operations on the PCI bus are not tagged as to which device requested the read, and if one *Load* (read) is issued to the PCI bus and retried and a *Load* to the same address issued by another processor is allowed to pass this retried *Load*, then the second processor could get the data that was requested by the first processor. Even this would not be a problem, providing the second processor did not first do a *Store* to that address followed by a *Load*. In this case, the *Load* would pick up the data from the first processor's *Load*, and therefore would get back something different than what it thought it had previously sent with its *Store* (that is, without requirement 5–6 the sequence of *Load* from processor 1, followed by a *Store* by processor 2, followed by a *Load* by processor 2, for example, could return the wrong data to processor 2). Of course, since it is not a requirement for a *Store* or *Load* to either the Peripheral Memory or Peripheral I/O Spaces to be allowed to pass a previous *Load*, one solution (and legitimate implementation as an implementation cost versus performance trade-off) is to never allow any *Load* or *Store* to pass any other *Load* or *Store* in a PHB's instruction queue (that is, order all *Load* and *Store* operations from the processors).

Personal Use Copy - Not for Reproduction

5.1.2.2 DMA Ordering

There are certain ordering rules that DMA operations must follow. In general, the ordering for the DMA path operations from the I/O bus to the processor side of the PHB is independent from the *Load* and *Store* path, with one exception as stated in requirement 5–11 below, and, indeed must be independent in the cases which are stated in requirements 5–7 and 5–8, below. Note that in the requirements, below, a *read request* is the initial request to the PHB and the *read completion* is the data phase of the transaction (that is, the data is returned). The PHB may signal a "retry" on the initial read request and later provide the data when the requester comes back to get it (that is, when the device requests it again). Thus, the read request and read completion may occur during different PCI bus transactions.

Requirements:

5–7. Data from a DMA read completion must be allowed to complete prior to *Load* or *Store* operation which was previously queued in the PHB, in order to prevent a possible deadlock.

5–8. *Load* data buffered in a PHB must not prevent a subsequent DMA write request from being posted into the PHB or from making progress through the PHB, in order to prevent a possible deadlock on the PCI bus.

5–9. A DMA write or read request from an I/O device to the processor side of the PHB must never be passed to the processor side of the PHB before the data from a previous I/O DMA write operation has been flushed to the processor side.

5–10. A previous DMA read request accepted by a PHB but not yet completed must not prevent a subsequent DMA write request from being posted into the PHB or from making progress through the PHB, in order to prevent a possible deadlock on the PCI bus.

5–11. All DMA write data (destined for the processor side of the PHB) in the PHB buffers must be flushed out of the PHB prior to delivering data from a *Load* operation which has come after the DMA write operations.

Requirement 5–11 is the one case where the *Load* and *Store* path is coupled to the DMA path. This requirement guarantees that the software has a method for forcing DMA write data out of any buffers in the path prior to servicing a completion interrupt from the device. Note that the device can do that, also, via

Personal Use Copy - Not for Reproduction

requirement 5–9, by following any write by a read through the same path prior to issuing a completion interrupt.

A DMA read operation is allowed to be processed prior to the completion of a previous DMA read operation, but is not required to be.

5.1.2.3 PCI Delayed Read Interaction

As mentioned previously, read operations on the PCI bus are not tagged as to which device requested the read. For PCI devices designed prior to version 2.1 of the *PCI Local Bus Specification* [14], this did not matter, since the PCI architecture did not allow delayed read operations. Version 2.1 does allow delayed read operations (a delayed read operation is one where the target device retries the PCI master, but goes off and does the read operation anyway, so that it can have the data ready when the master comes back and tries the operation again). This means that if two devices on the PCI bus (one of which could be the PHB) are reading from the exact same address (that is, the exact same byte address), and if one or both of those devices can write the data at that address, then the device(s) doing the writing can get stale data upon a read. This has implications in programming certain I/O operations. Several scenarios can occur:

1. Two (or more) processors reading the same address on a PCI device, with one of those device drivers also writing that address.
2. Two peer PCI devices reading the same address on another PCI device, with one of those devices also writing that address.
3. Two peer PCI devices reading the same address in System Memory, with one of those devices also writing that address.
4. The device driver (where the PHB acts as a PCI master on behalf of the device driver) and a PCI I/O device reading the same address on another PCI device, with one of those also writing that address.

Except for scenario 1, software must create the appropriate protocols to assure that the problem does not occur. Scenario 1 is prevented from happening by requirement 5–6.

5.1.3 Byte Ordering Conventions

A system can run in Little-Endian (LE) or Big-Endian (BE) mode. Depending on which mode the system is operating, the paths for data from the processor to the I/O or the System Memory to the I/O may need to do something special based on the mode of operation. Some of the LE and BE aspects that are unique

Personal Use Copy - Not for Reproduction

to the PHB will be discussed below. For more information about system requirements, see Section 2.3, “Bi-Endian Support,” on page 14 and Appendix C, “Bi-Endian Designs,” on page 265.

The PCI bus itself can be thought of as not inherently having an endianess associated with it (although its numbering convention indicates LE). It is the devices on the PCI bus that can be thought of as having endianess associated with them. Some PCI devices will contain a mode bit to allow them to appear as either a BE or LE device. Some devices will even have multiple mode bits; one for each data path (*Load* and *Store* versus DMA). In addition, some devices may have multiple concurrent apertures, or address ranges, where the device can be accessed as a LE device in one aperture and as a BE device in another.

Requirements:

5–12. The hardware platform must be designed such that software must follow Table 9 on page 76 while running in the various processor modes (LE = 0 and LE = 1) and issuing *Load* and *Store* operations to various entities with various endianess, including doing any necessary address un-modification when running with LE = 1 (for platforms using processors implementing LE mode via address modification).

5–13. When performing DMA operations through a PHB while running with the processor mode of LE = 1, if the platform is implemented with the true LE format in System Memory, or while running with the processor mode of LE = 0 with BE format in System Memory, then the platform must not modify the data during the transfer process; the lowest addressed byte in System Memory being transferred to the lowest addressed byte on the PCI bus, the second byte in System Memory being transferred as the second byte on the PCI bus, and so on.

5–14. When performing DMA operations through a PHB while running with the processor mode of LE = 1, if the platform is implemented with the PowerPC LE format in System Memory, then the platform must transform the data during the transfer process to true LE format by reflecting the bytes within a doubleword (that is, the DMA is done as though the data is accessed a byte at a time, with the address modified by the same modification as used by the processor for 1-byte *Loads* and *Stores*, namely, exclusive-or the address with 0b111).

Personal Use Copy - Not for Reproduction

Table 9. *Load* and *Store* Programming Considerations

Destination	PowerPC Processor Mode: LE=0 (processor operating in BE mode)	PowerPC Processor Mode: LE=1 (processor operating in LE mode)
BE scalar entity: For example, TCE or BE register in a PCI device	*Load* or *Store*	*Load* or *Store Reverse*
LE scalar entity: For example, LE register in a PCI device or PCI Configuration Registers	*Load* or *Store Reverse*	*Load* or *Store*

Hardware Implementation Note: Requirement 5–14, above, may have implications on the design of a PHB, depending on the hardware platform implementation and component partitioning (see Appendix C, “Bi-Endian Designs,” on page 265 for more information).

Software Implementation Note: When dealing with non-scalar (string) data, BE and true LE data is in System Memory in the same order (MSB at the lowest address). The PCI device will typically treat the data register as a non-scalar register, and thus, as the equivalent of a BE scalar register. If the processor is running in the BE mode, then essentially the BE to BE case in Table 9 on page 76 holds true, and in such cases, programmed transfers of data is accomplished by doing (depending on the direction of data transfer) either a *Load* from the System Memory buffer followed by a *Store* to the I/O device or a *Load* from the I/O device and a *Store* to System Memory. If the processor is running in the LE mode and is implemented using the address modification technique, then the string is stored basically reflected within a doubleword in System Memory. In such cases, a *Load* of something greater than 1-byte will bring the string into the processor register in reverse order (most significant character of the string in the LSB of the register), and when the *Store* is issued to the device, the platform will reverse the order of the bytes within the operation size (as required for scalar data to get true LE data; that is, in order to make the lower right hand corner operation of Table 9 on page 76 come out correctly), and the data will appear with the most significant character of the string on the lowest address byte of the PCI bus. Moves from the I/O to System Memory work in the reverse. In any case, the *Load* or *Store Reverse* instructions are not used for these string move operations.

Personal Use Copy - Not for Reproduction

5.1.4 PCI Bus Protocols

This section details the items from the "PCI Local Bus Specification" and "PCI System Design Guide" documents where there is variability allowed, and therefore further specifications, requirements, or explanations are needed.

5.1.4.1 Peripheral I/O Space

Although an HB in general does not have to implement a Peripheral I/O Space, the PCI bus contains an I/O address space, and therefore requires a Peripheral I/O Space.

Requirements:

5–15. There must be exactly one Peripheral I/O Space per PHB.

5.1.4.2 Dual Address Cycle (DAC)

The DAC capability of the PCI architecture allows PCI devices which are attached to a 32-bit PCI bus to use a 64-bit address (the address phase of the transaction extending to two clocks). PHB support for DAC when the PHB is the target of a DMA operation and the 64-bit addressing option is enabled is optional, but is specified by this architecture (see Chapter 3, "System Address Map," on page 23 and Figure 8 on page 43). PHB support for DAC when the PHB is the master on the PCI bus (that is, support for *Load* or *Store* operations at or above 4 GB to Peripheral Memory Space) is not required and is not specified by this architecture. If DAC DMA capability is supported, this support is reported by the existence of the *64-bit-dma* property in the HB node(s) of the OF device tree, and is enabled along with the 64-bit addressing option by the operating system by the *set-64-bit-addressing* OF method.

5.1.4.3 PCI Interrupt Acknowledge Cycle

The PCI Interrupt Acknowledge Cycle was provided in the PCI architecture for use with bridges which require 8259-like interrupt controller support.

Requirements:

5–16. If a PHB has an interrupt controller on the PCI side of the bridge which requires the PCI Interrupt Acknowledge Cycle generation, then that PHB must provide a 1-byte register which, when read by a processor

Personal Use Copy - Not for Reproduction

using a 1-byte *Load* instruction, will generate a PCI Interrupt Acknowledge cycle on the PCI bus.

If this register exists in a PHB, then so will the *8259-interrupt-acknowledge* property of the root node of the OF device tree, and the address of this register will be passed via this *8259-interrupt-acknowledge* property in the root node.

5.1.5 Programming Model

Normal memory mapped *Load* and *Store* instructions are used to access a PHB's facilities or PCI devices on the I/O side of the PHB. Chapter 3, "System Address Map," on page 23 defines the addressing model. Addresses of I/O devices are passed by OF via the device tree.

Requirements:

5–17. If a PHB defines any registers that are outside of the PCI Configuration space, then the address of those registers must be in the Peripheral Memory Space or Peripheral I/O Space for that PHB, or must be in the System Control Area.

PCI master DMA transfers refer to data transfers between a PCI master device and another PCI device, an ISA device (when ISA is implemented), or System Memory, where the PCI master device supplies the addresses and controls all aspects of the data transfer. Transfers from a PCI master to the PCI I/O Space are essentially ignored by a PHB (except for address parity checking). Transfers from a PCI master to PCI Memory Space are either directed at PCI Memory Space (for peer to peer operations) or need to be directed to the host side of the PHB. DMA transfers directed to the host side of a PHB may be to System Memory or may be to another I/O device via the Peripheral Memory Space of another HB. Transfers that are directed to the Peripheral I/O Space of another HB are considered to be an addressing error (see Chapter 10, "Error and Event Notification," on page 157). For information about decoding these address spaces and the address transforms necessary, see Chapter 3, "System Address Map," on page 23.

5.2 I/O Bus to I/O Bus Bridges

The PCI bus architecture was designed to allow for bridging to other slower speed I/O buses or to another PCI bus. The requirements when bridging from one I/O bus to another I/O bus in the platform are defined below.

Personal Use Copy - Not for Reproduction

Requirements:

5–18. All bridges must comply with the bus specification(s) of the buses to which they are attached.

5.2.1 What Must Talk to What

Since there are a number of different address spaces and multiple buses in the platform, this section describes which devices must be able to access which address spaces.

An *ISA DMA* device is one which requires a DMA controller in the system to provide an address and control the transfer (also called a third party DMA device), whereas an *ISA DMA master* (or I/O bus master) is a device which generates the address and controls the operation.

Requirements:

5–19. Table 10 on page 79 details the minimum requirements that the platform must implement relative to I/O device access to the various address spaces.

Table 10. Which I/O Devices Must Be Able to Access Which Address Spaces

Source Device	Destination Device	Platform Support Required?	Other Requirements and Comments
ISA DMA master	ISA memory	No	
	PCI memory	Limited	Platforms must support this if the io-hole is implemented and the source and destination devices reside on the same side of the same PHB and the operation does not traverse a PCI to PCI bridge and the device is configured in the io-hole (640 KB to (1 MB - 1) address range) and the io-hole is enabled.
	System Memory	Limited	System Memory addresses in the 0 to (16 MB - 1) range only and with a possible hole due to the io-hole (if enabled) or due to ISA devices configured in this range.

Personal Use Copy - Not for Reproduction

Table 10. Which I/O Devices Must Be Able to Access Which Address Spaces ***(Continued)***

Source Device	Destination Device	Platform Support Required?	Other Requirements and Comments
ISA DMA	ISA memory	No	
	PCI memory	Yes	Platforms must support this when the source and destination devices reside on the same side of the same PHB and the operation does not traverse a PCI to PCI bridge. The destination device must be configured either in the BPM to (BIM - 1) range, or if io-hole is enabled, in the 640 KB to (1 MB - 1) address range.
	System Memory	Yes	System Memory addresses in the 0 to (16 MB - 1) range should be accessed via the system-memory-alias area to get around possible holes in the address space due to io-hole being enabled or due to ISA devices configured in this range.
PCI Master	ISA memory	Yes	The PHB as a PCI master must be able to access the full 16 MB ISA Memory Space. Other PCI masters must support this when the ISA memory is in the 640 KB to (1 MB - 1) range and the io-hole is enabled.
	PCI memory	Yes	Platforms must support this when the source and destination devices reside on the same side of the same PHB. In this case, the destination device must be configured either in the BPMn to TPMn (n not equal to 0; BPM0 to (BIM - 1) for PHB0) range, or for PHB0, if io-hole is enabled, may also reside in the 640 KB to (1 MB - 1) address range. Platforms with multiple PHBs may also support this when the destination device resides in the Peripheral Memory Space of a different PHB than the source device.
	System Memory	Yes	For PHB0, if the io-hole is enabled, then the system-memory-alias must be enabled and used to access System Memory addresses in the 640 KB to (1 MB - 1) range.
ISA DMA or DMA Master	ISA or PCI I/O	No	
PCI Master	I/O Space of ISA or PCI	Yes	This is only required when the destination device resides on the same side of the same PHB as the source device (see also requirement 3–6).

Some operating systems may not support all these modes (refer to information available from the operating systems). Examples of what an operating sys-

Personal Use Copy - Not for Reproduction

tem might not support include:

- A PCI device which requires configuration below 1 MB when the operating system does not support the io-hole.
- PCI devices which require a PCI address below 1 MB and which will not configure into the io-hole due to their size may not be configured by the OF or the operating system.
- ISA devices which would cause holes in the ISA DMA master buffer area in System Memory or the corresponding ISA DMA master(s). Therefore system board devices should not be configured outside the io-hole range, and firmware should configure I/O devices in the io-hole range if possible.

ISA to PCI operations cannot traverse a PCI to PCI bridge or else a deadlock condition can occur.

5.2.2 PCI to PCI Bridges

The CHRP architecture allows the use of PCI to PCI bridges in the platform. If the 64-bit addressing option is enabled, the TCEs can be used with the devices attached to the other side of the PCI to PCI bridge when those devices are accessing something on the processor side of the PHB. After configuration, PCI to PCI bridges are basically transparent to the software as far as addressing is concerned (the exception is error handling). For more information, see the *PCI to PCI Bridge Architecture Specification* [16].

Requirements:

5–20. PCI to PCI bridges used on the base platform must implement the architecture as specified in the *PCI to PCI Bridge Architecture Specification* [16].

Hardware Implementation Note: PCI to PCI bridges may limit operations to the PCI I/O space to 16 bits on the side opposite the PHB (that is, addresses of 0 to (64K - 1) are passed through the bridge, but not addresses of 64K and above). Therefore, I/O devices will be limited to the first 64 KB of Peripheral I/O Space (with discontiguous mode disabled) in such cases.

Personal Use Copy - Not for Reproduction

5.2.3 PCI to ISA Bridges

The CHRP architecture allows for at most one ISA bridge attached to one (and only one) PCI bus in a platform. If the 64-bit addressing option is enabled, the TCEs can be used with ISA devices when those devices are accessing something on the processor side of the PHB.

Requirements:

5–21. There must be at most one ISA bus in a platform.

5–22. If there is an ISA bridge on a platform, it must be attached to the I/O side of HB0.

5–23. If there is an ISA bridge on a platform, a DMA controller must be available for the ISA DMA operations, and the DMA controller must be compatible with the register set defined in Chapter 9, “I/O Devices,” on page 151.

5–24. OF must program a PCI to ISA bridge such that all addresses for ISA DMA master operations get passed through the PCI to ISA bridge and do not get translated as they pass through the bridge.

5–25. If an ISA device is to participate in PCI to ISA peer to peer operations then the ISA device must be configured in the 640 KB to (1 MB - 1) address range (the io-hole) and the io-hole must be enabled.

5–26. PCI to ISA bridges must do a subtractive decode on the PCI side of the bridge in the PCI Memory Space from 0 to (16 MB - 1) and in the PCI I/O Space from 0 to (64 KB - 1) (that is, they must pass any PCI access in these address ranges to the ISA bus if the PCI cycles in these ranges are not first picked up by another PCI device).

Requirement 5–21 does not imply the number of ISA expansion slots and an ISA bus without slots counts as one ISA bus. It may be possible to meet requirement 5–21 with multiple physical ISA buses, but it must appear to software to be one logical ISA bus, including all programming models and the appearance in the OF device tree as one and only one PCI to ISA bridge.

Since the OF may not know which devices will participate in PCI to ISA peer to peer operations, requirement 5–25 says that OF should configure all ISA devices in the io-hole address range if possible. See also the requirements in Section 5.2.1, “What Must Talk to What,” on page 79 for additional requirements.

Personal Use Copy - Not for Reproduction

Hardware Implementation Notes:

- Type F DMA is recommended for multimedia operations.
- The IEEE definition of the ISA bus is in draft form and is called IEEE 996, *A Standard for an Extended Personal Computer Back Plane Bus* [13].
- The *PCI to PCI Bridge Architecture Specification* [16] does not support the attachment of PCI to ISA bridges to the PHB through a PCI to PCI bridge, and does not support ISA DMA or ISA DMA master transactions through a PCI to PCI bridge to another PCI device.

5.2.4 16-Bit PC Card (PCMCIA) and Cardbus PC Card Bridges

The CHRP architecture allows for 16-bit PC Card and Cardbus PC Card bridges. If the 64-bit addressing option is enabled, the TCEs can be used with 16-bit PC Card or Cardbus PC Card devices. For more information on PC Cards, see the *PC Card Standard* specification [17].

If a platform supports 16-bit PC Card or Cardbus PC Card cards, then the following requirement applies.

Requirements:

5–27. A platform which supports Cardbus PC Card devices must also support 16-bit PC Card devices.

Personal Use Copy - Not for Reproduction

Blank page when cut - 84

Interrupt Controller 6

This chapter specifies the requirements for the CHRP interrupt controller. It also proposes a distributed implementation of the interrupt controller.

6.1 Interrupt Controller Architecture

The Open PIC Multiprocessor Interrupt Controller Register Interface Specification[1] is the basis of the CHRP interrupt controller architecture. The Open PIC Specification contains a register-level architectural definition of an interrupt controller that supports up to 32 processors. The Open PIC specification defines means for assigning properties such as priority, vector, destination, etc., to I/O and interprocessor interrupts, as well as an interface for presenting them to processors. It supports both specific and distributed methods for interrupt delivery. The interrupt controller also provides timer-interrupt facilities and means to facilitate assertion of “Initialize” signals to processors.

Requirements:

6–1. Platforms must implement interrupt controllers that are in register-level architectural compliance with *Open PIC Multiprocessor Interrupt Controller Register Interface Specification,* Revision 1.2 [8] including its PowerPC architecture appendix.

[1] The PowerPC architecture Appendix in the Open PIC Specification Revision 1.2 defines options for PowerPC-based systems. However, more options are currently in the process of being included in the said appendix and are expected to appear in future revisions of the specification.

6–2. The Interrupt Acknowledge register implemented in the CHRP interrupt controller.

6–3. Platforms must make per-processor registers available in the “publicly accessible area” of the Open PIC register map.

6–4. All interrupt controller registers must be accessed via Caching-Inhibited and Guarded mapping.

6–5. The INIT signals defined in Open PIC must be connected to Soft Reset pins on PowerPC processors.

6–6. Interrupts must be disabled at the CHRP interrupt controller at the point of transfer of control to the operating system.

Software Implementation Note: Private access interface to the per-processor registers is optional. Software should not depend on it being available.

Software Implementation Note: The Who Am I register might not be implemented in a CHRP interrupt controller. An alternative system-specific method may be defined, instead, for a processor to find out its own Open PIC identity for use in specifying interrupt destinations. For an example of such a method, see Section Section 12.2, “An SMP Boot Process,” on page 218.

Hardware Implementation Note: Allocation of PCI interrupts is discussed in Section 9.1.3, “Assignment of Interrupts to PCI Devices,” on page 152.

Hardware Implementation Note: The number of I/O interrupts supported are platform dependent.

6.2 Distributed Implementation — A Proposal

It is an intention of the CHRP architecture to allow a variety of system structures and sizes. It is in keeping with this intention to expect system structures that include multiple I/O busses which are physically distant from each other and from the processors themselves. It would be impractical in such systems to have an interrupt controller implemented as a single centrally placed integrated circuit. The following distributed implementation of the Open PIC architecture is therefore proposed. The CHRP architecture provides addressing infrastructure to support it.

Personal Use Copy - Not for Reproduction

The interrupt controller is logically divided into two partitions: The I/O interrupt source partition containing the Interrupt Source Registers that accepts I/O interrupt signals and translates them into the vector, priority, and destination information to be presented to processors; and another partition that contains the rest of the controller functionality. The I/O interrupt source partition may be further partitioned into multiple sub-partitions. Each sub-partition represents an I/O interrupt group. An I/O interrupt group is defined by its lowest interrupt number and the number of interrupts in the group. These are specified via Open Firmware Interrupt Controller properties. See section on Open PIC interrupt controller node properties in *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10].

The interrupt controller may be partitioned into one or more physical units: A base unit called the Interrupt Delivery Unit (IDU), which houses the global, timer-related, and public-access per-processor registers, contains logic to deliver interrupts to processors, and may optionally house a single I/O interrupt source sub-partition. Additionally, there may be zero or more external Interrupt Source Units (ISUs), each housing Interrupt Source Registers for an I/O interrupt group.

The IDU and the external ISUs may communicate via a system-dependent means. Figure 11 on page 88 shows a system map with the IDU containing an I/O interrupt group and one external ISU.

Personal Use Copy - Not for Reproduction

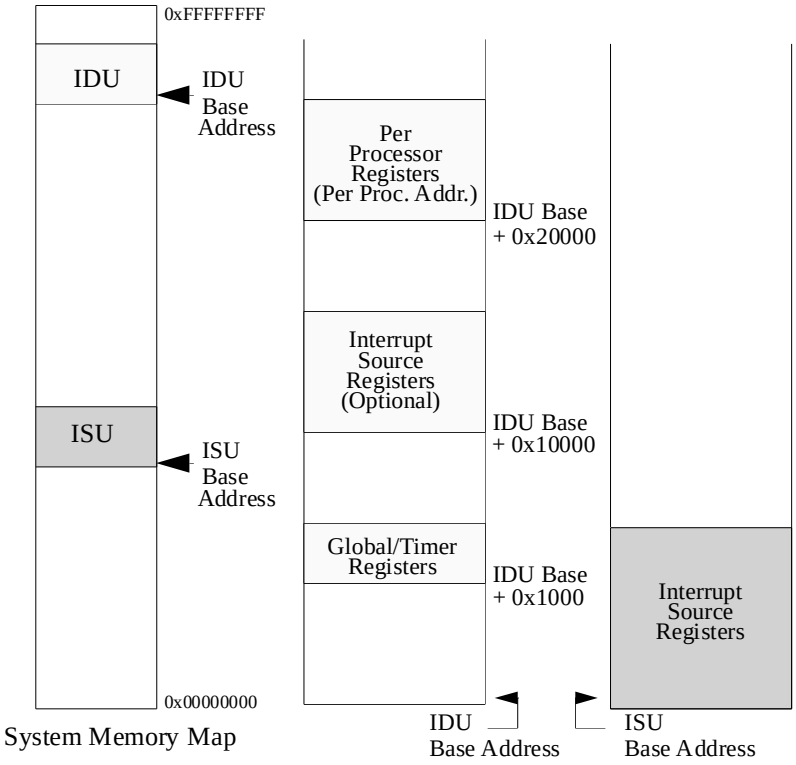


Figure 11. System Memory Map Showing Mapping of the IDU and an ISU

The IDU and the ISUs are individually mapped, in a non-overlapped manner, in the system memory. Typically, an external ISU will be mapped within the address range occupied by the bus, from whose devices it receives interrupts. Open Firmware properties for the interrupt controller specify the base addresses for the IDU and for the external ISUs, if any. See section on Open PIC interrupt controller node properties *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10].

The relative offsets of register addresses within the IDU with respect to its base address are identical to those specified in the Open PIC Specification. In the ISU, the ISRs for interrupt group are mapped contiguously starting at the

Personal Use Copy - Not for Reproduction

base address of the ISU with the lowest numbered interrupt first. All registers are aligned on 16-byte boundaries, as required by the Open PIC Specification.

Proposed requirements:

- Platforms must have an IDU.
- The registers in the IDU must be placed at offsets to the base address defined in the Open PIC Specification.
- Platforms must support at least one I/O interrupt group, either via the IDU or via an external ISU.
- The ISRs in an external ISU must be placed contiguously starting at the base address specified for it with the lowest numbered interrupt first.
- All registers in the CHRP interrupt controller must be aligned on 16-byte boundary and must be implemented in Little Endian format.

Hardware Implementation Note: A centralized, single-chip implementation is covered by the distributed scheme just outlined, as a special case. In such an implementation, the IDU supports the only I/O interrupt group in the system.

Software Implementation Note: Open Firmware *reg* properties for the interrupt controller supply base addresses of the IDU and of the external ISUs, if any. Open Firmware *interrupt-ranges* properties of the interrupt controller specify the interrupt groups supported by the IDU and by the external ISUs, if any. Software should use Open Firmware properties of the interrupt controller to locate the Open PIC registers.

Personal Use Copy - Not for Reproduction

Blank page when cut - 90

Run-Time Abstraction Services

7

7.1 RTAS Introduction

The Run-Time Abstraction Services (RTAS) functions are provided by CHRP platforms to insulate operating systems from having to know about and manipulate a number of key platform functions which ordinarily would require platform-dependent code. Operating systems call these functions rather than manipulating hardware registers directly, reducing the need for platform tailoring by OS suppliers. This method of abstracting access to these platform functions also permits hardware vendors considerable flexibility in hardware implementation. Since RTAS is provided by the hardware vendor, this approach places the responsibility for supporting the platform with the hardware vendor, not the OS vendor. This permits separating the schedules of hardware and software, permits different suppliers to provide each, and reduces the release and test requirements for OSs, since they can be tested to conform to the RTAS interfaces and not to every specific hardware configuration.

In order for platforms to achieve this separation of operating system code from platform dependencies, RTAS defines an interface between the platform and the operating systems that provides control of some of the common devices found on all CHRP platforms. RTAS is a system programming interface that is realized on a specific platform by an RTAS implementation. The RTAS implementation provides the platform specific processing of the common components while operating system drivers are necessary to provide device specific processing for I/O adaptors. In general, operating systems should not access RTAS resources directly. They should call RTAS to control the device.

RTAS limits itself to the run-time control of non-I/O, typically system board-resident hardware features. Traditionally, features such as these have almost always been implemented differently on different platforms. This list of features includes things like non-volatile memory, time-of-day clock, memory bridges, and power management control functions. Such differences traditionally require much effort and require platform-dependent code in OSs. RTAS permits an OS to operate over a much wider range of platforms without specialized code for each platform. The presence of RTAS does not prevent an OS from incorporating customized code to manipulate arbitrary platform hardware, provided such hardware is defined in the Open Firmware Device Tree as being present. We refer to such OSs as being "aware" of the platform feature.

The role of RTAS versus Open Firmware is very important to understand. Open Firmware and RTAS are both vendor-provided software, and both are tailored by the hardware vendor to manipulate the specific platform hardware. However, RTAS is intended to be present *during* the execution of the OS, and to be called by the OS to access platform hardware features on behalf of the OS, whereas Open Firmware *need not* be present when an OS is running. This frees Open Firmware's memory to be used by applications. RTAS is small enough to painlessly coexist with OSs and applications.

This chapter uses the term RTAS to refer both to the architected RTAS interface and to an RTAS implementation.

7.2 RTAS Environment

Since RTAS provides an interface definition between the operating system and the firmware provided by the platform vendor, both the operating system and the firmware on the hardware platform must use the calling convention as defined in this chapter.

RTAS must operate in an environment that does not interfere with the operating system. It can not cause any exceptions, nor can it depend on any particular virtual memory mappings. For this reason, it runs in real mode with exceptions disabled.

All RTAS functions are invoked from the operating system by calling the *rtas-call* function. The address of this function is obtained from Open Firmware when RTAS is instantiated. See requirement 7–13 for more details. RTAS will determine what function to invoke based on the data passed into the *rtas-call* function. This section describes the mechanisms used to invoke the *rtas-call* function, the machine state, register usage, resource allocation, and the invocation requirements.

Personal Use Copy - Not for Reproduction

7.2.1 Machine State

When RTAS functions are invoked, the calling processor shall have address translations, floating point, and most other exceptions disabled and it shall be running in privileged state.

Requirements:

7–1. RTAS must be called in "real mode," that is, all address translation must be disabled. Bits IR and DR of the MSR register must be zero.

7–2. RTAS must be called in privileged mode, and the PR bit of the MSR must be set to 0.

7–3. RTAS must be called with external interrupts disabled, and the EE bit of the MSR must be set to 0.

7–4. RTAS must be called with trace disabled, and the SE and BE bits of the MSR must be set to 0.

7–5. RTAS must be called with floating point disabled, and the FE0, FE1 and FP bits must be set to 0.

7–6. RTAS must be called with the SF and LE bits of the MSR set to the same values that were in effect at the time that RTAS was instantiated.

7–7. With the exception of the DR and RI bits, RTAS must not change the state of the machine by modifying the MSR.

7–8. If *rtas-call* is entered in a non-recoverable mode, indicated by having the RI bit of the MSR set equal to 0, then RTAS must not enter a recoverable mode by setting the RI bit to 1.

7–9. If called with RI of the MSR equal to 1, then RTAS must protect its own critical regions from recursion by setting the RI bit to 0 when in the critical regions.

Software Implementation Note: If the ME bit is left enabled, the operating system's exception handler must be aware that RTAS might have been running and that various processor registers might not be in the expected state for an interrupted operating system process. If the operating system cannot tolerate this, it should disable ME while RTAS is running, or set RI to zero which would preclude recoverability but permit logging machine checks.

Software Implementation Note: There are some provisions for recursive calls to RTAS error handling functions. Therefore, RTAS should set the

Personal Use Copy - Not for Reproduction

RI bit in the MSR if SRR0/SRR1 or any other RTAS resource is in a state where information would be lost and prohibit recovery.

Software Implementation Note: Requirement 7–6 implies that RTAS must be prepared to be instantiated in either Big-Endian or Little-Endian modes, and to be able to be instantiated in 64-bit mode on platforms that can support 64-bit execution.

7.2.2 Register Usage

Requirements:

7–10. Except as required by a specific function, RTAS must not modify the following operating environment registers: TB, DEC, SPRG0-SPRG3, EAR, DABR, SDR1, ASR, SR0-SR15, FPSCR, FPR0-FPR3, and any processor specific registers.

7–11. RTAS must preserve the following user mode registers: R1-R2, R13-R31, and CR.

7–12. RTAS must preserve the following operating environment registers: MSR, DAR, DSISR, IBAT0-IBAT3, and DBAT0-DBAT3.

Software Implementation Note: RTAS is entered in real mode (with address translation turned off). In this mode, all data accesses are assumed to be cached in copy back mode with memory coherence required. Since these settings may not be appropriate for all accesses, RTAS is free to use the Block Address Translation registers to set up its own virtual mappings. The operating system machine check handler can only depend on those registers that are required to be unchanged (see requirement 7–10).

Software Implementation Note: RTAS either may not change the preserved registers, or may save them as long as they are restored before returning to the operating system.

Software Implementation Note: The SRR0-SRR1, LR, CTR, XER registers, as well as any reservations made via the load and reserve instructions, need not be preserved.

Personal Use Copy - Not for Reproduction

7.2.3 RTAS Critical Regions

The operating system that uses RTAS is responsible for protecting RTAS and devices used by RTAS from any simultaneous accesses that could corrupt memory or device registers. Such corruption could be caused by simultaneous execution of RTAS code, or by a device driver accessing a control register that is also modified by RTAS. In a single processor system, most recursive calls to RTAS are prevented by clearing the EE bit of the MSR. This will limit recursive calls to RTAS to those made from the machine check handler. This handler may need to call various RTAS services such as *check-exception* or *system-reboot* even if the error was detected while in a RTAS service.

The operating system and RTAS must co-exist on the same platform. RTAS must not change device registers that are used by the operating system, nor may the operating system change device registers on devices used by RTAS. With the advent of more and more integration into common super parts, some of these registers may physically reside on the same component. In this section, device implies the collection of common registers that together perform a function. Each device must be represented in the Open Firmware Device Tree.

Requirements:

7–13. Except as noted in requirement 7–19, the operating system must ensure that RTAS calls are not re-entered and are not simultaneously called from different processors in a multi-processor system.

7–14. Any RTAS access to device or I/O registers specified in this document must be made in such a way as to be transparent to the operating system.

7–15. Any device that is used to implement the RTAS abstracted services must have the property *used-by-rtas* in the Open Firmware Device Tree. However, if the device is only used by the *suspend, hibernate, power-off,* and *system-reboot* calls, which do not return directly to the operating system, the property should not be set. The *display-character* device must be marked *used-by-rtas* only if it is a specialized device used only for *display-character.*

7–16. Platforms must be designed such that accesses to devices that are marked *used-by-rtas* have no side effects on other registers in the system.

7–17. Any operating system access to devices specified as *used-by-rtas* must be made in such a way as to be transparent to RTAS.

Personal Use Copy - Not for Reproduction

7–18. RTAS must not generate any exceptions (for example, no alignment exceptions, page table walk exceptions, etc.).

7–19. The operating system machine check and soft reset handlers may call the RTAS services:

- *nvram-fetch*
- *nvram-store*
- *check-exception*
- *set-indicator*
- *system-reboot*
- *set-power-level(0,0)*
- *power-off*

Software Implementation Note: While RTAS must not generate any exceptions, it is still possible that a machine check interrupt may occur during the execution of a RTAS function. In this case the machine check handler may be entered from the RTAS service.

Software Implementation Note: It is permissible for an operating system exception handler to make an RTAS call as long as requirements 7–13 and 7–10 are met. In particular, it is expected that the RTAS *check-exception* will be called from the operating system exception handlers.

7.2.4 Resource Allocation and Use

During execution, RTAS will require memory for both code and data. This memory may be in RAM, in a private memory area only known by the system firmware, or in memory allocated by the operating system for RTAS use. RTAS should use this memory for its stack and any state savings.

Requirements:

7–20. The operating system must allocate *rtas-size* bytes of contiguous real memory as RTAS private data area. This memory must be aligned on a 4096 byte boundary and may not cross a 256 megabyte boundary.

7–21. The RTAS private data area must not be accessed by the operating system.

Personal Use Copy - Not for Reproduction

7–22. Except for the RTAS private data area, the argument buffer, System Memory pointed to by any reference parameter in the argument buffer, and any other System Memory areas explicitly permitted in this chapter, RTAS must not modify any System Memory. RTAS may, however, modify System Memory during error recovery provided that such modifications are transparent to the operating system.

7–23. If the operating system moves or otherwise alters the addresses assigned to ISA or PCI devices that are marked *used-by-rtas* after it has instantiated RTAS, then the operating system must restart RTAS by calling *restart-rtas* prior to making any further RTAS calls.

7–24. RTAS must execute in a timely manner and may not sleep in any fashion nor busy wait for more than a very short period of time.

Software Implementation Note: A RTAS call should take the same amount of time to perform a service that it would take an operating system to perform the same function. A specific goal is that RTAS primitives should take less than a few tens of microseconds.

Software Implementation Note: Resources that need to be manipulated by RTAS, such as NVRAM, may reside on the PCI bus. If the operating system moves these devices, then RTAS must be restarted.

7.2.5 Instantiating RTAS

RTAS is instantiated by an explicit client interface service call into Open Firmware. The Open Firmware Device Tree contains a property (*rtas-size*, under the *rtas* node) which defines how much real memory RTAS requires. The operating system allocates *rtas-size* bytes of real memory, and then invokes the *instantiate-rtas* method of the RTAS node, passing this address as the *rtas-base-address* input argument. Firmware binds RTAS to that address, binds the addresses of devices that RTAS uses, performs any RTAS initialization, and returns the address of the *rtas-call* function that is appropriate for the current machine state.

Requirements:

7–25. The *instantiate-rtas* Open Firmware method must have the arguments specified in Table 11 on page 98.

Personal Use Copy - Not for Reproduction

Table 11. *instantiate-rtas* Argument Call Buffer

Parameter Type	Name	Values
In	*rtas-base-address*	Real Address of RTAS memory
Out	rtas-call	Real address used to invoke RTAS functions

7–26. RTAS must operate in the endian mode in effect at the time of the *instantiate-rtas* call.

Software Implementation Note: The firmware may provide two distinct implementations, one for each endianess, or may provide a single implementation that is bi-endian. The implementation technique is left up to the designer of the firmware.

7.2.6 RTAS Device Tree Properties

The Open Firmware Device Tree must contain an *rtas* device node that describes the implemented RTAS features and the output device supported by RTAS. Within this device node will be properties that describe the RTAS functions implemented by the firmware. For every implemented function, there will be an Open Firmware property whose name is the same as the RTAS function. The value of this property is passed into the *rtas-call* function when making a RTAS call. Note that some RTAS functions are optional and some are required. This is defined in Table 12 on page 99.

Requirements:

7–27. The Open Firmware Device Tree must contain a device node named *rtas* which describes the RTAS implementation.

7–28. The RTAS device node must have a property for each implemented RTAS function in Table 12 on page 99. The value of this property is a token that is passed into the *rtas-call* function to indicate which RTAS function to invoke.

Personal Use Copy - Not for Reproduction

Table 12. RTAS Tokens for Functions

RTAS property/function	Required?	Notes
restart-rtas	Required	
nvram-fetch	Required	Execution time proportional to amount of data
nvram-store	Required	Execution time proportional to amount of data
get-time-of-day	Required	
set-time-of-day	Required	
set-time-for-power-on		
event-scan	Required	
check-exception	Required	
read-pci-config	Required	
write-pci-config	Required	
display-character		
set-indicator	Required	Some specific indicators are required, and some are optional
get-sensor-state		
set-power-level	Required in Power Managed Platforms	
get-power-level		
assume-power-management		
relinquish-power-management		
power-off		Provided for platforms with software controlled power off capability, with or without other Power Management capability
hibernate		
suspend		
system-reboot	Required	

Personal Use Copy - Not for Reproduction

Table 12. RTAS Tokens for Functions ***(Continued)***

RTAS property/function	Required?	Notes
cache-control	Required in Power Managed Platforms	
	Required in SMP Platforms	
freeze_time_base	Required in SMP Platforms	Turn off time base
thaw_time_base		Turn time base back on
stop-self		
start-cpu		

7–29. The Open Firmware properties listed in Table 13 on page 100 must be in the RTAS Device Tree node prior to booting the operating system.

Table 13. Open Firmware Device Tree Properties

name	value
rtas-size	integer size of RTAS memory area in bytes
rtas-version	An integer encoding of the RTAS interface version. This document describes version 1.
rtas-event-scan-rate	The rate, in calls per minute, at which rtas-event-scan should be called by the operating system. See Section 7.3.4.1, “Event-scan,” on page 110.
rtas-display-device	The *phandle* of the device node used by the RTAS call, display-character
rtas-error-log-max	The maximum size of an extended error log.

7–30. All RTAS functions listed as “Required” in Table 12 on page 99 must be implemented in RTAS.

7–31. For the Power Management option: The functions listed as “Required in Power Managed Platforms” in Table 12 on page 99 must be implemented in RTAS.

7–32. For the Symmetric Multiprocessor option: The functions listed as “Required in SMP Platforms” in Table 12 on page 99 must be implemented in RTAS.

Personal Use Copy - Not for Reproduction

Software Implementation Note: It is permitted for RTAS not to implement those functions that are not appropriate or not needed on a particular platform. For example, if the system does not have power management features, then it is perfectly reasonable not to implement the RTAS power management calls.

Software Implementation Note: Vendors may introduce private RTAS calls of their own. If they do, the property names should be of the form *"vendor,property"* where *vendor* is a company name string as defined by Open Firmware. Future versions of this architecture will not choose RTAS property names that include a comma.

7.2.7 Calling Mechanism and Conventions

RTAS is called through a mechanism similar to the Open Firmware client interface service. An argument buffer is constructed which describes the desired RTAS call. This description includes an indication of the RTAS call that is being invoked, the number and value of the input parameters, the number of result values, and space for each of the result values.

Requirements:

7–33. In order to make an RTAS call, the operating system must construct an argument call buffer aligned on an eight byte boundary in physically contiguous real memory as described by Table 14 on page 101.

Table 14. RTAS Argument Call Buffer

Cell Number	Use
1	Token Specifying which RTAS Call
2	Number of Input Parameters
3	Number of Output Parameters
4	First Input Parameter
...	Other Input Parameters
4 + Number of Inputs -1	Last Input Parameter
4 + Number of Inputs	First Output Value

Personal Use Copy - Not for Reproduction

Table 14. RTAS Argument Call Buffer ***(Continued)***

Cell Number	Use
...	Other Output Parameters
4 + Number of Inputs + Number of Outputs -1	Last Output Value

7–34. If the system is a 32-bit system, or if the SF bit of the MSR was 0 when RTAS was instantiated, then all cells in the RTAS argument buffer must be 32-bit sign extended values that are aligned to 4 byte boundaries.

7–35. If the SF bit of the MSR was 1 when RTAS was instantiated, then all cells in the RTAS argument buffer must be 64-bit values that are aligned to 8 byte boundaries.

7–36. RTAS functions must be invoked by branching to the *rtas-call* address which is returned by the *instantiate-rtas* Open Firmware method (see Table 11 on page 98).

7–37. Register R3 must contain the argument buffer's real address when *rtas-call* is invoked.

7–38. Register R4 must contain the real address of the RTAS private data area when *rtas-call* is invoked (see requirement 7–20).

7–39. The Link Register must contain the return address when *rtas-call* is invoked.

Software Implementation Note: RTAS is not required to perform sanity checking of its input parameters. Using invalid values for any parameter in a RTAS argument buffer gives undefined results.

Software Implementation Note: The token that specifies the RTAS call is obtained by looking up the desired call from the *rtas* node of the Open Firmware Device Tree.

Software Implementation Note: Most operating systems will need to implement simple wrappers to provide an interface that is natural for the operating system (for example C function calls to the RTAS interface). These interfaces are operating system specific and are beyond the scope of this specification.

Personal Use Copy - Not for Reproduction

7.2.8 Return Codes

Requirements:

7–40. The first output value of all the RTAS functions must be a status word which denotes the result of the call. The status word will take on one of the values in Table 15 on page 103. Non-negative values indicate success.

Table 15. RTAS Status Word Values

Values	Status Word Meanings
0	RTAS function call succeeded.
-1	RTAS function call encountered a hardware error or failed for some unspecified reason.
-2	A necessary hardware device was busy, and the requested function could not be performed. The operation should be retried at a later time.
9000-9999	Reserved for vendor specific success codes.
-9000- (-9999)	Reserved for vendor specific error codes.
Additional Negative Numbers	An error was encountered. The meaning of this error is specific to the RTAS function that was invoked.
Additional Positive Numbers	The function succeeded. The meaning of the status word is specific to the RTAS function that was invoked.

7.3 RTAS Call Function Definition

This section specifies the semantics of all the RTAS calls. It specifies the RTAS function name, the contents of its argument call buffer (its token, input parameters, and output values) and semantics.

7.3.1 restart-rtas

If the operating system moves or otherwise alters addresses assigned to PCI or other buses, the resources used by RTAS may be assigned to different locations. RTAS needs to be informed of this change so it can continue to access its resources. It is the responsibility of RTAS to determine the new locations of its

Personal Use Copy - Not for Reproduction

devices by reading PCI configuration space or referencing unarchitected registers known to RTAS code.

Requirements:

7–41. RTAS must implement a *restart-rtas* function that uses the argument call buffer defined by Table 16 on page 104.

Table 16. *restart-rtas* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *restart-rtas*
	Number Inputs	0
	Number Outputs	1
Out	Status	0: Success -1: Hardware Error

7–42. If any device marked *used-by-rtas*, or a device used as the *rtas-display-device* if the OS will in the future call the *display-character* function, is moved or reconfigured, then the operating system must invoke *restart-rtas* before using any other RTAS function.

7–43. RTAS must update any necessary configuration state information based on the current configuration of the machine when *restart-rtas* is called.

7.3.2 NVRAM Access Functions

The architecture requires an area of non-volatile memory (NVRAM) to hold Open Firmware options, RTAS information, machine configuration state, operating system state, diagnostic logs, etc. The type and size of NVRAM is specified in the Open Firmware Device Tree. The format of NVRAM is detailed in Chapter 8, “Non-Volatile Memory,” on page 141.

In order to give a shrink wrapped operating system the ability to access NVRAM on different platforms that may use different implementations or locations for NVRAM, some layer of abstraction must be provided to the operating system. The functions in this section provide an interface for reading and writing NVRAM.

Personal Use Copy - Not for Reproduction

7.3.2.1 nvram-fetch

The RTAS function *nvram-fetch* will copy data from a given offset in NVRAM into the user specified buffer.

Requirements:

7–44. RTAS must implement an *nvram-fetch* function that returns data from NVRAM using the argument call buffer defined by Table 17 on page 105.

Table 17. *nvram-fetch* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *nvram-fetch*
	Number Inputs	3
	Number Outputs	2
	Index	Byte offset in NVRAM
	Buffer	Real address of data buffer
	Length	Size of data buffer (in bytes)
Out	Status	0: Success -1: Hardware Error -3: Parameter out of range
	Num	Number of bytes successfully copied

7.3.2.2 nvram-store

The RTAS function *nvram-store* will copy data from the user specified buffer to a given offset in NVRAM.

Requirements:

7–45. RTAS must implement an *nvram-store* function that stores data in NVRAM using the argument call buffer defined by Table 18 on page 106.

Personal Use Copy - Not for Reproduction

Table 18. *nvram-store* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *nvram-store*
	Number Inputs	3
	Number Outputs	2
	Index	Byte number in NVRAM
	Buffer	Real address of data buffer
	Length	Size of data buffer (in bytes)
Out	Status	0: Success -1: Hardware Error -3: Parameter out of range
	Num	Number of bytes successfully copied

7–46. If the *nvram-store* operation succeeded, the contents of NVRAM must have been updated to the user specified values. The contents of NVRAM are undefined if the RTAS call failed.

7–47. The caller of the *nvram-store* RTAS call must maintain the NVRAM partitions as specified in Chapter 8, “Non-Volatile Memory,” on page 141.

7.3.3 Time of Day

The minimum system requirements include a non-volatile real-time clock which maintains the time of day even if power to the machine is removed. Minimum requirements for this clock are described in Table 2 on page 19.

7.3.3.1 Time of Day Inputs/Outputs

Requirements:

7–48. The date and time inputs and outputs to the RTAS time of day function calls are specified with the year as the actual value (for example, 1995), the month as a value in the range 1-12, the day as a value in the range 1-31, the hour as a value in the range 0-23, the minute as a value in the

Personal Use Copy - Not for Reproduction

range 0-59, and the second as a value in the range 0-59. The date must also be a valid date according to common usage: the day range being restricted for certain months, month 2 having 29 days in leap years, etc.

7–49. Operating systems must account for local time, for daylight savings time when and where appropriate, and for leap seconds.

7–50. RTAS must account for leap years.

7.3.3.2 Get-time-of-day

Requirements:

7–51. RTAS must implement a *get-time-of-day* call using the argument call buffer defined by Table 19 on page 107.

Table 19. *get-time-of-day* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *get-time-of-day*
	Number Inputs	0
	Number Outputs	8
Out	Status	0: Success -1: Hardware Error -2: Clock Busy, Try again later
	Year	Year
	Month	1-12
	Day	1-31
	Hour	0-23
	Minute	0-59
	Second	0-59
	Nanoseconds	0-999999999

7–52. RTAS must read the current time and set the output values to the best resolution provided by the platform.

Personal Use Copy - Not for Reproduction

7.3.3.3 Set-time-of-day

Requirements:

7–53. RTAS must implement a *set-time-of-day* call using the argument call buffer defined by Table 20 on page 108:

Table 20. *set-time-of-day* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *set-time-of-day*
	Number Inputs	7
	Number Outputs	1
	Year	Year
	Month	1-12
	Day	1-31
	Hour	0-23
	Minute	0-59
	Second	0-59
	Nanosecond	0-999999999
Out	Status	0: Success -1: Hardware Error -2: Clock Busy, Try again later

7–54. RTAS must set the time of day to the best resolution provided by the platform.

Software Implementation Note: The operating system maintains the clock in UTC. This allows the various operating systems and diagnostics to co-exist with each other and provide uniform handling of time. Refer to Table 2 on page 19 for further details on the time of day clock.

Personal Use Copy - Not for Reproduction

7.3.3.4 Set-time-for-power-on

Some platforms provide the ability to set a time to cause the platform power on. The *set-time-for-power-on* call provides the interface to the operating system for setting this timer.

Requirements:

7–55. RTAS must implement the *set-time-for-power-on* call using the argument call buffer defined by Table 20 on page 108:

Table 21. *set-time-for-power-on* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *set-time-for-power-on*
	Number Inputs	7
	Number Outputs	1
	Year	Year
	Month	1-12
	Day	1-31
	Hour	0-23
	Minute	0-59
	Second	0-59
	Nanosecond	0-999999999
Out	Status	0: Success -1: Hardware Error -2: Clock Busy, Try again later

7–56. If the system is in a powered down state at the time scheduled by *set-time-for-power-on* (within the accuracy of the clock), then power must be reapplied and the system must go through its power on sequence.

7.3.4 Error and Event Reporting

The error and event reporting RTAS calls are designed to provide an abstract interface into hardware registers in the system that may contain correctable or

Personal Use Copy - Not for Reproduction

non-correctable errors and to provide an abstract interface to certain platform events that may be of interest to the operating system. Such errors and events may be detected either by a periodic scan or by an exception trap.

These functions are not intended to replace the normal error handling in the operating system. Rather, they enhance the operating system's abilities by providing an abstract interface to check for, report, and recover from errors or events on the platform that are not necessarily known to the operating system.

The operating system uses the error and event RTAS calls in two distinct ways:

1. Periodically, the operating system calls *event-scan* to have the system firmware check for any errors or events that have occurred.
2. Whenever the operating system receives an interrupt or exception that it can not fully process, it must call *check-exception.*

The first case covers all errors and events that do not signal their occurrence with an interrupt or exception. The second case covers those errors and events that do signal with an interrupt or exception. It is platform dependent whether any specific error or event causes an interrupt on that platform.

Requirements:

7–57. RTAS must return the event generated by a particular interrupt or event source by either *check-exception* or *event-scan*, but not both.

7.3.4.1 Event-scan

Requirements:

7–58. RTAS must implement an *event-scan* call using the argument call buffer defined by Table 22 on page 111.

Personal Use Copy - Not for Reproduction

Table 22. *event-scan* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *event-scan*
	Number Inputs	4
	Number Outputs	1
	Event Mask	Mask of event classes to process
	Critical	Indicates whether this call must complete quickly
	Buffer	Real address of error log
	Length	Length of error log buffer
Out	Status	1: No Errors Found 0: New Error Log returned -1: Hardware Error

7–59. The *event-scan* call must fill in the error log with a single error log formatted as specified in Section 10.3.2, “RTAS Error/Event Return Format,” on page 168. If necessary, the data placed into the error log must be truncated to *length* bytes.

7–60. RTAS must only check for errors or events that are within the classes defined by the *Event mask*. *Event mask* is a bit mask of error and event classes. Refer to Table 50 on page 159 for the definition of the bit positions.

7–61. If *Critical* is non-zero, then RTAS must perform only those operations that are required for continued operation. No extended error information will be returned.

7–62. The *event-scan* call must return the first found error or event and clear that error or event so it is only reported once.

7–63. The operating system must continue to call *event-scan* while a status of “New Error Log returned” is returned.

7–64. The *event-scan* call must be made at least *rtas-event-scan-rate* times per minute for each error and event class and must have the *Critical* parameter equal to 0 for this periodic call.

Personal Use Copy - Not for Reproduction

Software Implementation Note: In a multiprocessor system, each processor should call *event-scan* periodically, not always the same one. The *event-scan* function needs to be called a total of *rtas-event-scan-rate* times a minute.

Software Implementation Note: The maximum size of the error log is specified in the Open Firmware Device Tree as the *rtas-error-log-max* property of the RTAS node.

Software Implementation Note: This call does not log the error in NVRAM. It returns the error log to the operating system. It is the responsibility of the operating system to take appropriate action.

Software Implementation Note: For best system performance, the requested *rtas-event-scan* rate should be as low as possible, and as a goal should not exceed 120 scans per minute. Maximum system performance is obtained when no scans are required.

7.3.4.2 Check-exception

Requirements:

7–65. RTAS must implement a *check-exception* call using the argument call buffer defined by Table 23 on page 113.

Personal Use Copy - Not for Reproduction

Table 23. *check-exception* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *check-exception*
	Number Inputs	6
	Number Outputs	1
	Vector Offset	The vector offset for the exception. See *The PowerPC Architecture* [1], Section 5.4, “Interrupt Processing”.
	Additional Information	Information which RTAS may need to determine the cause of the exception, but which may be unavailable to it in hardware registers. See Table 24 on page 113 for details.
	Event Mask	Mask of event classes to process
	Critical	Indicates whether this call must complete quickly
	Buffer	Real address of error log
	Length	Length of error log
Out	Status	1: No Errors Found 0: New Error Log returned -1: Hardware Error

7–66. The operating system must provide the value specified in Table 24 on page 113 in the “Additional Information” parameter in the call to *check-exception*.

Table 24. Additional Information Provided to *check-exception* call

Source of Interrupt	Value of “Additional Information” Variable
External Interrupt	Interrupt number
Machine Check Exception	Value of register SRR1 at entry to machine check handler
System Reset Exception	Value of register SRR1 at entry to system reset handler
Other Exception	Value of register SRR1 at entry to exception handler

Personal Use Copy - Not for Reproduction

7–67. The *check-exception* call must fill in the error log with a single error log formatted as specified in Section 10.3.2, “RTAS Error/Event Return Format,” on page 168. The data in the error log must be truncated to *length* bytes.

7–68. If *Critical* is non-zero, then RTAS must perform only those operations that are required for continued operation. No extended error information will be returned.

7–69. The *check-exception* call must return the first found error or event and clear that error or event so it is only reported once.

7–70. RTAS must only check for errors or events that are within the classes defined by the *Event mask*. *Event mask* is a bit mask of error and event classes. Refer to Table 50 on page 159 for the definition of the bit positions.

Software Implementation Note: All operating system reserved exception handlers should call *check-exception* to process any errors that are unknown to the operating system.

Software Implementation Note: The interrupt number for external device interrupts is provided in the Open Firmware device tree as specified in the *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10].

7.3.5 PCI Configuration Space

Device drivers and system software need access to PCI configuration space. Chapter 3, “System Address Map,” on page 23 defines system address spaces for PCI memory and PCI I/O spaces. It does not define an address space for PCI configuration. Different PCI bridges may implement the mechanisms for accessing PCI configuration space in different ways. The RTAS calls in this section provide an abstract way of reading and writing PCI configuration spaces.

The PCI Local Bus Specification, Revision 2.1 defines two addressing modes for the PCI configuration space. The PCI access functions both take a *config_addr* input parameter which is similar to a Type 1 address. This address is a 24-bit quantity composed of bus, device, function, and register numbers, all concatenated (high to low). Bus is an 8-bit quantity, device a 5-bit quantity, function a 3-bit quantity, and register an 8-bit quantity. This allows the configuration of up to 256 buses (including sub-bridges), 32 devices per bus, 8 functions per device, and 256 bytes of register space per function. Thus, up to 8192

Personal Use Copy - Not for Reproduction

devices on various buses may be addressed. The *config_addr* for a device is obtained from the Open Firmware Device Tree.

The PCI Local Bus Specification, Revision 2.1 requires that unimplemented or reserved register space read as 0's, and that reads of the Vendor ID register of devices or functions which aren't present should be unambiguously reported (reading 0xFFFF is sufficient). Writes to unimplemented or reserved register space are specified as no-ops. Writes to devices or functions which aren't present are undefined. These operations are undefined if a bus is specified which doesn't exist.

Requirements:

7–71. For the RTAS PCI configuration space functions, the parameter *config_addr* must be a configuration address as specified by the *PCI Bus Binding to IEEE 1275 Standard for Boot (Initialization, Configuration) Firmware* [12].

7–72. All RTAS PCI Read/Write functions must follow the PCI Local Bus Specification, Revision 2.1 or later.

7.3.5.1 Read-pci-config

Requirements:

7–73. RTAS must implement a *read-pci-config* call using the argument call buffer defined by Table 25 on page 115.

Table 25. *read-pci-config* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *read-pci-config*
	Number Inputs	2
	Number Outputs	2
	Config_addr	Configuration Space Address
	Size	Size of Configuration Cycle in bytes, value can be 1, 2, or 4

Personal Use Copy - Not for Reproduction

Table 25. *read-pci-config* Argument Call Buffer ***(Continued)***

Parameter Type	Name	Values
Out	Status	0: Success -1: Hardware Error
	Value	Value Read from *config_addr*

7–74. The *read-pci-config* call must return the value from the configuration register which is located at *config_addr* in PCI configuration space.

7–75. The *read-pci-config* call must perform a 1-byte, 2-byte, or 4-byte configuration space read depending on the value of the *size* input argument.

7–76. The *config_addr* must be aligned to a 2-byte boundary if *size* is 2 and to a 4-byte boundary if *size* is 4.

7–77. The *read-pci-config* call of devices or functions which are not present must return *Success* with all ones as the output *value*.

7.3.5.2 Write-pci-config

Requirements:

7–78. RTAS must implement a *write-pci-config* call using the argument call buffer defined by Table 26 on page 116.

Table 26. *write-pci-config* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *write-pci-config*
	Number Inputs	3
	Number Outputs	1
	Config_addr	Configuration Space Address
	Size	Size of Configuration Cycle in bytes, can be 1, 2, or 4
	Value	Value to be written to *config_addr*

Personal Use Copy - Not for Reproduction

Table 26. *write-pci-config* Argument Call Buffer ***(Continued)***

Parameter Type	Name	Values
Out	Status	0: Success -1: Hardware Error

7–79. The *write-pci-config* call must store the value from the configuration register which is located at *config_addr* in PCI configuration space.

7–80. The *write-pci-config* call must perform a 1-byte, 2-byte, or 4-byte configuration space write depending on the value of the *size* input argument.

7–81. The *config_addr* must be aligned to a 2-byte boundary if *size* is 2 and to a 4-byte boundary if *size* is 4.

7–82. The *write-pci-config* call of devices or functions which aren't present must be ignored. A status of *Success* must be returned.

7.3.6 Operator Interfaces and Platform Control

The RTAS operator interface and platform control functions provide an OS with the ability to perform platform services in a portable manner. The RTAS operator interface provides the ability for an OS to notify the user about OS events during boot, to notify the user of abnormal events, and to obtain information from the platform. The platform control functions give the OS the ability to obtain platform-specific information and to control platform features.

These calls are all "best effort" calls. RTAS should make its best effort to implement the intent of the call. If the Platform Hardware does not implement some optional feature, it is permitted for RTAS to either return an error, or to virtualize the service in some way and return "Operation Succeeded."

Software Implementation Note: For example, a keyswitch could be virtualized by storing a keyswitch value in NVRAM and by providing a user interface to modify this value. The RTAS call *get-sensor-state* on the keyswitch would just return the value stored in NVRAM.

Software Implementation Note: If these services are only called prior to the use of any of the underlying devices by the OS, for example, during boot time, or only after the OS has finished using the devices, for example, during a crash, then the OS can avoid mutual exclusion and sharing concerns. Otherwise, synchronization per Section 7.2.3, "RTAS Critical Regions," on page 95, must be performed.

Personal Use Copy - Not for Reproduction

7.3.6.1 Display-character

The RTAS call *display-character* can be used by an operating system to display informative messages during boot, or to display error messages when an error has occurred and the operating system can not depend on its display drivers. This call is intended to display the characters on an LCD panel, graphics console, or attached tty. The precise implementation is platform vendor specific.

Requirements:

7–83. RTAS must implement a *display-character* call using the argument call buffer defined by Table 27 on page 118 to place a character on the output device.

7–84. The operating system must serialize all calls to *display-character* with any other use of the *rtas-display-device.*

Table 27. *display-character* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *display-character*
	Number Inputs	1
	Number Outputs	1
	Value	Character to be displayed
Out	Status	0: Success -1: Hardware error -2: Device busy, try again later

7–85. If a physical output device is used for the output of the RTAS *display-character* call, then it must have at least one line and 4 characters.

7–86. Certain ASCII control characters must have their normal meanings with respect to position on output devices which are capable of cursor positioning. In particular, ^M (0x0D) must position the cursor at column 0 in the current line, and ^J (0x0A) must move the cursor to the next line, scrolling old data off the top of the screen.

7–87. RTAS must not output characters to the *rtas-display-device* except for explicit calls to the *display-character* function.

Personal Use Copy - Not for Reproduction

Software Implementation Note: RTAS should try to produce output to the user. This could be to the system console, to an attached terminal, or to some other device. It could be implemented using a diagnostic processor or network. RTAS could also implement this call by storing the messages in a buffer in NVRAM so the user could determine the reason for a crash upon re-boot.

Software Implementation Note: This call will modify the registers associated with the *rtas-display-device*. The operating system may also access this device but must be aware that calls to *display-character* will change the state of the device.

7.3.6.2 Set-indicator

The RTAS *set-indicator* function provides the operating system with an abstraction for controlling various lights, indicators, and other resources on a hardware platform. If multiple indicators of a given type are provided by the platform, this function permits addressing them individually.

Requirements:

7–88. RTAS must implement a *set-indicator* call which sets the value of the indicator of type *Indicator* and index *Indicator-index* using the argument call buffer defined by Table 28 on page 119 and indicator types defined by Table 29 on page 120.

Table 28. *set-indicator* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *set-indicator*
	Number Inputs	3
	Number Outputs	1
	Indicator	Token defining the indicator
	Indicator-index	Index of specific indicator (0, 1, ...)
	State	Desired new state
Out	Status	0: Success -1: Hardware Error -2: Hardware busy, try again later -3: No such indicator implemented

Personal Use Copy - Not for Reproduction

Table 29. Defined Indicators

Indicator Name	Token Value	Defined Values	Default Value	Examples/Comments
Tone Frequency	1	Unsigned Integer (units are Hz)	1000	Generate an audible tone using the tone generator hardware (see Table 2 on page 19). RTAS will select the closest implemented audible frequency to the requested value.
Tone Volume	2	0-100 (units are percent), 0 = OFF	0	Set the percentage of full volume of the tone generator output, scaled approximately logarithmically.
System Power State	3	Off(0), On(1), Standby(2), Suspend(3), Hibernate(4)	On	An external indicator which may be provided to indicate the system power state. Colors and/or blinking may be used to indicate power states other than Off and On.
Warning Light	4	Off(0) Green(1) Amber(2) Red (3)	Off	Example Operating system policy: Green indicates no problems, amber indicates a near-abnormal voltage, temperature, or other environmental condition, red indicates an out-of-range condition which must be acted upon. Software should display the condition to the user before displaying amber or red.
Disk Activity Light	5	On (1), Off (0)	Off	Software may turn this light on when one or more disk transactions are in progress on any attached drive.
Hexadecimal Display Unit (LED)	6	0x0-0xFFFF: 4 Hex Digits to display 0xFFFFFFFF: Clear Display	Clear	Can be used to indicate boot progress, or fatal error indications when no other I/O device can be trusted. Values in the range 0xE000…0xEFFF are reserved for firmware and POST usage, and must be documented by the platform vendor if used. Other values are OS-specific, and must documented in OS documentation if used.
Battery Warning Time	7	Minutes Zero minutes = disabled	Zero	Sets the Battery Warning threshold. A Battery Warning event will be sent when approximately this many minutes of battery lifetime remains. The timing is more accurate with smaller parameter values. See Table 62 on page 199 and Section 11.2.3.5, “Battery-Related RTAS Calls,” on page 208 for further detail.
Condition Cycle Request	8	Disable (0) Enable (1)	Not in Progress	Setting this indicator requests the power circuitry to start a Battery Condition cycle. See Section 11.2.3.5, “Battery-Related RTAS Calls,” on page 208 for further detail.
Vendor Specific	9000-9999			Indicator values reserved for platform vendor use.

7.3.6.3 Get-sensor-state

The RTAS call *get-sensor-state* can be used by a shrink wrapped operating system to read the current state of various sensors on any Platform. If multiple

Personal Use Copy - Not for Reproduction

sensors of a given type are provided by the platform, this function permits addressing them individually.

Requirements:

7–89. RTAS must implement a *get-sensor-state* call which reads the value of the sensor of type *Sensor* which has index *Sensor-index* using the argument call buffer defined by Table 30 on page 121 and the sensor types defined by Table 31 on page 121.

Table 30. *get-sensor-state* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for get-sensor-state
	Number Inputs	2
	Number Outputs	2
	Sensor	Token defining the sensor type
	Sensor-index	Index of specific sensor (0, 1, ...)
Out	Status	0: Success -1: Hardware Error -2: Hardware Busy, Try again later -3: No such sensor implemented
	State	Current sensor's state

Table 31. Defined Sensors

Sensor Name	Token Value	Defined Values	Description
Key Switch	1	Off (0), Normal (1), Secure (2), Maintenance (3)	Key switch modes are tied to OS security policy. Suggested meanings: Maintenance mode permits booting from floppy or other external, non-secure media. Normal mode permits boot from any attached device. Secure mode permits no manual choice of boot device, and may restrict available functionality which can be accessed from the main operator station. Off completely disables the system.

Personal Use Copy - Not for Reproduction

Table 31. Defined Sensors ***(Continued)***

Sensor Name	Token Value	Defined Values	Description
Enclosure Switch	2	Open (1), Closed (0)	This is a switch which indicates that the computer enclosure is open. For safety reasons, opening the enclosure should generally cause power to be removed.
Thermal Sensor	3	Temperature (in Degrees Celsius)	If implemented, returns the internal temperature of the most temperature-sensitive portion of platform.
Lid Status	4	Open(1), Closed(2)	If implemented, permits a power managed OS to determine that a computer is not going to be used soon, and can be put into a reduced-power state
Power Source	5	AC (0), Battery (1) AC & Battery(2)	Indicates source of primary power.
Battery Voltage	6	Voltage (in units of Volts)	Current battery output voltage.
Battery Capacity Remaining	7	High(3) Mid(2) Low(1) Very Low(0)	Used mainly for batteries which do not provide information about their current state. The actual value may depend on the implementation, including battery chemistry.
Battery Capacity Percentage	8	0-1000 (in units of 0.1%)	Used mainly for batteries which do report their current state of charge.
Environmental and Power State (EPOW) Sensor	9	EPOW_Reset(0) Warn_Cooling(1) Warn_Power(2) System_Shutdown(3) System_Halt(4) EPOW_Main_Enclosure(5) EPOW_Power_Off(7)	RTAS assessment of the environment and power state of the platform. Refer to Section 11.2.1.2, “Definition of RTAS Abstracted Power Management Event Types,” on page 199 for further information.
Battery Condition Cycle State	10	None (0) In Progress (1) Requested (2)	Current state of the Battery Condition Cycle hardware. See Section 11.2.3.5, “Battery-Related RTAS Calls,” on page 208 for further information.
Battery Charging State	11	Charging (0) Discharging (1) No Current Flow (2)	Indicates whether the battery is currently being charged, being used as a power source (discharging), or neither.
Vendor Specific	9000-9999		Reserved for use by platform vendors.

Personal Use Copy - Not for Reproduction

7.3.7 Power Management

The ability to manage power consumption is a requirement for several conforming systems, such as laptop computers and "Green" or Energy Star computers. RTAS provides the mechanisms necessary for an operating system to manage the power consumption of the hardware. All policy decisions are up to the operating system.

Power management may be applied to an arbitrary power management domain. Power domains for a given platform are defined in the Open Firmware Device Tree. See Chapter 11, "Power Management," on page 185 and *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10].

Power management will involve changing clock speeds, lowering or removing power from devices, and setting certain control bits inside of processors. The RTAS power management calls only apply to power management services *outside* the processors. It is the responsibility of the operating system to manage individual CPUs.

Although a fundamental principle of power management is that system software as opposed to platform hardware/firmware provides all power management policy, there are times during the boot life of an operating system when system software is unable to provide this control (for example, prior to the loading of the power management system software). During these times the platform must assume responsibility for making certain policy decisions. The *assume-power-management* and *relinquish-power-management* RTAS calls provide a mechanism for handing responsibility back and forth between system software and the platform. Refer to the Chapter 11, "Power Management," on page 185 for a discussion of how these calls are intended to be utilized in a power managed system.

Requirements:

7–90. For the Power Management option: The operating system must control the power management features of the processors and any attached power-manageable devices for which it has device drivers.

Software Implementation Note: The RTAS Power Management calls provide a mechanism, not a policy. It is the responsibility of the operating system to implement the Power Management policy. It is also the responsibility of the operating system to maintain any necessary state information.

Personal Use Copy - Not for Reproduction

7.3.7.1 Set-power-level

Requirements:

7–91. For the Power Management option: RTAS must implement a *set-power-level* call which changes the power level setting of a power domain. This call must be implemented using the argument call buffer defined by Table 32 on page 124.

Table 32. *set-power-level* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *set-power-level*
	Number Inputs	2
	Number Outputs	2
	Power_domain	Token defining the power domain
	Level	Token for the desired level for this domain
Out	Status	0: Success -1: Hardware Error -2: Busy, Try again later
	Actual_level	The power level actually set

7–92. For the Power Management option: *Power_domain* must be a power domain identified in the Open Firmware Device Tree.

7–93. For the Power Management option: *Level* must be a power level as specified by Section 11.2.1.1, “Definition of Domain Power Levels,” on page 198.

7–94. For the Power Management option: The *set-power-level* call must set the level of the specified domain to the power level specified by *level* or to the next higher implemented level.

7–95. For the Power Management option: The *set-power-level* call must return the power level actually set in the *Actual_level* output parameter.

Software Implementation Note: The *set-power-level*(0,0) call, if implemented, removes power from the root domain, turning off power to all domains. The external events which can turn power back on are

Personal Use Copy - Not for Reproduction

platform specific. The RTAS primitive *power-off* also removes power from the system, but permits specifying the events which can turn power back on.

Software Implementation Note: The implemented values for the Level parameter for each power domain are defined in the Open Firmware device tree.

7.3.7.2 Get-power-level

Requirements:

7–96. For the Power Management option: RTAS must implement a *get-power-level* call which returns the current setting of a power domain. This call must be implemented using the argument call buffer defined by Table 33 on page 125.

Table 33. *get-power-level* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *get-power-level*
	Number Inputs	1
	Number Outputs	2
	Power_domain	Token defining the power domain
Out	Status	0: Success -1: Hardware Error -2: Busy, try again later -3: Can't determine current level
	Level	The current power level for this domain

7–97. For the Power Management option: *Power_domain* must be a power domain identified in the Open Firmware Device Tree.

Software Implementation Note: The *get-power-level* call only returns information about power levels whose state is readable in hardware. It does not need to remember the last set state and return that value.

Personal Use Copy - Not for Reproduction

7.3.7.3 Assume-power-management

This call transfers control of the power management policy from the platform to system software.

Requirements:

7–98. For the Power Management option: RTAS must implement an *assume-power-management* call which transfers control of the power management policy from the platform to system software, using the argument call buffer specified in Table 34 on page 126.

Table 34. *assume-power-management* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *assume-power-management*
	Number Inputs	0
	Number Outputs	1
Out	Status	0: Success -1: Hardware Error

Hardware Implementation Note: Even after the successful completion of the *assume-power-management* call, platform hardware/firmware may take actions (including turning off platform power) without the direction of system software to provide safety or prevent the destruction of platform hardware. In these cases, the platform should make an effort to inform system software of its imminent intervention.

7.3.7.4 Relinquish-power-management

This call transfers control of the power management policy from system software to the platform.

Requirements:

7–99. For the Power Management option: RTAS must implement a *relinquish-power-management* call which transfers control of the power management policy from system software to the platform, using the argument call buffer specified in Table 34 on page 126.

Personal Use Copy - Not for Reproduction

7–100. For the Power Management option: If the power management policy is under control of the operating system, the *relinquish-power-management* call must be performed prior to the completion of the transition into the Suspend, Hibernate, or Off system power states.

Table 35. *relinquish-power-management* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *relinquish-power-management*
	Number Inputs	0
	Number Outputs	1
Out	Status	0: Success -1: Hardware Error

7.3.7.5 Power-off

This primitive turns power off on a system which is equipped to perform a software-controlled power off function. It may be implemented on platforms which do not implement any other Power Management functions.

Requirements:

7–101. If software controlled power-off hardware is present: The *power-off* function must turn off power to the platform, using the argument call buffer described in Table 36 on page 128.

7–102. If software controlled power-off hardware is present: *Power_on_mask*, which is passed in two parts to permit a possible 64 events even on 32-bit implementations, must be a bit mask of power management events, refer to requirement 11–3. If a bit in the *resume_mask* is set to 1, then the hardware should enable the corresponding hardware power-on mechanism.

Personal Use Copy - Not for Reproduction

Table 36. *power-off* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *power-off*
	Number Inputs	2
	Number Outputs	1
	Power_on_mask_hi	Mask of events that can cause a power on event - event mask values [0:31] (right-justified if the cell size is 64 bits)
	Power_on_mask_lo	Mask of events that can cause a power on event - event mask values [32:63] (right-justified if the cell size is 64 bits)
Out	Status	On successful operation, does not return -1: Hardware error

7.3.8 Suspend and Hibernate

Suspension is the process of placing the memory subsystem in a low power data retentive state and placing the processor in a powered off state. When power is restored to the processor, the firmware recognizes that the system was in a suspend state. The firmware re-initializes all devices and transfers control back to the operating system.

Hibernation is the process of saving a system image on disk and powering down the system. When power is reapplied, the system firmware reboots the system. Operating system boot or initialization code recognizes that the system was hibernating, restores the saved image, and returns control to the point of hibernation. Firmware is not involved in wakeup from hibernation, other than its usual role in rebooting the system.

For more details on power management refer to Chapter 11, “Power Management,” on page 185.

Software Implementation Note: Hibernation can be performed without the use of the RTAS *hibernate* primitive, but the primitive can be valuable for saving the last part of the OS if the OS cannot save itself without corrupting state information that is to be part of the saved image.

Personal Use Copy - Not for Reproduction

7.3.8.1 Suspend

The RTAS function *suspend* preserves the System Memory image exclusive of that devoted to firmware use, records suspend restoration parameters in the system's configuration memory, and sets the system power state to suspend in a system dependent manner (this usually means powering down everything except the memory sub-system and its refresh, and some mechanism to trigger the resume). When the system is subsequently powered up, the firmware reads the suspend restoration parameters, resets its devices, and returns to the caller of suspend.

Requirements:

7–103. The RTAS *suspend* function must implement the power-saving Suspend state using the argument call buffer defined in Table 37 on page 129.

Table 37. *suspend* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *suspend*
	Number Inputs	3
	Number Outputs	2
	Resume_mask_hi	Mask of events that can cause a resume event - event mask values [0:31] (right-justified if the cell size is 64 bits)
	Resume_mask_lo	Mask of events that can cause a resume event - event mask values [32:63] (right-justified if the cell size is 64 bits)
	Processor_number	Token defining the processor to be awakened on a resume event. This is the *reg* value for the processor, as provided in the Open Firmware device tree
Out	Status	0: Success -1: Hardware error
	Resume_event	Event that cause the resume

7–104. The RTAS *suspend* call must be made only from the processor identified by *processor_number.*

Personal Use Copy - Not for Reproduction

7–105. *Resume_mask*, which is passed in two parts to permit a possible 64 events even on 32-bit implementations, must be a bit mask of power management events, refer to requirement 11–3. If a bit in the *resume_mask* is set to 1, then the hardware should enable the corresponding hardware wakeup mechanism.

7–106. *Resume_event* must be a power management event that caused the resume.

7–107. In an SMP system, the operating system must have invoked the RTAS *stop-self* function on all other processors prior to invoking *suspend.*

7–108. All elements of the I/O sub-system must be in a quiescent state at the time of the call: they must not be transferring data to or from memory, nor be able to cause an interrupt to any processor.

7.3.8.1.1 Suspend Restoration (Resume)

Upon receiving a wakeup event that was enabled using *resume_mask*, the system will resume execution on the processor identified by *processor_number.*

Requirements:

7–109. Upon return from the RTAS *suspend* call, the state of the memory locations, exclusive of the first 256 bytes of real memory and the RTAS private data area, must be in the same state they were in at the time of the call to *suspend***.**

7–110. Upon return from *suspend*, execution must be on the processor indicated by *processor_number.*

7–111. Upon return from *suspend*, all processors except the processor identified by *processor_number* must be in the *stopped* state.

7–112. Upon return from *suspend*, the firmware must restore the registers and devices which are reserved for firmware to the same state as before the *suspend*.

7–113. Upon return from *suspend*, the firmware must place all elements of the I/O system into a safe state.

7–114. The return from *suspend* must restore the registers listed in requirement 7–10 to the same state that existed prior to the *suspend* call.

Software Implementation Note: RTAS need not reprobe the I/O subsystem on resume, and in fact should avoid doing so in order to make resume

Personal Use Copy - Not for Reproduction

quicker. Firmware is NOT required to restore the I/O subsystem to the previous state; this is the responsibility of the operating system.

Software Implementation Note: This call may violate requirements stated in requirement 7–10 in that it actually saves and restores the values.

Software Implementation Note: The first 256 bytes of real memory are available for firmware use during the suspend/resume operations.

Software Implementation Note: The OS timebase must be reestablished using the Real-Time Clock upon resuming, since an unknown amount of time will have passed.

Software Implementation Note: RTAS is not required to permit configuration changes of devices which are reserved for firmware use while in a Suspend state. If such devices are removed or modified while suspended, the results are unspecified.

Software Implementation Note: Changing the hardware configuration while in a Suspend state may cause a Configuration Change event as defined in Table 62 on page 199 if the platform provides this capability.

Software Implementation Note: Since the Open Firmware device tree is not updated in a Suspend/Resume sequence, it is OS dependent whether an OS supports platform hardware configuration changes while in the Suspend state.

7.3.8.1.2 Hibernate

The RTAS *hibernate* call provides the capability to save the current system image for later restoration. One way that this could be implemented (using Open Firmware, with minimal support in RTAS) is to save the address of the memory list, to reset the I/O subsystem (avoiding resetting the memory subsystem, as this may disrupt memory contents), and then to re-enter Open Firmware. Open Firmware can then recognize the block list, and instead of performing a normal boot, can use its device drivers to write the data to the specified device. Once the data is recorded, Open Firmware causes the system to enter a low power hibernate state. Other implementations which meet the Requirements are also possible.

Requirements:

7–115. The RTAS *hibernate* call must be implemented using the argument call buffer described by Table 38 on page 132.

Personal Use Copy - Not for Reproduction

Table 38. *hibernate* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *hibernate*
	Number Inputs	3
	Number Outputs	1
	Block_list	A real pointer to a block list
	Wakeup_mask_hi	Mask of events that can cause a wakeup event - event mask values [0:31] (right-justified if the cell size is 64 bits)
	Wakeup_mask_lo	Mask of events that can cause a wakeup event - event mask values [32:63] (right-justified if the cell size is 64 bits)
Out	Status	On successful operation, does not return -1: Hardware error

7–116. The *wakeup_mask*, which is passed in two parts to permit a possible 64 events even on 32-bit implementations, must be a bit mask of power management events, refer to requirement 11–3. If a bit in the wakeup_mask is set to 1, then the hardware should enable the corresponding hardware wakeup mechanism.

7–117. The area of memory described by the *hibernate block_list* must be written to the device(s) as indicated.

7–118. The hibernate *block_list* must start with a list length in bytes and then must have quadruples of real address, length of memory block, device id, and block number as shown in Table 39 on page 132. Additional requirements on the block list are:

a. The block list must be a sequence of cells as defined in requirements 7–34 and 7–35.

b. Block numbers are in 512 byte units.

Table 39. Format of Block List

Length of list in bytes
Address of memory area 1

Personal Use Copy - Not for Reproduction

Table 39. Format of Block List ***(Continued)***

Length of memory area 1
Device for area 1
Block Number for area 1
Address of memory area 2
Length of memory area 2
Device for area 2
Block Number for area 2
...
Address of memory area n
Length of memory area n
Device for area n
Block number for area n

7–119. The *hibernate block_list* must be contained in system memory below 4 GB.

7–120. The *Device* fields of the *hibernate block_list* must be real pointers to System Memory below 4 GB that point to Open Firmware Path Names.

7–121. A device specified in the *hibernate block_list* must correspond to an device in the Open Firmware Device Tree that has *write* and *read* methods, and has a device type of *block*.

7–122. The memory areas defined by the *hibernate block_list* must only include System Memory outside that reserved for firmware (both the RTAS data area and Open Firmware's memory defined by *real-base* and *real-size*).

7–123. The memory areas defined by the *hibernate block_list* must only include System Memory below 4 GB.

7–124. Execution of the *hibernate* call must cause the System Memory areas described in the *block_list* to be saved on disk on the device(s) specified starting at the specified block numbers.

Personal Use Copy - Not for Reproduction

7–125. After system memory is saved, *hibernate* must place the system into its lowest power state.

7–126. In an SMP system, all other processors must be in the *stopped* state before invoking *hibernate*. This is the responsibility of the operating system.

7–127. Prior to making the *hibernate* call, the operating system must put all I/O devices into a quiescent state: they must not be transferring data to or from memory, nor be able to cause an interrupt to any processor.

7–128. Open Firmware must deterministically initialize a platform. The Open Firmware device tree and device addresses assigned on a given platform must be the same on successive reboots and hibernate wakeups if the platform hardware configuration has not changed.

Software Implementation Note: The firmware is free to move and otherwise modify System Memory during this call.

7.3.8.2 Hibernate Restoration (Wakeup)

From the standpoint of the platform firmware, wakeup is no different from any other system initialization process. It is the responsibility of the operating system to:

1. Recognize that this is a wakeup
2. Find the saved hibernation image on disk
3. Restore the hibernation image to memory
4. Transfer control back to the proper point for continued execution.

Software Implementation Note: Wakeup from hibernate is entirely an operating system implemented function, initiated by booting the system. Upon booting, OS boot or initialization code runs and notices a hibernation image. This early code must be careful to instantiate RTAS in an area of memory which will not be destroyed when the OS places the hibernation image into memory.

Software Implementation Note: In a Multiboot situation, it is possible for different operating systems resident on a system to be hibernating at the same time. This is a quick way to switch between operating system environments. In this scenario, the user would indicate to system firmware a different OS to boot/wakeup next, hibernate the system, and do a reboot operation to wakeup that OS image.

Personal Use Copy - Not for Reproduction

Software Implementation Note: The OS timebase must be reestablished using the Real-Time Clock upon wakeup, since an unknown amount of time will have passed.

7.3.9 Reboot

During execution, it may become necessary to shut down processing and re-boot the system in a new mode. For example, a different operating system may need to be loaded, or the same operating system may need to be re-booted with different settings of System Environment Variables.

7.3.9.1 System-reboot

Requirements:

7–129. RTAS must implement a *system-reboot* call which resets all processors and all attached devices. After reset, the system must be booted with the current settings of the System Environment Variables (refer to Section 8.4.3, “System (0x70),” on page 145 for more information).

7–130. The RTAS *system-reboot* call must be implemented using the argument call buffer defined by Table 40 on page 135.

Table 40. *system-reboot* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *system-reboot*
	Number Inputs	0
	Number Outputs	1
Out	Status	On successful operation, does not return -1: Hardware error

Hardware Implementation Note: The platform must be able to perform a system reset and reboot. On a multiprocessor system, this should be a hard reset to the processors.

Personal Use Copy - Not for Reproduction

7.3.10 Caches

The following interfaces allow simple control of internal and external caches, primarily for turning these caches on or off for power management or for management of a multiprocessor system.

When flushing and disabling caches, the operating system must take care to flush caches in order (see Section 4.2.4, “Cache Memory,” on page 65) and take care to synchronize with access by other elements of the system. These operations shall only be called by a processor connected to the cache. The current processor shall be the only active processor connected to the cache.

7.3.10.1 Cache-control

Requirements:

7–131. For the Symmetric Multiprocessor or Power Management option: RTAS must implement a *cache-control* call to place the cache into a new state, using the argument call buffer defined by Table 41 on page 136.

Table 41. *cache-control* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *cache-control*
	Number Inputs	2
	Number Outputs	1
	Cache_id	phandle of the cache (for L1 caches, the phandle of the processor)
	How	New state of the cache
Out	Status	0: Success -1: Hardware error -3: Bad state for this cache

7–132. For the Symmetric Multiprocessor or Power Management option: When entering the new state indicated by the *How* parameter, the actions specified in Table 42 on page 137 must be performed on the caches.

Personal Use Copy - Not for Reproduction

Table 42. *Cache-control* states

State Number	State Name	Action
1	Copy-back	The cache must be enabled in a full powered copy-back mode.
2	Write-Through	If necessary, the cache must be flushed. The cache must then be enabled in a full powered Write-Through mode.
3	Low-power	If necessary, the cache must be flushed and invalidated. The cache must then be placed in a low power but still functioning mode.
4	Disabled	The cache must be flushed, and then put in the lowest powered disabled mode.
5	Flush	The cache contents are flushed, and the cache remains in its current state.
9000-9999	Reserved	Reserved for platform vendor specific actions.

7–133. For the Symmetric Multiprocessor or Power Management option: The RTAS *cache-control* operations performed on the specified cache must appear to be a single atomic operation as seen from the operating system. A return status of success implies “committed,” that the operation is complete, and that any dirty data is safely out of the cache. These operations must not violate the cache coherency of the caches.

7.3.11 SMP Support

In a Symmetric Multiprocessor (SMP) system, the operating system needs the ability to synchronize the clocks on all the processors. To do this, it must have the ability to stop all clocks at the same time. For example, the TBEN signal provided on the PowerPC 604 microprocessor can be used to implement this clock control function.

7.3.11.1 freeze-time-base

Requirements:

7–134. For the Symmetric Multiprocessor option: RTAS must implement a *freeze-time-base* call which *freezes*, or keeps from changing, the time

Personal Use Copy - Not for Reproduction

base register on all processors. This call must be implemented using the argument call buffer defined by Table 43 on page 138.

Table 43. *freeze-time-base* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *freeze-time-base*
	Number Inputs	0
	Number Outputs	1
Out	Status	0: Success -1: Hardware Error

7–135. For the Symmetric Multiprocessor option: The *freeze-time-base* operation must simultaneously affect every processor of an SMP system.

7.3.11.2 thaw-time-base

Requirements:

7–136. For the Symmetric Multiprocessor option: RTAS must implement a *thaw-time-base* call which *thaws*, or permits the change of, the time base register on all processors. This call must be implemented using the argument call buffer defined by Table 44 on page 138.

Table 44. *thaw-time-base* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *thaw-time-base*
	Number Inputs	0
	Number Outputs	1
Out	Status	0: Success -1: Hardware Error

Personal Use Copy - Not for Reproduction

7–137. For the Symmetric Multiprocessor option: The *thaw-time-base* operation must simultaneously affect every processor of an SMP system.

7.3.11.3 stop-self

The stop-self primitive causes a processor to stop processing OS or user code, and to enter a state in which it is only responsive to the *start-cpu* RTAS primitive. This is referred to as the RTAS *stopped* state.

Requirements:

7–138. For the Symmetric Multiprocessor option: RTAS must implement a *stop-self* call which places the calling processor in the RTAS *stopped* state. This call must be implemented using the argument call buffer defined by Table 45 on page 139.

7–139. For the Symmetric Multiprocessor option: RTAS must insure that a processor in the RTAS *stopped* state will not check stop or otherwise fail if a machine check or soft reset exception occurs. Processors in this state will receive the exception, but must perform a null action and remain in the RTAS *stopped* state.

Table 45. *stop-self* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *stop-self*
	Number Inputs	0
	Number Outputs	1
Out	Status	If successful, this call does not return -1: Hardware Error

Software Implementation Note: If this call succeeds, it will not return. The CPU will wait for some other processor to issue a *start-cpu* targeted to this processor.

Personal Use Copy - Not for Reproduction

7.3.11.4 start-cpu

The *start-cpu* primitive is used to cause a processor which is currently in the RTAS *stopped* state to start processing at an indicated location.

Requirements:

7–140. For the Symmetric Multiprocessor option: RTAS must implement a *start-cpu* call which removes the processor specified by the *CPU_id* parameter from the RTAS *stopped* state. This call must be implemented using the argument call buffer defined by Table 46 on page 140.

7–141. For the Symmetric Multiprocessor option: The processor specified by the *CPU_id* parameter must be in the RTAS *stopped* state entered because of a prior call by that processor to the *stop-self* primitive.

7–142. For the Symmetric Multiprocessor option: When a processor exits the RTAS *stopped* state, it must begin execution in real mode, at the real location indicated by the *Start_location* parameter, with register R3 set to the value of parameter *Register_R3_contents*, in the endian mode of the processor executing the *start-cpu* primitive. All other register contents are indeterminate.

Table 46. *start-cpu* Argument Call Buffer

Parameter Type	Name	Values
In	Token	Token for *start-cpu*
	Number Inputs	3
	Number Outputs	1
	Cpu_id	Token identifying the processor to be started, obtained from the *reg* value for the CPU in the Open Firmware device tree
	Start_location	Real address at which the designated CPU will begin execution
	Register_R3_contents	Value which will be loaded into Register R3 before beginning execution at Start_location
Out	Status	0: Success -1: Hardware Error

Personal Use Copy - Not for Reproduction

Non-VolatileMemory 8

This chapter describes the requirements relating to the Common Hardware Reference Platform Non-Volatile Memory. Non-Volatile Memory is the repository for system information that must be persistent across reboots, power management activities, and power cycles.

8.1 System Requirements

Requirements:

8–1. Platforms must implement at least 8 KB of Non-Volatile Memory. This is sufficient for a system with a single operating system installed. Allow 1 KB for each additional operating system installed.

8–2. Non-Volatile Memory must maintain its contents in the absence of system power.

8–3. Firmware must reinitialize NVRAM to a bootable state if NVRAM data corruption is detected.

8–4. Operating systems must reinitialize their own NVRAM partitions if NVRAM data corruption is detected. Operating systems may create free space from the first corrupted partition header to the end of NVRAM and utilize this area to initialize their partitions.

Hardware Implementation Note: Non-volatile memory is normally implemented with battery-powered RAM and is generally called NVRAM. This terminology will be used throughout the remainder of this chapter, although it should be understood that this is not the only possible implementation.

Software Implementation Note: Refer to Section 7.3.2, "NVRAM Access Functions," on page 104 for information on accessing NVRAM.

8.2 Structure

NVRAM is formatted as a set of partitions that adhere to the structure in Table 47 on page 143. Partitions are prefixed with a *header* containing **`signature`**, **`checksum`**, **`length`**, and **`name`** fields. The structure of the **`data`** field is defined by the partition creator/owner (designated by "**`signature`**" and "**`name`**").

Requirements:

8–5. NVRAM partitions must be structured as shown in Table 47 on page 143.

8–6. All NVRAM space must be accounted for by partitions.

8.3 Signatures

The **`signature`** field is used as the first level of partition identification. Table 48 on page 144 lists all the currently defined signature types and their ownership classes. The ownership class determines the permission of a particular system software component to create and/or modify partitions and/or partition contents. All partitions may be read by any system software component, but the ownership class has exclusive write permission. Global ownership gives read/write permission to all system software components. These restrictions are made to minimize the possibility of corruption of NVRAM during update activities.

Hardware and Software Implementation Note: It is recommended that partitions be ordered on the signature field with the lowest value signature partition at the lowest NVRAM address (with the exception of signature = 0x7F, free space). This will minimize the effect of NVRAM data corruption on system operation.

Personal Use Copy - Not for Reproduction

Table 47. NVRAM Structure

Field Name	Size	Description
signature	1 byte	The **signature** field is used to identify the partition type and provide some level of checking for overall NVRAM contamination. Signature assignments are given in Table 48 on page 144.

checksum	1 byte	The **checksum** field is included to provide a check on the validity of the header. The checksum covers the **signature**, **length**, and **name** fields and is calculated (on a byte by byte or equivalent basis) by: add, and add 1 back to the sum if a carry resulted as demonstrated with the following program listing.
		
		`unsigned char sumcheck(bp,nbytes)`
		`unsigned char *bp /* buffer pointer */`
		`unsigned int nbytes; /* number of bytes to sum */`
		`{`
		`unsigned char b_data; /* byte data */`
		`unsigned char i_sum; /* intermediate sum */`
		`unsigned char c_sum; /* current sum */`
		`for (c_sum = 0; nbytes; nbytes--)`
		`{`
		`b_data = *bp++; /* read byte from buffer */`
		`i_sum = c_sum + b_data; /* add to current sum */`
		`if(i_sum < c_sum) /* did a carry out result? */`
		`i_sum += 1; /* if so, add 1 */`
		`c_sum = i_sum; /* copy to current sum */`
		`}`
		`return (c_sum);`
		`}`
		
		This checksum algorithm guarantees 0 to be an impossible calculated value.

length	2 bytes Big-Endian format	The **length** field designates the *total* length of the partition, in 16-byte blocks, beginning with the signature and ending with the last byte of the data area. This allows a maximum partition length of 16 x 2**16 = 1 MB. Software Implementation Note: The **length** fields must always provide valid offsets to the next header since an invalid length effectively causes the loss of access to every partition beyond it.
name	12 bytes	The **name** field is a null-terminated (if less than 12 bytes) string used to identify a particular partition within a signature group. There is no general uniqueness requirement, but each OS partition name must be prefixed with a 3 character Organizationally Unique Identifier (OUI) assigned by the IEEE Registration Authority Committee to the OS vendor, *followed by a comma*. If an OS vendor does not have an assigned OUI, a prefix of "," (comma) will be used to signify "other." Before assigning a new name to a partition, software should scan the existing partitions and ensure that an unwanted name conflict is not created.
data	**Length** minus 16 bytes	The structure of the **data** area is controlled by the creator/owner of the partition.

Personal Use Copy - Not for Reproduction

Table 48. NVRAM Signatures

Signature	Signature Type	Ownership Class	# Required	Description
0x01 - 0x0F	Support Processor	Any support processor	0 to n	Reserved for support processor use.
0x50	Open Firmware	Firmware	1	Open Firmware Configuration Variables.
0x51	Reserved	Firmware	n/a	Reserved
0x52	Hardware	Firmware	1 to n	These partitions are used to store administrative machine data such as serial numbers, ec levels, etc. This is often referred to as Vital Product Data (VPD).
0x53 - 0x5F	Firmware	Firmware	0 to n	General firmware usage.
0x70	System	Global	1	The System partition is used to store environment variables that must be accessed by firmware and operating systems.
0x71	Configuration	Global	0 to n	Configuration partitions are used to store configuration data generated by the system firmware or the operating system that is required at boot time to properly configure the system.
0x72	Error Log	Global	0 to n	This partition is used to store the error logs built as a result of various RTAS calls.
0x73	Multi-boot	Global	0 or 1	This partition will contain an entry for each OS installed on the system which will participate in a multiple boot process.
0x74 - 0x7E	Global	Global	0 to n	General global usage.
0x7F	Free Space	Global	0 to n	This signature is used to mark free space in the NVRAM array. The **name** field of all signature 0x7F partitions must be set to 0x7...77.
0xA0 - 0xAF	OS	Any OS	0 to n	General operating system usage.
0x00, 0x10 - 0x4F, 0x60 - 0x6F, 0x80 - 0x9F, 0xB0 - 0xFF				Reserved

8.4 Architected Partitions

8.4.1 Open Firmware (0x50)

This partition is required for storing Open Firmware Configuration Variables.

Requirements:

8–7. The system NVRAM must include a 0x50 partition with the name *of-config* to store Open Firmware Configuration Variables.

Personal Use Copy - Not for Reproduction

8.4.2 Hardware (0x52)

This partition type is used to store Vital Product Data (VPD).

Requirements:

8–8. VPD must be stored in the 0x52 partition using the format defined in the *PCI Local Bus Specification*, Revision 2.1, section 6.4 [14].

8.4.3 System (0x70)

The System partition is required for storing environment variables that are common to all platforms and operating systems.

Requirements:

8–9. Every system NVRAM must contain a System partition with the partition name = *common.*

8–10. Data in the *common* partition must be stored as null-terminated strings of the form: *name*=<*string*> and be terminated with a string of at least two null characters.

8–11. All *names* used in the *common* partition must be unique.

8–12. Device and file specifications used in the *common* partition must follow IEEE Std 1275 nomenclature conventions.

Software Implementation Note: Open Firmware will look for the *boot-script* variable in the *common* System partition; *boot-script=<string>* is a list of Open Firmware commands to be executed during the boot startup sequence. See *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10] for information on the use of this variable.

8.4.4 Configuration (0x71)

8.4.4.1 ISA Configuration Data

ISA configuration data (integrated devices and plug-in cards) is passed to the OS in the OF device tree. In order to enable OS reconfiguration of ISA device

Personal Use Copy - Not for Reproduction

resources, the ISA configuration database will also be entered into NVRAM. This gives the operating system the ability to observe and modify the configuration which will be used by the firmware on the next system boot.

Requirements:

8–13. System NVRAM must include a 0x71 partition with the name *isa-config* to store configuration data for all ISA devices.

8–14. The *isa-config* data area must use the resource format given in the ISA/EISA/ISA-PnP binding to IEEE 1275, *IEEE Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices* [9].

8–15. During boot, firmware must use the ISA configuration data stored in the *isa-config* NVRAM partition to generate the ISA portion of the OF device tree.

Architecture Note: This NVRAM partition is used to store configuration data for both Plug and Play (PnP) and non-PnP ISA adapters. For non-PnP devices, Vendor IDs will be assigned and documented in *PowerPC Microprocessor Hardware Reference Platform: I/O Device Reference* [20].

8.4.4.2 Power Management Configuration Data

Power Management configuration data provides power state and transition data necessary to power manage add-in devices.

Requirements:

8–16. For the Power Management option: NVRAM must include a 0x71 partition with the name *pm-config* to store power management configuration data.

8–17. For the Power Management option: The *pm-config* data area must use the format given in the *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10].

Personal Use Copy - Not for Reproduction

8.4.5 Error Log (0x72)

NVRAM error logs are optional.

It is recommended that operating systems record RTAS-returned error logs in NVRAM to provide error information in the event of a checkstop/reboot. It is also recommended that an error log be provided for errors discovered during IPL POST.

Requirements:

8–18. If an operating system implements an NVRAM error log partition for RTAS,

- **a.** the partition name must be *rtas-err-log*.
- **b.** the error log format must be as given in Table 54 on page 176.
- **c.** error log entries must be filled by the operating system in a manner that will make the most recent log visible.

8–19. If firmware implements an NVRAM error log partition for POST,

- **a.** the partition name must be *post-err-log*.
- **b.** the error log format must be as given in Table 54 on page 176.
- **c.** error log entries must be filled by the firmware in a manner that will make the most recent log visible. If multiple entries are provided, they must be filled in a manner that will make the first and last error occurrences visible to the OS.

8.4.6 Multi-Boot (0x73)

The multi-boot partition is only required if multi-boot is implemented on a platform. The partition will contain entries for each operating system installed on the platform that will participate in the multi-boot process. A configuration variable, *multi-boot?*, is defined in *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10] which instructs the firmware whether or not to present the multi-boot menu to the user during the boot process. This variable may be accessed by the OS through the *boot-script* variable in the NVRAM partition named *common*.

Personal Use Copy - Not for Reproduction

Requirements:

8–20. For the Multi-Boot option: The multi-boot partition must be named *multi-boot*.

8–21. For the Multi-Boot option: Each participating operating system must be represented in the *multi-boot* partition as a 4-tuple of null-terminated strings of the form *name*=<*string*> as represented in Table 49 on page 148.

8–22. For the Multi-Boot option: The *multi-boot* partition must be terminated with a string of at least two null characters.

8–23. Device and file specifications used in the *multi-boot* partition must follow IEEE 1275, *IEEE Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices* [9] nomenclature conventions.

Table 49. Multi-Boot String Definitions

String Name	String Description
boot-name	Text giving the name of the file(OS) to boot. This will be used by the multi-boot utility to build the user menu.
boot-path	The Open Firmware device path and file specification of the file to boot.
icon-path	The Open Firmware device path and file specification of icon to be used on the user menu.
config-var	These are overrides to the OF configuration variables. The string is a list of <*configuration variable name*>=<*value*> substrings separated by spaces.

8.4.7 Free Space (0x7F)

Requirements:

8–24. All unused NVRAM space must be included in a signature = 0x7F Free Space partition.

8–25. All Free Space partitions must have the **`name`** field set to 0x7...77.

Personal Use Copy - Not for Reproduction

8.5 NVRAM Space Management

A partition should be added or extended by consuming free space. Any signature 0x7F partition whose length is greater than or equal to the space requirements of the new partition may be all or partially consumed by this process. If there is no signature 0x7F partition with sufficient free space, partitions may be moved or deleted to provide the required space. As partitions are created and modified, it is likely that free space will become fragmented; free space consolidation may become necessary.

Requirements:

8–26. A system software component must not move or delete any NVRAM partition unless it is in the ownership class of that partition (see Table 48 on page 144). There are two exceptions to this requirement:

- **a.** Open Firmware may, with the appropriate backup precautions, modify any area of NVRAM in the interest of space management and/or maintaining NVRAM data integrity. Firmware must maintain the required 1 KB of contiguous NVRAM space for each installed operating system following any such modifications.
- **b.** Upon detection of a corrupted partition, the operating system may create free space beginning with the header of the corrupted partition through the end of the NVRAM space, and use this space to reinitialize its partitions.

8–27. The NVRAM partition header checksum must be calculated as shown in Table 47 on page 143.

Software Implementation Note: Operations that manipulate NVRAM partitions should be serialized to prevent conflict with other active processes that require NVRAM access.

Software Implementation Note: Restrictions are placed on which system software components may manipulate which partitions. This is to help ensure system integrity by protecting system-level information that is required by the boot process.

Software Implementation Note: If an operating system finds it necessary to remove a partition owned by another operating system, proper user notification and options should be provided.

Personal Use Copy - Not for Reproduction

Software Implementation Note: A partition should not be extended through internal linkages to other partitions. If this is done, unrecoverable orphan partitions may result. Since any operating system may have any number of partitions defined, a safe way to extend an OS partition is to create an additional partition if free space allows.

Personal Use Copy - Not for Reproduction

I/O Devices 9

This chapter describes requirements for PCI and ISA devices. It adds detail to areas of PCI and ISA architecture that are either unaddressed or optional. It also places some requirements on firmware and operating systems for device support. It provides references to specifications to which devices must comply and gives design notes for devices that run on CHRP systems. This chapter references the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20] frequently. This document is available from IBM and contains the detailed register level architecture for I/O devices.

9.1 PCI Devices

Requirements:

9–1. PCI devices must comply with the *PCI Local Bus Specification,* Revision 2.1 [14].

PCI is rapidly becoming the dominant bus in the computer industry and will increasingly become the foundation for all I/O in CHRP systems as ISA devices are migrated to PCI.

9.1.1 Resource Locking

Requirements:

9–2. PCI devices, excepting bridges, must not depend on the PCI LOCK# signal for correct operation nor require any other PCI device to assert LOCK# for correct operation.

There are some legacy devices on legacy buses which require LOCK#. Additionally, LOCK# is used in some implementations to resolve deadlocks between bridges. These uses of LOCK# are permitted.

9.1.2 PCI Expansion ROMs

Requirements:

9–3. PCI expansion ROMs must have a ROM image with a code type of 1 for Open Firmware as provided in the *PCI Local Bus Specification*, Revision 2.1 [14]. This ROM image must abide by the ROM image format for Open Firmware as documented in the *PCI Bus Binding to IEEE 1275 Standard for Boot (Initialization, Configuration) Firmware* [12]

CHRP systems rely on Open Firmware - not BIOS - to boot. This is why strong requirements for Open Firmware device support are made.

Vital Product Data (VPD) is put in the PCI expansion ROM in accordance with the *PCI Local Bus Specification*, Revision 2.1 [14] when it is provided. Although this is an optional feature, it is strongly recommended that VPD be included in all PCI expansion ROMs.

9.1.3 Assignment of Interrupts to PCI Devices

Requirements:

9–4. All PCI devices must use the Open PIC interrupt controller. They must not use the legacy (8259 derived) interrupt controller which is reserved for ISA. The programming model for the legacy interrupt controller is defined in *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

Personal Use Copy - Not for Reproduction

9–5. PCI devices that do not share Peripheral Memory Space and Peripheral I/O Space of the same PHB must not share the same Open PIC interrupt source.

9–6. When PCI-to-ISA bridges with embedded ISA devices are provided in a PCI part, the part must provide the capability to attach to the legacy interrupt controller defined in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

For further information on the Open PIC interrupt controller refer to Chapter 6, "Interrupt Controller," on page 85.

It is strongly advised that system board designers assign one Open PIC interrupt pin for each interrupt source. Additionally, multi-function PCI devices should have multiple interrupt source pins.

9.1.4 PCI Devices with Required Register Definitions

Requirements:

9–7. PCI devices must implement the programming model (register level definition, interrupts, and so forth) for those devices which are in the minimum system requirements and are specified in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

9–8. The following PCI devices, when used, must adhere to the programming model provided in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20]:

- MESH SCSI Controller
- ADB (Apple Desktop Bus) Controller
- SCC (Serial Communications Controller)
- Bus master IDE Controller
- VGA Compatible Graphics Controller

9–9. If a PCI device is defined in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20] and a specific implementation of that device has additional registers beyond those defined as used by programming, then those additional registers must

Personal Use Copy - Not for Reproduction

be reported in the Open Firmware device tree as reserved (see Section 3.1, “Address Areas,” on page 23).

With regard to requirement 9–7, there may be additional registers defined for the initialization of these PCI devices which the firmware has knowledge of but programming does not. Requirement 9–9 addresses how this situation is handled in the Open Firmware device tree.

For MESH SCSI, ADB, and SCC see the *Macintosh Technology in the Common Hardware Reference Platform* [23] for specific physical, timing, and electrical requirements.

9.1.5 PCI-PCI Bridge Devices

Requirements:

9–10. PCI-to-PCI bridges must be compliant with the *PCI to PCI Bridge Architecture Specification* [16]

9–11. Firmware must initialize PCI-to-PCI bridges to work in CHRP systems. See *PCI Bus Binding to IEEE 1275 Standard for Boot (Initialization, Configuration) Firmware* [12].

9.1.6 Graphics Controller and Monitor Requirements for Clients

The graphics requirements for servers are different from those for portable and personal systems.

Requirements:

9–12. Portable and personal platforms must provide Bi-Endian graphics aperture support as described in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

9–13. Plug-in graphics controllers for portable and personal platforms must provide graphics mode sets in the Open Firmware PCI expansion ROM image in accordance with the *PCI Bus Binding to IEEE 1275 Standard for Boot (Initialization, Configuration) Firmware* [12].

Personal Use Copy - Not for Reproduction

Portable and personal platforms are strongly urged to support some mechanism which allows the platform to electronically sense the display capabilities of monitors.

For graphics controllers that are placed on the system board, the graphics mode sets can be put in system ROM. The mode set software put in the system ROM in this case would be FCode and would be largely or entirely the same as the FCode that would be in the PCI expansion ROM if the same graphics controller was put on a plug-in PCI card.

Refer to Section C.4, "Bi-Modal Devices," on page 276 and the PCI Multimedia Design Guide [29] for information on Bi-Endian graphics apertures.

9.2 ISA Devices

There is a legacy of programming for ISA devices which has little platform performance impact and thus will be preserved for the foreseeable future. The ISA devices of primary interest are listed here and their programming model in given in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

Requirements:

9–14. The following ISA devices, when used, must adhere to the programming model provided in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

- Legacy Interrupt Controller
- Serial Port Controller
- Parallel Port Controller
- Floppy Disk Controller
- DMA Controller
- Audio Controller
- Keyboard and Mouse Controller

9–15. When an ISA device is included in a PCI part, is required by the minimum requirements, and is among those specified in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20], it must completely retain the programming model.

Personal Use Copy - Not for Reproduction

9–16. The system tone and the system audio must be able to be used concurrently.

9–17. ISA devices included in a PCI part must route their interrupt signals to the legacy interrupt controller defined in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

Personal Use Copy - Not for Reproduction

Error and Event Notification

10

10.1 Introduction

RTAS provides a mechanism which helps operating systems avoid the need for platform-dependent code that checks for, or recovers from, errors or exceptional conditions. The mechanism is used to return information about hardware errors which have occurred as well as information about non-error events, such as power-management interrupts or environmental conditions (for example, temperature or voltage out-of-bounds) which may need OS attention. This permits RTAS to pass hardware event information to the operating system in a way which is abstracted from the platform hardware. This mechanism primarily presents itself to the operating system via two RTAS functions, *event-scan* and *check-exception*, which are described further in Section 7.3.4, “Error and Event Reporting,” on page 109.

The *event-scan* function is called periodically to check for the presence or past occurrence of a hardware event, such as a soft failure or voltage condition, which did not cause a program exception or interrupt (for example, an ECC error detected and corrected by background scrubbing activity). The *check-exception* function is called to provide further detail on what platform event has occurred when certain exceptions or interrupts are signalled. The events reported by these two functions are mutually exclusive on any given platform; that is, a platform may choose to notify the OS of a particular event type either through *event-scan* or through an interrupt and *check-exception*, but not both.

Since RTAS is platform-specific, it can examine hardware registers, can often diagnose many kinds of hardware errors down to a root cause, and may even perform some very limited kinds of error recovery on behalf of the operat-

ing system. The reporting format, described in this chapter, permits RTAS to report the type of error which has occurred, what entities in the platform were involved in the error, and whether RTAS has successfully recovered from the error without the need for further operating system involvement. RTAS may not, in many cases, be able to determine all the details of an error, so there are also returned values which indicate this fact. RTAS may optionally provide extended error diagnostic information, as described in Section 10.3.2.2, "Extended Error Log Format Returned by RTAS," on page 174.

A platform-aware OS can handle errors and events with as much sophistication as desired by providing platform-specific code and drivers, and a hardware vendor can provide such code and drivers for specific platforms to make an OS platform-aware. However, the abstractions provided by this architecture enable the handling of most platform errors and events without integrating platform-specific code into each supported OS.

Architecture Note: It is not a goal of the RTAS to diagnose all hardware failures. Most I/O device failures, for example, will be detected and recovered by an associated device driver. RTAS attempts to determine the cause of a problem and report what it finds, to aid the end user (by providing meaningful diagnostic data for messages) and to prevent the loss of error syndrome information. RTAS is never required to correct any problem, but in some cases may attempt to do so. System vendors who want more extensive error diagnosis may create operating system error handlers which contain specific hardware knowledge, or could use RTAS to collect a minimum set of error information which could then be used by diagnostics to further analyze the cause of the error.

10.2 RTAS Error and Event Classes

Table 50 on page 159 describes the predefined classes of error and event notifications that can be presented through the *check-exception* and *event-scan* RTAS functions. More detailed descriptions of these classes are given later in this chapter. Table 50 defines nodes in the Open Firmware device tree which, through an *open-pic-interrupt* property, may list the platform-dependent interrupts related to each class. From this information, operating systems know which interrupts may be handled by calling *check-exception*. The Open Firmware structure for describing these interrupts is defined in the *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10]. Table 50 also defines the mask parameter for the *check-exception* and *event-scan* RTAS functions which limits the search for errors and events to the classes specified.

Personal Use Copy - Not for Reproduction

Table 50. Error and Event Classes with RTAS Function Call Mask

Class Type	Open Firmware Node Name (where the *open-pic-interrupt* property lists the interrupts)	RTAS Function Call Mask (value = 1 enables class)
Internal Errors	*internal-errors*	bit 0
Environmental and Power Warnings	*epow-events*	bit 1
Power Management Events	*power-management-events*	bit 2

Requirements:

10–1. Platform-specific error and event interrupts that a platform provider wants the operating system to enable must be listed in the *open-pic-interrupt* property of the appropriate Open Firmware event class node, as described in Table 50 on page 159.

10–2. To enable platform-specific error and event interrupt notification, operating systems must find the list of interrupts (described in Table 50 on page 159) for each error and event class in the Open Firmware device tree, and enable them.

10–3. Operating systems must have interrupt handlers for the enabled interrupts described in requirement 10–2, which call the RTAS *check-exception* function to determine the cause of the interrupt.

10–4. Platforms which support error and event reporting must provide information to the OS via the RTAS *event-scan* and *check-exception* functions, using the reporting format described in Table 53 on page 172.

10–5. Optional Extended Error Log information, if returned by the *event-scan* or *check-exception* functions, must be in the reporting format described in Table 54 on page 176.

10–6. To provide control over performance, the RTAS event reporting functions must not perform any event data gathering for classes not selected in the event class mask parameter, nor any extended data gathering if the time critical parameter is non-zero or the log buffer length parameter does not allow for an extended error log.

10–7. To prevent the loss of any event notifications, the RTAS event reporting functions must be written to gather and process error and event data

Personal Use Copy - Not for Reproduction

without destroying the state information of events other than the one being processed.

10–8. Any interrupts or interrupt controls used for error and event notification must not be shared between error and event classes, or with any other types of interrupt mechanisms. This allows the operating system to partition its interrupt handling and prevents blocking of one class of interrupt by the processing of another.

10–9. If a platform chooses to report multiple event or error sources through a single interrupt, it must ensure that the interrupt remains asserted or is re-asserted until *check-exception* has been used to process all outstanding errors or events for that interrupt.

10.2.1 Internal Error Indications

Hardware may detect a variety of problems during operation, ranging from soft errors which have already been corrected by the time they are reported, to hard errors of such severity that the OS (and perhaps the hardware) cannot meaningfully continue operation. The mechanisms described in Section 10.1, “Introduction,” on page 157 are used to report such errors to the OS. This section describes the architectural sources of errors, and describes a method that platforms can use to report the error. All OSs need to be prepared to encounter the errors reported as they are described here. However, in some platforms more sophisticated handling may be introduced via RTAS, and the OS may not have to handle the error directly. More robust error detection, reporting, and correcting are at the option of the hardware vendor.

The primary architectural mechanism for indicating hardware errors to an operating system is the machine check interrupt. If an error condition is surfaced by placing the system in check stop, it precludes any immediate participation by the operating system in handling the error (that is, no error capture, logging, recovery, analysis, or notification by the operating system). For this reason, the machine check interrupt is preferred over going to the check stop state. However, check stop may be necessary in certain situations where further processing represents an exposure to data integrity. To better handle such cases, a special hardware mechanism may be provided to gather and store residual error data, to be analyzed when the system goes through a subsequent successful reboot.

Less critical internal errors may also be signalled to the operating system through a platform-specific interrupt in the “Internal Errors” class, or by periodic polling with the *event-scan* RTAS function.

Personal Use Copy - Not for Reproduction

Architecture Note: The machine check interrupt will not be listed in the Open Firmware node for the “Internal Errors” class, since it is a standard architectural mechanism. The machine check interrupt mechanism is enabled from software by setting the MSR_{ME} bit =1. Upon the occurrence of a machine check interrupt, bits in SRR1 will indicate the source of the interrupt and SRR0 will contain the address of the next instruction that would have been executed if the interrupt had not occurred. Depending on where the error is detected, machine check interrupt may be surfaced from within the processor, via logical connection to the processor machine check interrupt pin, or via a system bus error indicator (for example, Transfer Error Acknowledge - TEA).

Requirements:

10–10. Operating systems must set MSR_{ME}=1 prior to the occurrence of a machine check interrupt in order to enable machine check processing via the *check-exception* RTAS function.

10–11. For hardware-detected errors, platforms must generate error indications as described in Table 51 on page 162, unless the error can be handled through a less severe platform-specific interrupt, or the nature of the error forces a check stop condition.

10–12. Platforms which detect and report the errors described in Table 51 on page 162 must provide information to the OS via the RTAS *check-exception* function, using the reporting format described in Table 53 on page 172.

10–13. To prevent error propagation and allow for synchronization of error handling, all processors in a multi-processor system must receive any machine check interrupt signalled via the external machine check interrupt pin.

10.2.1.1 Error Indication Mechanisms

Table 51 on page 162 describes the mechanisms by which software will be notified of the occurrence of operational failures during the types of data transfer operations listed below. The assumption here is that the error notification can occur only if a hardware mechanism for error detection (for example, a parity checker) is present. In cases where there is no specific error detection mechanism, the resulting condition, and whether the software will eventually recognize that condition as a failure, is undefined.

Personal Use Copy - Not for Reproduction

Table 51. Error Indications for System Operations

Initiator	Target	Operation	Error Type (if detected)	Indication to Software	Comments
Processor	N/A	Internal	Various	Machine check	Some may cause check stop
Processor	Memory	Load	Invalid address	Machine check	
			System bus time-out	Machine check	
			Address parity on system bus	Machine check	
			Data parity on system bus	Machine check	
			Memory parity or uncorrectable ECC	Machine check	
		Store	Invalid address	Machine check	
			System bus time-out	Machine check	
			Address parity on system bus	Machine check	
			Data parity on system bus	Machine check	
		External cache load	Memory parity or uncorrectable ECC	Machine check	Associated with Instruction Fetch or Data Load
		External cache flush	Cache parity or uncorrectable ECC	Machine check	
		External cache access	Cache parity or uncorrectable ECC	Machine check	Associated with Instruction Fetch or Data Transfer

Personal Use Copy - Not for Reproduction

Table 51. Error Indications for System Operations ***(Continued)***

Initiator	Target	Operation	Error Type (if detected)	Indication to Software	Comments
Processor	I/O	Load	Data parity on I/O bus	Machine check	
			Data parity on system bus	Machine check	
		Store	Data parity on I/O bus	Machine check	
			Data parity on system bus	Machine check	
		Load or Store	Address parity on system bus	Machine check	
			Invalid address	Machine check	
			I/O bus time-out	Machine check	
			Retry count expired	Machine check	
			Target-abort	Machine check	
			Invalid size	Machine check	
			No response from device	Ignore a Store, all-1's returned on a Load	Invalid address or configuration cycle to non-existent device
Processor	Invalid target address	Load or Store	No response from system	Machine check	
I/O	Memory	DMA - I/O to memory	Data parity on PCI bus	PHB Target-aborts operation	May be recoverable; see note following table
			Data parity on system bus	Machine check	
		DMA - memory to I/O	Data parity on PCI Bus	PHB signals PERR#	May be recoverable; see note following table
			Data parity on system bus	Machine check or PHB signals	
			Memory parity or uncorrectable ECC	Machine check	

Personal Use Copy - Not for Reproduction

Table 51. Error Indications for System Operations ***(Continued)***

Initiator	Target	Operation	Error Type (if detected)	Indication to Software	Comments
I/O	Memory	DMA - either direction	Address parity on I/O bus	Machine check	
			Address parity on system bus	Machine check	
			I/O bus time-out	Machine check	
			Invalid address	Machine check	
			PHB TCE extent	PHB target-aborts operation	May be recoverable; see note following table
			PHB page fault	PHB target-aborts operation	May be recoverable; see note following table
			PHB unauthorized access	PHB target-aborts operation	May be recoverable; see note following table
I/O	I/O	DMA - either direction	Data parity on PCI bus	Device signals PERR#	May be recoverable; see note following table
			Data parity on system bus	Machine check	
			Address parity on I/O bus	Machine check	
			Address parity on system bus	Machine check	
			I/O bus time-out	Machine check	
I/O	Invalid target-address	DMA - either direction	No response from any device	PCI device master-aborts	Signal device driver as an external interrupt,
PCI I/O Device		Any	Internal, indicated by SERR#	SERR#, causing machine check	Use of SERR# for internal errors is discouraged
ISA I/O Device		Any	ISA bus IOCHK#	Machine check	

Implementation Note: I/O devices should detect the occurrence of PCI Target-abort or PCI data parity errors on DMA and report them via an external interrupt (for possible device driver recovery) or retry the

Personal Use Copy - Not for Reproduction

operation. Since system state has not been lost, reporting these errors via a Machine Check to the CPUs is inappropriate. Some devices or device drivers may cause a catastrophic error. Systems which wish to recover from these types of errors should choose devices and device drivers which are designed to handle them correctly.

10.2.2 Environmental and Power Warnings

Environmental and Power Warnings (EPOW) is an option that provides a means for the platform to inform the operating system of these types of events. The intent is to enable the operating system to provide basic information to the user about these problems and to minimize the logical damage done by these problems. For example, an operating system might want to abort all disk I/O operations in progress to ensure that disk sectors are not corrupted by the loss of power.

These warnings include action codes that the platform can use to influence the operating system behavior when various hardware components fail. For example, a fan failure where the system can continue to operate in the safe cooling range may just generate an action code of WARN_COOLING, but a fan failure where the system cannot operate in the safe cooling range may generate an action code of SYSTEM_HALT.

Implementation Note: Hardware can not assume that the operating system will process or take action on these warnings. These warnings are only provided to the operating system in order to allow the operating system a chance to cleanly abort operations in progress at the time of the warning. Hardware still assumes responsibility for preventing hardware damage due to environmental or power problems.

An operating system that wants to be EPOW-aware will look for the *epow-events* node in the Open Firmware device tree, enable the interrupts listed in its *open-pic-interrupt* property, and provide an interrupt handler to call *check-exception* when one of those interrupts are received.

Requirements:

10–14. If the platform supports Environmental and Power Warnings by including a EPOW device tree entry, then the platform must support the EPOW sensor for the *get-sensor-state* RTAS function.

10–15. The EPOW sensor, if provided, must contain the EPOW action code (defined in Table 52 on page 166) in the least significant 4 bits. In cases where multiple EPOW actions are required, the action code with the

Personal Use Copy - Not for Reproduction

highest numerical value (where 0 is lowest and 7 is highest) must be presented to the operating system. The platform may implement any subset of these action codes, but must operate as described in Table 52 for those it does implement.

10–16. To ensure adequate response time, platforms which implement the EPOW_MAIN_ENCLOSURE or EPOW_POWER_OFF action codes must do so via interrupt and *check-exception* notification, rather than by *event-scan* notification.

10–17. For interrupt-driven EPOW events, the platform must ensure that an EPOW interrupt is not lost in the case where a numerically higher-priority EPOW event occurs between the time when check-exception gathers the sensor value and when it resets the interrupt.

Implementation Note: One way for hardware to prevent the loss of an EPOW interrupt is by deferring the generation of a new EPOW interrupt until the existing EPOW interrupt is reset by a call to the RTAS *check-exception* function. Another way is to ignore resets to the interrupt until all EPOW events have been reported.

Table 52. EPOW Action Codes

Action Code	Value	Description
EPOW_RESET	0	No EPOW event is pending. This action code is the lowest priority.
WARN_COOLING	1	A non-critical cooling problem exists. An EPOW-aware operating system logs the EPOW information.
WARN_POWER	2	A non-critical power problem exists. An EPOW-aware operating system logs the EPOW information.
SYSTEM_SHUTDOWN	3	The system must be shut down. An EPOW-aware operating system logs the EPOW error log information, then schedules the system to be shut down in 10 minutes.
SYSTEM_HALT	4	The system must be shut down quickly. An EPOW-aware operating system logs the EPOW error log information, then schedules the system to be shut down in 20 seconds.

Personal Use Copy - Not for Reproduction

Table 52. EPOW Action Codes ***(Continued)***

Action Code	Value	Description
EPOW_MAIN_ENCLOSURE	5	The system may lose power. The hardware ensures that at least 4 milliseconds of power within operational thresholds is available. An EPOW-aware operating system performs any desired functions, masks the EPOW interrupt, and monitors the sensor to see if the condition changes. Hardware does not clear this action code until the system resumes operation within safe power levels.
EPOW_POWER_OFF	7	The system will lose power. The hardware ensures that at least 4 milliseconds of power within operational thresholds is available. An EPOW-aware operating system performs any desired operations, then attempts to turn system power off. An EPOW-aware operating system does not clear the EPOW interrupt for this action code. This action code is the highest priority.

Software Note: A recommended operating system processing method for an *EPOW_MAIN_ENCLOSURE* event is as follows: Prepare for shutdown, mask the EPOW interrupt, and wait for 50 milliseconds. Then call *get-sensor-state* to read the EPOW sensor. If the EPOW action code is unchanged, wait an additional 50 milliseconds. If the action code is *EPOW_POWER_OFF*, attempt to power off. Otherwise, the power condition may have stabilized, so interrupts may be enabled and normal operation resumed.

10.2.3 Power Management Events

Power Management Events, when implemented, are also reported through either the *check-exception* or *event-scan* functions, depending on whether the events are implemented to report through interrupts or not. The architected events are defined in Chapter 11, “Power Management,” on page 185 and are returned in the fixed portion of the RTAS return value (see Table 53 on page 172).

10.3 RTAS Error and Event Information Reporting

Architecture Note: All data formats listed in this section are either referenced as byte fields (and therefore are independent of Endian orientation), or an indicator in the data structure describes their Endian

Personal Use Copy - Not for Reproduction

orientation. Bits are numbered from left (high-order:0) to right (low-order:7).

10.3.1 Introduction

This section describes the data formats used to report events and errors from RTAS to the Operating System. A common format is used for errors and events to simplify software both in RTAS and in the OS. Both errors and events may have been analyzed to some degree by RTAS, and value judgments may have been applied to decide how serious an error is, or even how to describe it to the OS. These judgments are made by platform providers, since only they know enough about the hardware to decide how serious a problem it is, whether and how to recover from it, and how to map it onto the abstracted set of events and errors that a system is required to know about. There will be cases with some platforms where no reasonable mapping exists, and platform features may not be fully supported by the OS (an example might be a platform with power management features beyond those provided for in this specification). In such cases, error reports may also be non-specific, leaving platform-specific details to platform-aware software.

10.3.2 RTAS Error/Event Return Format

This section describes in detail the return value retrieved by an RTAS call to either the *event-scan* or *check-exception* function.

The return value consists of a fixed part and an optional Extended Error Report, described in the next section, which contains full details of the error. The fixed part is intended to allow reporting the most common problems in a simple way, which makes error detection and recovery simple for operating systems that want to implement a very simple error handling strategy. At the same time, the mechanism is capable of providing full disclosure of the error syndrome information for operating systems which have a more complete error handling strategy.

RTAS can return at most one return code per invocation. If multiple conditions exist, RTAS returns them in descending order of severity on successive calls.

10.3.2.1 Reporting and Recovery Philosophy, and Description of Fields

All RTAS implementations use a common error and event reporting scheme, as described in detail below. It is not required that error recovery be present in

Personal Use Copy - Not for Reproduction

RTAS implementations, nor is it required that a high degree of error recovery or survival be undertaken by operating systems. If such behavior is desired, then specific platform-dependent handlers can be loaded into the OS. However, this section defines return result codes and a philosophy which can be used if aggressive error handling is implemented in RTAS. This section describes the fields of the Error Report format, and the philosophy which should be applied in generating return values from RTAS or interpreting such return codes in an OS.

In general, an OS would look at the Disposition field first to see if an error has been corrected already by RTAS. If not corrected to the OS's satisfaction, the OS would examine the Severity field. Based on that value, and optionally on any information it can use from the Type and other fields, the OS will make a determination of whether to continue or to halt operations. In either case, it may choose to log information regarding the error, using the remaining fields and optional Extended Error Log.

The following sections describe the field values in Table 53 on page 172.

10.3.2.1.1 Version Field

This field is used only to distinguish among present and potential future formats for the remainder of the error report. This value will be incremented if extensions are made to the format described here. Future versions will be backward-compatible; the primary function of this field is for future OSs to identify whether an error report may contain some (unknown at present) feature that was added after the initial version of this specification.

10.3.2.1.2 Severity

This field represents the value judgment of RTAS of how serious the problem being reported should be considered by the OS. A platform-aware OS may choose to ignore the RTAS judgment, but non-platform-aware OSs have no way to make adequate value judgments, and should in general believe the RTAS assessment of the situation.

Errors which are believed to represent a permanent hardware failure affecting the entire system are considered "FATAL." OSs would not attempt to continue normal operation after receiving notice of such an error. OSs may not even be able to perform an orderly shutdown in the presence of a Fatal error, though they may make a policy decision to try.

Less serious errors, but still causing a loss of data or state, are considered "ERRORs." In general, continuing after such an error is questionable, since details of what has failed may not be available, or if available, may not map nicely onto any ongoing activity with which the OS can associate it. However, OSs may make a policy decision (for example, based on the error Type, the Initiator, or the Target) to continue operation after an Error.

Personal Use Copy - Not for Reproduction

There are some types of errors, such as parity errors in memory or a parity error on a transfer between CPU and memory, which occur synchronously with the current process execution context. Such errors are sometimes fatal only to the current thread of execution; that is, they affect only the current CPU state and possibly that of any memory locations being currently referenced. If that context of execution is not essential to the system's operation (for example, if an error trap mechanism is available in the OS and can be triggered to recover the OS to a known state), recovery and continuation may be possible. Or at least, since the memory of the machine is in an undamaged state, the system may be able to be brought down in an orderly fashion. Such errors are reported as having severity "ERROR_SYNC." It is OS dependent whether recovery is possible after such an error, or whether the OS will treat it as a fatal problem.

The "WARNING" return value indicates that a non-state-losing error, either fully recovered by RTAS or not needing recovery, has occurred. No OS action is required, and full operation is expected to continue unhindered by the error. Examples include corrected ECC errors or bus transfer failures which were retried successfully.

The "EVENT" return value is the mechanism RTAS uses to communicate event information to the OS. The event may have been detected by polling using *event-scan* or on the occurrence of an interrupt by calling *check-exception*. In either case, the Error Return value indicates the event which has occurred in the Type field. See the Type description below for a description of specific events and their expected handling.

The "NO ERROR" return value indicates that no error was present. In this case, the remainder of the Error Return fields are not valid and should not be referenced.

10.3.2.1.3 RTAS Disposition

An aggressive RTAS implementation may choose to attempt recovery for some classes of error so a non-platform-aware OS can continue operation in the face of recoverable errors. If RTAS detects an error for which it has recovery code, it attempts such action before it returns a value to the OS (that is, the mechanisms are linked in RTAS and cannot be separately accessed). Note that Disposition is nearly independent from Severity. Severity says how serious an error was, and Disposition says, regardless of severity, whether or not the OS has to even look at it. In general, an OS will first examine Disposition, then Severity.

A return value of "FULLY RECOVERED" means that RTAS was able to completely recover the machine state after the error, and OS operation can continue unhindered. The severity of the problem in this case is irrelevant, though for consistency a "FATAL" error can never be "FULLY RECOVERED."

A return value of "LIMITED RECOVERY" means that RTAS was able to recover the state of the machine, but that some feature of the machine has been

Personal Use Copy - Not for Reproduction

disabled or lost (for example, error checking), or performance may suffer (for example, a failing cache has been disabled). The RTAS implementation may return further information in the extended error log format regarding what action was done or what corrective action failed. In general, a conservative OS will treat this return the same as “NOT RECOVERED,” and initiate shutdown. A less conservative OS may choose to let the user decide whether to continue or to shut down.

A value of “NOT RECOVERED” indicates that the RTAS either did not attempt recovery, or that it attempted recovery but was unsuccessful.

10.3.2.1.4 Optional Part Presence

This is a single flag, valid only if the 32-bit Error Return value is located in memory, which indicates whether or not an Extended Error Log Length field and the Extended Error Log follows it in memory. It will be set on an in-memory return result from RTAS if and only if the RTAS call indicated sufficient space to return the Extended Error Log, and the RTAS implementation supports the Extended Error Log.

10.3.2.1.5 Initiator

This field indicates, to the best ability of RTAS to determine it, the initiator of a failed transaction. (Note that in the “Initiator” field of Table 51 on page 162, the value “I/O” indicates one of the defined I/O buses or devices. This field contains finer-grained details of which type of I/O bus failed, if known, and “UNKNOWN” if RTAS cannot tell.)

10.3.2.1.6 Target

If RTAS can determine it, this field indicates the target of a failed transaction.

10.3.2.1.7 Type

This field identifies the general type of the error or event. In some cases (for example, INTERN_DEV_FAIL, or power management events), multiple possible events are grouped together under a common return value. In such cases, a platform-aware OS may use the Extended Error Log to distinguish them. A non-platform-aware OS will generally treat all errors of a given type the same, so it generally will not need to access the Extended Error Log information.

In the table, the EPOW and power management values are associated with a Severity of EVENT. All other values will be associated with Severity values of FATAL, ERROR, ERROR_SYNC, or WARNING, and may or may not be corrected by RTAS.

EPOW is an event type which indicates the potential loss of power or environmental conditions outside the limits of safe operation of the platform. See “Environmental and Power Warnings” on page 165 for more information.

Personal Use Copy - Not for Reproduction

Power Management event types are described in Section 11.2.1.2, “Definition of RTAS Abstracted Power Management Event Types,” on page 199. OSs implementing power management features will pass these events to the appropriate power management software.

Additional Type values will be added in future revisions of the specification. If an OS does not recognize a particular event type, it can examine the severity first, and then choose to ignore the event if it is not serious.

10.3.2.1.8 Extended Error Log Length

This optional 32-bit field is present in memory following the 32-bit Error Return value only if the Optional Part Presence flag is “PRESENT.” If present, it indicates the length in bytes of the Extended Error Log information which immediately follows it in memory. The length does not include this field or the Error Return field, so it may be zero.

10.3.2.1.9 RTAS Error Return Format Fixed Part

The summary portion of the error return is designed to fit into a single 32-bit integer. When used as a data return format in memory, an optional Length field and Extended Error Log data may follow the summary. The fixed part contains a “presence” flag which identifies whether an extended report is present.

In the table below, the location of each field within the integer is included in parentheses after its name. Numerical field values are indicated in decimal unless noted otherwise.

Table 53. RTAS Error Return Format (Fixed Part)

Bit Field Name (bit number(s))	Description, Values (Described in Section 10.3.2.1, “Reporting and Recovery Philosophy, and Description of Fields,” on page 168)
Version (0:7)	A distinct value used to identify the architectural version of message. Current version = (1)
Severity (8:10)	Severity level of error/event being reported: FATAL (5) ERROR (4) ERROR_SYNC (3) WARNING (2) EVENT (1) NO_ERROR (0) reserved for future use (6-7)

Personal Use Copy - Not for Reproduction

Table 53. RTAS Error Return Format (Fixed Part) ***(Continued)***

Bit Field Name (bit number(s))	Description, Values (Described in Section 10.3.2.1, "Reporting and Recovery Philosophy, and Description of Fields," on page 168)
RTAS Disposition (11:12)	Degree of recovery which RTAS has performed prior to return after an error (value is FULLY_RECOVERED if no error is being reported): FULLY_RECOVERED(0) Note: Cannot be used when Severity is "FATAL". LIMITED_RECOVERY(1) NOT_RECOVERED(2) reserved for future use (3)
Optional_Part_Presence (13)	Indicates if an Extended Error Log follows this 32-bit quantity in memory: PRESENT (1): The optional Extended Error Log is present. NOT_PRESENT (0): The optional Extended Error Log is not present.
Reserved (14:15)	Reserved for future use (0:3)
Initiator (16:19)	Abstract entity that initiated the event or the failed operation: UNKNOWN (0): Unknown or Not Applicable CPU (1): A CPU failure (in an MP system, the specific CPU is not differentiated here) PCI (2): PCI host bridge or PCI device ISA (3): ISA bus bridge or ISA device MEMORY (4): Memory subsystem, including any caches POWER_MANAGEMENT (5): Power Management sub-system Reserved for future use (6-15)
Target (20:23)	Abstract entity that was apparent target of failed operation (UN-KNOWN if Not Applicable): Same values as Initiator field

Personal Use Copy - Not for Reproduction

Table 53. RTAS Error Return Format (Fixed Part) ***(Continued)***

Bit Field Name (bit number(s))	Description, Values (Described in Section 10.3.2.1, "Reporting and Recovery Philosophy, and Description of Fields," on page 168)
Type (24:31)	General event or error type being reported: Internal Errors: RETRY (1): too many tries failed, and a retry count expired TCE_ERR (2): range or access type error in an access through a TCE INTERN_DEV_FAIL (3): some RTAS-abstracted device has failed (for example, TODC) TIMEOUT (4): intended target did not respond before a time-out occurred DATA_PARITY (5): Parity error on data ADDR_PARITY(6): Parity error on address CACHE_PARITY (7): Parity error on external cache ADDR_INVALID(8): access to reserved or undefined address, or access of an unacceptable type for an address ECC_UNCORR (9): uncorrectable ECC error ECC_CORR (10): corrected ECC error RESERVED (11-63): Reserved for future use Environmental and Power Warnings: EPOW(64): See Extended Error Log for sensor value RESERVED (65-95): Reserved for future use Power Management Events(96-159): power management event occurred - see Table 62 on page 199 for defined event values Reserved for future use (160-223) Vendor-specific events(224-255): Non-architected Other (0): none of the above
Extended Error Log Length (32:63)	Length in bytes of Extended Error Log information which follows (see 10.3.2.2, "Extended Error Log Format Returned by RTAS," on page 174)

Typically, most operating systems care about, and have handlers for, only a few specific errors. Since coding of an error is unique in the above scheme, an operating system can check for specific errors, then if nothing matches exactly, look at more generic parts of the error message. This permits generic error message generation for the user, providing the basic information that RTAS delivered to the operating system. Platforms may provide more complete error diagnosis and reporting in RTAS, combined with off-line diagnostics which take advantage of the information reported from previous failures.

10.3.2.2 Extended Error Log Format Returned by RTAS

The following tables define an extended error log format by which the RTAS can optionally return detailed information to the software about a hardware error condition. This format is also intended to be usable as residual error log

Personal Use Copy - Not for Reproduction

data in NVRAM, so that the operating system could alternatively retrieve error data after an error event which caused a reboot.

Figure 12 and Table 54 on page 176 shows the general layout for the extended error log format, while Tables 55 through 60 show the detailed layout of bytes 12 through 39. The detail area format is determined by bits 4:7 of byte 2, which indicate the error log type.

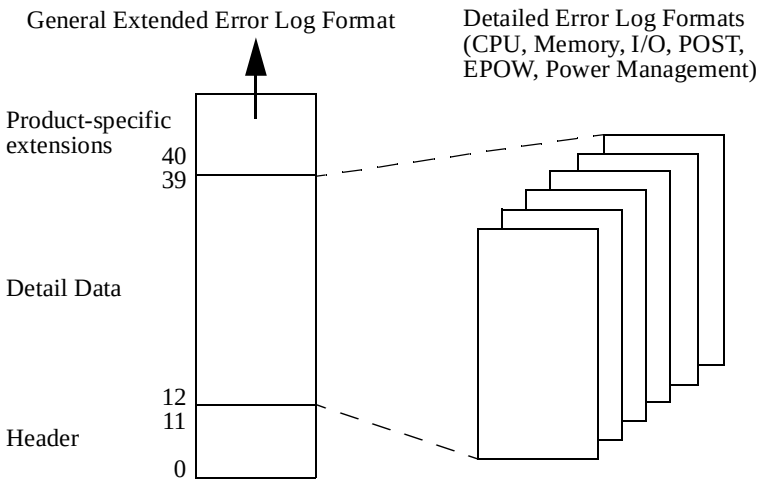


Figure 12. Layout of extended error log format from RTAS.

Extended product-unique data may be added to the end of the error log buffer (starting at byte 40) for capture and logging by the operating system. Platforms which provide extended data indicate the maximum length of the error log buffer in the *rtas-error-log-max* RTAS property in the Open Firmware device tree, so that the operating system can allocate a buffer large enough to hold the extended error log data when calling the RTAS *event-scan* or *check-exception* functions. If the allocated buffer is not large enough to hold all the error log data, the data is truncated to the size of the buffer.

Personal Use Copy - Not for Reproduction

Table 54. RTAS General Extended Error Log Format

Error Log Format		
Byte	Bit(s)	Description
0	0	1 = Log Valid
	1	1 = Unrecoverable Error
	2	1 = Recoverable (correctable or successfully retried) Error
	3	1 = Unrecoverable Error, Bypassed - Degraded operation (for example, Single CPU taken off-line, bad cache bypassed)
	4	1 = Predictive Error - Error is recoverable, but indicates a trend toward unrecoverable failure (for example, correctable ECC error threshold)
	5	1 = “New” Log (always 1 for data returned from RTAS)
	6	1 = Addresses/Numbers are Big-Endian format, 0 = Little-Endian Note: This bit will always be set to the Endian mode in which RTAS was instantiated.
	7	Reserved
1	0:7	Reserved
2	0	Set to 1 - (Indicates log is in PowerPC format)
	1:2	Reserved
	3	1 = No failing address was available for recording within the log’s Detailed Log Data, so the address field is invalid
2	4:7	Log format indicator, defines format used for bytes 12-39: (0) Reserved (1) CPU-detected failure, see Table 55 on page 178 (2) Memory-detected failure, see Table 56 on page 178 (3) I/O-detected failure, see Table 57 on page 180 (4) Power-On Self Test (POST) failure, see Table 58 on page 182 (5) Environmental and Power Warning, see Table 59 on page 183 (6) Power Management Event, see Table 60 on page 183 (7-11) Reserved (12-15) Vendor-specific

Note: Some of these elements are sensitive to Endian orientation, and are indicated by an “*” in the Byte column. Byte 0, Bit 6 of the log structure indicates the Endian orientation.

Personal Use Copy - Not for Reproduction

Table 54. RTAS General Extended Error Log Format ***(Continued)***

Error Log Format		
Byte	Bit(s)	Description
3	0:3	Reserved
	4	1 = Error is residual information from a failure which occurred prior to the last boot (for example, stored information about a machine check that crashed the system before RTAS could report it to the OS)
	5	1 = Error detected during IPL process (If neither bit 5 nor bit 7 is on, the error occurred after control was passed to the operating system)
	6	1 = Configuration changed since last boot.
	7	1 = Error detected prior to IPL (in POST or firmware extended diagnostics)
4-7	Note: Time and Date are based upon the same values and time base as the RTAS Time-of-Day functions. Time of most recent error in BCD format: HHMMSS00, where HH=00-23, MM=00-59, SS=00-59	
8-11	Date of most recent error in BCD format: YYYYMMDD, where YYYY=1995-future, MM=01-12, DD=01-31	
12-39	Detailed log data (See Detail log formats, Tables 55 through 60)	
Note: Some of these elements are sensitive to Endian orientation, and are indicated by an "*" in the Byte column. Byte 0, Bit 6 of the log structure indicates the Endian orientation.		

Personal Use Copy - Not for Reproduction

Table 55. Error Log Detail for CPU-Detected Error

CPU-detected error log format, bytes 12-39		
Byte	Bit(s)	Description
12	0	1 = CPU internal Error, other than cache Note: If failure cannot be isolated, these bits may all be 0
	1	1 = CPU internal cache error
	2	1 = External (L2) cache parity or multi-bit ECC error
	3	1 = External (L2) cache ECC single-bit error
	4	1 = Time-out error, waiting for memory controller
	5	1 = Time-out error, waiting for I/O
	6	1 = Address/Data parity error on Processor Bus
	7	1 = Transfer error on Processor Bus
13	Physical CPU ID number	
14-15 *	Identifier number of sender of data/address parity error, or element which timed out	
16-23 *	64-bit Memory Address for cache error (High-order bytes =0 if 32-bit addressing)	
24-39	Reserved	

Note: Some of these elements are sensitive to Endian orientation, and are indicated by an "*" in the Byte column. Byte 0, Bit 6 of the log structure indicates the Endian orientation.

Table 56. Error Log Detail for Memory Controller-Detected Error

Memory Controller-detected error log format, bytes 12-39		
Byte	Bit(s)	Description
12	0	1 = Uncorrectable Memory Error (parity or multiple bit ECC) Note: If failure cannot be isolated, these bits may all be 0
	1	1 = ECC correctable error
	2	1 = Correctable error threshold exceeded

Note: Some of these elements are sensitive to Endian orientation, and are indicated by an "*" in the Byte column. Byte 0, Bit 6 of the log structure indicates the Endian orientation.

Personal Use Copy - Not for Reproduction

Table 56. Error Log Detail for Memory Controller-Detected Error ***(Continued)***

Memory Controller-detected error log format, bytes 12-39		
Byte	Bit(s)	Description
12	3	1 = Memory Controller internal error
	4	1 = Memory Address (Bad address going to memory)
	5	1 = Memory Data error (Bad data going to memory)
	6	1 = Memory bus/switch internal error
	7	1 = Memory time-out error
13	0	1 = Processor Bus parity error, detected by Memory Controller
	1	1 = Processor time-out error, detected by Memory Controller
	2	1 = Processor bus Transfer error
	3	1 = I/O Host Bridge time-out error, detected by Memory Controller
	4	1 = I/O Host Bridge address/data parity error, detected by Memory Controller
	5:7	Reserved
14	Physical Memory Controller number which detected error (0 if only one controller)	
15	Physical Memory Controller number which caused error (0 if only single memory controller, or if the error source is in main memory, not another memory controller)	
16-23 *	64-bit Memory Address (High-order bytes =0 if only 32-bit address)	
24-25 *	Syndrome bits (included if single-bit correctable error)	
26	Memory Card Number (0 if on system board)	
27	Reserved	
28-31 *	0:31	Memory sub-elements (for example, SIMMs/DIMMs) implicated on this card (or system board), 1 bit per sub-element
32-33 *	Identifier number of sender of data/address parity error, or element which timed out	
34-39	Reserved	

Note: Some of these elements are sensitive to Endian orientation, and are indicated by an “*” in the Byte column. Byte 0, Bit 6 of the log structure indicates the Endian orientation.

Personal Use Copy - Not for Reproduction

Table 57. Error Log Detail for I/O-Detected Error

I/O-detected error log format, bytes 12-39

Byte	Bit(s)	Description
12	0	1 = I/O Bus Address Parity Error Note: If failure cannot be isolated, these bits may all be 0
	1	1 = I/O Bus Data Parity Error
	2	1 = I/O Bus Time-out Error
	3	1 = I/O Device Internal Error
	4	1 = Signaling device is a PCI to non-PCI bridge chip, indicating an error on the secondary bus (for example, ISA IOCHK#)
	5	1 = Mezzanine/Processor Bus Address Parity Error
	6	1 = Mezzanine/Processor Bus Data Parity Error
	7	1 = Mezzanine/Processor Bus Time-out Error
13	0	1 = Bridge is connected to Processor Bus
	1	1 = Bridge is connected to Memory Controller via Mezzanine Bus
	2:7	Reserved
14	PCI Bus ID of the device signaling the error	
15	0:4	PCI Device ID of the device signaling the error
	5:7	PCI Function ID of the device signaling the error
16-17 *	PCI “Device ID” of the device signaling the error (from configuration register)	
18-19 *	PCI “Vendor ID” of the device signaling the error (from configuration register)	
20	PCI “Revision ID” of the device signaling the error (from configuration register)	
21	Slot Identifier number of the device signaling the error ’00’ if system board device ’FF’ if multiple devices signaling an error	
22	PCI Bus ID of the sending device at the time of error	

Note: Some of these elements are sensitive to Endian orientation, and are indicated by an “*” in the Byte column. Byte 0, Bit 6 of the log structure indicates the Endian orientation.

Personal Use Copy - Not for Reproduction

Table 57. Error Log Detail for I/O-Detected Error ***(Continued)***

I/O-detected error log format, bytes 12-39

Byte	Bit(s)	Description
23	0:4	PCI Device ID of the sending device at the time of error
	5:7	PCI Function ID of the sending device at the time of error
24-25 *	PCI “Device ID” of the sending device at the time of error (from configuration register)	
26-27 *	PCI “Vendor ID” of the sending device at the time of error (from configuration register)	
28	PCI “Revision ID” of the sending device at the time of error (from configuration register)	
29	Slot Identifier number of the sending device at the time of error ’00’ if system board device ’FF’ if sender cannot be identified, or if no sender (for example, internal SERR#)	
30-39	Reserved	

Note: Some of these elements are sensitive to Endian orientation, and are indicated by an “*” in the Byte column. Byte 0, Bit 6 of the log structure indicates the Endian orientation.

Personal Use Copy - Not for Reproduction

Table 58. Error Log Detail for Power-On Self Test-Detected Error

Power-On Self Test error log format, bytes 12-39		
Byte	Bit(s)	Description
12	0	1 = Firmware Error
	1	1 = Configuration Error
	2	1 = CPU POST Error
	3	1 = Memory POST Error
	4	1 = I/O Subsystem POST Error
	5	1 = Keyboard POST Error
	6	1 = Mouse POST Error
	7	1 = Graphic Adapter / Display POST Error
13	0	1 = Diskette Initial Program Load (IPL) Error
	1	1 = Drive Controller IPL Error (SCSI, IDE, etc.)
	2	1 = CD-ROM IPL Error
	3	1 = Hard disk IPL Error
	4	1 = Network IPL Error
	5	1 = Other IPL Device Error (Tape, Flash Card, etc.)
	6	Reserved
	7	1 = Self-test error in firmware extended diagnostics
14-25	Device Name (Open Firmware Device for which self-test failed. Name truncated if longer than 12 bytes.)	
26-29 *	POST Error Code	
30-31 *	Firmware Revision Level	
32-39	Location Name (platform-specific identifier which points to specific instance of failing device)	

Note: Some of these elements are sensitive to Endian orientation, and are indicated by an “*” in the Byte column. Byte 0, Bit 6 of the log structure indicates the Endian orientation.

Personal Use Copy - Not for Reproduction

Table 59. Event Log Detail for Environmental and Power Warnings

Environmental and Power Warning event log format, bytes 12-39		
Byte	Bit(s)	Description
12-15 *	EPOW Sensor Value (low-order 4 bits contain the action code)	
16-39	Reserved	

Note: Some of these elements are sensitive to Endian orientation, and are indicated by an "*" in the Byte column. Byte 0, Bit 6 of the log structure indicates the Endian orientation.

Table 60. Event Log Detail for Power Management Events

Power Management event log format, bytes 12-39		
Byte	Bit(s)	Description
12-15 *	Integer identifier of the source of the power management event (product-specific)	
16-39	Reserved	

Note: Some of these elements are sensitive to Endian orientation, and are indicated by an "*" in the Byte column. Byte 0, Bit 6 of the log structure indicates the Endian orientation.

Personal Use Copy - Not for Reproduction

Blank page when cut - 184

Power Management 11

Power management is a set of hardware, firmware, and system software facilities employed in the efficient utilization of electrical power while continuing to meet the computational needs of the user. Support for power management is an important new feature of this architecture. Although the initial impetus for power management was the extension of useful operational time for battery-powered portable computing, the scope of power management has been broadened to include energy conservation in non-portable personal computers and servers. Power management facilities can also be utilized to perform certain network resource management functions such as remote shutdown/power-off and reboot/power-on.

The fundamental principle of reducing average power consumption is quite simple — turn off any device which is not currently in use. This policy is very straight-forward to implement, but ignores the reality that any device which is currently unused may be required at any instant in the future. If it is required immediately after it was turned off, both the time required to turn the device off and the time required to restore it to its power-on state are wasted. This delay will impact the responsiveness of the system. The key to power management which is unobtrusive is the anticipation of the future device utilization requirements of the user.

11.1 Power Management Concepts

A common understanding of power management concepts and terminology will facilitate the description of power management facilities to be provided by

hardware, firmware, and system software. The following sections contrast power management policy and mechanism, present the concepts of device and system power states, and discuss power domains and control points. This is followed by sections describing power sources and batteries, power management events, the explicit transfer of policy responsibility, and EPA Energy Star Compliance.

11.1.1 Power Management Policy Versus Mechanism

Briefly *mechanism* is the *how* of power management and *policy* is the *when and why*. Policy determines the conditions under which power state transitions are initiated. Mechanism deals with the means by which these transitions are carried out and the sensing of events which trigger these transitions.

The term *policy* is often used to refer to both the strategies used to control power as well as the entity executing these strategies. Policy can be implemented in hardware, firmware, system software, or application software, but is usually implemented via a combination of two or more of these. The rule employed in this architecture is to allow hardware and firmware to set policy until system and/or application software explicitly assumes control. Once this takes place, platform hardware/firmware is designed to totally surrender control to system and/or application software except in cases where it must intervene to insure the safety of persons, property or the platform hardware.

The underlying principle is that the operating system as the central service in the computing system has the most information concerning future device access demands of the current computational workload and thus is in the best position to optimize energy usage while maintaining overall system responsiveness and/or computational throughput. The operating system is also best equipped to resolve conflicts between system power state transition requests from multiple active power management aware (PM-aware) applications.

Power management mechanism involves both the hardware facilities which control the consumption of electrical power in devices and the methods of routing control information from the policy to these facilities. Mechanism also refers to methods of routing information from devices or sensors to the policy. Mechanism is implemented by hardware, Run-Time Abstraction Services, and system software (primarily device drivers and interrupt handlers).

To intelligently utilize these mechanisms the policy must have access to an internal model of the power consumption of the devices which constitute the system and understand the topology of devices as they are organized in *power domains*.

Personal Use Copy - Not for Reproduction

11.1.2 Device Power States

A device is any physical entity which consumes power in a computer system. All devices appear as nodes in the Open Firmware device tree. Some devices provide software controllable modes of operation which consume varying levels of power. These devices will be called power-manageable devices. Other devices have no such capability. Power savings can still be achieved in systems which employ this second class of devices by either removing power from these devices or clocking them at slower speeds. These devices will be called indirectly-managed devices.

An indirectly-managed device is defined to possess a single power state. Power-manageable devices have two or more power states. An important attribute of each power state is whether the device retains its internal functional parameters. An indirectly-managed device always retains its internal functional parameters. A power-manageable device is required to retain its internal functional parameters in all supported power states with the possible exception of its lowest power state.

Most devices have associated with them a device driver through which the operating system controls the device and obtains services from it. A power-management-enabled device driver is responsible for providing to the operating systems mechanisms for controlling the power state of a device. The number and characteristics of these device power states are recorded in the device tree and are made available to the operating system via the client interface of Open Firmware. The device driver associated with a power-manageable device is also responsible for restoring the internal operational parameters of the device when directed to bring the device to an operational state if the prior device power state may have caused these internal parameters to be lost.

Certain devices employ internal power conservation techniques which are not software controllable. It is recommended that such a device provide a means of disabling its power conservation mode if the utilization thereof effects the device's service time in any way. These power conservation techniques must not impact the correct operation of software.

11.1.3 System Power Management States

The concept of a system power state facilitates the description of the problems associated with managing power and the individual responsibilities of hardware, firmware, and system software. The following sections present definitions for six "static" power states. Section 11.1.4, "System Power Transitory States," on page 190 defines three transitory power states.

Personal Use Copy - Not for Reproduction

11.1.3.1 Power Management Disabled

In this state the system is its most responsive to application and system software. It may also be the state where power usage is maximum. All power-manageable devices are inhibited from entering any of their supported lower power states. All self-managed devices are inhibited from entering their energy conservation modes, if this capability is available. All power domains (see Section 11.1.5.1, “Power Domains,” on page 191) are set to their Full On power level (domain power levels are described in section 11.1.5.4, “Domain Power Levels,” on page 194).

11.1.3.2 Power Management Enabled

The Power Management Enabled state is the state in which a power managed system spends the majority of its operational time. While the system is in the Power Management Enabled state the power management software is actively engaged in the control of the device power states with the goal of minimizing energy consumption.

In this state the system is completely operational. Various devices may be commanded by the power management software to move to lower power states. This may increase latency for applications requiring service from these devices.

11.1.3.3 Standby

Standby is intended to be the lowest power state where the system is responsive to interrupts from all sources. The CPU will be placed in a low power mode if one is available. However, the CPU is still in control of the system and will respond (perhaps sluggishly) to interrupts. The power management software will place all I/O devices in the lowest power state supported by the device unless it is involved in the generation or routing of power management events. Operational parameters are retained within the devices. No data loss occurs in Standby.

An operating system implementation specific set of I/O interrupts will cause an immediate transition back to the Power Management Enabled State. This transition should be imperceptible to the user and certainly less than one second.

Personal Use Copy - Not for Reproduction

11.1.3.4 Suspend

Suspend is defined to be a system state which consumes very little power and yet allows the very rapid resumption of software activity based on the detection of a platform specific and software maskable set of events.

After operational parameters of peripheral devices and the contents of both the L1 and write-back L2 caches are saved to main memory, these devices may be powered off. Main memory is placed in a low power data retention state (for Dynamic RAM, slow refresh). All I/O devices not necessary to the detection of Resume events are normally powered off. Finally the main processor itself may be powered off.

If the processor does not retain its internal state, the re-application of power followed by a reset is required to move to a more responsive state. The transition from the Suspend state to one of the more responsive states is called Resume. This process should require less than 10 seconds.

Depending on the amount of time the system spends in the Suspend state and the amount of time required to return to an operational state after entering Suspend, the system may not be able to maintain all communication sessions established prior to entering this state.

11.1.3.5 Hibernate

In the Hibernate state the system should consume no more power than when the system is in the Off state, yet the preparation described below allows the restoration of system and operational software upon the detection of one or more of a platform specific and software maskable set of events. This set may be different than the set of events which trigger Resume.

Prior to entering the Hibernate state, the operational state of the system must first be saved to main memory. This process is the same as the process of preparing for Suspend described above. Preparation for Hibernate, however, requires an additional step. After the operational state of the system is contained in main memory, an image of main memory must be written (perhaps in compressed form) to a nonvolatile secondary storage medium. During this time, devices which are necessary to the process will be brought to whatever device power states are required to execute the process.

If this process is successful, it will be possible on reboot of the system to restore the operational state of the system to one which is indistinguishable to the user from the state of the system prior to the time that the transition to Hibernate took place. An exception is communication sessions which will have been broken and will need to be reestablished by the user after the operational state of the system has been restored.

Personal Use Copy - Not for Reproduction

After the successful completion of preparing for hibernation, the system should move immediately to the Hibernate state. In the Hibernate state the system is not operational. Power usage may not be zero in this state, but to the user all indication that the system is powered will be absent.

The restoration of a former operational state from the Hibernate state is called Wakeup. Although the time required for this process is highly dependent on the installed operating system and will increase for larger system memory sizes, it should require less time than that required to perform a cold boot. A goal for this process is less than 100 seconds from the trigger event to the time when the system appears responsive to user input.

11.1.3.6 Off

In this state the system is not operational. Power usage may not be zero in this state, but to the user all indication that the system is powered will be absent. One or more of a platform specific and software maskable set of events initiates the transition out of this state. This set may be different than the set of events which trigger Resume or Wakeup. On exiting the Off state, the boot processor will experience a hardware reset.

11.1.4 System Power Transitory States

A system power transitory state is a process which if successfully completed moves the system from one static power state to another. These processes are initiated by either explicit user request or one or more power management events. Some of these actions are very quick and simple, others are more involved. This section presents three of the more involved system power transitory states. Section 11.1.8, “Power Management Events,” on page 195 defines the concept of power management events.

11.1.4.1 Resume

Resume is defined to be the process of restoring the system to the operational state that existed prior to the transition to the Suspend system state. In order to exit the Suspend state the boot processor may experience a reset at the outset of the Resume process.

After reinitializing all standard devices, firmware transfers control back to the system software which was active prior to the system Suspend request.

Personal Use Copy - Not for Reproduction

11.1.4.2 Wakeup

Wakeup is defined to be the process of restoring the system to the operational state that existed prior to the transition to the Hibernate state. The process is normally initiated by the detection of a wakeup event by hardware. Hardware and firmware reset and disable or initialize and enable system support logic and system board I/O devices. A possibly abbreviated power on self test follows. The remainder of the process proceeds just like a cold boot. The operating system is responsible to discriminate this from a cold boot, find the image of the system state which has been saved to a secondary nonvolatile storage medium, and bring it back into main memory.

11.1.4.3 Powerup

Powerup is defined as the process, also known as cold boot, of moving from the Off state to either the Power Management Enabled or Power Management Disabled system power state. Hardware and firmware are responsible to initialize all system devices and generally performs some type of hardware test. The actuation of the system power switch is an example of action which could initiate the Powerup transitory state.

11.1.5 Power Domains and Domain Control Points

A power domain groups one or more devices together and represents a dependency relationship. This dependency relationship often cuts across device drivers and therefore must be managed at a level which transcends the device drivers. The power level of a power domain affects the power consumption of one or more devices within a domain in a manner which is outside the normal capability of a manageable device or its device driver. It may accomplish this by interrupting or restoring the flow of power to a device or lowering or raising its clock frequency. A power domain control point is a device in the Open Firmware device tree which provides software control over a power domain. Power domains, domain control points, and domain power levels are discussed in the following sections.

11.1.5.1 Power Domains

Logically a power domain is a set of devices in the Open Firmware device tree that share the same power domain name. Membership is specified via the Open Firmware property *power-domains*.

Personal Use Copy - Not for Reproduction

Any device whose power consumption is dependent on a mechanism of which the device driver is unaware must belong to a power domain. Domains may have hierarchical relationships with one another.

A device may be the member of more than one power domain. Membership in multiple domains may be used to represent dependency upon multiple different external provisions such as multiple voltages or clocks. Devices are not considered members of multiple power domains, however, simply because they are in a subdomain.

The device power state of power-manageable devices affects the total average and peak power consumption of the domain to which it belongs. Additionally the various legal power state transitions of these devices will impose different transitory power demands on the domain.

A platform which supports the *set-power-level* RTAS function must support the root power domain. This domain is the parent of all other power domains in the system (or contains all physical devices if there are no subordinate domains). If a device in the Open Firmware device tree does not have domain membership explicitly defined by encoding the *power-domains* property in a system with software control of the main power switch, it defaults to being a member of the root domain.

11.1.5.2 Power Domain Control Point

A power domain control point is a device which appears in the device tree and exercises control over one or more power domains. The Open Firmware property *controls-power-domains* enumerates these domains for a given control point. Control points may themselves be members of other power domains. This membership is expressed through the *power-domains* property just like other devices. A power domain controller is not allowed, however, to be a member of a domain that it controls.

11.1.5.3 Power Domain Dependencies

The power domain dependency tree is a data structure which represents the functional dependencies of power domains within a platform. A functional dependency exists when the power level of one domain affects the operation or usefulness of another domain. The value of the *power-domains-tree* property of the *rtas* node encodes the power domain dependency tree data structure.

Refer to Figure 13 on page 193 which provides a representation of a simple set of dependency relationships between power domains and devices in the Open Firmware device tree. The figure shows three power domains and three devices. The devices are not formally part of the domain dependency tree and

Personal Use Copy - Not for Reproduction

are not explicitly represented in the *power-domains-tree* property. Devices have membership in one or more power domains. This membership implies a dependency relationship between a device and its containing domain. Devices may have dependencies on other devices. One device is dependent on another if the power state of the first affects the operation of the other.

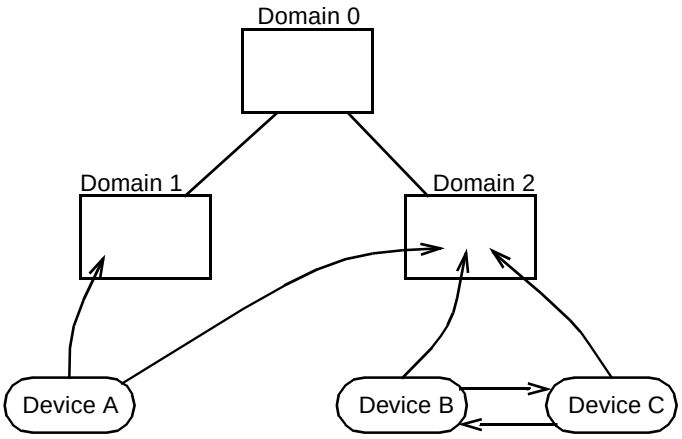


Figure 13. Example Domain and Device Dependency Relationships

In the figure, Device A belongs to two domains — Domain 1 and Domain 2. Device B and Device C, which share a mutual dependency, belong to Domain 2. Domain 1 and Domain 2 are children of Domain 0. This means that the power level of Domain 0 affects the operation of Domain 1 and Domain 2. Because Domain 1 and Domain 2 are siblings, the power state of Domain 1 must not affect the operation of Domain 2 and vice versa. While it is legal for Device B and Device C to share either a one way or mutual dependency, the requirements for the power domain dependency tree do not allow a dependency relationship to exist between a device which is a member of one domain and another device which has membership in a sibling or ancestral domain. For example, if a dependency relationship were to exist between Device A and Device B, Domains 1 and 2 would need to be combined to maintain compliance to the requirements for a power domain dependency tree. These requirements are given in Section 11.2.2.1, “Power Domain Dependency Tree,” on page 201.

Personal Use Copy - Not for Reproduction

11.1.5.4 Domain Power Levels

There are four defined domain power levels: Full On, Reduced, Freeze, and Off. Domains may support all levels, but must support at least two. Switching domain power levels affects operation of devices within the domain and any subdomains. This switching must not, however, affect the correct operation of other domains. The states are defined so that an operating system knows how devices are affected and can take appropriate action before changing domain power levels. The implementation of power domain levels must insure that devices within the domain and any subdomains will operate in a known fashion. The four domain power levels are described below.

- Full On: In this level devices in the domain and any subdomain have full power available to them.
- Reduced: This level must consume less power than Full On. Power available to devices within the domain and all subdomains must allow the devices to be fully operational although they may experience increased service times. Operating systems should assume that devices in the domain will not work as quickly as in Full On.
- Freeze: This level must consume less power than Reduced. Power available to devices in the domain and any subdomains must allow those devices to retain their internal operational parameters, but the devices are not required to be functional. Operating systems should assume that the devices in the domain can no longer be used.
- Off: This level must consume less power than Freeze and ideally will consume no power. Devices in the domain and any subdomains are not functional and will not retain their internal parameters. Operating systems need to be aware that device parameters may need to be reset or restored after switching the domain to another level.

A given power domain may not implement all four levels defined above. The power domain dependency tree specifies the power levels supported for each domain.

Hardware Implementation Note: Domain power levels are separate from device power states. Changing power levels only changes the power available to devices within the domain. The implementation of the RTAS method *set-power-level* must not directly operate on a device and the indirect effects must be constrained to those described above.

Personal Use Copy - Not for Reproduction

The defined values for the Level argument used in the *set-power-level* and *get-power-level* RTAS calls are specified in Section 11.2.1.1, “Definition of Domain Power Levels,” on page 198.

11.1.6 Power Sources

A power source is either a DC/DC convertor or a subsection of an AC power supply which sources power to a given set of devices via a distribution plane or cabling network. Power sources have associated with them the values *peak-power* and *average-power.* Each defined power source in a system appears in the property value list of the *platform-power-sources* property of the *rtas* node of the Open Firmware device tree. Each device which consumes power sourced by a power source encodes the *power-sources* property to list the power sources upon which it depends.

Power sources and their lists of dependent consuming devices are intended to be used by system software in the budgeting of power.

11.1.7 Batteries

Battery management is a vital aspect of power management in a battery-operated platform. The *platform-battery-sources* property of the Open Firmware device tree *rtas* node encodes important characteristics of the batteries which are present in a system. This data structure provides the identification and manufacturer’s rated capacity for each battery in the platform.

11.1.8 Power Management Events

A power management event (PM event) is a condition which is reported to the policy entity and, based on the specific policy algorithm, may result in device and/or system power state transitions. PM events are generated by devices, device drivers, or application programs. A generator of a PM event is called an event source. PM events which are generated internal to a device driver or application program are beyond the scope of this architecture.

A key feature of this architecture which enables a shrink-wrap operating system to effectively manage power is the grouping of power management events into abstract event types. Because these event types have well-defined semantics, it becomes possible for an operating system to set reasonable policy algorithms without knowing the details of the specific platform.

There are three basic mechanisms for getting processor attention when an event occurs. The first is employed when the boot processor is not powered as

Personal Use Copy - Not for Reproduction

during Suspend, Hibernate, or Off. Here the mechanism is the re-application of power followed by a reset which is directed to the boot processor. In the case of a Resume from Suspend, the event type which triggered the Resume is returned via a parameter in the RTAS *suspend* call buffer. For Wakeup and Powerup a subsequent call to the RTAS function *event-scan* may provide more information as to the specific causal event.

The second attention mechanism is employed when the processor(s) is (are) powered as in the system power states Standby and Power Management Enabled. In this case the platform asserts the power management interrupt. On receipt of a power management interrupt the processor can use the RTAS call *check-exception* to identify the event type and source. A third mechanism is a polled one provided via the RTAS call *event-scan*. In addition other PM event delivery mechanisms may be employed which are unique to the operating system.

Architecture Note: Although the *change* in state of some sensors may be reported as a power management event, the *get-sensor-state* RTAS call always returns the current state of the sensor, not necessarily its value at the time that the event was asserted. Power management events are reported either via an interrupt which is discriminated by a *check-exception* call or via *event-scan.* Which mechanism an event source employs is platform dependent, but a specific event will not be reported by both methods. If more than one event source of a given type exists on a platform, the event presented is the logical OR of these events. Thus a given event type may employ both event reporting mechanisms. Properties in the device tree specify which interrupts must be enabled by system software for each of the defined error/event classes. The event type is reported in the type field of the error/exception return buffer fixed part (see Chapter 10, “Error and Event Notification,” on page 157) and the event source is specified in the extended part of same (refer to Section 10.3.2.2, “Extended Error Log Format Returned by RTAS,” on page 174).

11.1.9 Explicit Transfer of Power Management Policy

A pair of RTAS calls allow the explicit transfer of power management policy responsibility between the platform and the system software. The *assume-power-management* call transfers responsibility for power management control from the platform hardware/firmware to system software. The *relinquish-power-management* call transfers responsibility back to the platform. The default platform policy is not intended to provide active device power management which is the responsibility of system software.

Personal Use Copy - Not for Reproduction

On hard reset the platform hardware/firmware owns the responsibility for power management policy. This normally involves very simple rules concerning the effects of a limited number of events such as depressing the system power switch or the restoration of AC power after an outage. When system software is loaded and the power management software is operational, it signifies its readiness to assume responsibility for the system power management policy by executing the *assume-power-management* RTAS call.

In the termination phase of the typical operation of a system, system software should explicitly transfer responsibility for system power management back to the platform. This is accomplished by calling the *relinquish-power-management* RTAS function. See Section 7.3.7, “Power Management,” on page 123 for a description of these functions.

11.1.10 EPA Energy Star Compliance

The EPA Memorandum of Understanding (MOU) as amended in October, 1994, states that a computer system must possess a low-power “sleep” mode in which the system unit consumes less than 30 watts. The utilization of the Suspend or Hibernate system power states described above in a system design are possible means by which this may be achieved. The EPA MOU is subject to change at any time.

11.2 Power-Managed Platform Requirements

The design of a power-managed system presents the designer with new requirements based on the desired level of power management functionality. The following sections present these requirements. First parameters defined for use by RTAS are given. Then current and future Open Firmware properties required in a power-managed system are presented. Next general requirements for all platforms which support power management are given. Finally some optional power management features and recommendations are presented.

Personal Use Copy - Not for Reproduction

11.2.1 Definition of Power Management Related Parameters Utilized by RTAS

Platforms which support power management are required to implement all RTAS calls designated for power-managed platforms as defined in Section 7.2.6, “RTAS Device Tree Properties,” on page 98. The following sections define the values of parameters associated with power management which are employed in specific RTAS calls.

11.2.1.1 Definition of Domain Power Levels

The *set-power-level* and *get-power-level* RTAS calls use the argument *Level*. The following defines the requirements for the implementation of this value.

Requirements:

11–1. For the Power Management option: The *set-power-level* and *get-power-level* RTAS calls must respectively accept as input and return the values of the argument *Level* as defined in Table 61 on page 198.

Table 61. Defined Power Levels

Level	Value	Description
Full On	100	Platform insures that power and clocking meets the requirements of all devices within the domain for maximal performance.
Reserved	21-99	Reserved for future use.
Reduced Power	20	Platform uses whatever techniques available to lower power consumption within the specified domain, but does nothing which prevents correct operation.
Reserved	11-19	Reserved for future use.
Freeze	10	Platform may use power conservation techniques which stop operation of devices within the domain, but does nothing which prevents devices within the domain from retaining their internal functional parameters.
Reserved	1-9	Reserved for future use.
Off	0	Platform may remove power from all devices within the specified domain.

Personal Use Copy - Not for Reproduction

11.2.1.2 Definition of RTAS Abstracted Power Management Event Types

The RTAS calls *event-scan* and *check-exception* are both capable of reporting power management events and event types using the error log format defined in Section 10.3.2, “RTAS Error/Event Return Format,” on page 168.

The error log format field designated “Type” has the values 96 through 159 allocated for reporting power management event types. These event types and their Error Log Type field values are listed in Table 62 on page 199.

Requirements:

11–2. For the Power Management option: The Type field of the error log within the range 96 to 159 must be defined as specified in Table 62 on page 199.

Table 62. Defined Power Management Event Types

Event	Type Field Value	Semantics
Power Switch On	96	This event means that the user has requested a change in system power state. The precise target state is based on current policy settings. This event is presented only when the current system state is Suspend, Hibernate, or Off
Power Switch Off	97	This event means that the user has requested a change in system power state. The precise target state is based on current policy settings. This event is presented only when the current system state is PM enabled, PM disabled, or Standby. Platform designers should not rely upon software response to this event to deenergize power within an enclosure if safety is a concern.
Lid Open	98	A lid as used here is something physical which covers or protects the standard user input (keyboard or pad and/or pointing device) and output (display) and prevents their usage. The Lid Open event signifies that these user interface devices are now available.
Lid Close	99	User interface devices as defined above are now unavailable. The user has signalled that he no longer desires to interact with the system by closing the lid.
Sleep Button	100	User is requesting a system power state transition from the current state to a policy dependent lower power state.
Wake Button	101	A transition from the current system power state to a more responsive state is requested.
Battery Warning	102	The battery status sensor is signalling that remaining time to battery empty condition has dropped below that set via the *set-indicator* RTAS call.
Battery Critical	103	A battery energy level has been reached which is below a platform dependent level. Generally this means that there is not sufficient energy remaining to perform a Hibernate.

Personal Use Copy - Not for Reproduction

Table 62. Defined Power Management Event Types ***(Continued)***

Event	Type Field Value	Semantics
Switch to Battery	104	This event signifies that a dual AC/battery powered system has lost AC and has switched automatically to internal battery power.
Switch to AC	105	This signifies that a dual powered system has automatically switched its power source from its internal battery back to available AC power.
Keyboard or mouse activity	106	The user has pressed keys on the keyboard or pad, moved the mouse, or depressed a mouse click button. This event is generated only when the system is in the Suspend or Hibernate states
Enclosure Open	107	An enclosure is a physical barrier which constrains the variability of the machine configuration. This event signifies that the enclosure has been opened and that the system configuration may have been modified. If the opening of the enclosure involves any safety concern, this event should not be relied upon to ensure safety. Hardware interlocks should be employed in this case.
Enclosure Closed	108	This event signifies that the physical barrier as defined above is again in place. If there exists a safety concern, this event should not be used to energize power within an enclosure.
Ring Indicate	109	Either the ring indicate signal of an external serial port connector has been asserted or an internal modem has detected the ring pulses.
LAN Attention	110	A local area network connection has signalled indicating a request for a power state transition to a more responsive state.
Time Alarm	111	The time threshold set via the *set-time-for-power-on* RTAS call has been met or exceeded.
Configuration change	112	A significant configuration change such as attachment to a docking station has occurred. This event would be generated when the system power state is Suspend or Hibernate.
Reserved	113-159	Reserved for future use.

Software Implementation Note: The *power-type* property of the *power-management-events* node of the Open Firmware device tree provides a list of power management event types which are supported on a given platform.

Personal Use Copy - Not for Reproduction

11.2.1.3 Definition of Power Management Event Mask Argument

Requirements:

11–3. For the Power Management option: The Resume_mask of the *suspend* RTAS call, the Wakeup_mask of the *hibernate* RTAS call and the Power-on-mask of the *power-off* RTAS call must be defined by the 64 bit quantity generated by 0x8000000000000000 right shifted by (n-96), where n equals the Type field value specified in Table 62 on page 199.

11.2.2 Open Firmware Device Tree Properties

The *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10] specifies several new power related device tree properties. This section specifies the manner in which they are to be used to convey power management information from firmware to system software. This information includes a description of all the platform power domains and all supported abstract event sources.

11.2.2.1 Power Domain Dependency Tree

The power domain dependency tree is given via the Open Firmware property *power-domains-tree*. This property is specified by the platform designer and takes into account all the functional interdependencies between members of each domain as well as dependencies on the containing domain and members of other domains. A dependency domain tree must meet the following requirements.

Requirements:

11–4. For the Power Management option: The power domain dependency tree must have a single root node which is the root power domain.

11–5. For the Power Management option: Each node of the power domain dependency tree must have a single parent except the root domain which has no parent.

11–6. For the Power Management option: The effects of changing the power level of a domain must be limited to member devices and subdomains only.

Personal Use Copy - Not for Reproduction

11.2.2.2 Properties for Power Domain Control Points

Power domain control points require properties describing the power domains they control, the domains they are a member of, and properties describing the power levels they support.

Requirements:

11–7. For the Power Management option: System firmware must provide the Open Firmware *controls-power-domains* property for all domain control points. The value of this property is a list of domain numbers specifying the domains over which this device exercises control.

Hardware Implementation Note: A power domain controller is not a member of the domain it controls. A control point of the root power domain is not a member of any power domain. If *controls-power-domains* property list includes domain 0, this overrides the default domain membership rule of requirement 11–18.

11.2.2.3 Properties for Power-Manageable Devices

Power-manageable devices are those that support software controllable switching among different power states. These states are separate from domain power levels. The following gives the requirements for a platform which implements power-manageable devices.

Requirements:

11–8. For the Power Management option: System firmware must provide the Open Firmware *power-domains* property describing the membership of a power-manageable device in the platform power domains.

11–9. For the Power Management option: If system firmware provides device power state information, it must use the Open Firmware *device-power-states* property to describe the supported device states. This property is described in the *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10] and contains the following information for each state:

- Initial device power state

Personal Use Copy - Not for Reproduction

- Power consumption of the device per power source while idle in this state
- Power consumption of the device per power source while in use in this state
- Manufacturer's rated lifetime of the device while in this state

11–10. For the Power Management option: If system firmware provides device power state information, it must use the *device-state-transitions* property describing the allowable transitions between the state described in *device-power-states*. This property is described in the *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10] and contains the following information:

- A value specifying the source state of this transition
- A value specifying the destination state of this transition
- Power consumed per power source to make this transition
- Wall clock time required to make this transition
- Manufacturer's rated tolerance on the number of cumulative occurrences of this transition

11–11. For the Power Management option: If system firmware provides device power state information, each device in the Open Firmware device tree which consumes power provided by a power source must encode the *power-sources* property which is a list of power sources (indices into the *platform-power-sources* data structure).

Software Implementation Note: Each power-manageable device may provide the Open Firmware property *power-management-mapping* to define the set of device power states and the mapping from domain power levels to device power state for the device. This property value is device dependent and is defined in the Open Firmware binding for each device type.

11.2.2.4 Properties for Indirectly-Managed Devices

Devices that are not directly power manageable still require information about power consumption and domain membership.

Personal Use Copy - Not for Reproduction

Requirements:

11–12. For the Power Management option: System firmware must provide the Open Firmware *power-domains* property describing the power domains of which this device is a member.

11–13. For the Power Management option: If system firmware provides device power state information, it must use the Open Firmware *device-power-states* property to provide this information.

11–14. For the Power Management option: If system firmware provides device power state information, each device must encode the *power-sources* property which is a list of power sources (indices into the *platform-power-sources* data structure) from which the device draws power.

11.2.2.5 Properties for Power Sources

Requirements:

11–15. For the Power Management option: If system firmware provides device power state information, the *platform-power-sources* property of the Open Firmware device tree must specify all defined power sources of the platform.

Software Implementation Note: The *platform-power-sources* property provides the following information for each power source in the platform:

- peak-power in milliwatts
- average-power in milliwatts

11.2.2.6 Properties for Batteries

Requirements:

11–16. For the Power Management option: If system firmware of a battery-operated platform provides device power state information, it must also provide the *platform-battery-sources* property for its main batteries.

Software Implementation Note: The *platform-battery-sources* property provides the following information for each battery in the platform:

Personal Use Copy - Not for Reproduction

- Battery Identifier
- Rated Capacity in milliwatt-hours

11.2.2.7 Other Open Firmware Property Usage Rules

Requirements:

11–17. For the Power Management option: System firmware must provide the Open Firmware *power-domains* property for each physical (non-system) node of the device tree unless the device is a member of the root power domain.

11–18. For the Power Management option: If the root node of the device tree encodes the *power-domains* property, the membership of any physical node which does not encode this property must be assigned to the root power domain.

Architecture Note: The root power domain (power domain 0) is defined to contain all the physical devices of the system. Thus setting the power level of domain 0 to Off turns off the entire system. If this is the only domain which exists on a given platform, only the root node of the device tree is required to have the *power-domains* property.

11.2.2.8 Power Management NVRAM Partition

An NVRAM configuration partition named *pm-config* is defined in Section 8.4.4.2, “Power Management Configuration Data,” on page 146. The contents of this partition are defined in the *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10]. System Firmware is responsible for creating the nodes defined in this binding in the Open Firmware device tree.

11.2.3 General Hardware Requirements

The following sections present hardware requirements for systems which are designed to support power management.

Personal Use Copy - Not for Reproduction

11.2.3.1 Hardware Functionality

While hardware may be employed as an aid to system software in the monitoring of device activity, hardware should not effect device power state transitions without the direction of system software except in situations where it must act to protect platform hardware or the safety of persons and property.

Requirements:

11–19. For the Power Management option: Hardware/firmware must not change the power state of a device unless this change is not observable by system software and does not affect the operation of software or impact the predictability of device service time. This requirement is void if hardware/firmware must act to protect life or property.

11–20. For the Power Management option: Power domain power level transitions must not affect the correct operation of sibling or ancestral power domains.

11–21. For the Power Management option: The indirect effects of power level changes upon device within a given power domain must be limited to the following:

- Devices may present increased service time when the power level is Reduced.
- Devices may be rendered inoperative when the domain is placed in the Freeze or Off power levels.
- Devices may loose internal functional parameters when the domain in placed in the Off power state.

11.2.3.2 Power Management Controller

The power management controller is a mechanism which carries out the functions of sensing Wakeup events and optionally Resume events, controlling the power supply secondary voltages, and controlling any power state related indicators. The implementation of a power management controller is optional.

A power management controller is powered at any time the system is capable of being turned on. For battery-powered systems, this logic presents a constant drain on the battery. For AC-powered systems, power for this logic may be provided by an auxiliary secondary derived from the AC main.

Personal Use Copy - Not for Reproduction

After responsibility for platform power management policy is assumed by system software, software failures may impede some fundamental operations such as turning off system power when the system power switch is pressed. To prevent this, consideration may be given to providing a watchdog timer which would provide the command to the power supply to turn off the secondary voltages in the event that the power management software fails to complete its task.

11.2.3.3 System Power Switch and Power Supply

If the system power switch is used as a power management event source instead of hard-wiring it to the power supply, the function of the switch and power supply require modification.

Requirements:

11–22. For the Power Management option: To support the usage of the system power switch as a power management event source, the actuation of this switch must not interrupt the secondary voltages to the system.

11–23. For the Power Management option: In support of requirement 11–22, the secondary voltages from the power supply must be software controllable.

Hardware Implementation Note: An unavoidable result of these requirements for an AC-powered system is that the primary AC circuit is not broken when the system power switch is actuated. This may raise a safety concern. The recommended mechanism for safety interlock during service is the removal of the AC line cord. It should be noted that in the majority of implementations of these requirements the safety provisions will actually improve from industry standard practice due to the fact that the AC power cord will attach directly to the system power supply which is contained in a grounded metal enclosure. Since all secondary voltages are controlled by logic level signals, users and/or service personnel will not be exposed to any AC line voltages even when the system unit covers are removed.

Architecture Note: It is strongly recommended that systems choose to support the requirements of this section. Many operating systems cache file system data in system memory (NT and AIX, for example). Unexpected loss of power can lead to damage to the file systems in these operating systems. Implementation of these requirements can help reduce the frequency of occurrence of this problem.

Personal Use Copy - Not for Reproduction

11.2.3.4 Indicators

Requirements:

11–24. For the Power Management option: The platform must establish control of all indicators following a hardware reset until system software assumes control of power management policy via the *assume-power-management* RTAS call.

Software Implementation Note: Prior to the transfer of power management responsibility from the platform to system software, system software may not be guaranteed exclusive control of indicators even if it uses the *set-indicator* RTAS call.

11.2.3.5 Battery-Related RTAS Calls

The RTAS calls *set-indicator* and *get-sensor-state* may be utilized by an operating system to monitor battery charge status and general battery health in a battery operated system. This section gives guidance on the expected usage of these calls in managing a battery. Section 7.3.6.2, “Set-indicator,” on page 119 and Section 7.3.6.3, “Get-sensor-state,” on page 120 give detailed descriptions of these functions.

When called with the token value corresponding to the sensor Battery Capacity Percentage the calling system software is returned a value which represents the current percentage of remaining battery energy. System software may use this value to estimate remaining battery life based on the current or anticipated application workload.

Batteries of varying chemistry are used in these systems. Some of these batteries require conditioning cycles to eliminate charging “memory” effects which reduce the effective capacity of a battery. Intelligent batteries contain integrated circuitry which detects when these effects are degrading charging performance. The *get-sensor-state* function when called using the token corresponding to the Battery Condition Cycle State sensor provides a mechanism for system software to query an intelligent battery to determine if a conditioning cycle is required. This sensor should be polled on a periodic basis (period less than 5 minutes).

Figure 14 on page 209 gives a condition cycle state transition diagram which details how the calls are used to sense the need for a condition cycle, initiate a condition cycle, determine its progress, and handle error conditions. The following sections explain the conditions shown on the diagram edges which initiate state transitions.

Personal Use Copy - Not for Reproduction

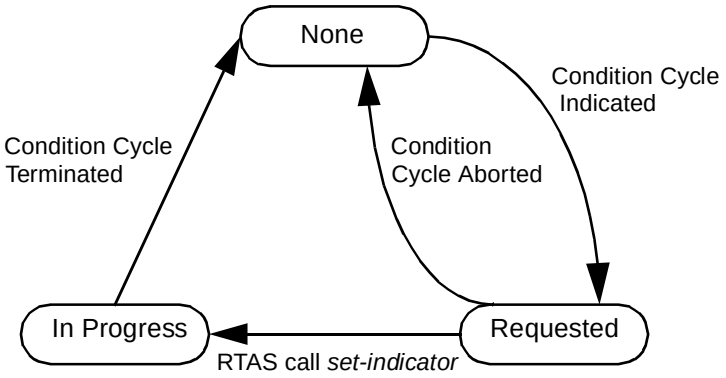


Figure 14. Battery Condition Cycle State Transition Diagram

11.2.3.5.1 Condition Cycle Indicated

The battery condition monitor circuitry has determined that a conditioning cycle would help restore the battery's charging capacity. At the same time the AC charger is connected which is a prerequisite to initiate a condition cycle. Factors which contribute to the need to initiate a condition cycle vary depending on battery technology.

System software can sense the need for a condition cycle by calling *get-sensor-state* with the sensor token for the Battery Condition Cycle State sensor. A return value of *requested* indicates that permission to initiate a condition cycle is being requested.

11.2.3.5.2 Condition Cycle Aborted

The following conditions will initiate a condition cycle abort:

- The AC charger is disconnected.
- The battery is removed from the system.
- An error is detected.
  The reasons for this error depend on the battery and/or the power module implementation.

11.2.3.5.3 RTAS call *set-indicator*

One possible response of system software to the receipt of a request to initiate a condition cycle is to indicate to the platform that a condition cycle is autho-

Personal Use Copy - Not for Reproduction

rized. The mechanism which RTAS provides to carry this out is the *set-indicator* function. When this is called with the indicator value set to Condition Cycle Request and the State value set to one (enable), the platform is directed to initiate a condition cycle.

11.2.3.5.4 Condition Cycle Terminated

There are many reasons for terminating the condition cycle. These are:

- The condition cycle has reached successful completion.
- The AC charger has been disconnected.
- The battery has been removed.
- The condition cycle has been terminated by system software.
  System software uses the *set-indicator* call with the indicator value set to Condition Cycle Request and the State set to 0 (disable).
- An error has been detected.

11.2.3.6 Other Implementation Recommendations

In a power-managed system the reliability of any electromechanical storage subsystems must be examined carefully. For example, it may be necessary to specify a hard disk which has been designed and tested to endure more than the number of start/stop cycles than would be required in a system without power management.

Consideration should be given to providing power domains subordinate to the root on single board systems to allow the removal of power from certain subsystems to reduce power consumption. Consideration should also be given to defining a separate power domain for add-in adapters to allow power to unused adapters to be turned off. This does, however, require self-identifying and automatically configurable adapters.

It is recommended that the system preserve security features during power management. For instance, if the system has a mechanism to check for a user password when powering on from an Off state, then this facility should also exist when waking up from a Hibernate state.

11.3 Operating System Requirements

To facilitate the following discussion of architectural aspects of power management software, please refer to Figure 15 on page 212 which gives a conceptual view of a power-management-enabled operating system.

Personal Use Copy - Not for Reproduction

At the highest level an application called the PM User Interface provides the user's interface to Power Management. The policy execution module is called the PM Executive. The PM Executive receives system power state transition requests from the PM User Interface and from PM-aware applications. It receives event messages from device drivers or via interrupt handlers. The PM Executive is responsible to initiate commands to the various PM extended device drivers to control the current power state of the various system resources. The PM Executive can be integrated in the operating system kernel, or exist as a kernel extension or privileged user-level application. It serves as an arbiter to resolve possible conflicts between system power state transition requests from multiple active PM-aware applications.

Although the power management policy may be implemented in a centralized executive, the power management mechanism should be implemented in a distributed manner via power management extensions to the individual device drivers. System software is shielded from selected hardware differences across system implementations by code called the power management Run-Time Abstraction Services.

Personal Use Copy - Not for Reproduction

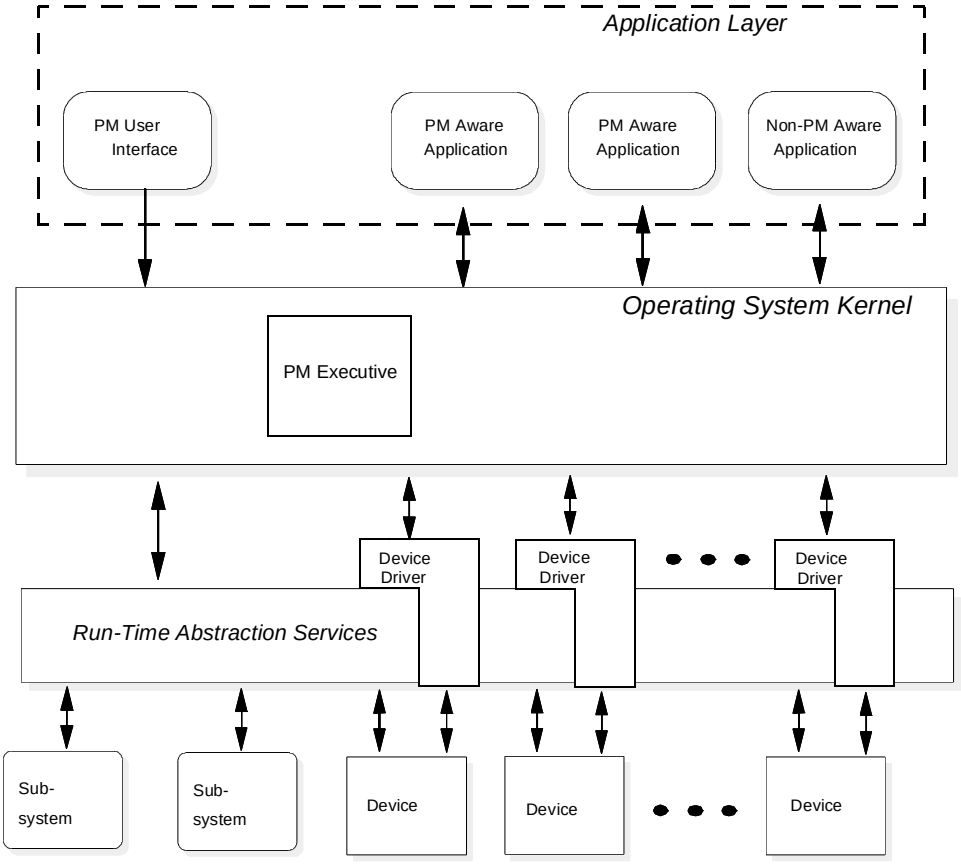


Figure 15. Example Power Management Software Structure

11.3.1 General Requirements

Although the implementation of power management is not required, requirement 11–25 must be followed in order to claim power management capability.

Requirements:

11–25. For the Power Management option: To claim power management capability an operating system must implement at least one of the following system power states:

Personal Use Copy - Not for Reproduction

- Power Management Enabled (and PM Disabled)
- Standby
- Suspend
- Hibernate

Software Implementation Note: It is recommended that operating systems which are targeted for portable computers implement all the defined system power states. An operating system targeted for AC-powered non-mobile computers should implement either Suspend or Hibernate. An operating system targeted for servers only should implement the Power Management Enabled system power state.

Personal Use Copy - Not for Reproduction

Blank page when cut - 214

The Symmetric Multiprocessor Option

12

The Common Hardware Reference Platform supports the implementation of symmetric multiprocessor (SMP) systems as an optional feature. This Chapter provides information concerning the design and programming of such systems.

SMP systems provide increased computational power in a broadly industry-accepted manner that is supported by numerous operating systems as well as middleware such as database systems. This added power is often applied to server systems. However, it is also appropriate for high-end client workstations, where it can provide necessary increased performance for tasks such as graphics, audio signal processing, speech recognition, etc. The ability to add processors to increase performance can also provide increased investment protection for users of client workstations, as well as server systems.

SMP systems differ from uniprocessors in a number of ways. These differences are not all covered in this chapter. Other chapters that cover SMP-related topics include:

- non-processor-related initialization and other requirements: Chapter 2, “System Requirements,” on page 7
- interrupts: Chapter 6, “Interrupt Controller,” on page 85
- error handling: Chapter 10, “Error and Event Notification,” on page 157

Many other general characteristics of SMPs—such as interprocessor communication, load/store ordering, and cache coherence—are defined in *The PowerPC Architecture* [1]. Requirements and recommendations for system

organization and time base synchronization are discussed here, along with SMP-specific aspects of the boot process.

SMP hardware platforms require SMP-specific operating system support. An operating system supporting only uniprocessor platforms will not automatically be usable on an SMP, even when an SMP platform has only a single processor installed; conversely, an SMP-supporting operating system will not automatically be usable on a uniprocessor. It is, however, a requirement that uniprocessor operating systems be able to run on one-processor SMPs, and that SMP-enabled operating systems also run on uniprocessors. See the next section.

12.1 SMP System Organization

The only multiprocessor defined by the Common Hardware Reference Platform architecture is an SMP. This is a computer system in which multiple processors equally share functional and timing access to and control over all other system components, including memory and I/O, as defined in the requirements below. Other multiprocessor organizations ("asymmetric multiprocessors," "attached processors," etc.) are not included in the Common Hardware Reference Platform architecture. These might, for example, include systems in which only one processor can perform I/O operations; or in which processors have private memory that is not accessible by other processors.

Requirements 12–4 through 12–7 below further require that all processors be of (nearly) equal speed, type, cache characteristics, etc. These requirements are included to avoid a combinatorial explosion of software testing. For example, it is manifestly impossible for all operating system, subsystem, and application software to be certified against all possible simultaneously installed combinations of processor type, speed, and cache size. This obviously does not stop particular specific combinations from being certified for particular specific software; but "shrink-wrapped" software cannot be expected to have been tested against all possible combinations.

Note that the above issues of symmetry refer only to the hardware platform. System or application software certainly may, should it be considered desirable for some purpose, dedicate processors to particular tasks or implement other functional asymmetries.

Requirements and implementation notes related to SMPs follow.

Personal Use Copy - Not for Reproduction

Requirements:

12–1. Operating systems that do not explicitly support the SMP option must support SMP-enabled hardware platforms, actively using only one processor.

12–2. **For the Symmetric Multiprocessor option:** SMP Operating Systems must support uniprocessor platforms.

12–3. **For the Symmetric Multiprocessor option:** The "SMP Extensions" sections of the Open Firmware binding for the CHRP architecture and for the PowerPC, and the SMP support section of the RTAS (see Section 7.3.11, "SMP Support," on page 137) must be implemented.

12–4. **For the Symmetric Multiprocessor option:** All processors in the configuration must have equal functional access and "quasi-equal" timing access to all of system memory, including other processors' caches, via cache coherence. "Quasi-equal" means that the time required for processors to access memory is sufficiently close to being equal that all software can ignore the difference without a noticeable negative impact on system performance; and no software is expected to profitably exploit the difference in timing.

12–5. **For the Symmetric Multiprocessor option:** All processors in the configuration must have equal functional and "quasi-equal" timing access to all I/O devices and adaptors. "Quasi-equal" is defined as in requirement 12–4 above, with I/O access replacing memory access for this case.

12–6. **For the Symmetric Multiprocessor option:** SMP Operating Systems must at least support SMPs with the same PVR contents and speed (for example, 133 MHz). The PVR contents includes both the PVN and the revision number.

12–7. **For the Symmetric Multiprocessor option:** All caches at the same hierarchical level must have the same Open Firmware properties.

12–8. Hardware for SMPs must provide a means of "freezing" and "thawing" the processor time base for use by RTAS. See Section 7.3.11, "SMP Support," on page 137. This is for purposes of clock synchronization at initialization.

Software Implementation Note: Requirement 12–1 has implications on the design of uniprocessor operating systems, particularly regarding the handling of interrupts. See the sections that follow, particularly Section 12.2.2, "Finding the Processor Configuration," on page 220.

Personal Use Copy - Not for Reproduction

Software Implementation Note: While requirement 12–6 does not require this, Operating Systems are encouraged to support processors of the same type but different PVR contents as long as their programming models are compatible.

Hardware Implementation Note: Particularly when used as servers, SMP systems make heavy demands on the I/O and memory subsystems. Therefore, it is strongly recommended that the I/O and memory subsystem of an SMP platform should either be expandable as additional processors are added, or else designed to handle the load of the maximum system configuration.

Software Implementation Note: Because of performance penalties associated with inter-processor synchronization, the weakest synchronization primitive that produces correct operation should be used. For example, *eieio* can often be used as part of a sequence that unlocks a data structure, rather than the higher-overhead but more general *sync* instruction.

Hardware Implementation Note: Defining an exact numeric threshold for "quasi-equal" is not feasible because it depends on the application, compiler, subsystem, and operating system software that the system is to run. It is highly likely that a wider range of timing differences can be absorbed in I/O access time than in memory access time. An illustrative example that is deliberately far from an upper bound: A 2% timing difference is certainly quasi-equal by this definition. While significantly larger timing differences are undoubtedly also quasi-equal, more conclusive statements must be the province of the operating system and other software.

12.2 An SMP Boot Process

Booting an SMP entails considerations not present when booting a uniprocessor. This section indicates those considerations by describing a way in which an SMP system can be booted. It does not pretend to describe "the" way to boot an SMP, since there are a wide variety of ways to do this, depending on engineering choices that can differ from platform to platform. To illustrate the possibilities, several variations on the SMP booting theme will be described after the initial description.

This section concentrates solely on SMP-related issues, and ignores a number of other initialization issues such as hibernation and suspension. See Section 2.1, "System Operation," on page 7 for a discussion of those other issues.

Personal Use Copy - Not for Reproduction

12.2.1 SMP-Safe Boot

The basic booting process described here is called “SMP-Safe” because it tolerates the presence of multiple processors, but does not exploit them. This process proceeds as follows:

1. At power on, one or more finite state machines (FSMs) built into the system hardware initialize each processor independently. FSMs also perform basic initialization of other system elements, such as the memory and interrupt controllers.

2. After the FSM initialization of each processor concludes, it begins execution at a location in ROM that the FSM has specified. This is the start of execution of the system firmware that eventually provides the Open Firmware interfaces to the operating system.

3. One of the first things that firmware does is establish one of the processors as the *master*: The *master* is a single processor which continues with the rest of the booting process; all the others are placed in a *stopped* state. A processor in this *stopped* state is out of the picture; it does nothing that affects the state of the system and will continue to be in that state until awakened by some outside force, such as an inter-processor interrupt (IPI).[1]

   One way to choose the *master* is to include a special register at a fixed address in the memory controller. That special register has the following properties:

   — The FSM initializing the memory controller sets this register’s contents to 0 (zero).

   — The first time that register is read, it returns the value 0 and then sets its own contents to non-zero. This is performed as an atomic operation; if two or more processors attempt to read the register at the same time, exactly one of them will get the 0 and the rest will get a non-zero value.

   — After the first attempt, all attempts to read that register’s contents return a non-zero value.

   The *master* is then picked by having all the processors read from that special register. Exactly one of them will receive a 0 and thereby become the *master*.

[1] Another characteristic of the *stopped* state, defined by the Open Firmware PowerPC binding, is that the processor remembers nothing of its prior life when placed in a *stopped* state; this distinguishes it from the *idle* state. That isn’t strictly necessary for this booting process; *idle* could have been used. However, since the non-*master* processor must be in the *stopped* state when the operating system is started, *stopped* might as well be used.

Personal Use Copy - Not for Reproduction

Note that the operation of choosing the *master* cannot be done using the PowerPC memory locking instructions, since at this point in the boot process the memory is not initialized. The advantage to using a register in the memory controller is that system bus serialization can be used to automatically provide the required atomicity.

4. The *master* chosen in step 3 then proceeds to do the remainder of the system initialization. This includes, for example, the remainder of Power-On Self Test, initialization of Open Firmware, discovery of devices and construction of the Open Firmware device tree, loading the operating system, starting it, and so on. Since one processor is performing all these functions, and the rest are in a state where they are not affecting anything, code that is at least very close to the uniprocessor code can be used for all of this (but see Section 12.2.2, “Finding the Processor Configuration,” on page 220 below).
5. The operating system begins execution on the single *master* processor. It uses the Open Firmware Client Interface Services to start each of the other processors, taking them out of the *stopped* state and setting them loose on the SMP operating system code.

This completes the example SMP boot process. Variations are discussed beginning at Section 12.2.3, “SMP-Efficient Boot,” on page 222. Before discussing those variations, an element of the system initialization not discussed above will be covered.

12.2.2 Finding the Processor Configuration

Unlike uniprocessor initialization, SMP initialization must also discover the number and identities of the processors installed in the system. By “identity” is meant the interrupt address of each processor as seen by the interrupt controller; without that information, a processor cannot reset interrupts directed at it. This identity is determined by board wiring: The processor attached to the “processor 0” wire from the interrupt controller has identity 0. For information about how this identity is used, see the “Symmetric Multiprocessor (SMP)” section of the Open Firmware system binding for the CHRP architecture.

12.2.2.1 The Very Special Register Technique

One way to find the processor configuration is to use a very special register in the memory controller, rather than a merely “special” one as described in the prior section. This very special register returns not just 0 or non-0, but an indicator of the identity of the processor. (This can be discovered by the memory

Personal Use Copy - Not for Reproduction

controller through logic that inspects the bus grant lines.) Note that returning this identity alone does not suffice to pick a *master* as defined above. Suppose, for example, a processor discovers it is number 0. Does that mean it should declare itself the *master*? That had better not be written into the code unless the hardware guarantees that the physical slot corresponding to number 0 is always occupied by a processor; this would be the case if, for example, processor 0 were on a motherboard and additional processors were on plug-in optional daughter boards. However, it is not unusual in SMPs for all processors to be on plug-in daughter boards; in that case, the first processor to access the register must receive some data to indicate that it was first, no matter what the identity of the processor. (Also, what should happen if the motherboard-mounted processor is out of commission?)

12.2.2.2 The IPI Technique

There is a way to discover the number and identities of SMP processors that doesn't use a very special register.

In this technique, Inter-Processor Interrupts (IPIs) are used. The *master* is chosen as was done in step 3 of Section 12.2.1, "SMP-Safe Boot," on page 219. The *master* then performs the remainder of its own self-test (if required). It also does at least the minimum initialization of memory and the interrupt controller, interrupt vectors, and so on, required to allow all possible processors to individually respond sanely to IPIs and perform the operations indicated below. The *master* also sets its own Processor ID Register (PIR) to a value that can designate no possible processor.

The *master* then performs the following operation for P=0 to 32 (which is the limit of the number of processors in the Open PIC specification):

1. Set a lock named L.
2. Set a memory value YOU_ARE to P.
3. IPI processor P (whoever that is; it might be yourself).
4. Wait for L to be reset. If L is not reset within a suitably chosen fixed time period, declare processor P to be missing or dead and resume the loop.
5. Since L was reset, somebody responded; declare processor P alive and resume the loop.

Corresponding to this, the IPI interrupt handler, executed by each processor in response to an IPI, does the following:

1. Loads the value in YOU_ARE into its PIR.

Personal Use Copy - Not for Reproduction

2. Resets lock L.
3. Returns from the interrupt, possibly back into the dormant state.

   If necessary, the non-dormant *master* can be differentiated from the other processors, which must go dormant, by noting back in step 1 what its initial PIR value was. This assumes, of course, that all the other processors' PIRs were set to a value different from the one set either by the FSM or by some other means.

At the conclusion of this process, every processor has its identity in its own PIR and the processors in the configuration have been identified.

12.2.3 SMP-Efficient Boot

The booting process as described so far tolerates the existence of multiple processors but does not attempt to exploit them. It is not impossible that the booting process can be sped up by actively using multiple processors simultaneously. In that case, the pick-a-*master* technique must still be used to perform sufficient initialization that other inter-processor coordination facilities—in-memory locks and IPIs—can be used. Once that is accomplished, normal parallel SMP programming techniques can be used within the initialization process itself.

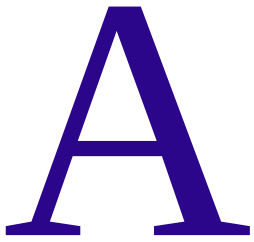
12.2.4 Use of a Service Processor

A system might contain a service processor that is distinct from the processors that form the SMP. If that service processor has suitably intimate access to and control over each of the SMP processors, it can perform the operations of choosing a *master* and discovering the SMP processor configuration. In that case, special or very special registers are unnecessary, as is the IPI technique.

In the rather unlikely event that the service processor is itself an SMP, recurse.

Personal Use Copy - Not for Reproduction

Operating System Information



Six operating systems are targeted for support of the CHRP architecture. Table 63 on page 223 lists these operating systems and gives sources of information for them. The second column in this table gives telephone contacts for obtaining additional information. The third column gives the location of any electronic form of documents describing the support of the CHRP architecture by these operating systems. System vendors must carefully consider the requirements which operating systems place on platforms. Platforms which conform to the CHRP architecture could be configured to support only some of the operating systems or with enough resources the platform could support any of the operating systems. Platforms may also supply hardware capabilities which are not taken advantage of by some or all of the operating systems.

Table 63. Sources of Operating Systems Information

Operating System	Contact for Information	Location of On-line Documentation
AIX	AIX product license and product information is available from M. Dane Dixon, AIX OEM Relations with IBM, at 1-512-838-2445. Application porting information is available from the IBM AIX Power Team General Information Line at 1-800-222-2363. AIX information is available from Motorola at 1-800-759-1107 extension PR.	

Table 63. Sources of Operating Systems Information ***(Continued)***

Operating System	Contact for Information	Location of On-line Documentation
Mac OS	Mac OS Licensing Department, Apple Computer, Inc., 1 Infinite Loop, MS 305-1DS, Cupertino, CA 95014, Telephone 1-408-974-4645.	
NetWare PowerPC Edition		
OS/2 Warp Connect PowerPC Edition	For information on the microkernel call 1-800-816-7493 or 1-407-443-6805. For information on OS/2 Warp Connect PowerPC Edition call 1-800-426-4579 extension 50 or e-mail ips@otirmg.mhs.compuserve.com.	
Solaris PowerPC Edition	Solaris product information is available from Abe Ellenberg, Manager of OEM Engineering with Sun, at 1-310-348-6057, FAX 1-310-348-8605, or e-mail abe.ellenberg@west.sun.com.	
Windows NT	For information pertaining to developing hardware abstractions for these platforms contact Motorola RISC software support at 1-512-891-2999. For information on Windows NT call your Microsoft representative.	

Personal Use Copy - Not for Reproduction

Requirements Summary

B

This appendix lists all the requirements of this architecture. The requirements are collected under the chapter heading. This appendix may be used for a quick reference list of all requirements for building a standard platform. The requirements list follows:

Chapter 2 System Requirements

2–1. I/O devices must adhere to the reset states given in Table 1 on page 10 when control of the system is passed from firmware to an operating system.

2–2. Prior to passing control to the operating system, firmware or hardware must initialize all registers not visible to the operating system to a state that is consistent with the system view represented by the OF device tree.

2–3. Hardware must provide a mechanism, callable by software, to hard reset all processors and I/O subsystems in order to facilitate the implementation of the RTAS *system-reboot* function.

2–4. **For the Power Management option:** Hardware must provide a software-controllable mechanism to reset the I/O subsystems without affecting the state of the processors or memory to facilitate implementation of the RTAS hibernate function.

2–5. Platforms must implement Open Firmware as defined in *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10].

2–6. Platforms must implement the Run-Time Abstraction Services (RTAS) as described in Chapter 7, “Run-Time Abstraction Services,” on page 91.

2–7. Operating systems must use Open Firmware and the RTAS functions to be compatible with all platforms.

2–8. Platforms must support operation in Big-Endian mode.

2–9. Platforms must support operation in Little-Endian mode.

2–10. Platforms must contain the minimum required components given in Table 2 on page 19.

2–11. Portable and personal CHRP operating systems must support all the following:

a. PS/2™ *and* ADB™ keyboard/mouse interfaces.

b. 16550-compatible *and* SCC serial ports.

c. SCSI *and* IDE hard disk interfaces.

2–12. An option, if implemented, must operate as specified in this architecture.

2–13. Extensions, if implemented, must come up passively, such that an operating system which does not use the extension will not be affected.

2–14. Options, if implemented, must come up passively or as otherwise specified in this architecture.

2–15. An extension, if implemented, must not contradict this architecture.

Chapter 3 System Address Map

3–1. All unavailable addresses in the Peripheral Memory and Peripheral I/O Spaces must be conveyed in the OF device tree.

a. A device type of *reserved* must be used to specify areas which are not to be used by software and not otherwise reported by OF.

Personal Use Copy - Not for Reproduction

b. Shadow aliases must be communicated as specified by the appropriate OF bus binding.

3–2. There must not be any address generated by the system which causes the system to hang.

3–3. Processor Load and Store operations must be routed and translated as shown in Table 5 on page 32.

3–4. DMA operations in the Memory Space of an I/O bus must be routed and translated as shown in Table 6 on page 32.

3–5. In addition to Table 6 on page 32, if the platform is designed to support System Memory configured at an address of 4 GB or above, then the Translation Control Entry (TCE) translation mechanism, described in Section 3.2.2, “Translation of 32-Bit DMA Addresses in 64-Bit Addressing Systems,” on page 38 must be implemented on all HBs. If the operating system enables the platform to access System Memory at or above 4 GB, then TCEs must be used to translate all DMA operations in the Memory Space of the I/O bus of the HB which use a 32-bit address.

3–6. An HB must not act as a target for operations in the I/O Space of an I/O bus.

3–7. The following are the System Control Area requirements:

a. Each platform must have exactly one System Control Area.

b. The System Control Area must not overlap with the System Memory Space(s), Peripheral Memory Space(s), or the Peripheral I/O Space(s) in the platform.

3–8. The following are the System Memory Space requirements:

a. Each platform must have at least one System Memory Space.

b. The System Memory Space(s) must not overlap with the Peripheral I/O Space(s), Peripheral Memory Space(s), the System Control Area, or other System Memory Space(s) in the platform.

c. The first System Memory Space must start at address 0 (BSM0 = 0), must be at least 16 MB before a second System Memory Space is added, and must be contiguous except that if the processor-hole is enabled, then there will be a hole from 640 KB to (768 KB - 1).

d. Each of the additional (optional) System Memory Space(s) must start on a 4 KB boundary.

Personal Use Copy - Not for Reproduction

e. Each of the additional (optional) System Memory Space(s) must be contiguous within itself.

f. There must be at most eight System Memory Spaces below BSCA and at most eight at or above 4 GB.

g. If multiple System Memory Spaces exist below BSCA, then they must not have any Peripheral Memory or Peripheral I/O Spaces interspersed between them.

3–9. The following are the Peripheral Memory Space requirements:

a. The Peripheral Memory Space(s) must not overlap with the System Memory Space(s), Peripheral I/O Space(s), the System Control Area, or other Peripheral Memory Space(s) in the platform.

b. When the OF passes control to the operating system, there must be no I/O device configured in the address range (TPM0 - 16 MB + 1) to TPM0 in the Peripheral Memory Space for PHB0, in order to assure that 16 MB of space is available for the initial memory alias spaces.

c. The size of each Peripheral Memory Space (TPMn - BPMn + 1) must be a power of two for sizes up to and including 256 MB, with the minimum size being 16 MB, and an integer multiple of 256 MB plus a power of two which is greater than or equal to 16 MB for sizes greater than 256 MB (for example, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB, 256 + 16 MB, 256 + 32 MB,..., 512 + 16 MB,...).

d. The boundary alignment for each Peripheral Memory Space must be an integer multiple of the size of the space up to and including 256 MB and must be an integer multiple of 256 MB for sizes greater than 256 MB.

e. There must be exactly one Peripheral Memory Space per HB.

f. The Peripheral Memory Space for every HB defined by the CHRP architecture must reside below BSCA.

3–10. The following are the Peripheral I/O Space requirements:

a. The Peripheral I/O Space(s) must not overlap with the System Memory Space(s), Peripheral Memory Space(s), the System Control Area, or other Peripheral I/O Space(s) in the platform.

b. The size of each Peripheral I/O Space (TIOn - BIOn + 1) must be a power of two with the minimum size being 8 MB (that is, sizes of 8 MB, 16 MB, 32 MB, 64 MB, and so on, are acceptable).

Personal Use Copy - Not for Reproduction

c. The boundary alignment for each Peripheral I/O Spaces must be an integer multiple of the size of the space.

d. There must be at most one Peripheral I/O Space per HB.

e. Peripheral I/O Spaces for all HBs defined by the CHRP architecture must reside below BSCA.

3–11. The following are the initial memory alias space requirements:

a. The peripheral-memory-alias and system-memory-alias spaces must be implemented on PHB0, and these initial memory alias spaces must have the capability to be enabled or disabled by the *set-initial-aliases* OF method for PHB0 node.

b. The initial state of the peripheral-memory-alias and system-memory-alias spaces, when control passes from OF to the operating system, must be enabled if there is any device on the I/O side of PHB0 which is configured in the 0 to (16 MB - 1) address range of the Memory Space of the I/O bus, and must be disabled otherwise.

3–12. The following are the compatibility hole requirements:

a. The initial state of the io-hole and the processor-hole, when control passes from OF to the operating system, must be disabled (if implemented).

b. If a platform implements the PC Emulation option, then the io-hole must be implemented. See Section 3.3, “PC Emulation Option,” on page 44 for more information.

c. Platforms for which VGA support is provided or which have an ISA bus must also implement the io-hole (see Table 2 on page 19 for the platform VGA requirements).

3–13. I/O devices which cannot be configured in the Peripheral Memory Space address range must be located on the I/O bus of PHB0, or on another I/O bus which is generated by a bridge attached to this bus, and must be configured in the 0 to (16 MB - 1) address range.

3–14. When the discontiguous I/O mode is enabled, Processor *Load* and *Store* addresses in the first 8 MB of Peripheral I/O Space experience the following translation. The high order seven bits of each 4 KB page offset are ignored. Thus, all page offsets wrap to the same 32 bytes within that page. Successive page numbers, starting at BIO, reference successive 32-byte blocks of Peripheral I/O Space, starting at address 0 (see Figure 7 on page 38).

Personal Use Copy - Not for Reproduction

3–15. When the discontiguous I/O mode is disabled (that is, in the contiguous mode), addresses are translated so that BIO to (BIO + 64 KB - 1) is sent through to the I/O Space of the I/O bus starting at address 0, and (BIO + 8 MB) to TIO is sent through to the I/O Space of the I/O bus starting at 8 MB (see Figure 7 on page 38).

3–16. The discontiguous I/O mode must be disabled when control is passed to the operating system.

3–17. **For 64-bit addressing option in HBs:** In platforms which are designed to support System Memory configured at an address of 4 GB or above, the TCE mechanism, described in Section 3.2.2, “Translation of 32-Bit DMA Addresses in 64-Bit Addressing Systems,” on page 38, must be implemented on all HBs. After a DMA operation is accepted by the HB and pre-translated as per Table 6 on page 32, if the 64-bit addressing option of the HB is enabled, the HB must use the TCE mechanism to translate the address when the I/O device presents a 32-bit address to the Memory Space of the HB.

3–18. **For 64-bit addressing option in HBs:** The bits of the TCE must be implemented as defined in Table 5 on page 32.

3–19. **For 64-bit addressing option in HBs:** Enough bits must be implemented in the TCE so that I/O DMA devices are able to access all System Memory addresses.

3–20. **For 64-bit addressing option in HBs:** TCEs must be stored as Big-Endian entities.

3–21. **For 64-bit addressing option in HBs:** When the 64-bit addressing option is enabled, an HB must not accept 32-bit accesses unless they would also be accepted under the requirements in Table 6 on page 32.

3–22. **For 64-bit addressing option in HBs:** If an HB accepts 64-bit addresses on DMA accesses (as reported by the existence of the 64-bit-dma property in the HB node of the OF device tree for that specific HB) and if the 64-bit addressing option of the HB is enabled, then that HB must not use TCEs to translate I/O bus Memory Space DMA addresses which support a full 64-bit address (for example, Dual Address Cycle (DAC) accesses on the PCI bus do not use the TCE translation mechanism in the PCI HB (PHB)).

3–23. **For 64-bit addressing option in HBs:** If an HB accepts 64-bit addresses on DMA accesses and if the 64-bit addressing option of the HB is enabled, that HB must translate 64-bit I/O bus Memory Space DMA ac-

Personal Use Copy - Not for Reproduction

cesses (for example, DAC PCI accesses for the PHB) in the upper 4 GB of the 64-bit Memory Space of the I/O bus to the 0 to (4 GB - 1) address range before passing the access to the host side of that HB (for example, for a PHB, if AD[63:32] (PCI notation) are equal to 0xFFFF FFFF, then the PHB must set AD[63:32] to 0), otherwise the HB must not translate 64-bit accesses (see Figure 8 on page 43).

3–24. **For 64-bit addressing option in HBs:** After translation of the address via requirements 3–17 or 3–23, above, an HB must use the translated address to access the system, unless that address would re-access the same HB (for example, is in the Peripheral Memory Space or Peripheral I/O Space of that HB), in which case the HB should generate an invalid address error. (See Chapter 10, “Error and Event Notification,” on page 157)

3–25. **For 64-bit addressing option in HBs:** TCEs must be located in System Memory or appear to software as though they are in System Memory, the memory must be a contiguous real address range, and the memory must be coherent.

3–26. **For 64-bit addressing option in HBs:** Each HB must provide the capability of having its own independent TCE table.

3–27. **For 64-bit addressing option in HBs:** Any non-recoverable error while an HB is accessing its TCE table must result in a TCE access error; the action to be taken by the HB being defined under the TCE access error in Chapter 10, “Error and Event Notification,” on page 157.

3–28. **For 64-bit addressing option in HBs:** In implementations which cache TCEs, if software issues a *Store* instruction to a TCE, then the hardware must perform the following steps: First, if any data associated with the page represented by that TCE is in an I/O bridge cache or buffer, the hardware must write the data, if modified, to System Memory. Secondly, it must invalidate the data in the cache. Finally, it must invalidate the TCE in the cache.

3–29. **For 64-bit addressing option in HBs:** Neither an I/O device nor an HB must ever modify a TCE.

3–30. **For 64-bit addressing option in HBs:** If the page mapping and control bits in the TCE are set to 0b00, the hardware must not change its state based on the values of the remaining bits of the TCE.

3–31. **For PC Emulation option:** The platform must implement the TEMR, and all of the following must be true:

Personal Use Copy - Not for Reproduction

a. The TEMR must contain the largest address in System Memory which is set aside for the image of PC memory.

b. The granularity of the contents of TEMR must be 1 MB.

c. The HB must not respond to I/O bus Memory Space DMA operations in the address range of (TEMR + 1) to the top of System Memory, or must respond and signal an invalid address error.

3–32. **For PC Emulation option:** PCI devices must not be configured between (TEMR + 1) and the top of System Memory which is below BSCA.

3–33. **For PC Emulation option:** The platform must implement the ERR, and all of the following must be true:

a. The value of *exception-relocation-size* OF property must be the amount of address space above 0xFFF00000 that is relocated down into System Memory and is a fixed value, as determined by the platform.

b. The granularity of *exception-relocation-size* must be 4 KB and the minimum size must be 12 KB.

c. The contents of the ERR must be the address of the base of the region in System Memory to which references to the interrupt/exception handling area (normally at the top of the 32-bit address space) are relocated.

d. The granularity of the ERR must be 1 MB.

3–34. **For PC Emulation option:** When the PC Emulation option is enabled, the peripheral-memory-alias space must be enabled, the system-memory-alias space must be disabled, and the io-hole must be enabled.

Chapter 4 Processor and Memory

4–1. Platforms must incorporate only processors which comply fully with the PowerPC architecture.

4–2. **For the Symmetric Multiprocessor option:** Multiprocessing platforms must use only processors which implement the processor identification register. See *PowerPC 604 RISC Microprocessor User's Manual* [6] for a definition of this register.

Personal Use Copy - Not for Reproduction

4–3. Platforms must incorporate only processors which implement *tlbie* and *tlbsync*, and *slbie* and *slbia* for 64-bit implementations.

4–4. Except where specifically noted otherwise in Section 4.1.4, “PowerPC Architecture Features Deserving Comment,” on page 53, platforms must support all functions specified by the PowerPC architecture.

4–5. Operating systems must use the properties of the *cpu* node of the OF device tree to determine the programming model of the processor implementation.

4–6. Operating systems must provide an execution path which uses the properties of the *cpu* node of the OF device. The PVN is available to the platform aware operating system for exceptional cases such as performance optimization and errata handling.

4–7. Operating systems must support both of the page table formats (32-bit and 64-bit) defined by the PowerPC architecture.

4–8. Processors which exhibit the *64-bit* property of the *cpu* node of the OF device tree must also implement the “bridge architecture,” an option in the PowerPC architecture. See the updates on the Internet associated with *The PowerPC Architecture* [1].

4–9. Platforms must restrict their choice of processors to those whose programming models may be described by modifications to that of the 604 by the properties defined for the *cpu* node of the OF device tree in *PowerPC processor binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [11] (for example, *64-bit* and *603-translation*).

4–10. Platform firmware must initialize the second and third pages above *Base* correctly for the processor in the platform, and in the correct endian mode, prior to giving control to the operating system.

4–11. Operating system and application software must not alter the state of the second and third pages above *Base*.

4–12. **For the Symmetric Multiprocessor option:** The 603 family of processors must not be used in a symmetric multiprocessing complex.

4–13. The 603 family of processors must be used only with system logic that does not reorder data transfers.

4–14. Operating systems must provide an alignment interrupt handler which correctly emulates the execution of any unaligned LE accesses that are

Personal Use Copy - Not for Reproduction

not implemented in hardware, except those which may be generated by *lwarx, ldarx, stwcx., stdcx., eciwx, ecowx,* and the multiple scalar operations (see 4.1.4.2, “Little-Endian Multiple Scalar Operations,” on page 54).

4–15. Software must not use multiple scalar operations in LE mode. Results of multiple scalar operations in LE mode are undefined.

4–16. Platforms must not use direct-store segments to implement interfaces defined by the CHRP architecture.

4–17. Operating systems must not depend on direct-store segment support when using interfaces specified by the CHRP architecture.

4–18. If the external control facility defined by the PowerPC architecture is supported, that support will be described as properties of the appropriate nodes (for example, memory controller and host bridge) of the OF device tree. See *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10] for more details.

4–19. Platforms must not use external control instructions as the sole interface to functions specified by the CHRP architecture.

4–20. Operating systems must not require the external control instructions when using interfaces specified by the CHRP architecture.

4–21. Platforms must provide at least 8 MB of System Memory. (Also see Chapter 3, “System Address Map,” on page 23 for other requirements which apply to memory within the first 16 MB of System Memory.)

4–22. Platforms must support the expansion of System Memory to 32 MB or more.

4–23. Platforms must provide the ability to maintain System Memory Coherence.

4–24. I/O transactions to System Memory through a Host Bridge must be made with coherence required.

4–25. Software must assume only the Weakly Consistent storage model.

4–26. Platforms must guarantee that a processor’s accesses to the same location are kept strongly ordered, unless the location is accessed by the pro-

Personal Use Copy - Not for Reproduction

cessor with WIM-bit aliasing which prohibits the assumption of hardware-maintained coherence.

4–27. Platforms must guarantee that, for a particular processor, accesses to the same location beyond the PowerPC coherency domain are performed in a strongly ordered manner, given that no I/O device is allowed to make concurrent accesses to the location. Note that it is not sufficient to merely produce an appearance of strong ordering with respect to the processor performing the accesses.

4–28. Platforms must guarantee that the storage ordering semantics of *eieio* are preserved for accesses leaving the PowerPC coherency domain at any given host bridge, all the way to destination I/O devices. That is, *Load* and *Store* accesses (in any combination) by a processor to an I/O device which are separated by an *eieio*, must complete in the same order that the processor issued those accesses.

4–29. Platforms must guarantee that accesses entering the PowerPC coherency domain that are from the same I/O device and to the same location are completed in a sequentially consistent manner.

4–30. Platforms must guarantee that multiple write operations entering the PowerPC coherency domain that are issued by the same I/O device are completed in a sequentially consistent manner.

4–31. The *Load And Reserve* and *Store Conditional* instructions must not be assumed to be supported for Write-Through storage.

4–32. Memory controller(s) must support the accessing of System Memory as defined in Chapter 3, “System Address Map,” on page 23.

4–33. Memory controller(s) must be fully initialized and set to full power mode prior to the transfer of control to the operating system. This requirement applies to normal boot and wakeup. (See Chapter 11, “Power Management,” on page 185 for an explanation of these terms.)

4–34. All allocations of System Memory space among memory controllers must have been done prior to the transfer of control to the operating system.

4–35. Memory controller(s) must maintain System Memory in Big-Endian byte order when the system is operating in Big-Endian mode (MSR-LE=0) and in Little-Endian byte order when the system is operating in Little-Endian mode (MSRLE=1). Whether System Memory is maintained in “True Little-Endian” or “PowerPC Little-Endian” form while

Personal Use Copy - Not for Reproduction

in Little-Endian mode is platform dependent, but must be transparent to software. See Section 2.3, “Bi-Endian Support,” on page 14 and Appendix C, “Bi-Endian Designs,” on page 265 for more information on Bi-Endian designs.

4–36. All caches must meet the requirements of the CHRP architecture coherency model, as stated in “Storage Ordering Models” on page 57.

4–37. **For the Symmetric Multiprocessor or Power Management option:** Each cache must be able to be completely flushed and invalidated via the RTAS *cache-control* function. (See Section 7.3.10.1, “Cache-control,” on page 136 for more information.)

4–38. **For the Power Management option:** External caches must be able to be placed in low power or disabled states via the RTAS *cache-control* function.

4–39. **For the Symmetric Multiprocessor or Power Management option:** To ensure compatibility with all CHRP implementations, operating systems must use the RTAS *cache-control* function to flush an entire cache rather than code directly to any specific system or processor implementation.

4–40. If a platform implementation elects not to cache portions of the address map in all external levels of the cache hierarchy, the result of not doing so must be transparent to the operation of the software, other than a difference in performance.

4–41. All caches must be fully initialized and enabled, and they must have accurate state bits prior to the transfer of control to the operating system.

4–42. If an in-line external cache is used, it must support one reservation as defined for the *Load And Reserve* and *Store Conditional* instructions.

4–43. **For the Symmetric Multiprocessor or Power Management option:** Platforms must implement their cache hierarchy such that all caches at a given level in the cache hierarchy can be flushed and disabled before any caches at the next level which may cache the same data are flushed and disabled (that is, L1 first, then L2, and so on).

4–44. **For the Symmetric Multiprocessor or Power Management option:** If a cache implements snarfing, then the cache must be capable of disabling the snarfing during flushing in order to implement the RTAS *cache-control* function in an atomic way.

4–45. Software must not depend on being able to change a cache from copy-back to write-through.

Personal Use Copy - Not for Reproduction

Chapter 5 I/O Bridges

5–1. All PHB implementations must be compliant with the *PCI Local Bus Specification*, revision level 2.1 [14].

5–2. All requirements defined in Chapter 3, “System Address Map,” on page 23 for HBs must be implemented by all PHBs in the platform.

5–3. HB0 must be a PHB (PHB0).

5–4. PHB implementations which include buffers or queues for DMA, *Load*, and *Store* operations must make sure that these are transparent to the software, with a few exceptions which are allowed by the PCI architecture, by the PowerPC architecture, and in Section 4.2.2.1, “Memory Coherence,” on page 57.

5–5. A *Load* or *Store* to either the Peripheral Memory Space or the Peripheral I/O Space of a PHB must never be passed to the I/O bus before a previous *Store* to either the Peripheral Memory Space or the Peripheral I/O Space of that same PHB (that is, multiple *Stores* to the I/O bus generated by one PHB must be kept in order and a *Load* must not pass a *Store*).

5–6. A *Load* to either the Peripheral Memory Space or the Peripheral I/O Space of a PHB must never be passed to the I/O bus before a previous *Load* to either the Peripheral Memory Space or the Peripheral I/O Space of that same PHB when both of those *Loads* go to the exact same address.

5–7. Data from a DMA read completion must be allowed to complete prior to *Load* or *Store* operation which was previously queued in the PHB, in order to prevent a possible deadlock.

5–8. *Load* data buffered in a PHB must not prevent a subsequent DMA write request from being posted into the PHB or from making progress through the PHB, in order to prevent a possible deadlock on the PCI bus.

5–9. A DMA write or read request from an I/O device to the processor side of the PHB must never be passed to the processor side of the PHB before the data from a previous I/O DMA write operation has been flushed to the processor side.

5–10. A previous DMA read request accepted by a PHB but not yet completed must not prevent a subsequent DMA write request from being posted into the PHB or from making progress through the PHB, in order to prevent a possible deadlock on the PCI bus.

Personal Use Copy - Not for Reproduction

5–11. All DMA write data (destined for the processor side of the PHB) in the PHB buffers must be flushed out of the PHB prior to delivering data from a *Load* operation which has come after the DMA write operations.

5–12. The hardware platform must be designed such that software must follow Table 9 on page 76 while running in the various processor modes (LE = 0 and LE = 1) and issuing *Load* and *Store* operations to various entities with various endianess, including doing any necessary address un-modification when running with LE = 1 (for platforms using processors implementing LE mode via address modification).

5–13. When performing DMA operations through a PHB while running with the processor mode of LE = 1, if the platform is implemented with the true LE format in System Memory, or while running with the processor mode of LE = 0 with BE format in System Memory, then the platform must not modify the data during the transfer process; the lowest addressed byte in System Memory being transferred to the lowest addressed byte on the PCI bus, the second byte in System Memory being transferred as the second byte on the PCI bus, and so on.

5–14. When performing DMA operations through a PHB while running with the processor mode of LE = 1, if the platform is implemented with the PowerPC LE format in System Memory, then the platform must transform the data during the transfer process to true LE format by reflecting the bytes within a doubleword (that is, the DMA is done as though the data is accessed a byte at a time, with the address modified by the same modification as used by the processor for 1-byte *Loads* and *Stores*, namely, exclusive-or the address with 0b111).

5–15. There must be exactly one Peripheral I/O Space per PHB.

5–16. If a PHB has an interrupt controller on the PCI side of the bridge which requires the PCI Interrupt Acknowledge Cycle generation, then that PHB must provide a 1-byte register which, when read by a processor using a 1-byte *Load* instruction, will generate a PCI Interrupt Acknowledge cycle on the PCI bus.

5–17. If a PHB defines any registers that are outside of the PCI Configuration space, then the address of those registers must be in the Peripheral Memory Space or Peripheral I/O Space for that PHB, or must be in the System Control Area.

Personal Use Copy - Not for Reproduction

5–18. All bridges must comply with the bus specification(s) of the buses to which they are attached.

5–19. Table 10 on page 79 details the minimum requirements that the platform must implement relative to I/O device access to the various address spaces.

5–20. PCI to PCI bridges used on the base platform must implement the architecture as specified in the *PCI to PCI Bridge Architecture Specification* [16].

5–21. There must be at most one ISA bus in a platform.

5–22. If there is an ISA bridge on a platform, it must be attached to the I/O side of HB0.

5–23. If there is an ISA bridge on a platform, a DMA controller must be available for the ISA DMA operations, and the DMA controller must be compatible with the register set defined in Chapter 9, “I/O Devices,” on page 151.

5–24. OF must program a PCI to ISA bridge such that all addresses for ISA DMA master operations get passed through the PCI to ISA bridge and do not get translated as they pass through the bridge.

5–25. If an ISA device is to participate in PCI to ISA peer to peer operations then the ISA device must be configured in the 640 KB to (1 MB - 1) address range (the io-hole) and the io-hole must be enabled.

5–26. PCI to ISA bridges must do a subtractive decode on the PCI side of the bridge in the PCI Memory Space from 0 to (16 MB - 1) and in the PCI I/O Space from 0 to (64 KB - 1) (that is, they must pass any PCI access in these address ranges to the ISA bus if the PCI cycles in these ranges are not first picked up by another PCI device).

5–27. A platform which supports Cardbus PC Card devices must also support 16-bit PC Card devices.

Chapter 6 Interrupt Controller

6–1. Platforms must implement interrupt controllers that are in register-level architectural compliance with *Open PIC Multiprocessor Interrupt Controller Register Interface Specification,* Revision 1.2 [8] including its PowerPC architecture appendix.

Personal Use Copy - Not for Reproduction

6–2. The Interrupt Acknowledge register implemented in the CHRP interrupt controller.

6–3. Platforms must make per-processor registers available in the “publicly accessible area” of the Open PIC register map.

6–4. All interrupt controller registers must be accessed via Caching-Inhibited and Guarded mapping.

6–5. The INIT signals defined in Open PIC must be connected to Soft Reset pins on PowerPC processors.

6–6. Interrupts must be disabled at the CHRP interrupt controller at the point of transfer of control to the operating system.

Chapter 7 Run-Time Abstraction Services

7–1. RTAS must be called in “real mode,” that is, all address translation must be disabled. Bits IR and DR of the MSR register must be zero.

7–2. RTAS must be called in privileged mode, and the PR bit of the MSR must be set to 0.

7–3. RTAS must be called with external interrupts disabled, and the EE bit of the MSR must be set to 0.

7–4. RTAS must be called with trace disabled, and the SE and BE bits of the MSR must be set to 0.

7–5. RTAS must be called with floating point disabled, and the FE0, FE1 and FP bits must be set to 0.

7–6. RTAS must be called with the SF and LE bits of the MSR set to the same values that were in effect at the time that RTAS was instantiated.

7–7. With the exception of the DR and RI bits, RTAS must not change the state of the machine by modifying the MSR.

7–8. If *rtas-call* is entered in a non-recoverable mode, indicated by having the RI bit of the MSR set equal to 0, then RTAS must not enter a recoverable mode by setting the RI bit to 1.

7–9. If called with RI of the MSR equal to 1, then RTAS must protect its own critical regions from recursion by setting the RI bit to 0 when in the critical regions.

Personal Use Copy - Not for Reproduction

7–10. Except as required by a specific function, RTAS must not modify the following operating environment registers: TB, DEC, SPRG0-SPRG3, EAR, DABR, SDR1, ASR, SR0-SR15, FPSCR, FPR0-FPR3, and any processor specific registers.

7–11. RTAS must preserve the following user mode registers: R1-R2, R13-R31, and CR.

7–12. RTAS must preserve the following operating environment registers: MSR, DAR, DSISR, IBAT0-IBAT3, and DBAT0-DBAT3.

7–13. Except as noted in requirement 7–19, the operating system must ensure that RTAS calls are not re-entered and are not simultaneously called from different processors in a multi-processor system.

7–14. Any RTAS access to device or I/O registers specified in this document must be made in such a way as to be transparent to the operating system.

7–15. Any device that is used to implement the RTAS abstracted services must have the property *used-by-rtas* in the Open Firmware Device Tree. However, if the device is only used by the *suspend*, *hibernate*, *power-off*, and *system-reboot* calls, which do not return directly to the operating system, the property should not be set. The *display-character* device must be marked *used-by-rtas* only if it is a specialized device used only for *display-character*.

7–16. Platforms must be designed such that accesses to devices that are marked *used-by-rtas* have no side effects on other registers in the system.

7–17. Any operating system access to devices specified as *used-by-rtas* must be made in such a way as to be transparent to RTAS.

7–18. RTAS must not generate any exceptions (for example, no alignment exceptions, page table walk exceptions, etc.).

7–19. The operating system machine check and soft reset handlers may call the RTAS services:

- *nvram-fetch*
- *nvram-store*
- *check-exception*
- *set-indicator*
- *system-reboot*

Personal Use Copy - Not for Reproduction

- *set-power-level(0,0)*
- *power-off*

7–20. The operating system must allocate *rtas-size* bytes of contiguous real memory as RTAS private data area. This memory must be aligned on a 4096 byte boundary and may not cross a 256 megabyte boundary.

7–21. The RTAS private data area must not be accessed by the operating system.

7–22. Except for the RTAS private data area, the argument buffer, System Memory pointed to by any reference parameter in the argument buffer, and any other System Memory areas explicitly permitted in this chapter, RTAS must not modify any System Memory. RTAS may, however, modify System Memory during error recovery provided that such modifications are transparent to the operating system.

7–23. If the operating system moves or otherwise alters the addresses assigned to ISA or PCI devices that are marked *used-by-rtas* after it has instantiated RTAS, then the operating system must restart RTAS by calling *restart-rtas* prior to making any further RTAS calls.

7–24. RTAS must execute in a timely manner and may not sleep in any fashion nor busy wait for more than a very short period of time.

7–25. The *instantiate-rtas* Open Firmware method must have the arguments specified in Table 11 on page 98.

7–26. RTAS must operate in the endian mode in effect at the time of the *instantiate-rtas* call.

7–27. The Open Firmware Device Tree must contain a device node named *rtas* which describes the RTAS implementation.

7–28. The RTAS device node must have a property for each implemented RTAS function in Table 12 on page 99. The value of this property is a token that is passed into the *rtas-call* function to indicate which RTAS function to invoke.

7–29. The Open Firmware properties listed in Table 13 on page 100 must be in the RTAS Device Tree node prior to booting the operating system.

7–30. All RTAS functions listed as “Required” in Table 12 on page 99 must be implemented in RTAS.

Personal Use Copy - Not for Reproduction

7–31. **For the Power Management option:** The functions listed as "Required in Power Managed Platforms" in Table 12 on page 99 must be implemented in RTAS.

7–32. **For the Symmetric Multiprocessor option:** The functions listed as "Required in SMP Platforms" in Table 12 on page 99 must be implemented in RTAS.

7–33. In order to make an RTAS call, the operating system must construct an argument call buffer aligned on an eight byte boundary in physically contiguous real memory as described by Table 14 on page 101.

7–34. If the system is a 32-bit system, or if the SF bit of the MSR was 0 when RTAS was instantiated, then all cells in the RTAS argument buffer must be 32-bit sign extended values that are aligned to 4 byte boundaries.

7–35. If the SF bit of the MSR was 1 when RTAS was instantiated, then all cells in the RTAS argument buffer must be 64-bit values that are aligned to 8 byte boundaries.

7–36. RTAS functions must be invoked by branching to the *rtas-call* address which is returned by the *instantiate-rtas* Open Firmware method (see Table 11 on page 98).

7–37. Register R3 must contain the argument buffer's real address when r*tas-call* is invoked.

7–38. Register R4 must contain the real address of the RTAS private data area when *rtas-call* is invoked (see requirement 7–20).

7–39. The Link Register must contain the return address when *rtas-call* is invoked.

7–40. The first output value of all the RTAS functions must be a status word which denotes the result of the call. The status word will take on one of the values in Table 15 on page 103. Non-negative values indicate success.

7–41. RTAS must implement a *restart-rtas* function that uses the argument call buffer defined by Table 16 on page 104.

7–42. If any device marked *used-by-rtas*, or a device used as the *rtas-display-device* if the OS will in the future call the *display-character* function, is moved or reconfigured, then the operating system must invoke *restart-rtas* before using any other RTAS function.

Personal Use Copy - Not for Reproduction

7–43. RTAS must update any necessary configuration state information based on the current configuration of the machine when *restart-rtas* is called.

7–44. RTAS must implement an *nvram-fetch* function that returns data from NVRAM using the argument call buffer defined by Table 17 on page 105.

7–45. RTAS must implement an *nvram-store* function that stores data in NVRAM using the argument call buffer defined by Table 18 on page 106.

7–46. If the *nvram-store* operation succeeded, the contents of NVRAM must have been updated to the user specified values. The contents of NVRAM are undefined if the RTAS call failed.

7–47. The caller of the *nvram-store* RTAS call must maintain the NVRAM partitions as specified in Chapter 8, “Non-Volatile Memory,” on page 141.

7–48. The date and time inputs and outputs to the RTAS time of day function calls are specified with the year as the actual value (for example, 1995), the month as a value in the range 1-12, the day as a value in the range 1-31, the hour as a value in the range 0-23, the minute as a value in the range 0-59, and the second as a value in the range 0-59. The date must also be a valid date according to common usage: the day range being restricted for certain months, month 2 having 29 days in leap years, etc.

7–49. Operating systems must account for local time, for daylight savings time when and where appropriate, and for leap seconds.

7–50. RTAS must account for leap years.

7–51. RTAS must implement a *get-time-of-day* call using the argument call buffer defined by Table 19 on page 107.

7–52. RTAS must read the current time and set the output values to the best resolution provided by the platform.

7–53. RTAS must implement a *set-time-of-day* call using the argument call buffer defined by Table 20 on page 108:

7–54. RTAS must set the time of day to the best resolution provided by the platform.

7–55. RTAS must implement the *set-time-for-power-on* call using the argument call buffer defined by Table 20 on page 108:

Personal Use Copy - Not for Reproduction

7–56. If the system is in a powered down state at the time scheduled by *set-time-for-power-on* (within the accuracy of the clock), then power must be reapplied and the system must go through its power on sequence.

7–57. RTAS must return the event generated by a particular interrupt or event source by either *check-exception* or *event-scan*, but not both.

7–58. RTAS must implement an *event-scan* call using the argument call buffer defined by Table 22 on page 111.

7–59. The *event-scan* call must fill in the error log with a single error log formatted as specified in Section 10.3.2, “RTAS Error/Event Return Format,” on page 168. If necessary, the data placed into the error log must be truncated to *length* bytes.

7–60. RTAS must only check for errors or events that are within the classes defined by the *Event mask*. *Event mask* is a bit mask of error and event classes. Refer to Table 50 on page 159 for the definition of the bit positions.

7–61. If *Critical* is non-zero, then RTAS must perform only those operations that are required for continued operation. No extended error information will be returned.

7–62. The *event-scan* call must return the first found error or event and clear that error or event so it is only reported once.

7–63. The operating system must continue to call *event-scan* while a status of “New Error Log returned” is returned.

7–64. The *event-scan* call must be made at least *rtas-event-scan-rate* times per minute for each error and event class and must have the *Critical* parameter equal to 0 for this periodic call.

7–65. RTAS must implement a *check-exception* call using the argument call buffer defined by Table 23 on page 113.

7–66. The operating system must provide the value specified in Table 24 on page 113 in the “Additional Information” parameter in the call to *check-exception*.

7–67. The *check-exception* call must fill in the error log with a single error log formatted as specified in Section 10.3.2, “RTAS Error/Event Return Format,” on page 168. The data in the error log must be truncated to *length* bytes.

Personal Use Copy - Not for Reproduction

7–68. If *Critical* is non-zero, then RTAS must perform only those operations that are required for continued operation. No extended error information will be returned.

7–69. The *check-exception* call must return the first found error or event and clear that error or event so it is only reported once.

7–70. RTAS must only check for errors or events that are within the classes defined by the *Event mask. Event mask* is a bit mask of error and event classes. Refer to Table 50 on page 159 for the definition of the bit positions.

7–71. For the RTAS PCI configuration space functions, the parameter *config_addr* must be a configuration address as specified by the *PCI Bus Binding to IEEE 1275 Standard for Boot (Initialization, Configuration) Firmware* [12].

7–72. All RTAS PCI Read/Write functions must follow the PCI Local Bus Specification, Revision 2.1 or later.

7–73. RTAS must implement a *read-pci-config* call using the argument call buffer defined by Table 25 on page 115.

7–74. The *read-pci-config* call must return the value from the configuration register which is located at *config_addr* in PCI configuration space.

7–75. The *read-pci-config* call must perform a 1-byte, 2-byte, or 4-byte configuration space read depending on the value of the *size* input argument.

7–76. The *config_addr* must be aligned to a 2-byte boundary if *size* is 2 and to a 4-byte boundary if *size* is 4.

7–77. The *read-pci-config* call of devices or functions which are not present must return *Success* with all ones as the output *value*.

7–78. RTAS must implement a *write-pci-config* call using the argument call buffer defined by Table 26 on page 116.

7–79. The *write-pci-config* call must store the value from the configuration register which is located at *config_addr* in PCI configuration space.

7–80. The *write-pci-config* call must perform a 1-byte, 2-byte, or 4-byte configuration space write depending on the value of the *size* input argument.

7–81. The *config_addr* must be aligned to a 2-byte boundary if *size* is 2 and to a 4-byte boundary if *size* is 4.

Personal Use Copy - Not for Reproduction

7–82. The *write-pci-config* call of devices or functions which aren't present must be ignored. A status of *Success* must be returned.

7–83. RTAS must implement a *display-character* call using the argument call buffer defined by Table 27 on page 118 to place a character on the output device.

7–84. The operating system must serialize all calls to *display-character* with any other use of the *rtas-display-device*.

7–85. If a physical output device is used for the output of the RTAS *display-character* call, then it must have at least one line and 4 characters.

7–86. Certain ASCII control characters must have their normal meanings with respect to position on output devices which are capable of cursor positioning. In particular, ^M (0x0D) must position the cursor at column 0 in the current line, and ^J (0x0A) must move the cursor to the next line, scrolling old data off the top of the screen.

7–87. RTAS must not output characters to the *rtas-display-device* except for explicit calls to the *display-character* function.

7–88. RTAS must implement a *set-indicator* call which sets the value of the indicator of type *Indicator* and index *Indicator-index* using the argument call buffer defined by Table 28 on page 119 and indicator types defined by Table 29 on page 120.

7–89. RTAS must implement a *get-sensor-state* call which reads the value of the sensor of type *Sensor* which has index *Sensor-index* using the argument call buffer defined by Table 30 on page 121 and the sensor types defined by Table 31 on page 121.

7–90. **For the Power Management option:** The operating system must control the power management features of the processors and any attached power-manageable devices for which it has device drivers.

7–91. **For the Power Management option:** RTAS must implement a *set-power-level* call which changes the power level setting of a power domain. This call must be implemented using the argument call buffer defined by Table 32 on page 124.

7–92. **For the Power Management option:** *Power_domain* must be a power domain identified in the Open Firmware Device Tree.

Personal Use Copy - Not for Reproduction

7–93. **For the Power Management option:** *Level* must be a power level as specified by Section 11.2.1.1, “Definition of Domain Power Levels,” on page 198.

7–94. **For the Power Management option:** The *set-power-level* call must set the level of the specified domain to the power level specified by *level* or to the next higher implemented level.

7–95. **For the Power Management option:** The *set-power-level* call must return the power level actually set in the *Actual_level* output parameter.

7–96. **For the Power Management option:** RTAS must implement a *get-power-level* call which returns the current setting of a power domain. This call must be implemented using the argument call buffer defined by Table 33 on page 125.

7–97. **For the Power Management option:** *Power_domain* must be a power domain identified in the Open Firmware Device Tree.

7–98. **For the Power Management option:** RTAS must implement an *assume-power-management* call which transfers control of the power management policy from the platform to system software, using the argument call buffer specified in Table 34 on page 126.

7–99. **For the Power Management option:** RTAS must implement a *relinquish-power-management* call which transfers control of the power management policy from system software to the platform, using the argument call buffer specified in Table 34 on page 126.

7–100. **For the Power Management option:** If the power management policy is under control of the operating system, the *relinquish-power-management* call must be performed prior to the completion of the transition into the Suspend, Hibernate, or Off system power states.

7–101. **If software controlled power-off hardware is present**: The *power-off* function must turn off power to the platform, using the argument call buffer described in Table 36 on page 128.

7–102. **If software controlled power-off hardware is present**: *Power_on_mask*, which is passed in two parts to permit a possible 64 events even on 32-bit implementations, must be a bit mask of power management events, refer to requirement 11–3. If a bit in the *resume_mask* is set to 1, then the hardware should enable the corresponding hardware power-on mechanism.

Personal Use Copy - Not for Reproduction

7–103. The RTAS *suspend* function must implement the power-saving Suspend state using the argument call buffer defined in Table 37 on page 129.

7–104. The RTAS *suspend* call must be made only from the processor identified by *processor_number*.

7–105. *Resume_mask,* which is passed in two parts to permit a possible 64 events even on 32-bit implementations, must be a bit mask of power management events, refer to requirement 11–3. If a bit in the *resume_mask* is set to 1, then the hardware should enable the corresponding hardware wakeup mechanism.

7–106. *Resume_event* must be a power management event that caused the resume.

7–107. In an SMP system, the operating system must have invoked the RTAS *stop-self* function on all other processors prior to invoking *suspend.*

7–108. All elements of the I/O sub-system must be in a quiescent state at the time of the call: they must not be transferring data to or from memory, nor be able to cause an interrupt to any processor.

7–109. Upon return from the RTAS *suspend* call, the state of the memory locations, exclusive of the first 256 bytes of real memory and the RTAS private data area, must be in the same state they were in at the time of the call to *suspend***.**

7–110. Upon return from *suspend,* execution must be on the processor indicated by *processor_number*.

7–111. Upon return from *suspend,* all processors except the processor identified by *processor_number* must be in the *stopped* state.

7–112. Upon return from *suspend,* the firmware must restore the registers and devices which are reserved for firmware to the same state as before the *suspend*.

7–113. Upon return from *suspend,* the firmware must place all elements of the I/O system into a safe state.

7–114. The return from *suspend* must restore the registers listed in requirement 7–10 to the same state that existed prior to the *suspend* call.

7–115. The RTAS *hibernate* call must be implemented using the argument call buffer described by Table 38 on page 132.

Personal Use Copy - Not for Reproduction

7–116. The *wakeup_mask*, which is passed in two parts to permit a possible 64 events even on 32-bit implementations, must be a bit mask of power management events, refer to requirement 11–3. If a bit in the wakeup_mask is set to 1, then the hardware should enable the corresponding hardware wakeup mechanism.

7–117. The area of memory described by the *hibernate block_list* must be written to the device(s) as indicated.

7–118. The hibernate *block_list* must start with a list length in bytes and then must have quadruples of real address, length of memory block, device id, and block number as shown in Table 39 on page 132. Additional requirements on the block list are:

a. The block list must be a sequence of cells as defined in requirements 7–34 and 7–35.

b. Block numbers are in 512 byte units.

7–119. The *hibernate block_list* must be contained in system memory below 4 GB.

7–120. The *Device* fields of the *hibernate block_list* must be real pointers to System Memory below 4 GB that point to Open Firmware Path Names.

7–121. A device specified in the *hibernate block_list* must correspond to an device in the Open Firmware Device Tree that has *write* and *read* methods, and has a device type of *block*.

7–122. The memory areas defined by the *hibernate block_list* must only include System Memory outside that reserved for firmware (both the RTAS data area and Open Firmware's memory defined by *real-base* and *real-size*).

7–123. The memory areas defined by the *hibernate block_list* must only include System Memory below 4 GB.

7–124. Execution of the *hibernate* call must cause the System Memory areas described in the *block_list* to be saved on disk on the device(s) specified starting at the specified block numbers.

7–125. After system memory is saved, *hibernate* must place the system into its lowest power state.

7–126. In an SMP system, all other processors must be in the *stopped* state before invoking *hibernate*. This is the responsibility of the operating system.

Personal Use Copy - Not for Reproduction

7–127. Prior to making the *hibernate* call, the operating system must put all I/O devices into a quiescent state: they must not be transferring data to or from memory, nor be able to cause an interrupt to any processor.

7–128. Open Firmware must deterministically initialize a platform. The Open Firmware device tree and device addresses assigned on a given platform must be the same on successive reboots and hibernate wakeups if the platform hardware configuration has not changed.

7–129. RTAS must implement a *system-reboot* call which resets all processors and all attached devices. After reset, the system must be booted with the current settings of the System Environment Variables (refer to Section 8.4.3, “System (0x70),” on page 145 for more information).

7–130. The RTAS *system-reboot* call must be implemented using the argument call buffer defined by Table 40 on page 135.

7–131. **For the Symmetric Multiprocessor or Power Management option:** RTAS must implement a *cache-control* call to place the cache into a new state, using the argument call buffer defined by Table 41 on page 136.

7–132. **For the Symmetric Multiprocessor or Power Management option:** When entering the new state indicated by the *How* parameter, the actions specified in Table 42 on page 137 must be performed on the caches.

7–133. **For the Symmetric Multiprocessor or Power Management option:** The RTAS *cache-control* operations performed on the specified cache must appear to be a single atomic operation as seen from the operating system. A return status of success implies “committed,” that the operation is complete, and that any dirty data is safely out of the cache. These operations must not violate the cache coherency of the caches.

7–134. **For the Symmetric Multiprocessor option:** RTAS must implement a *freeze-time-base* call which *freezes*, or keeps from changing, the time base register on all processors. This call must be implemented using the argument call buffer defined by Table 43 on page 138.

7–135. **For the Symmetric Multiprocessor option:** The *freeze-time-base* operation must simultaneously affect every processor of an SMP system.

7–136. **For the Symmetric Multiprocessor option:** RTAS must implement a *thaw-time-base* call which *thaws*, or permits the change of, the time base register on all processors. This call must be implemented using the argument call buffer defined by Table 44 on page 138.

Personal Use Copy - Not for Reproduction

7–137. **For the Symmetric Multiprocessor option:** The *thaw-time-base* operation must simultaneously affect every processor of an SMP system.

7–138. **For the Symmetric Multiprocessor option:** RTAS must implement a *stop-self* call which places the calling processor in the RTAS *stopped* state. This call must be implemented using the argument call buffer defined by Table 45 on page 139.

7–139. **For the Symmetric Multiprocessor option:** RTAS must insure that a processor in the RTAS *stopped* state will not check stop or otherwise fail if a machine check or soft reset exception occurs. Processors in this state will receive the exception, but must perform a null action and remain in the RTAS *stopped* state.

7–140. **For the Symmetric Multiprocessor option:** RTAS must implement a *start-cpu* call which removes the processor specified by the *CPU_id* parameter from the RTAS *stopped* state. This call must be implemented using the argument call buffer defined by Table 46 on page 140.

7–141. **For the Symmetric Multiprocessor option:** The processor specified by the *CPU_id* parameter must be in the RTAS *stopped* state entered because of a prior call by that processor to the *stop-self* primitive.

7–142. **For the Symmetric Multiprocessor option:** When a processor exits the RTAS *stopped* state, it must begin execution in real mode, at the real location indicated by the *Start_location* parameter, with register R3 set to the value of parameter *Register_R3_contents*, in the endian mode of the processor executing the *start-cpu* primitive. All other register contents are indeterminate.

Chapter 8 Non-Volatile Memory

8–1. Platforms must implement at least 8 KB of Non-Volatile Memory. This is sufficient for a system with a single operating system installed. Allow 1 KB for each additional operating system installed.

8–2. Non-Volatile Memory must maintain its contents in the absence of system power.

8–3. Firmware must reinitialize NVRAM to a bootable state if NVRAM data corruption is detected.

8–4. Operating systems must reinitialize their own NVRAM partitions if NVRAM data corruption is detected. Operating systems may create free

Personal Use Copy - Not for Reproduction

space from the first corrupted partition header to the end of NVRAM and utilize this area to initialize their partitions.

8–5. NVRAM partitions must be structured as shown in Table 47 on page 143.

8–6. All NVRAM space must be accounted for by partitions.

8–7. The system NVRAM must include a 0x50 partition with the name of-config to store Open Firmware Configuration Variables.

8–8. VPD must be stored in the 0x52 partition using the format defined in the *PCI Local Bus Specification*, Revision 2.1, section 6.4 [14].

8–9. Every system NVRAM must contain a System partition with the partition name = *common*.

8–10. Data in the common partition must be stored as null-terminated strings of the form: *name=<string>* and be terminated with a string of at least two null characters.

8–11. All *names* used in the *common* partition must be unique.

8–12. Device and file specifications used in the *common* partition must follow IEEE Std 1275 nomenclature conventions.

8–13. System NVRAM must include a 0x71 partition with the name *isa-config* to store configuration data for all ISA devices.

8–14. The *isa-config* data area must use the resource format given in the ISA/EISA/ISA-PnP binding to IEEE 1275, *IEEE Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices* [9].

8–15. During boot, firmware must use the ISA configuration data stored in the *isa-config* NVRAM partition to generate the ISA portion of the OF device tree.

8–16. **For the Power Management option:** NVRAM must include a 0x71 partition with the name *pm-config* to store power management configuration data.

8–17. **For the Power Management option:** The *pm-config* data area must use the format given in the *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10].

Personal Use Copy - Not for Reproduction

8–18. If an operating system implements an NVRAM error log partition for RTAS,

a. the partition name must be *rtas-err-log*.

b. the error log format must be as given in Table 54 on page 176.

c. error log entries must be filled by the operating system in a manner that will make the most recent log visible.

8–19. If firmware implements an NVRAM error log partition for POST,

a. the partition name must be *post-err-log*.

b. the error log format must be as given in Table 54 on page 176.

c. error log entries must be filled by the firmware in a manner that will make the most recent log visible. If multiple entries are provided, they must be filled in a manner that will make the first and last error occurrences visible to the OS.

8–20. **For the Multi-Boot option:** The multi-boot partition must be named *multi-boot*.

8–21. **For the Multi-Boot option:** Each participating operating system must be represented in the *multi-boot* partition as a 4-tuple of null-terminated strings of the form *name*=<*string*> as represented in Table 49 on page 148.

8–22. **For the Multi-Boot option:** The *multi-boot* partition must be terminated with a string of at least two null characters.

8–23. Device and file specifications used in the *multi-boot* partition must follow IEEE 1275, *IEEE Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices* [9] nomenclature conventions.

8–24. All unused NVRAM space must be included in a signature = 0x7F Free Space partition.

8–25. All Free Space partitions must have the **name** field set to 0x7...77.

8–26. A system software component must not move or delete any NVRAM partition unless it is in the ownership class of that partition (see Table 48 on page 144). There are two exceptions to this requirement:

a. Open Firmware may, with the appropriate backup precautions, modify any area of NVRAM in the interest of space management and/or

Personal Use Copy - Not for Reproduction

maintaining NVRAM data integrity. Firmware must maintain the required 1 KB of contiguous NVRAM space for each installed operating system following any such modifications.

b. Upon detection of a corrupted partition, the operating system may create free space beginning with the header of the corrupted partition through the end of the NVRAM space, and use this space to reinitialize its partitions.

8–27. The NVRAM partition header checksum must be calculated as shown in Table 47 on page 143.

Chapter 9 I/O Devices

9–1. PCI devices must comply with the *PCI Local Bus Specification*, Revision 2.1 [14].

9–2. PCI devices, excepting bridges, must not depend on the PCI LOCK# signal for correct operation nor require any other PCI device to assert LOCK# for correct operation.

9–3. PCI expansion ROMs must have a ROM image with a code type of 1 for Open Firmware as provided in the *PCI Local Bus Specification*, Revision 2.1 [14]. This ROM image must abide by the ROM image format for Open Firmware as documented in the *PCI Bus Binding to IEEE 1275 Standard for Boot (Initialization, Configuration) Firmware* [12]

9–4. All PCI devices must use the Open PIC interrupt controller. They must not use the legacy (8259 derived) interrupt controller which is reserved for ISA. The programming model for the legacy interrupt controller is defined in *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

9–5. PCI devices that do not share Peripheral Memory Space and Peripheral I/O Space of the same PHB must not share the same Open PIC interrupt source.

9–6. When PCI-to-ISA bridges with embedded ISA devices are provided in a PCI part, the part must provide the capability to attach to the legacy interrupt controller defined in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

Personal Use Copy - Not for Reproduction

9–7. PCI devices must implement the programming model (register level definition, interrupts, and so forth) for those devices which are in the minimum system requirements and are specified in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

9–8. The following PCI devices, when used, must adhere to the programming model provided in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20]:

- MESH SCSI Controller
- ADB (Apple Desktop Bus) Controller
- SCC (Serial Communications Controller)
- Bus master IDE Controller
- VGA Compatible Graphics Controller

9–9. If a PCI device is defined in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20] and a specific implementation of that device has additional registers beyond those defined as used by programming, then those additional registers must be reported in the Open Firmware device tree as reserved (see Section 3.1, “Address Areas,” on page 23).

9–10. PCI-to-PCI bridges must be compliant with the *PCI to PCI Bridge Architecture Specification* [16]

9–11. Firmware must initialize PCI-to-PCI bridges to work in CHRP systems. See *PCI Bus Binding to IEEE 1275 Standard for Boot (Initialization, Configuration) Firmware* [12].

9–12. Portable and personal platforms must provide Bi-Endian graphics aperture support as described in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

9–13. Plug-in graphics controllers for portable and personal platforms must provide graphics mode sets in the Open Firmware PCI expansion ROM image in accordance with the *PCI Bus Binding to IEEE 1275 Standard for Boot (Initialization, Configuration) Firmware* [12].

9–14. The following ISA devices, when used, must adhere to the programming model provided in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

Personal Use Copy - Not for Reproduction

- Legacy Interrupt Controller
- Serial Port Controller
- Parallel Port Controller
- Floppy Disk Controller
- DMA Controller
- Audio Controller
- Keyboard and Mouse Controller

9–15. When an ISA device is included in a PCI part, is required by the minimum requirements, and is among those specified in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20], it must completely retain the programming model.

9–16. The system tone and the system audio must be able to be used concurrently.

9–17. ISA devices included in a PCI part must route their interrupt signals to the legacy interrupt controller defined in the *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference* [20].

Chapter 10 Error and Event Notification

10–1. Platform-specific error and event interrupts that a platform provider wants the operating system to enable must be listed in the *open-pic-interrupt* property of the appropriate Open Firmware event class node, as described in Table 50 on page 159.

10–2. To enable platform-specific error and event interrupt notification, operating systems must find the list of interrupts (described in Table 50 on page 159) for each error and event class in the Open Firmware device tree, and enable them.

10–3. Operating systems must have interrupt handlers for the enabled interrupts described in requirement 10–2, which call the RTAS *check-exception* function to determine the cause of the interrupt.

10–4. Platforms which support error and event reporting must provide information to the OS via the RTAS *event-scan* and *check-exception* functions, using the reporting format described in Table 53 on page 172.

Personal Use Copy - Not for Reproduction

10–5. Optional Extended Error Log information, if returned by the *event-scan* or *check-exception* functions, must be in the reporting format described in Table 54 on page 176.

10–6. To provide control over performance, the RTAS event reporting functions must not perform any event data gathering for classes not selected in the event class mask parameter, nor any extended data gathering if the time critical parameter is non-zero or the log buffer length parameter does not allow for an extended error log.

10–7. To prevent the loss of any event notifications, the RTAS event reporting functions must be written to gather and process error and event data without destroying the state information of events other than the one being processed.

10–8. Any interrupts or interrupt controls used for error and event notification must not be shared between error and event classes, or with any other types of interrupt mechanisms. This allows the operating system to partition its interrupt handling and prevents blocking of one class of interrupt by the processing of another.

10–9. If a platform chooses to report multiple event or error sources through a single interrupt, it must ensure that the interrupt remains asserted or is re-asserted until *check-exception* has been used to process all outstanding errors or events for that interrupt.

10–10. Operating systems must set MSRME=1 prior to the occurrence of a machine check interrupt in order to enable machine check processing via the *check-exception* RTAS function.

10–11. For hardware-detected errors, platforms must generate error indications as described in Table 51 on page 162, unless the error can be handled through a less severe platform-specific interrupt, or the nature of the error forces a check stop condition.

10–12. Platforms which detect and report the errors described in Table 51 on page 162 must provide information to the OS via the RTAS *check-exception* function, using the reporting format described in Table 53 on page 172.

10–13. To prevent error propagation and allow for synchronization of error handling, all processors in a multi-processor system must receive any machine check interrupt signalled via the external machine check interrupt pin.

Personal Use Copy - Not for Reproduction

10–14. If the platform supports Environmental and Power Warnings by including a EPOW device tree entry, then the platform must support the EPOW sensor for the *get-sensor-state* RTAS function.

10–15. The EPOW sensor, if provided, must contain the EPOW action code (defined in Table 52 on page 166) in the least significant 4 bits. In cases where multiple EPOW actions are required, the action code with the highest numerical value (where 0 is lowest and 7 is highest) must be presented to the operating system. The platform may implement any subset of these action codes, but must operate as described in Table 52 for those it does implement.

10–16. To ensure adequate response time, platforms which implement the EPOW_MAIN_ENCLOSURE or EPOW_POWER_OFF action codes must do so via interrupt and *check-exception* notification, rather than by *event-scan* notification.

10–17. For interrupt-driven EPOW events, the platform must ensure that an EPOW interrupt is not lost in the case where a numerically higher-priority EPOW event occurs between the time when check-exception gathers the sensor value and when it resets the interrupt.

Chapter 11 Power Management

11–1. **For the Power Management option:** The *set-power-level* and *get-power-level* RTAS calls must respectively accept as input and return the values of the argument *Level* as defined in Table 61 on page 198.

11–2. **For the Power Management option:** The Type field of the error log within the range 96 to 159 must be defined as specified in Table 62 on page 199.

11–3. **For the Power Management option:** The Resume_mask of the *suspend* RTAS call, the Wakeup_mask of the *hibernate* RTAS call and the Power-on-mask of the *power-off* RTAS call must be defined by the 64 bit quantity generated by 0x8000000000000000 right shifted by (n-96), where n equals the Type field value specified in Table 62 on page 199.

11–4. **For the Power Management option:** The power domain dependency tree must have a single root node which is the root power domain.

Personal Use Copy - Not for Reproduction

11–5. **For the Power Management option:** Each node of the power domain dependency tree must have a single parent except the root domain which has no parent.

11–6. **For the Power Management option:** The effects of changing the power level of a domain must be limited to member devices and subdomains only.

11–7. **For the Power Management option:** System firmware must provide the Open Firmware *controls-power-domains* property for all domain control points. The value of this property is a list of domain numbers specifying the domains over which this device exercises control.

11–8. **For the Power Management option:** System firmware must provide the Open Firmware *power-domains* property describing the membership of a power-manageable device in the platform power domains.

11–9. **For the Power Management option:** If system firmware provides device power state information, it must use the Open Firmware *device-power-states* property to describe the supported device states. This property is described in the *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10] and contains the following information for each state:

- Initial device power state
- Power consumption of the device per power source while idle in this state
- Power consumption of the device per power source while in use in this state
- Manufacturer's rated lifetime of the device while in this state

11–10. **For the Power Management option:** If system firmware provides device power state information, it must use the *device-state-transitions* property describing the allowable transitions between the state described in *device-power-states*. This property is described in the *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10] and contains the following information:

- A value specifying the source state of this transition
- A value specifying the destination state of this transition

Personal Use Copy - Not for Reproduction

- Power consumed per power source to make this transition
- Wall clock time required to make this transition
- Manufacturer's rated tolerance on the number of cumulative occurrences of this transition

11–11. **For the Power Management option:** If system firmware provides device power state information, each device in the Open Firmware device tree which consumes power provided by a power source must encode the *power-sources* property which is a list of power sources (indices into the *platform-power-sources* data structure).

11–12. **For the Power Management option:** System firmware must provide the Open Firmware *power-domains* property describing the power domains of which this device is a member.

11–13. **For the Power Management option:** If system firmware provides device power state information, it must use the Open Firmware *device-power-states* property to provide this information.

11–14. **For the Power Management option:** If system firmware provides device power state information, each device must encode the *power-sources* property which is a list of power sources (indices into the *platform-power-sources* data structure) from which the device draws power.

11–15. **For the Power Management option:** If system firmware provides device power state information, the *platform-power-sources* property of the Open Firmware device tree must specify all defined power sources of the platform.

11–16. **For the Power Management option:** If system firmware of a battery-operated platform provides device power state information, it must also provide the *platform-battery-sources* property for its main batteries.

11–17. **For the Power Management option:** System firmware must provide the Open Firmware *power-domains* property for each physical (non-system) node of the device tree unless the device is a member of the root power domain.

11–18. **For the Power Management option:** If the root node of the device tree encodes the *power-domains* property, the membership of any physical node which does not encode this property must be assigned to the root power domain.

Personal Use Copy - Not for Reproduction

11–19. **For the Power Management option:** Hardware/firmware must not change the power state of a device unless this change is not observable by system software and does not affect the operation of software or impact the predictability of device service time. This requirement is void if hardware/firmware must act to protect life or property.

11–20. **For the Power Management option:** Power domain power level transitions must not affect the correct operation of sibling or ancestral power domains.

11–21. **For the Power Management option:** The indirect effects of power level changes upon device within a given power domain must be limited to the following:

- Devices may present increased service time when the power level is Reduced.
- Devices may be rendered inoperative when the domain is placed in the Freeze or Off power levels.
- Devices may loose internal functional parameters when the domain in placed in the Off power state.

11–22. **For the Power Management option:** To support the usage of the system power switch as a power management event source, the actuation of this switch must not interrupt the secondary voltages to the system.

11–23. **For the Power Management option:** In support of requirement 11–22, the secondary voltages from the power supply must be software controllable.

11–24. **For the Power Management option:** The platform must establish control of all indicators following a hardware reset until system software assumes control of power management policy via the *assume-power-management* RTAS call.

11–25. **For the Power Management option:** To claim power management capability an operating system must implement at least one of the following system power states:

- Power Management Enabled (and PM Disabled)
- Standby
- Suspend

Personal Use Copy - Not for Reproduction

- Hibernate

Chapter 12 The Symmetric Multiprocessor Option

12–1. Operating systems that do not explicitly support the SMP option must support SMP-enabled hardware platforms, actively using only one processor.

12–2. **For the Symmetric Multiprocessor option:** SMP Operating Systems must support uniprocessor platforms.

12–3. **For the Symmetric Multiprocessor option:** The “SMP Extensions” sections of the Open Firmware binding for the CHRP architecture and for the PowerPC, and the SMP support section of the RTAS (see Section 7.3.11, “SMP Support,” on page 137) must be implemented.

12–4. **For the Symmetric Multiprocessor option:** All processors in the configuration must have equal functional access and “quasi-equal” timing access to all of system memory, including other processors’ caches, via cache coherence. “Quasi-equal” means that the time required for processors to access memory is sufficiently close to being equal that all software can ignore the difference without a noticeable negative impact on system performance; and no software is expected to profitably exploit the difference in timing.

12–5. **For the Symmetric Multiprocessor option:** All processors in the configuration must have equal functional and “quasi-equal” timing access to all I/O devices and adaptors. “Quasi-equal” is defined as in requirement 12–4 above, with I/O access replacing memory access for this case.

12–6. **For the Symmetric Multiprocessor option:** SMP Operating Systems must at least support SMPs with the same PVR contents and speed (for example, 133 MHz). The PVR contents includes both the PVN and the revision number.

12–7. **For the Symmetric Multiprocessor option:** All caches at the same hierarchical level must have the same Open Firmware properties.

12–8. Hardware for SMPs must provide a means of “freezing” and “thawing” the processor time base for use by RTAS. See Section 7.3.11, “SMP Support,” on page 137. This is for purposes of clock synchronization at initialization.

Personal Use Copy - Not for Reproduction

Personal Use Copy - Not for Reproduction

Bi-Endian Designs

C

This appendix discusses ways in which platforms can meet the requirement of operating in both of the PowerPC endian modes, as required by the CHRP architecture. Current PowerPC processors assume that storage is Big-Endian when the processor is in Big-Endian mode or PowerPC Little-Endian when the processor is in Little-Endian mode. Therefore, when the processor is in Little-Endian mode, a translation of Little-Endian data must be made somewhere in the platform to PowerPC Little-Endian before the data reaches the PowerPC processor. Later sections discuss the design of graphics adaptors which support both Big-Endian and Little-Endian accesses and the effect on platform designs if the processor architecture changes.

C.1 Little-Endian Address and Data Translation

In order for PowerPC processors running in Little-Endian mode to correctly access an object in Little-Endian-organized storage, the object must have its bytes show up in the correct processor byte lanes, and the Big-Endian address must be changed to refer to the correct location (after the object has had its byte lanes reordered). This translation puts the data into PowerPC Little-Endian format and has three parts.

The first part of the translation achieves address conversion and is done by the PowerPC processor. It is summarized below (a more detailed discussion can be found in Book I, Appendix D of *The PowerPC Architecture* [1]).

The PowerPC architecture defines two status bits that determine whether a PowerPC processor uses Big-Endian or Little-Endian modes. The endian mode of the kernel and the endianess of the current operating mode are recorded in

two status bits. When Little-Endian mode is enabled, the effective address (EA) is modified in the PowerPC processor as shown in Table 64 on page 266 before it is used to reference memory.

Table 64. Address Modification for Little-Endian Mode

Data Access	EA Modifier
Byte	XOR with 0b111
Halfword	XOR with 0b110
Word	XOR with 0b100
Doubleword	(no change)

This address modification results in correct Little-Endian addresses being presented to memory for aligned accesses (as will become clearer in the example below). The PowerPC architecture allows unaligned accesses in Little-Endian mode to interrupt, so incorrect memory references will not be generated if the processor does not support them. One subtlety here is that string operations and load and store multiple are considered unaligned accesses, and thus may interrupt in Little-Endian mode also. There is also a transformation which can modify the EA to support unaligned accesses in Little-Endian mode, but it is beyond the scope of this discussion.

The second part of the translation ensures that the bytes of the object addressed show up on the correct byte lanes of the processor. In Little-Endian mode, this translation requires that the bytes of a doubleword be reversed between I/O storage and the processor. This byte reversal may happen either as the data is read from or written to a Little-Endian I/O device and put into System Memory, or it may occur as the data in Little-Endian order in System Memory is brought into the processor. The required byte alignment as a function of endian mode and access is summarized in Table 65 on page 267. The table assumes that the doubleword value 0x1011121314151617 is stored at address 0 (actually any doubleword-aligned address.)

Thus, a processor in Big-Endian mode that accesses the halfword at address 4 expects to see the value 0x1415. The most significant byte of the halfword, 0x14, appears in byte lane 4 and the least significant byte, 0x15, in byte lane 5. The table shows that applying the address mapping of Table 64 on page 266 to the address and reversing the bytes between storage and the processor will result in referencing the quantity 0x1415 in byte lanes 3 and 2 at halfword address 2 in a Little-Endian storage platform, as required.

Personal Use Copy - Not for Reproduction

Table 65. Bytes Accessed Versus Endian Mode

Big-Endian	Byte Address								Little-Endian
	0	1	2	3	4	5	6	7	
	7	6	5	4	3	2	1	0	
Byte at addr 0	10								Byte at addr 7
Byte at addr 1		11							Byte at addr 6
Byte at addr 2			12						Byte at addr 5
Byte at addr 3				13					Byte at addr 4
Byte at addr 4					14				Byte at addr 3
Byte at addr 5						15			Byte at addr 2
Byte at addr 6							16		Byte at addr 1
Byte at addr 7								17	Byte at addr 0
Halfword at 0	10	11							Halfword at 6
Halfword at 2			12	13					Halfword at 4
Halfword at 4					14	15			Halfword at 2
Halfword at 6							16	17	Halfword at 0
Word at addr 0	10	11	12	13					Word at addr 4
Word at addr 4					14	15	16	17	Word at addr 0
Doubleword at 0	10	11	12	13	14	15	16	17	Doubleword at 0
Instr at addr 0	i00	i01	i02	i03					Instr at addr 4
Instr at addr 4					i10	i11	i12	i13	Instr at addr 0

The third part of the translation requires that, in Little-Endian mode, addresses be generated correctly when addressing data in Little-Endian storage or when addressing I/O device addresses. For instance, since the effective address generated by the Little-Endian program as modified by the processor is used to access I/O, the programmer writing a device driver would have to pre-compensate the effective address used to access an adapter so that, after the

Personal Use Copy - Not for Reproduction

modification by the processor, the real address used to access the adapter is the real address of the target storage/register on the adapter. This translation must be done in every device driver and makes writing device drivers very error-prone. A better solution is to solve the address modification problem one time in hardware. Logic must be added to the I/O interface such that, when Little-Endian mode is selected, the added logic undoes the modification to the three low-order bits of the address. Then the unmodified (remodified) address used to access the I/O adapter is the same address as generated by the program. This platform design is required, since a programmer writing a device driver is able to use the control register addresses as specified in the adapter hardware reference manual.

In summary, whenever the machine is running in an endian mode different from the native mode of the processor (that is, the PowerPC processor expects Big-Endian storage order, but the platform is running in a Little-Endian mode) the address must be remapped and the byte lanes reversed somewhere on the way to or from the I/O subsystems.

C.2 Conforming Bi-Endian Designs

Several designs for implementing a Bi-Endian architecture are possible. Two approaches are described in the following material. The first approach is one in which both the memory and I/O subsystems are Bi-Endian, and the second approach is one in which memory is Big-Endian and the I/O subsystems are Bi-Endian.

C.2.1 Processor and I/O Mode Control

The mode of the platform will be changed by firmware, which has to perform the required functions to place the processor and I/O subsystem in the mode desired by the operating system. These designs are based on the assumption that the processor and I/O are in the same endian mode. No attempt has been made to design a platform where Little-Endian data is read and translated for a Big-Endian-mode processor running Big-Endian applications. The processor comes up in Big-Endian mode at power on or after a hardware reset. The synchronization requirements for changing the processor from one endian mode to the other are processor implementation-dependent and are specified in the Book IV, *Processor Implementation Features* document for the processor implementation.

Since the processor does not provide an external signal indicating the endian mode selected, the platform must provide a mechanism to allow firmware to control the endian mode of other subsystems. During configuration,

Personal Use Copy - Not for Reproduction

firmware will use this mechanism to select the endian mode to be used by storing the appropriate control value to the address of the control mechanism. The platform design may place the control mechanism address in the System Control Area or in Peripheral Memory space, whichever is more convenient. Firmware must be able to address and alter the mode control mechanism in both endian modes.

C.2.2 Approach #1—Bi-Endian Memory and Bi-Endian I/O Design

The Bi-Endian Memory and Bi-Endian I/O design for a Bi-Endian platform is shown in Figure 16 on page 271. I/O devices are categorized as "Transportable I/O," which consists of devices such as disks, tapes, and networks, and "Presentation I/O," which consists of devices such as graphics, audio, and video adaptors. For this design, memory, the processor, and Transportable I/O interfaces must support both endian modes. Data may exist on the Transportable devices in either Big-Endian or Little-Endian format; it is brought in, and sent out to the outside world, in either form. Presentation I/O are by design either Big-Endian or Little-Endian (or with extra hardware they may be Bi-Endian).

The processor accesses both storage and I/O through a controlled byte reversal multiplexor and controlled address modification function (called Xpose and Mod in Figure 16 on page 271). The address modification algorithm is shown in Table 64 on page 266. The byte reversal multiplexor reverses the position of each byte in a doubleword as shown in Table 67 on page 270. Similarly, when the transfer between the processor and I/O is on a 4-byte I/O bus, the byte reversal is performed as shown in Table 68 on page 270.

In this design, the memory and I/O are connected as if they were Big-Endian. In this case, byte 0 of storage or I/O is considered the most significant byte and passes through the controlled byte reversal multiplexor to byte 0 of the processor (MSB of the processor). The rules for applying the byte reversal multiplexor and the address modification are as shown in Table 66 on page 269.

Table 66: Rules for Byte Reversal and Address Modification

Mode	Function
Big-Endian	No byte reversal of data and no address translation
Little-Endian	Byte reversal of data and address translation

Personal Use Copy - Not for Reproduction

Table 67. Endian Mode Data Byte Reversal

I/O Byte	Storage Byte	
	BE Mode	LE Mode
0	0	7
1	1	6
2	2	5
3	3	4
4	4	3
5	5	2
6	6	1
7	7	0

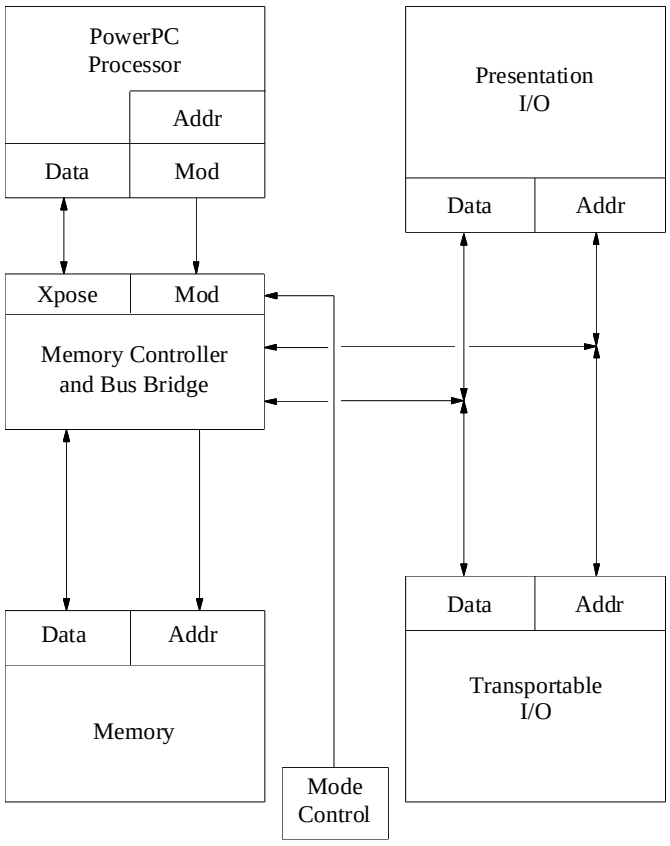
Table 68. Byte Reversal, Unequal Bus Widths

I/O		Storage Byte		Description
Word	Byte	BE Mode	LE Mode	
0	0	0	7	Word 0: Even addressed word
	1	1	6	
	2	2	5	
	3	3	4	
1	0	4	3	Word 1: Odd addressed word
	1	5	2	
	2	6	1	
	3	7	0	

From the viewpoint of the endianess of the data, I/O interfaces access only storage, not the processor or other I/O interfaces. I/O master transfers that move data from one device to another do not enter into the endianess translation.

The following three sections describe the interfaces that must be designed to perform this Bi-Endian support. For the three interfaces, both address processing and data handling are described.

Personal Use Copy - Not for Reproduction



Note: These are functional blocks, not chips.

Figure 16. Bi-Endian Platform with Bi-Endian Memory and I/O

C.2.2.1 Processor-to-Memory Interface

In this form of the design, memory is assumed to be in the same endian mode as the processor. Memory is accessed in two modes: as a result of cache-inhibited loads and stores, or as a result of cache reads or writes. Each of these operations will be described below.

Operations between cache and memory are always doublewords or larger, and as such the address in either endian mode is unaffected. Data is byte

Personal Use Copy - Not for Reproduction

reversed for Little-Endian mode and not byte reversed for Big-Endian mode. The result of these operations is that data brought into cache from Big-Endian memory is unchanged, and data brought into cache from memory in Little-Endian mode is byte reversed and stored as the PowerPC processor expects to see it (that is, most significant byte first). Writing data back to memory from cache in Little-Endian mode reverses the byte order and restores the data in Little-Endian order.

Cache-inhibited loads and stores have to adjust the data and addresses for the expected processor formats of data and addresses. For Little-Endian mode, the address generated by an instruction points to where the data exists in Little-Endian storage. The address is translated by the processor, and then translated again outside the processor which returns it to its original value. The data is byte reversed, putting it in the order expected by the processor logic on fetches or expected in storage for stores. For Big-Endian mode, the address is unchanged and the data is placed in memory in the order contained in the processor.

C.2.2.2 I/O-to-Memory Interface

There are no address or data transformations between memory and I/O. Data is placed in memory in the same order as it exists on the I/O devices independent of endian mode. The one exception might be Presentation I/O devices. Adaptors for these devices could be designed to accept data from Little-Endian and Big-Endian storage, or to always accept data only in one format. In this case, software would have to handle the data transformation. For example, a graphics adaptor that expected data in Little-Endian format would need software to byte reverse the data into Big-Endian mode before putting the data in memory.

C.2.2.3 Processor-to-I/O Interface

In Little-Endian mode, the address as modified by the processor must be modified again by the I/O interface such that the address used for the I/O access is the address computed by the storage instruction. Within the processor, the I/O addresses computed by a storage instruction are modified by the processor before the access is performed. Regardless of whether the access is to I/O space memory or a device control register, the address originally computed by the instruction is the address that must be used to access I/O space. The address modification algorithm shown in Table 64 on page 266 is used to remodify this I/O address. This function is shown in Figure 16 on page 271 in the box labeled “Mod,” which is controlled by the box labeled “Mode Control.”

Personal Use Copy - Not for Reproduction

The data transfer between the processor and I/O is managed in the same manner as the transfer between the processor and memory. Bytes are crossed between source and destination byte channels as indicated in Table 67 on page 270 and Table 68 on page 270. This byte transposition will occur once in Little-Endian mode to transform the processor-held data in Big-Endian format to Little-Endian format for I/O. In Big-Endian mode, the byte order is not transformed.

C.2.3 Approach #2—Bi-Endian I/O Design

The Bi-Endian I/O design for a Bi-Endian platform is shown in Figure 17 on page 274. For this design, storage is always Big-Endian, which means data is always stored with the most significant byte at the lowest address. Transportable I/O interfaces must support both endian modes. Data may exist on the Transportable devices in either Big-Endian or Little-Endian format; it is brought in, and sent out to the outside world, in either form. Presentation I/O are by design either Big-Endian or Little-Endian (or with extra hardware they may be Bi-Endian).

The processor accesses both storage and I/O. I/O master transfers that move data from one device to another do not enter into the endianess translation. From a platform viewpoint, I/O interfaces access only storage, not the processor or other I/O interfaces.

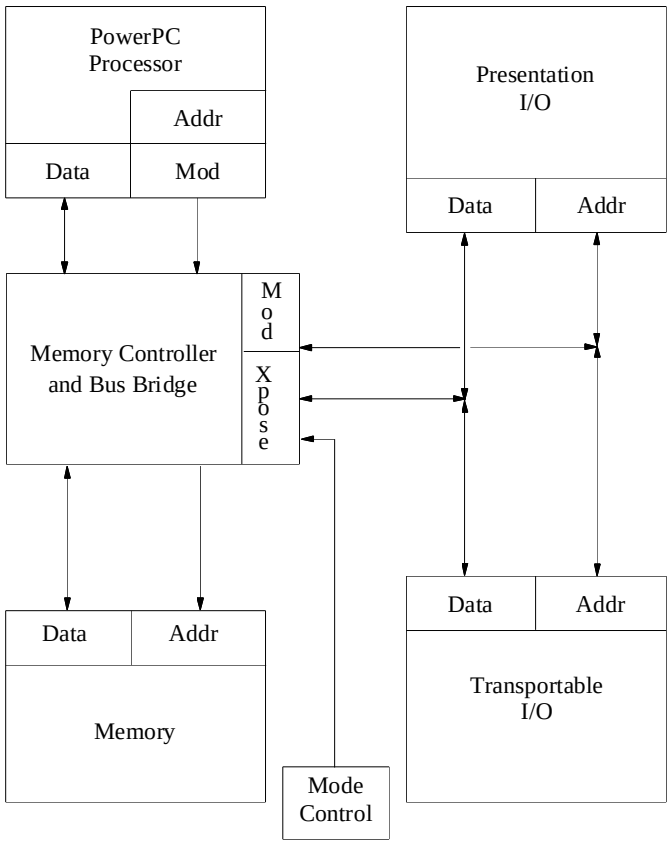
I/O accesses storage through a controlled byte reversal multiplexor and controlled address modification function (called Xpose and Mod in Figure 17 on page 274). This is further explained in Section C.2.3.2, “I/O-to-Memory Interface,” on page 274.

The following three sections describe the interfaces that must be designed to perform this Bi-Endian support. For the three interfaces, both address processing and data handling are described.

C.2.3.1 Processor-to-Memory Interface

Address translations are performed by the processor. Cache block transfers between the processor and storage are always a doubleword or larger, and the address is not affected by the endian mode. With Big-Endian storage, Little-Endian-mode loads and stores of aligned scalars with caching enabled work correctly after the address translation. When Little-Endian mode is enabled, loads and stores with caching inhibited use the address as modified by the processor. These instructions work the same and have the same constraints as for loads and stores out of cache.

Personal Use Copy - Not for Reproduction



Note: These are functional blocks, not chips.

Figure 17. Bi-Endian Platform with Bi-Endian I/O

For the processor-to- or from-memory interface, the data handling is independent of endian mode. The memory stores multi-byte scalars in Big-Endian order, which has the most significant byte at the first byte of the address.

C.2.3.2 I/O-to-Memory Interface

In Little-Endian mode, storage addresses generated by I/O devices are modified using the address modification described in Table 64 on page 266 prior to

Personal Use Copy - Not for Reproduction

performing the access. This address modification is shown in the block labeled "Mod" on the right side of the Memory Controller and Bus Bridge, shown in Figure 17 on page 274. This address modification is performed for both Transportable I/O and Presentation I/O adaptors. This address modification is required to adjust for the format of the data in memory that has been byte reversed to place the MSB first.

Data transferred to and from I/O devices that must switch endian modes must have data that is stored in Little-Endian format converted to Big-Endian format. I/O transfers may be done at any implementation-determined width, but this byte reversal of data in Little-Endian mode is done by treating the data as a string of bytes that must be reversed within a doubleword (see Table 67 on page 270). For unaligned transfers or transfers of less than a doubleword, bytes must be crossed from the source byte channel to the destination byte channel as shown in this table. This design places a byte-reversing multiplexor in the path from I/O to memory. This byte reversal multiplexor is shown as the "Xpose" box in Figure 17 on page 274.

When the interface between main storage and I/O requires a conversion from a two-word bus to a one-word bus, the byte reversal should be done in accordance with Table 68 on page 270, which assumes a two-word bus to main storage and a one-word bus to I/O.

An unaligned access (for example, read or write of System Memory) that crosses a doubleword boundary must be performed as multiple accesses on processors which do not support unaligned accesses. The address modification algorithm described above does not work for unaligned accesses. One approach to handling unaligned accesses is to perform the access as multiple aligned accesses using byte, halfword, and word operations for which the main storage address is modified as described above.

The specific design of the Presentation I/O device adaptors will determine whether a second byte-reversing multiplexor is present on the device, and will influence how device drivers must interface with this device. For instance, an audio adaptor that has byte reversal logic in it may be placed on a bus, while a graphics adaptor that does not have byte reversal logic may be on the same or a different bus. Depending upon the mode of the processor and memory, neither device may need byte reversal or both may need it. For example, a graphics adaptor with a Little-Endian design (that is, registers and data are expected in Little-Endian order) could be addressed by a Big-Endian processor via software performing the byte reversal as the Big-Endian data was sent to the graphics adaptor. Alternatively, the graphics adaptor could be designed to perform the byte reversal. In this case, the adaptor would pass data straight through when the processor is in Little-Endian mode and do a byte reversal when the processor is in Big-Endian mode.

Personal Use Copy - Not for Reproduction

C.2.3.3 Processor-to-I/O Interface

In Little-Endian mode, the address as modified by the processor must be modified again by the I/O interface such that the address used for the I/O access is the address computed by the storage instruction. Within the processor, the I/O addresses computed by a storage instruction are modified by the processor before the access is performed. Regardless of whether the access is to I/O space memory or a device control register, the address originally computed by the instruction is the address that must be used to access I/O space. The address modification algorithm shown in Table 64 on page 266 is used to remodify this I/O address. This function is shown in Figure 17 on page 274 in the boxes labeled "Mod."

The data transfer between the processor and I/O is managed in the same manner as the transfer between I/O and memory except that the processor is the master (it provides address and control) rather than an I/O mechanism. Bytes are crossed between source and destination byte channels as indicated in Table 67 on page 270 and Table 68 on page 270. This byte reversal is performed on all I/O transfers. The processor performs unaligned accesses as multiple accesses to aligned doublewords and may transfer an odd number of bytes within an aligned doubleword.

C.3 Software Support for Bi-Endian Operation

The endian mode-switching logic is a function provided by firmware before control is passed to the operating system. The specific set of instructions is processor- and platform-implementation-dependent.

As pointed out above, data for Presentation I/O and multi-byte control data (registers) for any I/O device may have to be byte reversed by software. Services to perform these operations should be provided by the support software. Typically, this would be language syntax and compiler support for a load and store with byte reversal of 2-, 4- and 8-byte scalars. If the compilers for all languages do not support these forms of loads and stores, then the operating system should supply services that perform the byte reversal.

C.4 Bi-Modal Devices

For performance reasons, applications in many operating environments write directly to graphics adaptors. Graphics adaptors for CHRP implementations will provide both Big-Endian and Little-Endian data transfer methods. (See Ta-

Personal Use Copy - Not for Reproduction

ble 2 on page 19) It is recommended that graphics subsystems implementing this support use the design shown in Figure 18 on page 278, which applies to both the frame buffer and the graphics subsystem's register/command space. This should provide good performance when using the hardware acceleration features of the graphics subsystem. In this design, the graphics subsystem is accessed through address "apertures" which perform the endian switch of data as it passes through the aperture, as necessary.

The register/command apertures provide two ways (Big- and Little-Endian) to access the control data. They are defined address ranges within the address map of the adaptor. While two dedicated apertures are desired, one aperture may be used if it can programmably accommodate Big- or Little-Endian. The aperture labelled "BE" would always provide byte swapping in a two-aperture implementation. The aperture labelled "LE/BE" would provide no byte swapping in a two-aperture implementation, and provide byte swapping in a single-aperture implementation based on one of the following:

1. The size of the facility being accessed would automatically determine how swapping is to occur for a particular access (that is, half- or full-word, or no swapping).
2. Software would programmably set the swapping function to either half-word, full-word, or none as appropriate.

The frame buffer apertures provide two ways to access the pixel data. The frame buffer apertures are defined address ranges within the address map of the adaptor. Each frame buffer aperture can be independently controlled to provide one of the following modes, under software control:

1. No swapping.
2. Byte swapping within each halfword, for example, $ABCD \Rightarrow BADC$. This becomes $ABCDEFGH \Rightarrow BADCFEHG$ for a platform with a 64-bit data interface, such as the PCI bus with optional 64-bit bus extension.
3. Byte swapping across the entire word, for example, $ABCD \Rightarrow DCBA$. This becomes $ABCDEFGH \Rightarrow DCBAHGFE$ for a platform with a 64-bit data interface, such as the PCI bus with optional 64-bit bus extension.
4. Byte swapping across a doubleword, for example, $ABCDEFGH \Rightarrow HGFEDCBA$ for 64-bit pixels on a 64-bit extended data bus.

Hardware Implementation Note: Determination of the correct swapping mode for a given access may be performed automatically in the hardware. This approach is not recommended if there is a possibility

Personal Use Copy - Not for Reproduction

that the pixel depth in the frame buffer can be interpreted differently by multiple devices, or is not guaranteed to be “known” or implied by the state of the graphics subsystem hardware.

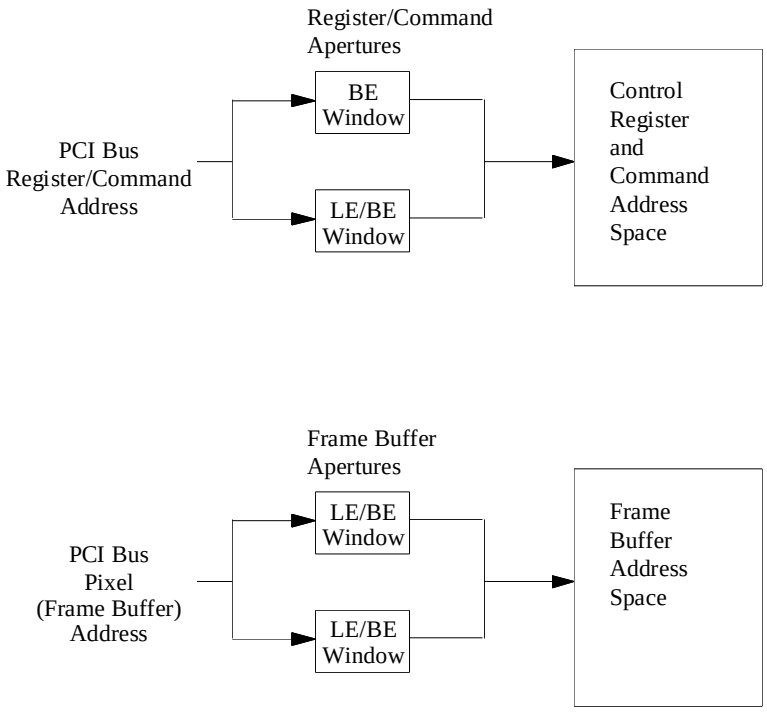


Figure 18. Bi-Endian Apertures for the Graphics Subsystem

With this capability, one aperture could access the frame buffer in Little-Endian mode, while the other could access it in one of the Big-Endian modes. Similarly, one aperture could be defined to swap for 16-bit pixels, while the other could be defined to swap for 24- or 32-bit pixels. Alternatively, several apertures may be defined to support the various pixel depths.

Platforms employing only one frame buffer aperture would provide the previously described swapping options under software control.

24-bit pixels are defined to occupy the least significant 3 bytes of a full word. The most significant byte may be used as an alpha byte, or may be

Personal Use Copy - Not for Reproduction

unused. 16-bit pixels are always halfword aligned, and 24- or 32-bit pixels are always full-word aligned.

Pixels between 32 and 64 bits are defined to occupy the least significant bytes in the 64-bit field, and are doubleword aligned.

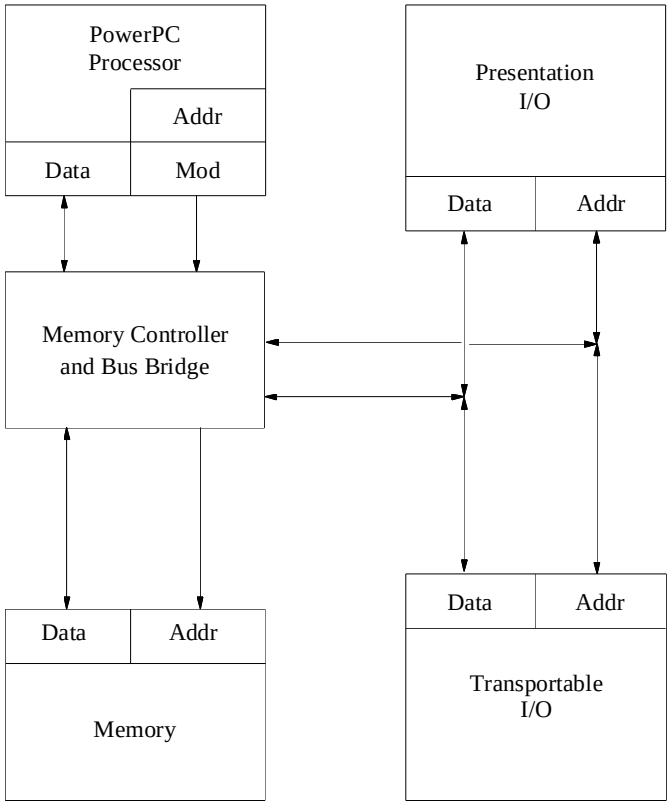
C.5 Future Directions in Bi-Endian Architecture

This section discusses possible future directions in the design of PowerPC microprocessors and the effect these changes would have on Bi-Endian platform design. The current implementations of Little-Endian mode in PowerPC microprocessors reduce internal processor complexity by moving some of the Bi-Endian support out of the processor.

Future implementations of the PowerPC architecture may support a true Little-Endian mode. In this true Little-Endian mode, data would go to the processor in Little-Endian format and be addressed from the processor with the Little-Endian address. In the interim, some PowerPC processor designs may implement support for Little-Endian unaligned operations.

Figure 19 on page 280 shows a design which could be used with future versions of the PowerPC processor implementing full Bi-Endian support. No external address modifications or byte reversal multiplexors are required. The previous designs may be migrated to this design by physically removing the byte reversal multiplexors and address modification logic components from the design, or by functionally removing them by changing the rules under which they are applied. For instance, Approach #1 (Figure 16 on page 271) would require the address modification and byte-transposing multiplexor to always be off. The address modification is no longer needed, because the processor would not modify the address since it deals with Little-Endian data in Little-Endian order. The byte reversal is not required because the processor accepts data in true Little-Endian order. The same applies to Approach #2.

Personal Use Copy - Not for Reproduction



Note: These are functional blocks, not chips.

Figure 19. Design with a Full Bi-Endian Processor

Personal Use Copy - Not for Reproduction

Architecture Migration Notes

D

This appendix provides information describing the migration from legacy systems. The PowerPC Reference Platform, Apple RISC architecture, and IBM RISC server systems were used in the development of the CHRP architecture. The objective was to reduce the porting effort of operating systems and applications coming from each of these environments. The information below describes the relationship of the Apple RISC architecture to the CHRP architecture and of the PowerPC Reference Platform to this architecture. The PowerPC Reference platform used IBM RISC client and server information.

Many components of this architecture are included for compatibility with the second generation Power Macintosh desktop products. Features of the second generation Power Macintosh and references to their use in this architecture are listed below:

- Based on the PowerPC microprocessor family. See Section 4.1, “Processor Architecture,” on page 49.
- Use of the PCI bus to support all I/O and system expansion. See 5.1 , “PCI Host Bridge (PHB) Architecture,” on page 69.
- Use of bus to bus bridges to support other buses such as NuBus™, SCSI, and IDE. See Section 5.2, “I/O Bus to I/O Bus Bridges,” on page 78.
- Use of Open Firmware for system start-up and to allow use of expansion cards from other architectures. See Section 2.2, “Firmware,” on page 13.
- Function of processor bus coherency. See Section 4.2.2, “Storage Ordering Models,” on page 57.

- Support for both Big-Endian and Little-Endian modes. See Section 2.3, “Bi-Endian Support,” on page 14.
- Provide Macintosh style I/O such as ADB, SCC, and LocalTalk. See Section 2.5, “Minimum System Requirements,” on page 16 and Chapter 9, “I/O Devices,” on page 151.

Some material used in this document was originally published in the *PowerPC Reference Platform Specification*, Version 1.1 [28]. For those currently developing products for the *PowerPC Reference Platform*, this evolution information will aid in understanding the architecture in this document. Table 69 on page 282 shows the sections in the *PowerPC Reference Platform Specification* and indicates their allocation in this document or the reason some material was not carried forward. Material carried from the *PowerPC Reference Platform Specification*, Version 1.1 has been modified and new material inserted.

Table 69. PowerPC Reference Platform Specification Evolution

PowerPC Reference Platform Section	Presentation in this Document
Chapter 1, “Introduction”	Chapter 1, “Introduction,” on page 1
Chapter 2, “Hardware Configuration”	Section 2.5, “Minimum System Requirements,” on page 16
Chapter 3, “Architecture through Section 3.1 System Topology”	Chapter 1, “Introduction,” on page 1
Section 3.2, “System Memory”	Section 4.2.2.1, “Memory Coherence,” on page 57
Section 3.3, “I/O Memory”	Section 4.2.2.1, “Memory Coherence,” on page 57
Section 3.4, “System I/O,” and Section 3.5 Self Modifying Code”	Section 4.2.2.1, “Memory Coherence,” on page 57
Section 3.6, “Resource Locking”	Section 9.1.1, “Resource Locking,” on page 152
Section 3.7. “Bus Errors”	Chapter 10, “Error and Event Notification,” on page 157
Section 3.8, “Memory Map”	Section 4.2.1, “System Memory,” on page 56
Section 3.8.1, “Example Memory Maps”	Chapter 3, “System Address Map,” on page 23
Section 3.9, “Memory Ordering”	Section 4.2.2.1, “Memory Coherence,” on page 57
Section 3.10, “Configuration and Diagnostics”	Not covered in the Architecture
Section 3.11, “Power Management”	Chapter 11, “Power Management,” on page 185
Section 3.12, “Bi-Endian Support”	Section 2.3, “Bi-Endian Support,” on page 14

Personal Use Copy - Not for Reproduction

Table 69. PowerPC Reference Platform Specification Evolution ***(Continued)***

PowerPC Reference Platform Section	Presentation in this Document
Section 3.13, “Multiprocessor Considerations”	Chapter 12, “The Symmetric Multiprocessor Option,” on page 215
Section 3.14, “Alignment Considerations”	Section 4.1.4.1, “Unaligned Little-Endian Scalar Operations,” on page 53 and Section 4.1.4.2, “Little-Endian Multiple Scalar Operations,” on page 54
Section 3.15, “Support for Loads and Stores to System I/O Bus,” and Section 3.16, “Cache-Inhibited Loads and Stores to System Memory”	Section 4.1.4.3, “Direct-Store Segment Support,” on page 54
Section 3.17, “PowerPC Architecture Features Not Recommended”	Section 4.1.4, “PowerPC Architecture Features Deserving Comment,” on page 53
Chapter 4, “Machine Abstraction”	Chapter 7, “Run-Time Abstraction Services,” on page 91
Chapter 5, “Boot Process,” through Section 5.4, “Transferring Control”	The legacy process is defined in the current version of the *PowerPC Reference Platform Specification* [28]
Section 5.5, “NVRAM”	NVRAM for the legacy process is described in the *PowerPC Reference Platform Specification* [28]. NVRAM for the Open Firmware process is described in Chapter 8, “Non-Volatile Memory,” on page 141
Section 5.6, “Residual Data”	Residual data for the legacy process is described in the *PowerPC Reference Platform Specification* [28]
Section 5.7, “Open Firmware Extension”	Refer to the *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10]
Chapter 6, “Reference Implementation”	Hardware system vendors may publish separate books with implementation examples for compliant platforms
Appendix A, “Implementation Examples”	Separate books prepared by hardware system vendors
Appendix A.7, “Proposed Diagnostic Strategy”	Not covered in the Architecture
Appendix B, “Bi-Endian Design Guidance”	Appendix C, “Bi-Endian Designs,” on page 265
Appendix C, “Additional Compliant Subsystems and Devices”	Separate books prepared by hardware system vendor
Appendix D, “Windows NT”	Obtain information from the operating systems
Appendix E, “AIX”	Obtain information from the operating systems
Appendix F, “Workplace OS”	Obtain information from the operating systems
Appendix G, “Solaris”	Obtain information from the operating systems

Personal Use Copy - Not for Reproduction

Table 69. PowerPC Reference Platform Specification Evolution ***(Continued)***

PowerPC Reference Platform Section	Presentation in this Document
Appendix H, “Taligent”	Obtain information from Taligent™
Appendix I, “PowerPC Supplement to IEEE 1275”	See *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10]
Appendix J, “Plug and Play Extensions”	See the *PowerPC Reference Platform Specification* [28]
Appendix K, “Dump of Residual Data”	Residual data is described in the *PowerPC Reference Platform Specification* [28]
Obtaining Additional Information	Section , “Obtaining Additional Information,” on page 300
Bibliography	Section , “Bibliography,” on page 297
Acronyms and Abbreviations	Section , “Glossary,” on page 285
Glossary	Section , “Glossary,” on page 285
Trademark Information	Section , “Trademark Information,” on page 295
Index	Section , “Index,” on page 303

Personal Use Copy - Not for Reproduction

Glossary

This glossary contains an alphabetical list of terms, phrases, and abbreviations used in this document.

Term	Definition
AC	Alternating current
AD	Address data line
ADB	Apple Desktop Bus
addr	Address
Architecture	The hardware/software interface definition or software module to software module interface definition.
ASCII	American National Standards Code for Information Interchange
ASR	Address Space Register
BAT	Block address translation
BE	Big-Endian or Branch Trace Enable bit in the MSR
BIM	Bottom of initial memory
BIO	Bottom of Peripheral Input/Output Space
BIOS	Basic input/output system
Boundedly undefined	Describes some addresses and registers which when referenced provide one of a small set of predefined results.

BPM	Bottom of Peripheral Memory
BSCA	Bottom of System Control Area
BSM	Bottom of System Memory
CD-ROM	Compact disk read-only memory
CHRP	Common Hardware Reference Platform
CIS	Client interface service
COM	Communication
CPU	Central processing unit
CR	Condition Register
CTR	Count Register
DABR	Data Address Breakpoint Register
DAC	Dual address cycle
DAR	Data Address Register
DBAT	Data block address translation
DBDMA	Descriptor based direct memory access
DDC 1	Display data channel for unidirectional information from displays.
DEC	Decrementer
DIMM	Dual in-line memory module
DOS	Disk operating system
DR	Data relocate bit in MSR
DSISR	Data Storage Interrupt Status Register
DMA	Direct memory access
EA	Effective address
EAR	External Access Register
ECC	Error checking and correction
ECP	Extended capability port
EE	External interrupt enable bit in the MSR
EISA	Extended industry standard architecture

Personal Use Copy - Not for Reproduction

EPA	Environmental protection agency
EPOW	Environment and power warning
ERR	Exception Relocation Register
ESCD	Extended system configuration data
FCode	A computer programming language defined by the Open Firmware standard which is semantically similar to the Forth programming language, but is encoded as a sequence of binary byte codes representing a defined set of Forth words.
FE0	Floating-Point exception mode 0 bit in the MSR
FE1	Floating-Point exception mode 1bit in the MSR
FP	Floating-Point available bit in the MSR
FPR	Floating-Point register
FPSCR	Floating-Point Status And Control Register
FSM	Finite state machine
GB	Gigabytes - as used in this document it is 2 raised to the power of 30
HB	Host Bridge
Hz	Hertz
IBAT	Instruction block address translation
ID	Identification
IDE	Integrated device electronics
IDU	Interrupt delivery unit
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/output
I/O bus master	Any entity other than a processor, cache, memory controller, or host bridge which supplies both address and data in write transactions or supplies the address and is the sink for the data in read transactions.
ILE	Interrupt Little-Endian bit in MSR
Instr	Instruction
IP	Interrupt prefix bit in MSR

Personal Use Copy - Not for Reproduction

IPI	Interprocessor interrupt
IPL	Initial program load
IR	Instruction relocate bit in MSR register or infrared
IrDA	Infrared Data association which sets standards for infrared support including protocols for data interchange.
ISA	Industry standard architecture (typically refers to the PC local bus)
ISO	International Standards Organization
ISR	Interrupt source register
ISU	Interrupt source unit
JEIDA	Japan Electronic Industry Development Association
KB	Kilobytes - as used in this document it is 2 raised to the power of 10
KHz	Kilo Hertz
LAN	Local area network
LCD	Liquid crystal display
LE	Little-Endian bit in MSR or Little-Endian
LED	Light emitting diode
LFB	Linear frame buffer
LR	Link Register
L1	Primary cache
L2	Secondary cache
MB	Megabytes - as used in this document it is 2 raised to the power of 20
ME	Machine check enable
MESH	Macintosh enhanced SCSI hardware
MFM	Modified frequency modulation
MHz	Mega Hertz
MOD	Address modification bit in the MSR
MOU	Memorandum of understanding

Personal Use Copy - Not for Reproduction

MP Multiprocessor

MSB Most significant byte

MSR Machine State Register

N/A Not applicable

Nibble Refers to the first or last four bits in an 8 bit byte

NVRAM Nonvolatile random access memory

OF Open Firmware

OS Operating system

OUI Organizationally unique identifier

PC Personal computer

PC Card A memory or I/O card compatible with the *PC Card Standard* [17]. When cards are referred to as PC Cards, what is being addressed are those characteristics common to both 16-bit PC Cards and CardBus PC Cards.

PCI Peripheral Component Interconnect

PCMCIA Personal Computer Memory Card International Association (see PC Card)

Peripheral I/O Space

The range of real addresses which are assigned to the I/O Space of a Host Bridge (HB) and which are sufficient to contain all of the *Load* and *Store* address space requirements of all the devices in the I/O Space of the I/O bus that is generated by the HB. A keyboard controller is an example of a device which may require Peripheral I/O Space addresses. This portion of the system address space was referred to as “System I/O” in the *PowerPC Reference Platform Specification*.

Peripheral Memory Space

The range of real addresses which are assigned to the Memory Space of a Host Bridge (HB) and which are sufficient to contain all of the *Load* and *Store* address space requirements of the devices in the Memory Space of the I/O bus that is generated by the HB. The frame buffer of a graphics adapter is an example of a device which may require Peripheral Memory Space addresses. This portion of the system address space was re-

Personal Use Copy - Not for Reproduction

ferred to as “I/O Memory” in the *PowerPC Reference Platform Specification* [28].

Peripheral Space
: Refers to the physical address space which may be accessed by a processor, but which is controlled by a host bridge. At least one peripheral space must be present and it is referred to by the suffix 0. A host bridge will typically provide access to at least a memory space and possibly to an I/O space.

PHB
: PCI Host Bridge

PIC
: Programmable interrupt controller

PIR
: Processor Identification Register

Platform
: Refers to the hardware plus firmware portion of a system composed of hardware, firmware, and operating system.

PM
: Power management

PnP
: Plug and play

POST
: Power-on self test

PR
: Privileged bit in the MSR

Processor revision number
: A 16-bit number that distinguishes between various releases of a particular processor version, for example different engineering change levels.

PVN
: Processor version number. Uniquely determines the particular processor and PowerPC architecture version.

PVR
: Refers to the Processor Version Register. A register in each processor that identifies its type. The contents of the PVR include the processor version number and processor revision number.

RAM
: Random access memory

Real address
: A real address results from doing address translation on an effective address when address translation is enabled. If address translation is not enabled, the real address is the same as the effective address. An attempt to fetch from, load from, or store to a real address that is not physically present in the machine may result in a Machine Check interrupt.

Personal Use Copy - Not for Reproduction

Reserved — The term “reserved” is used within this document to refer to bits in registers or areas in the address space which should not be referenced by software except as described in this document.

Reserved for firmware use — Refers to a given location or bit which may not be used by software, but are used by firmware.

Reserved for future use — Refers to areas of address space or bits in registers which may be used by future versions of the architecture.

RFU — Reserved for future use

RI — Recoverable interrupt bit in the MSR

RISC — Reduced instruction set computing

ROM — Read only memory

RPN — Real page number

RTAS — Run-time abstraction services

RTC — Real time clock

SCC — Serial communications controller

SCSI — Small computer system interface

SDR — Storage Description Register

SE — Single-step trace enabled bit in the MSR

SF — Processor 32-bit or 64-bit processor mode bit in the MSR

Shrink-wrap OS — A single version of an operating system that runs on all compliant platforms.

Shrink-wrap Application — A single version of an application program that runs on all compliant platforms with the applicable operating system.

SIMM — Single in-line memory module

SMP — Symmetric multiprocessor

Snarf — An industry colloquialism for cache-to-cache transfer. A typical scenario is as follows: (1) cache miss from cache A, (2) line found modified in cache B, (3) cache B performs castout of

Personal Use Copy - Not for Reproduction

modified line, and (4) cache A allocates the modified line as it is being written back to memory.

Snoop — The act of interrogating a cache for the presence of a line, usually in response to another party on a shared bus attempting to allocate that line.

SPRG — Special Purpose Registers for General use

SR — System Registers

SRR — Save/Restore Register

System — Refers to the collection of hardware, system firmware, and operating system software which comprise a computer model.

System address space — The total range of addressability as established by the processor implementation.

System Control Area — Refers to a range of addresses which contains the system ROM(s) and an unarchitected, reserved, platform-dependent area used by firmware and Run-Time Abstraction services for control of the platform. The ROM areas are defined by the OF properties in the *openprom* and *os-rom* nodes of the OF device tree.

System firmware — Refers to the collection of all firmware on a system including Open Firmware, RTAS and any legacy firmware.

System Memory — Refers to those areas of memory which form a coherency domain with respect to the PowerPC processor or processors that execute application software on a system.

System software — Refers to the combination of operating system software, device driver software, and any hardware abstraction software, but excludes the application software.

TB — Time base

TBD — To be determined

TBEN — Time base enable signal on a PowerPC processor

TCE — Translation Control Entry

Personal Use Copy - Not for Reproduction

TEA	Transaction error acknowledge signal on a PowerPC processor
TEMR	Top of Emulated Memory Register
TIO	Top of Peripheral Input/Output Space
Third party DMA	The process by which an entity independent of the data source and sink generates addresses for transfers from the source to the sink; or a noun referring to the independent entity; or a noun referring to the transfer done using this technique.
TLB	Translation lookaside buffer
TODC	Time of day clock
TPM	Top of Peripheral Memory
TSM	Top of System Memory
tty	Teletypewriter or ASCII character driven terminal device
UCT	Universal coordinated time
URL	Universal resource locator
VESA	Video Electronics Standards Association
VGA	Video graphics array
VIA	Versatile interface adapter
VPD	Vital product data
XER	Fixed-Point Exception Register
Xpose	Transpose data during LE mode transfer
x86	Intel processors such as 80386, 80486,...

Personal Use Copy - Not for Reproduction

Blank page when cut - 294

Trademark Information

The following terms, denoted by a registration symbol (®) or trademark symbol (™) on the first occurrence in this publication, are registered trademarks or trademarks of the companies as shown in the list below:

Trademark	Company
Apple Desktop Bus	Apple Computer, Inc.
AIX	International Business Machines Corporation
Apple	Apple Computer, Inc.
CHRP	Apple Computer, Inc., International Business Machines Corporation, and Motorola, Inc.
Ethernet	Xerox
IBM	International Business Machines Corporation
Intel	Intel Corporation
LocalTalk	Apple Computer, Inc.
Macintosh	Apple Computer, Inc.
Mac OS	Apple Computer, Inc.
Micro Channel	International Business Machines Corporation
Motorola	Motorola, Inc.
NetWare	Novell, Inc.

NuBus	Texas Instruments
Power Macintosh	Apple Computer, Inc.
PowerPC	International Business Machines Corporation
PS/2	International Business Machines Corporation
Solaris	SunSoft
Taligent	Taligent, Inc.
Windows	Microsoft Corporation
Windows NT	Microsoft Corporation

Personal Use Copy - Not for Reproduction

Bibliography

This section lists documents which were referenced in this specification or which provide additional information. Following this list are two sections. The first section gives useful information for obtaining these documents. The second section lists contacts and sources for additional helpful information. The documents are listed below:

1. *The PowerPC Architecture: A Specification for a New Family of RISC Processors,* Second Edition, Morgan Kaufmann Publishers, Inc., San Francisco, CA, ISBN 1-55860-316-6
2. *Computer Architecture: A Quantitative Approach,* Second Edition Morgan Kaufmann Publishers, Inc., San Francisco, CA, ISBN 1-55860-329-8
3. *How To Create Endian-Neutral Software for Portability,* IBM TR54.837 (also published in *Dr. Dobb's Journal* for October and November 1994)
4. *PowerPC 603 RISC Microprocessor User's Manual,* IBM order number MPR603UMU-01, Motorola order number MPC603UM/AD
5. *PowerPC 603 RISC Microprocessor Technical Summary,* Rev 3 IBM order number MPR603TSU-03, Motorola order number MPC603/D
6. *PowerPC 604 RISC Microprocessor User's Manual,* IBM order number MPR604UMU-01, Motorola order number MPC604UM/AD
7. *PowerPC 604 RISC Microprocessor Technical Summary,* Rev 1 IBM order number MPR604TSU-02, Motorola order number MPC604/D
8. *Open PIC Multiprocessor Interrupt Controller Register Interface Specification,* Revision 1.2

9. IEEE 1275, *IEEE Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices,* IEEE part number DS02683, ISBN 1-55937-426-8

10. *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware*

11. *PowerPC processor binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware*

12. *PCI Bus Binding to IEEE 1275 Standard for Boot (Initialization, Configuration) Firmware*

13. IEEE 996, *A Standard for an Extended Personal Computer Back Plane Bus*

14. *PCI Local Bus Specification*, Revision 2.1

15. *PCI System Design Guide*, Revision 1.0, September 1993

16. *PCI to PCI Bridge Architecture Specification*

17. *PC Card Standard*, February 1995

18. *PowerPC 60x Microprocessor Interface Specification*, by IBM and has limited availability

19. *PowerPC 6xx Bus Definition,* by IBM and has limited availability

20. *PowerPC Microprocessor Common Hardware Reference Platform: Hardware Technical Reference*, available from IBM

21. *Plug and Play ISA Specification,* Version 1.0a, Microsoft Corporation

22. *Extended System Configuration Data Specification*, Version 1.01, available with other Plug and Play information

23. *Macintosh Technology in the Common Hardware Reference Platform*, Morgan Kaufmann Publishers, Inc., San Francisco, CA, ISBN 1-55860-393-X

24. *Inside Macintosh*, Addison-Wesley Publishing Company, Reading, MA.

25. *Designing PCI Cards and Drivers for Power Macintosh Computers*, APDA, P.O. Box 319, Buffalo, NY 14207

26. *Technical Introduction to the Macintosh Family,* Second Edition, Addison-Wesley Publishing Company, Reading, MA, ISBN 0-201-62215-7

Personal Use Copy - Not for Reproduction

27. *RISC System/6000 PowerPC System Architecture*, Morgan Kaufmann Publishers, San Francisco, CA., ISBN 1-55860-344-1

28. *PowerPC Reference Platform Specification*, Version 1.1

29. *PCI Multimedia Design Guide*

Sources for Documents

This section provides useful information for obtaining the documents listed above.

The manuals for the PowerPC microprocessors or the *PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture* are available from the following sources:

- IBM at 1-800-PowerPC (1-800-769-3772) in the U.S.A
  - If the 1-800-PowerPC number can not be reached or if multilingual operators are required, use 1-708-296-6767
- IBM at (39)-39-600-4455 in Europe
- Motorola at 1-800-845-MOTO (6686)

The *PowerPC Reference Platform Specification* in PostScript format is available via anonymous FTP and on CompuServe. The FTP server address is ftp.austin.ibm.com and the material is placed in the directory /pub/technology/spec. The document is maintained on a library of the PowerPC forum.

The *PowerPC Reference Platform Specification* and *PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture* are available in Europe electronically from a bulletin board service (BBS). The BBS may be reached at +39-39-600-5076 and supports up to 28.8 KBPS.

The documents published by Morgan Kaufmann Publishers may be purchased in bookstores and may be ordered from Morgan Kaufmann at 1-800-745-7323. The PowerPC versions may be available from IBM publications sources at 1-800-426-6477 in the US only. In other countries, order from your local source for IBM literature.

The *Open PIC Multiprocessor Interrupt Controller Register Interface Specification* is available from Advanced Micro Devices, Inc. or Cyrix, Inc. Send a request which includes your postal address to openpic@amd.com.

The *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* and *PowerPC processor binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* are available via

Personal Use Copy - Not for Reproduction

anonymous FTP to playground.sun.com. These documents are located on the path /pub/p1275/bindings/postscript/*.ps.

Call 1-212-642-4900 for orders or inquiries pertaining to ANSI and ISO standards.

To order copies of EIA standards, contact Global Engineering Documents at 1-800-854-7179.

Copies of IEEE standards may be obtained by calling 1-800-678-IEEE. For information about IEEE standards, call 908-562-3800.

To purchase PCI documents, call 1-800-433-5177 in the U.S.A. or 1-503-797-4207 outside the U.S.A.

Copies of PCMCIA standards including the *PC Card Standard* may be obtained by calling PCMIA at 1-408-720-0107 or Fax to 1-408-720-9416 or by calling JEIDA at +81-3-3433-1923 or Fax to +81-3-3433-6350.

Plug and Play information is available on CompuServe Plug and Play forum (GO PLUGPLAY).

Obtaining Additional Information

Several sources exist for obtaining additional information about the PowerPC microprocessor and the *PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture*.

Hardware system vendors may obtain information on IBM components or the IBM design kits which give further information on their reference implementation by contacting IBM at the following numbers:

- IBM at 1-800-PowerPC (1-800-769-3772) in the U.S.A
- Within Europe (33)-6713-5757 in French
- Within Europe (33)-6713-5756 in Italian
- Within Europe (49)-511-516-3444 in English
- Within Europe (49)-511-516-3555 in German
- In Asia (81)-755-87-4745 in Japanese

Updates to the *The PowerPC Architecture: A Specification for a New Family of RISC Processors* are available at http://www.austin.ibm.com/tech/ppc-chg.html.

Updates to *The PowerPC Microprocessor Common Hardware Reference Processor: A System Architecture* are available via anonymous FTP. The FTP server address is ftp.austin.ibm.com and the material is placed in the directory /pub/technology/spec. This document and other related documents are

Personal Use Copy - Not for Reproduction

described on the home page at www.austin.ibm.com at the directory /pow-rinfo.html.

Several white papers are available from IBM. These papers expand on the information in this specification. These papers are available via anonymous FTP to ftp.austin.ibm.com and in the directory /pub/technology/spec. The papers which were available at the time of publishing this document are as follows:

- *Bi-Endian Designs in PowerPC Reference Platform* by Shien-Tai Pan
- *PowerPC Endian Switch Code* by Gary Tsao
- *Plug and Play for PowerPC Reference Platform* by Gary Tsao
- *L2 Cache Design for PowerPC-Based Systems* by Allan Steel
- *The PowerPC™ Hardware Reference Platform,* by Steve MacKenzie, David Tjon, Allan Steel, and Steve Bunch

OEMs, IHVs, and ISVs may obtain information on Motorola products and Motorola system design kits by contacting their local Motorola sales office or 1-800-845-MOTO (6686).

Information on the full range of Motorola's semiconductor, software, design kits, and system products may be found at Motorola's PowerPC World Wide Web home page located at http://www.mot.com/PowerPC/.

Information on the full range of IBM products may be found at IBM's World Wide Web home page located at http://www.ibm.com/.

Information on the full range of Apple products may be found at Apple's World Wide Web home page located at http://www.apple.com/.

An electronic forum on CompuServe has been established for the discussion of PowerPC reference platform topics and for obtaining answers to questions on the PowerPC reference platform. Go to the "PowerPC" forum on CompuServe and join the "Reference Platform" topic.

Information on IrDA documents is available on the World Wide Web at ftp://hplose.hpl.hp.com.

VESA documents including the DDC 1 standard are available on the World Wide Web at www.vesa.org.

Personal Use Copy - Not for Reproduction

Blank page when cut - 302

Index

Personal Use Copy - Not for Reproduction

Personal Use Copy - Not for Reproduction

Personal Use Copy - Not for Reproduction

R

Personal Use Copy - Not for Reproduction

Personal Use Copy - Not for Reproduction

Personal Use Copy - Not for Reproduction