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