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.

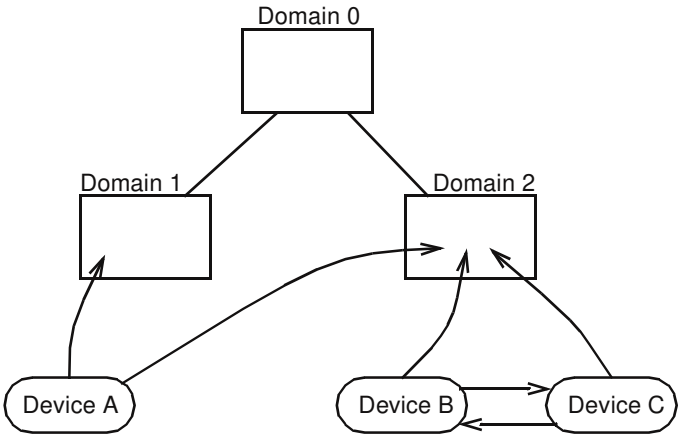

Figure 13. Example Domain and Device Dependency Relationships

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

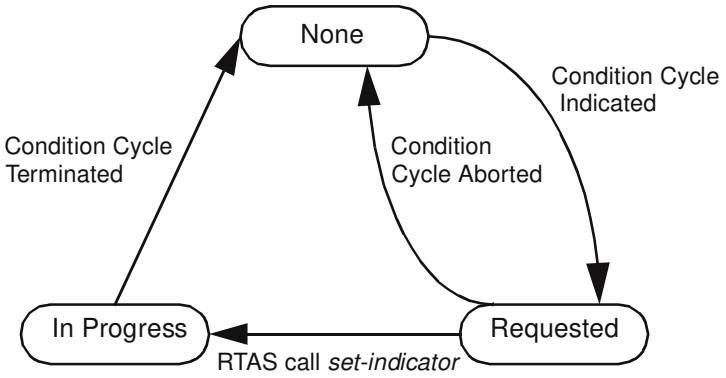

Figure 14. Battery Condition Cycle State Transition Diagram

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

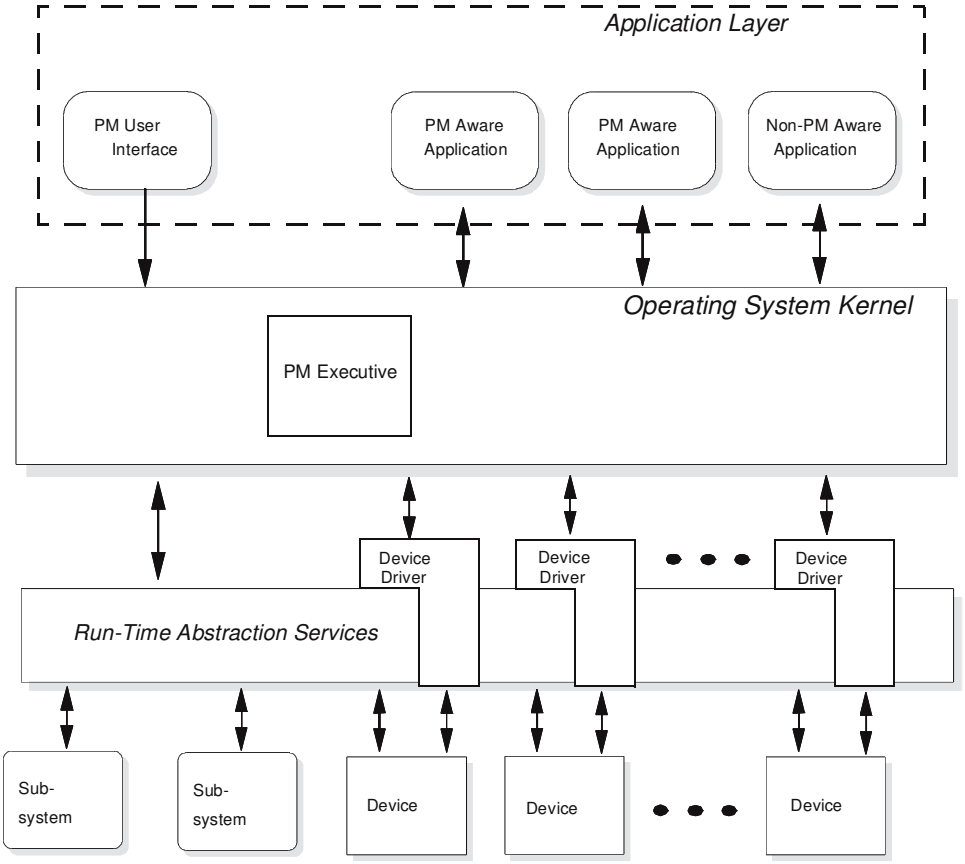

Figure 15. Example Power Management Software Structure

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