PowerPC™ Reference Platform Architectural Aspects of Power Management

Final DRAFT of White Paper for Publication

February 16, 1995

Andrew R. Rawson, Advisory Engineer
Power Personal Systems Architecture, Austin, TX

Released for external publication, Feb. 15, 1995

Abstract

Power management is a set of hardware and software facilities used to optimize power consumption in a computer system or peripheral. The particular focus of this paper is power management in PowerPC™ microprocessor–based computers.

Although power management originated in mobile computers, it has become an important consideration in other environments as well. This paper first gives an overview of the motivating factors which led to the extension of the principles of power management to non–portable computers. It then presents a specific philosophy of power management and defines system–level requirements for both the hardware and software. Next a structural framework for power management software is presented to facilitate the discussion of architectural issues. It is demonstrated that the concepts of power management impact both operating system, and ultimately application software. A recommended set of system power states and allowable transitions is defined and the concept of device power state is discussed. Finally the issues of requisite extensions to system configuration management and system firmware are considered.

It is argued that power utilization can be optimized only through synergy between the operating system power management facilities and power management–aware application software. This paper gives a staged strategy for reaching this cooperative situation.

1.0 Introduction

The *PowerPC™ Reference Platform Specification* [1] provides a methodology which allows developers of PowerPC™ microprocessor–based computer systems to design platforms which can run a wide range of operating systems and share applications. As will be explained, power management is an important new element of system design not only for portable computing, but now also for non–mobile home and office environments.

This paper presents architectural aspects of power management that are of interest to both hardware and software developers. The requirements outlined in this paper should be seriously considered in the design of future PowerPC™ Reference Platform–compliant systems and are embodied in products being designed by IBM's Power Personal Systems development organizations in Austin, Texas and Yamato, Japan.

2.0 Power Management Motivation

Although the initial impetus for power management was the extension of useful operating time for mobile computers, with the growing number of personal computers and a heightened awareness of the need to conserve energy, the scope of power management has broadened to include non–mobile computers. One of the main tenets of the personal computer revolution has been to increase the availability of computing power to the user while at the same time decreasing the cost per user. While this effort has been very successful, the down side of increased availability is that most personal computers sit idle the vast majority of the time.[2] During these times of inactivity most desktop computers continue to consume electrical power at a level only slightly less than when in use. So while the cost per user of the hardware has declined, another important aspect of the total operating cost associated with providing this very high level of availability has largely been ignored.

As the industry moves into the world of multimedia, video conferencing, and real–time applications, the trend is toward an increase in demand for reserve computational power. The semiconductor technology of microprocessors and support logic has moved from bipolar transistor to CMOS which inherently consumes less power. Despite this fact, as clock speeds escalate to attain more performance, systems continue to increase their power consumption. Moreover, the addition of more and more complex add–in adapters and external peripherals compounds the problem.

At odds with the drive toward more powerful computers is the desire to have smaller, cooler, quieter machines. Power management is a significant part of the solution to these conflicting requirements. The goal of power management is to reduce the average power usage of a computer system without reducing availability. Lower power consumption naturally results in lower heat generation, which allows components to be packed more densely and require less supplemental cooling. Denser packaging enables smaller–sized systems. Since supplemental cooling is normally supplied via fans and fans generate noise, eliminating the need for a fan is tantamount to the goal of eliminating noise.

Another force at work is concern for the environment. The generation of power impacts the environment due to the production of pollutants, carbon dioxide, and waste heat. Eliminating any unnecessary usage of electrical power is thus an opportunity to reduce pollution. Electrical power generation also consumes increasingly scarce non–renewable energy reserves.

In the United States, the Environmental Protection Agency has addressed this opportunity by instituting a program called Energy Star Partners.[3] The EPA has taken the approach of encouraging industry to respond voluntarily to improve the energy efficiency of computer systems instead of requesting new legislation. The manufacturers of computer equipment have responded by bringing Energy Star–compliant computers to market. Success of these products encourages other manufacturers to offer Energy Star–compliant products.

3.0 History of Power Management

The most popular operating system for the IBM–compatible personal computer originally had no provision for power management. As this operating system was placed on battery–powered mobile computers, the system designers were forced to add power management in a manner that did not impact the existing system software. This led to some hardware–only solutions to the problem of power management in portable systems. These hardware mechanisms relied on reducing the clocking rate to various subsystems (the CPU, memory controller, and the peripheral bus controller) to reduce their power consumption during times when the demand for these resources was low.

The current state of the art in power management employs changes to both system software and firmware[4,5]. System software is augmented with a power management device driver, system utility programs, and extensions to the system firmware. Once a cooperative interface has been established between the power management device driver and system firmware, the power management device driver takes over control the power state of system I/O devices. Power management event notification messages are passed up from devices through the power management extended system firmware to the power management device driver from whence they can be broadcast as system messages. These messages can be monitored by power–management–aware application programs. Applications may also send requests for device power state transitions to the power management device driver which forwards these to individual device drivers to effect the control of device power states. The approach presented in this paper extends and enhances this model.

4.0 Power Management Philosophy

The basic principle of power management is very simple — turn off anything not currently in use. This same principle applies to conserving energy in a home, for example.

The complication in applying this principle to computer systems is that an application may need access to a resource immediately after it is powered down. This would be no problem if the resource that was powered down could come back online instantaneously. While electro–mechanical devices like hardfiles and CD–ROMs obviously need time to come back up to an operational level of responsiveness, even non–mechanical devices such as I/O bridge chips require time and attention from software to return to an operational state after a period of power loss. Any application that is waiting on a resource to come back online is not getting its intended task accomplished.

A second fundamental problem is increased wear in devices and subsystems due to turning power off during times of inactivity. Hardfiles, for example, experience significant wear and resultant increases in failure rate when power to the spindle motor is cycled frequently. This is due to read/write head erosion as it repeatedly lands on the platter surface. Even solid state semiconductor devices are not exempt from this problem. These devices show wear effects primarily due to the thermal cycling which results from repeatedly removing and applying power. Finally, the energy required to bring a device from a lower power state to a higher, more responsive power state must be taken into consideration.

In the power management approach described in this paper, the operating system bears the responsibility of allocating system resources in a manner that takes into consideration both the current power state of all system resources and the resource requirements of all currently active applications. For example, the operating system may prioritize a task which requires only resources which are currently available over one which requires one or more resources that are not available having been placed in a power–savings mode. At the same time it would initiate the process of bringing the required resource up to a state which meets the requirement of the waiting application. Carrying this out requires knowledge of the resource requirements of all the tasks at hand. This type of information is not usually available.

There is really only one reliable way of knowing the resource requirements of an application — the application developer must provide this information. A mechanism needs to be set up to convey such information to the operating system. This can be accomplished either by the application itself via operating system calls or by a task description language which can execute in a command script prior to the invocation of the application.

In lieu of this explicit resource description, the operating system is forced to make a prediction based on experience with the resource utilization of a given application. Taking advantage of the principles of physical and temporal locality we can make a guess about the future resource requirements of an application based on history. This type of knowledge is always imperfect, however, leading to performance bottlenecks.

As performance becomes more of an issue (as it is any real-time application), application developers will be motivated to use these new mechanisms to inform the operating system of application requirements both at invocation and during runtime as requirements change dynamically. As applications are enhanced to take advantage of the power management capabilities of the operating system, it becomes possible to view power as a schedulable resource. It will also be possible to inform the user of resource requirement conflicts which may impact real–time performance.

5.0 System Hardware Requirements

The design of a power–managed computer presents the system designer with several new requirements. These include the following changes to system hardware:

- Redefinition of the electro–mechanical function of the main power switch
- Functional changes to the power supply
- Additional power supply and, possibly, clock control logic
- Changes to the system firmware
- Possible addition of new indicators
- Consideration of power cycling on reliability

In addition the system hardware designer must supply new types of information to the operating system which were not required in non–power–managed systems. Finally, the designer must work to encourage the incorporation of power management facilities in new support chips and subsystems.

5.1 System Functional Requirements

The electrical and mechanical function of the main power switch is redefined in a power–managed system. In a conventional non–power–managed computer, the position of the main power switch usually represents the current state of the power in the system. In the power–managed system, this is not necessarily the case. The power switch, therefore, should be a mechanically stateless momentary contact switch.

The significance of actuating the power switch changes. It now becomes a system power state transition request to the power management software. Software must be in control of the state of the secondary power circuits. This requires changes in the function of the power supply. A logic signal input to the power supply must be provided to allow turning on and off the secondary voltages. For AC–powered systems, this has the side benefit of eliminating the internal wiring of AC power from the rear of the machine to the front where the power switch is located.

Since opening and closing the primary circuit no longer controls the system power, logic to turn system power on and off must be provided. For AC–powered systems, this logic may be battery–powered, powered by an always–on auxiliary secondary, or a combination of both. This logic will most likely be supplied by a microcontroller and its embedded firmware. This is referred to as the power management controller. This controller may need to provide general purpose I/O pins for the sensing and control of subsystems that have no direct software–controllable power management facilities.

If power consumption in certain subsystems is reduced by slowing or stopping one or more clock signals, the system designer must supply software–controllable logic to provide this function. Consideration should be given to providing power islands on the motherboard to allow the removal of power from certain subsystems to reduce power consumption. The control of power to add–in adapters will become a possibility as the industry moves to self–identifying adapters. Logic could be added to remove power from adapters during periods of inactivity defined by the power management software.

Another change is that the power indicator which is present on most systems will be used to indicate more than just two states (on and off) of system power. The power management software must be able to control this indicator. It may be desirable to add a suspend indicator and/or a suspend request button. See Section 7 for a description of system Suspend.

Since turning off the power is now the responsibility of the power management software, it may be desirable to provide a watchdog timer. This timer would provide the control signal to the power supply to turn off the secondary voltages in the event that the power management software fails to complete its task. The designer may also wish to include a system wake–up alarm within the power management controller.

The power management logic will not provide I/O activity monitoring for the purpose of determining the idleness of a device or subsystem. In general this is the responsibility of the power management software. However, if the CPU clock is stopped during suspend to save power, logic must be supplied to detect any supported resume events in order to wake up the CPU on these events. Resume events are discussed in Section 7 below.

The power management controller is further discussed in Appendix A. Required changes to the system firmware will be discussed in Section 12.0 below.

In a power–managed system the reliability of packaging and electromechanical storage subsystems must be examined carefully. For example, it may be necessary to specify a hardfile which has been designed and tested to endure more than the normal number of start/stop cycles for use in a power–managed system than would be required in a system without power management.

5.2 New Information Provided to Operating System

The onus is on the hardware designer to describe platforms in enough detail to enable the operating system provider to determine how to manage the system power usage. The system designer must characterize the power usage of subsystems in detail. Not only does the operating system need to know the power utilization of all supported device power states, it also needs to know the penalty in terms of time, power, and wear to move from one device power state to another. The system designer must describe the topology of any power distribution and/or clocking control structures.

This information must be conveyed from the system firmware to the operating system either via residual data (see reference 1 for a description of residual data) or data returned via boot time calls.

5.3 Future Requirements

System designers must work with support chip manufacturers to propagate power management facilities into all motherboard subsystems and work with independent hardware designers to provide add–in adapters which support power management. In addition system designers should encourage electromechanical data storage subsystem manufacturers to consider new methods of conserving power in the larger form–factor devices while mitigating the detrimental effects of wear on device reliability just as they have in the smaller form–factor devices used in portable battery–powered systems.

6.0 Operating System Requirements

Placing the responsibility for the power management policy implementation on the operating system provider leads to some interesting problems for the system designer. The system designer has a vested interest in complying with domestic and international standards for energy conservation. Although certain strategies are bound to reduce the usage of electrical power (such as spinning down any hardfiles which are not in use), no one strategy can guarantee compliance with the EPA's Energy Star guidelines, for example, on all computing systems. Thus each system designer must place requirements on the operating system provider that will insure compliance with these standards.

According to the philosophy presented in this paper, the operating system software bears the responsibility to manage power within the computer system. The following sections elaborate on the specific system level structures and facilities required to carry this out.

7.0 Definition of Power Management System States

Since the system software provides the power management policy of the system, it can implement as many system–level power states as the designer of this software can conceive. However, for consistency the following system power states are recommended. In some implementations one or more of these states may be combined.

7.1 System Power States

The suggested system power states are: Power Management Disabled, Power Management Enabled, Standby, Suspend, Hibernate, and Off.

7.1.1 Power Management Disabled

In this state the system is its most responsive to the application. It may also be the state where power usage is maximum. All devices are inhibited from entering any of their supported lower power states.

7.1.2 Power Management Enabled

The Power Management Enabled state is the state in which a power–managed system spends the majority of its operational time. This state has a very large number of substates because the current power utilization of the system is dependent on the current power state of every device which constitutes the system. While the power management software is in the Power Management Enabled state it is not static, but is actively engaged in the control of the device states with the goal of maximizing some given set of system performance parameters.

In this state the system is completely operational. Various devices may be commanded by the power management software to move to lower power states — either Power Managed or Suspend. (See the definition of device power states below.) This may increase latency when the application(s) require service from these devices. In general, however, the increase in response time should not be noticeable to the user.

7.1.3 Standby

Standby is intended to be the lowest power state where the system is responsive to interrupts from all sources. The CPU is placed in a low power mode if one is available. However, the CPU is still in control of the system and does not require a hard reset on resumption. Power management software will place all I/O devices in either Suspend state or Device Off state (see the definition of device power states below) unless the device is involved in generating or routing interrupts and needs to be in a higher power state. Operational parameters (register values) are retained within the devices. No data loss occurs in Standby. Any I/O interrupt will cause an immediate transition back to the Power Management Enabled State. All communication sessions are maintained in this state.

7.1.4 Suspend

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 will be powered off, if possible. Main memory itself is put in a low power data retention state (e.g., for Dynamic RAM, slow refresh). All I/O devices not necessary to the detection of suspend resume events will be placed in their Off states.

The CPU is not executing instructions and will not respond to external interrupts. The CPU may not retain its internal state, and thus may require a hard reset and subsequent reinitialization to move to a more responsive state.

Communication sessions are not maintained when the system is in the Suspend state. Any communication sessions will have to be reestablished by the user upon returning to the Power Management Disabled or Power Management Enabled states.

The transition from the Suspend state to one of the more responsive states is called Resume. A suspend resume event initiates the transition to the Power Management Enabled state.

7.1.5 Hibernate

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 (usually in compressed form) to a non–volatile secondary storage medium. During this time, devices that are necessary to the process will be brought to their Device On states.

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.

If this preparation for Hibernate process is not successful, due to some internal error or insufficient space on the non–volatile secondary storage medium, the power management software should abort the transition to the Hibernate state and give the user an indication that an error has occurred. The power management software should also allow the user to abort the prepare for hibernate process, especially if this process requires a perceivable amount of time (say, > 30 seconds). This is known as a user–initiated Hibernation Abort event.

If none of the exception conditions described above occur, the system should move immediately to the Hibernate state. In the Hibernate state the system is not operational. All communication channels are inoperative. Power usage will not be zero in this state, but to the user all indication that the system is powered will be absent. Power usage should be at an absolute minimum, especially for battery–powered mobile systems. For non–mobile systems, power may be drawn from a battery or, if the unit is currently attached to an active AC main, from this source.

User input (e.g. depressing the power switch) will be required to leave this state. The CPU will experience a hardware reset. At least an abbreviated Power–On Self Test will be performed. The transition from Hibernation shall be to the Power Management Enabled state. The restoration of a former operational state from the Hibernate state is called Wakeup.

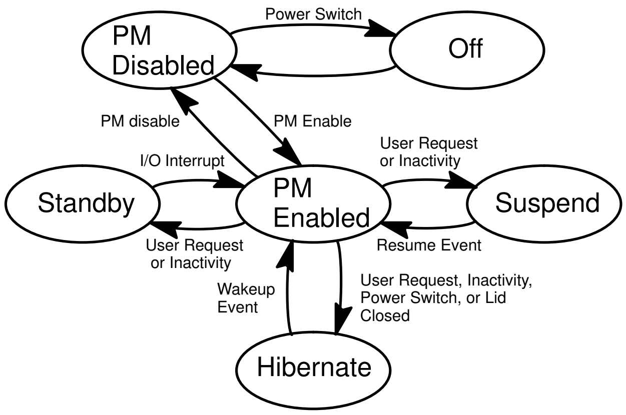
7.1.6 Off

In this state the system is not operational. All communication channels are inoperative. Power usage will not be zero in this state, but to the user all indication that the system is powered will be absent. Power usage should be at an absolute minimum, especially for battery–powered mobile systems. For non–mobile systems, power may be drawn from a battery or, if the unit is currently attached to an active AC main, from this source.

User input (e.g. actuating the power switch) is required to leave this state. The CPU experiences a hardware reset and A Power–On Self Test is be performed. The terminal state of this transition shall be the Power Management Disabled system state.

7.2 Power Management System Power State Transition Diagram

The following figure gives the suggested system power state transitions and the events that cause the transitions.



NOTE: Depressing the Power Switch in the system power–managed states of Standby, or Suspend causes a transition back to PM Enabled to confirm that a transition to the Hibernation state is desired.

Figure 1. Recommended System Power State Transition Diagram

Suggested resume events are

- power switch actuation
- keyboard input
- mouse input on a non–portable system[1]
- click button depression on a mobile computer system
- lid open in computers that have a lid/display which covers the keyboard or pad
- modem ring indicate
- one time or periodic timer alarm

Suggested wake–up events are

- power switch actuation
- modem ring indicate
- timer alarm

7.3 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 many system implementations the CPU will experience a hard reset at the outset of the Resume process. A hard reset causes control of the CPU to be passed to system firmware. Firmware must determine whether control was passed to it as a result of a Resume event or if a cold boot is required. A discussion of firmware considerations is given in Section 12.0 below.

After reinitializing all standard motherboard devices, firmware transfers control to a restart entry point which is passed to firmware via system NVRAM. Code at this entry point will effect a task switch back to the process which was active prior to the system Suspend request.

7.4 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 image of the system state which has been saved to a secondary non–volatile storage medium is brought back into main memory. The remainder of the process proceeds just like the Resume process.

Wakeup may or may not require changes to system firmware depending on the implementation of the boot process. An operating system which normally employs a boot loader will not require firmware support for Wakeup. An operating system which has no boot loader and relies on the firmware to load an executable image of the system software requires modification to the firmware to redirect the loading process to read the saved system image instead of the normal cold boot image.

8.0 Defined Device States

At any given moment the current power utilization depends on the power state of all devices in the system. To optimally utilize certain devices the power management software may need to have special knowledge about a device's power management attributes. However, to deal with devices in a

1. Mouse movement on a portable system is not a reliable indicator of user interaction. Vibration during transportation of portable systems can cause mouse input changes.

uniform way, it is useful to conceptualize device power states.

Devices may have built–in power management functions that are not controllable via software. These functions are beyond the scope of this document since they are transparent to the operation of software.

The following section defines power management states for devices. Devices must implement the full set of power states or a proper subset thereof. A device that implements such a subset must make its capabilities known to its device driver which will handle requests for transitions to unsupported states by substituting an appropriate supported state.

8.1 Device States

The suggested device power states are: On, Power Managed, Suspend, and Off. These are discussed in the next sections.

8.1.1 Device On

In this state a device is fully functional. Latency in servicing requests is at its minimum.

8.1.2 Device Power Managed

The device is in this state is fully operational although some requests for service from the software may experience increased latency. All operational parameters are retained within the device. The device responds to software commands to move to higher or lower power states. Within this state there may be a number of substates that differ in terms of latency and power usage.

8.1.3 Device Suspend

In this state the device is not immediately functional although all internal operational parameters are retained. Power is utilized in the Device Suspend state only to maintain the static state of internal operational parameters. This should be the minimum power usage level other than Device Off.

Restoration to Power Managed and On states is possible, but requires software intervention. This process is called device resume. If suspend involves the freezing of the clock(s) driving the device, power management software needs to control this process by communicating with the parent of the device. See the discussion of the clock distribution tree in Section 11.0 below.

8.1.4 Device Off

In this state power is removed from the device and the device is not functional. Internal operational parameters are not retained by the device. Reinitialization of the device by software is required.

Restoration to Power Managed, Suspend, and On states is possible, but requires software intervention. This process is called device wakeup. Turning device power on or off requires the agency of a third party. This third party is identified as the parent of the device in the power distribution tree. See the discussion of the power distribution tree in Section 11.0 below.

9.0 Power Management Event Sources

A power management event (PM Event) is any event which may affect the power state of a device or the system.

PM Events can be classified as:

- Requests for transition to a more responsive system state (e.g. modem ring indicate, real–time clock alarm, or power switch depressed)
- Timer underflow events indicating a period of software–detected inactivity
- Power loss imminent warnings (battery low or critical, AC outage). These will generally result in an attempt to move to a lower system power state.

A PM Event is abstracted as PM Event Source Object which encapsulates information concerning how the event is signalled in the hardware and how the event is cleared.

The following is a list of some of the possible PM events:

Mouse movement

Keyboard activity

Ring indication

Real–Time Clock Alarm

Backlighting adjusted (user is adjusting the intensity control knob)

Power switch actuation

Hard drive activity

CD–ROM activity

External PM request (could be used to control power remotely)

Battery low

Suspend timer underflow

Hibernate timer underflow

General purpose timer underflow

Lid open, close

PCMCIA socket services required

10.0 Power Management Software Structure

To facilitate the following discussion of architectural aspects of power management software, please refer to Figure 2 which gives a conceptual view of a power management–enabled operating system.

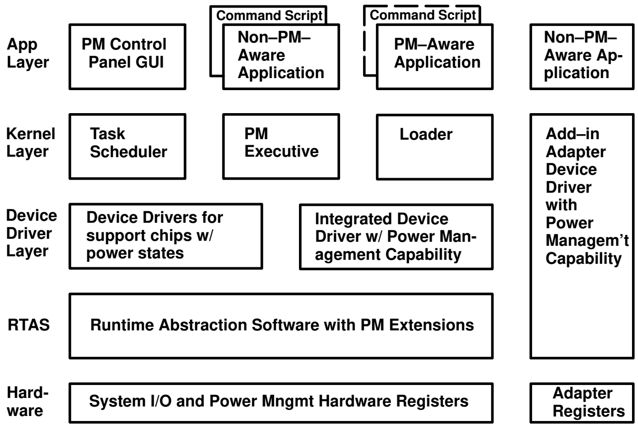


Figure 2. Power Management Software Structure

Power management software should be structured to separate the user interface from the power management policy and to separate the policy from the mechanism. This allows for modularity, code reuse, and ease of differentiating the look (by supplying a new user interface) or the function (by supplying a new policy) of the power management subsystem.

At the highest level this conceptual view shows several types of applications. All of these are currently active and sharing the resources of the computer system. A PM–aware application is one that has been written to take advantage of the power management capabilities of the operating system. Also active in the system may be one or more non–PM–aware applications. A special PM–aware application called the PM Control Panel provides the user's interface to power management. This provides a graphical user interface (GUI) and employs all the interaction metaphors standard to the particular operating system. A command script is an interpretive language used to manage application programs. Embedded within a command script may be a task description which is employed to register the resource requirements of an associated application. This application may either be PM–aware or non–PM–aware.

The central control point for power management is called the PM Executive. The PM Executive receives system power state transition requests from the PM Control Panel, PM–aware applications, and possibly task descriptions embedded in command scripts and must resolve possible conflicts between these requests. It receives event messages from device drivers or via interrupt handlers and broadcasts significant events to the PM–aware applications. The PM Executive generates requests for device power state transitions which it sends to the appropriate device drivers. The PM Executive can be integrated in the operating system kernel, or exist as a kernel extension or privileged user–level application, but must coordinate its activity with that of the Task Scheduler.

The purpose of the Task Scheduler is to select a process from a queue of waiting processes and grant the selected process control of the CPU. The Task Scheduler of a power management enabled operating system uses knowledge about the application mix's resource requirements to optimize performance within the current power constraints. These requirements can come from PM–aware applications directly or be communicated on an application's behalf by a task description language. The Task Scheduler uses this information to build a resource list which it associates with the process set up by the Loader to execute the application. This is maintained in a data structure called the Task List. Task Scheduler informs the PM Executive when various system resources need to be brought to new power states to satisfy the requirements of the executing applications. The Task Scheduler maintains queues of processes waiting on the availability of system resources and cooperates with the PM Executive in making decisions to effect the transition of resources to higher or lower power states.

Although the power management policy may be implemented in a centralized executive, the power management mechanism should be implemented in a distributed manner by power management extensions to the individual device drivers. Device drivers are shielded from hardware differences across system implementations by code called runtime abstraction software (RTAS).

10.1 PM Control Panel

The following specifies some of the user interface requirements for the PM Control Panel.

The PM Control Panel should provide:

- user control of enabling or disabling power management
- a user–selectable option or icon to go immediately to the Suspend system state
- a user–selectable option or icon to immediately go to the Hibernate state
- a dialog box to set the time-out value to trigger a transition to the Suspend state
- a dialog box to set the time-out value to trigger a transition to the Hibernate state
- for systems in which the lid/display folds down and covers the keyboard, a means to select a transition to Suspend, Hibernate, or Off on lid closure
- for all systems, a means to select a transition to Suspend, Hibernate, or Off on power switch actuation
- for all battery–operated systems, a graphical ”fuel gauge” representation of remaining battery energy and/or a digital presentation of estimated time to battery critical low condition
- on battery–operated systems, a selectable option to go to Suspend or Hibernate on a battery critical condition
- a means to enable and disable various resume and wake–up events

10.2 Power Management–Aware Applications

Applications that are written to take advantage of the power management capabilities of the system have one or both of the following characteristics:

- The application registers its resource requirements early in the initialization section.
- The application informs the operating system when its requirements for various system resources change.

Existing applications and new applications which are not written to take advantage of the power management capabilities of the operating system are accommodated in this architecture. These applications can be adapted to take advantage of the power management–capable operating system by supplying a task description to be associated with the application.

10.3 Power Management Extensions to Device Drivers

Device drivers for both integrated and add–in devices require extensions to handle requests from the PM Executive to effect the transition of a device between its supported device power states. The device driver must understand the capabilities of the device it services.

Since most devices do not retain their operational state when power is removed, the device driver is responsible for restoring the operational state of a device after a period when power has been removed. The device driver must either retain the state of any hardware registers which affect the operational characteristics of the device and provide a method for reinitializing these registers, or be able to reconstruct the contents of all device registers algorithmically. If the device driver does not modify the state of a register from the value set by boot time firmware, this restoration process is not required.

The device driver is also responsible for monitoring its attached device(s) for inactivity. The power–management–enabled device driver will signal the PM Executive when a device has been inactive for a specified period of time. The device driver should also support queries from the PM Executive concerning the current level of activity of devices.

Certain devices that do not normally need device drivers (e.g. L2 cache controllers, memory controllers, I/O bridges, programmable interrupt controllers, and DMA controllers) may require them in a power–managed system to control the saving and restoration of the hardware state when these devices are turned off and on. These device drivers also control any supported lower power modes of these devices.

11.0 Power Management and System Configuration

This discussion must assume that there exists a global configuration manager. A configuration manager maintains a possibly dynamic list of all the devices currently configured and available in the system. The power management attributes of all devices must be included in this database. Specifically, the configuration database tells if the integrated device or plug–in adapter and device driver supports power management. It also specifies any power management event sources that might be associated with the device.

This database must describe the topology of any power and/or clock trees. In many systems the distribution and switching of power will have a single control point. However, to make the model more general, power distribution and control is conceptualized as a hierarchical tree. The parent controls all the power to its children.

Also involved in the management of power consumption in a system is the control of clocking. Clocking rates to various subsystems can be controlled. The structure of this control is conceptualized as a second hierarchical tree. Since the power distribution and clocking structures of systems are usually different, the clocking tree is represented in a manner independent of the power tree.

11.1 Power–Managed Device Attributes

The configuration database contains information describing the power management attributes of all the devices currently configured.

For each supported power state of each device the following information is available:

- power dissipation at this state
- power required to make the transition to the next higher or lower state
- delay to complete the transition to the next higher and next lower power state
- any restrictions on the cumulative number or frequency of transitions between device power states due to device wear characteristics

12.0 System Firmware

Power Management must be considered in the design of system firmware. System firmware receives control whenever the CPU sees a hard reset. In a power–managed system, however, a reset to the CPU no longer necessarily means that a cold boot is desired. The firmware must make a determination during the boot process as to whether this is a cold boot or a Resume from System Suspend.

The boot process for a Resume must differ from a cold boot. For example, since main memory contains an image of the system state at the time of suspension, certain areas of memory should not be written by the firmware. Self test routines that write to memory should be skipped or directed to avoid areas of memory known to contain essential data. Also, since the operating system is already resident in memory, it does not need to be loaded from the boot device.

The Wakeup process may require a change to system firmware to support operating systems which do not employ a boot loader.

Firmware must handle changes made to residual data to support the description of the power–managed device attributes. In the future as boot–time calls to firmware from the operating system are supported, firmware must return power attributes in response to device tree queries.

13.0 Summary

This paper has proposed a common framework for power management in both non–mobile AC–powered computer systems as well as battery–operated mobile computers. It has explored how power management principles permeate the hardware, firmware, operating system, and application software of a power–managed system. A new level of cooperation is proposed between power management–aware applications and the power management–capable system software and hardware. This new synergy portends the design of an optimally power–managed system. The user will benefit as the reserve computational capacity of systems increase while at the same time the average power requirements decrease. Ultimately the environment benefits as well.

Reaching this goal, however, requires the cooperation of component manufacturers, system hardware designers, operating system providers, and application software developers.

Appendix A: Power Management Controller System Interface

The power management controller must sense PM events and combine them in software–programmable ways to generate a power management interrupt to the CPU.

The power management controller must provide a command/status interface to the power management software. The following is a list of generic commands:

Generic Commands

- enable/ disable power switch scanning
- read the current (debounced) state of the power switch (open or closed)
- read information that gives the event or reason that power was turned on
- control watchdog timer, if one is provided
- load a value into a timer, reset the timer, determine what event the expiration of a timer will generate (turn power on, generate an interrupt, or just set a bit which software can poll)
- determine which external events generate a PM interrupt
- control and sense any general purpose I/O provided by the PM controller hardware
- control any indicators
- set number of ring indicate pulses required to generate an event

References

1. *PowerPC™ Reference Platform Specification*, Version 1.1, IBM Corporation, Austin, TX, January 1995. Posted on the IBM forum on Compuserv and Prodigy.

2. L. Norford and C. Dandridge, ”Near–Term Technology Review of Electronic Office Equipment”, *Conference Record — IAS Annual Meeting*, vol. 2, 1993.

3. For information on the EPA's Energy Star Partner's program dial the EPA Energy Star Fax line at (202) 233–9659.

4. S. Pancoast and D. Crump, ”White Paper on IBM's Smart Energy System™ and Rapid Resume™”, dated September 2, 1994. Posted on the IBM forum on Prodigy and Compuserv, October 1994.

5. Advanced Power Management (APM) – BIOS Interface Specification, Revision 1.1, Intel Corporation/ Microsoft Coporation, September 1993.