Chapter 6 The Power Manager

This section includes and introduction, a description of the power manager states, power management hints to hardware developers (software developers, see Chapter 2), and a list of power manager ROM calls. ▪

6.1 Introduction

The power manager is an intelligent assistant to the 68000 CPU. The Macintosh Portable power manager provides these functions:

- Power management activities (i.e., enabling or disabling clocks to peripheral chips—like the SWIM chip—to reduce power consumption during idle or sleep periods, and physically enabling or disabling the various power planes).
- Performing the Macintosh real-time clock (RTC) functions.
- Performing the transceiver functions for the Apple Desktop Bus (ADB).
- Generating the contrast signal level (via the PWM function) for the flat panel display.
- Monitoring the temperature (via an A/D input and a thermistor) inside the case to enable the CPU to either warn the customer, or take some other protective action, in the event that the machine becomes too hot.
- Monitoring the level of battery charge so that the customer can be warned if it becomes too low, or if the machine should shut down all activity in order to preserve its memory contents.
- Generating the 1 sec interrupt signal to the CPU.

The power manager communicates with the CPU by using an asynchronous handshake mechanism and an 8-bit parallel data bus in conjunction with the VIA. The power manager primarily responds to commands from the 68HC000.

6.2 Power manager states—idling, sleeping, and waking

An important requirement of any portable computer is to conserve battery power. Macintosh Portable accomplishes this in a number of ways: powering down peripheral subsystems when not needed, slowing (idling condition) or stopping (sleeping condition) the 68000 clock when full speed is not required.

The power manager goes into the sleep state for one of two reasons: it detects a very low battery condition, or it receives the Sleep command from the 68000. The 68000 sends the Sleep command when the Macintosh Portable operating system has determined that there is no user activity or when the user decides to stop work and to shut down the system. Before sending the command, the operating system and drivers save their state variables in RAM. These variables are used later to restore the system when power is restored. The power manager sets a flag indicating that the Sleep command has been received and confirmed, then turns down all system power with the exception of the keyboard processor and the ring-detect circuitry of a modem card, if one is installed.

At the end of the power manager sleep code are two instructions that toggle one of the power manager processor's output lines. This line is connected to a circuit that disables the power manager processor's own 3.9 MHz clock. This essentially "stops time" for the processor, halting its execution and lowering its power consumption by two orders of magnitude. The processor internal state is frozen, with all internal RAM and control registers remaining intact. This is defined as the power manager's **sleep** state—powered, yet stopped.

The 60 Hz external clock used as the basic power manager processor time base, is also used to re-enable the 3.9 MHz clock. The rising edge of the 60 Hz clock re-enables the 3.9 MHz clock, which starts the power manager processor executing code. The power manager resumes execution at the exact spot where it turned itself off and begins the wake-up check loop. First, the clock and times are updated, then the environment is checked for a reason to wake up (return to the operating state).

There are three possible conditions that will end the sleep state:

1. If a key is down on the keyboard
2. If the wake-up timer is enabled and matches the real-time clock
3. If the Macintosh Portable internal modem is installed and set up to watch for "ring detect"

If any one of these conditions is met, the power manager restores itself and the system to normal activity (powers up the 68000, RAM, ROM, etc). When main power returns, the 68000 restores itself and the rest of the system to where it was before the sleep state. This is called **waking**.

BatteryMonitor Code on the 68000

The BatteryMonitor is a segment of code on the 68000 that is part of the ROM image. The BatteryMonitor is called by the 1 sec interrupt (from the power manager processor) and it monitors the CPU for activity. More specifically it

- checks the battery level and looks to see if it is necessary to alert the user that a low-power condition exists
- looks at sleep and hard disk spin-down times (user selected) to see if they've been changed, and updates them as necessary. These times are stored in parameter RAM, which is implemented in the power manager processor.
- turns off the high current drain sound circuit when it is not needed
- updates the real-time clock, which is implemented in the power manager processor
- checks the internal computer temperature to see if it is necessary to alert the user that a high-temperature condition exists
- watches for when it is time to go into the sleep state, that is, it waits for a sleep timeout to occur
- puts the system in the sleep state

Clearly, this code segment does a great deal more than its name implies.

Idle/sleep code function

Idle/sleep code monitors, with the help of other parts of the system ROM, the indications of activity. Any one of these types of activity will prevent a sleep timeout:

- a call to Read or Write through IOCore
- calling PostEvent
- OSEventAvail returning true (events are in the queue)
- making sound (any access of an ADB sound chip)
- execution od an ADB completion routine
- a call to SetCursor that changes the cursor
- whenever the watch cursor is the cursor

Criterion for idle

The criterion for idle is 15 seconds without user activity of any kind (including communication through the serial port; for example, modem use).

Software developers, you need to ensure that routines involving a large amount of calculation perform at least one of the listed types of activity to prevent the power manager commanding the 68000 into the idle state.

In idle, the 68000 processor inserts 64 wait states into RAM and other accesses to lower the processor effective frequency to near 1 MHz, even though its clocking continues at 16 MHz. Interrupts still get processed at 16 MHz, full speed, in the interrupt handler.

Criterion for sleep state

For the sleep state, the variable time used as an inactivity criterion is user selected in the Battery desk accessory (see Figure 4-2). The same measure of inactivity is used for the sleep state as for idle, but in considering the transition from idle to sleep, the user established sleep-time is counted down; if the countdown reaches zero without indication of activity, the Macintosh Portable goes into sleep state.

During sleep there are four sections that remain turned on (powered for full functionality):

1. RAM (both main memory and video memory) is backed up
2. Portions of a modem card, if one is installed
3. The VIA and the SCC are powered but their clocks have been reduced to zero frequency, thereby reducing their power consumption to almost zero.
4. The power manager processor, once the sleep call is made, saves some state variables, systematically powers down all the peripheral subsystems over which it has control (by stopping the system clock signal to them), and then puts itself to sleep (powers itself down).

1/60th of a second interrupts are generated continuously by the 60 Hz clock as long as the battery is charged. When the power manager processor receives an interrupt, it does the following:

- updates the real-time clock
- checks the wake-up timer to see if it matches the real-time clock
- checks for evidence of events that should end the sleep state, such as 1) any keystroke at the built-in keyboard, 2) wake-up timer timeout, or 3) Ring Detect from the modem is true and the modem feature has been enabled at the Control Panel.

If no indication of activity survives the validity check then the power manager powers itself down again. The periodic functions take perhaps 200 microseconds out of the 16.7 milliseconds between interrupts, so the power manager is powered down most of the time.

When someone presses a key or some other event occurs, the power manager will recognize that event, restore itself, hold down (assert) the RESET signal to the 68000, turn on power to the system, and then raise (deassert) the RESET signal.

The 68000 has now been reset and its program counter returned to location $00 0000; the execution of the second instruction asks the question “Is this a return from sleep (a wake-up) or a true reset (from activation of the reset switch) ”. If it's a return from sleep, the program jumps off to the wake-up code, which restores each subsystem to its state before it went into the sleep state and returns to the calling program.

6.3 Power management hints (hardware)

The following hints may be of interest to hardware designers: Chapter 2, “Software Developer Guidelines,” contains hints more likely to be of interest to application designers.

Microprocessor

The 68HC000 is a mostly static cell CMOS design and as such has a DC component to the current drawn during operation (as well as an AC component). This DC component (~10 mA) is always present while power is on to the processor and is the reason the power to the microprocessor is turned off during sleep. For the AC component, three factors determine the amount of current drawn during idle or full speed operation:

- Frequency of operation
- Number of gates switching
- Capacitance to be charged when switching

The frequency of operation cannot be taken to DC because there are some dynamic nodes present in the design. Instead there is a provision (in the Normandy ASIC, $FE xxxx space) to put the processor into an *idle mode* where there are 64 wait states injected into every RAM/ROM cycle. Introduction of the wait states reduces the effective clock frequency for a large portion of the chip. The result of power management is that the number of gates that are switching is reduced as is the frequency at which they switch. The frequency at which internal and external capacitive loads are being charged is also proportionately reduced; therefore the contribution to overall power consumption by address/data/control bus capacitance, and by the RAM and ROM arrays, is reduced.

Permanent main memory

The RAM is a static cell CMOS design and as such has a significant DC component (as well as an AC component) to the current drawn only while it is selected. The RAM array is configured using x8 parts in order to reduce this selected current (only 2 chips are selected in the array, the rest are in standby—no matter how deep the array is). The current drawn while operating is approximately proportional to the duty cycle of use. For instance, when the 68HC000 processor is put into idle mode (64 wait states are injected into each bus cycle) the RAM will be selected only for the minimum amount of time to guarantee proper operation (i.e., the select time remains about the same but the overall duty cycle is reduced dramatically).

ROM memory

The ROM is a static cell CMOS design and as such has a significant DC component to the current drawn only while it is selected (as well as having an AC component). The current drawn while operating is approximately proportional to the duty cycle of use. For instance, when the 68HC000 processor is put into 'idle' mode (64 wait states are injected into each bus cycle) the ROM will be selected only for the minimum amount of time to guarantee proper operation (i.e., the select time remains about the same but the overall duty cycle is reduced dramatically).

Floppy disk interface

The SWIM chip is a static cell CMOS design and as such has a negligible DC component to the current drawn during operation (the major contributing factor is the AC component). For the AC component, three factors determine the amount of current drawn:

- Frequency of operation
- Number of gates switching
- Capacitance to be charged when switching

The strategy for power management on the SWIM chip is to control the frequency of the clock (power is left on to this device even during the sleep state). Specifically it is either on and switching at 16 MHz or it is off (DC). Savings by using this technique have been measured to be about 35 mA. The state of the clock is controlled via the power manager processor through the Normandy ASIC. The normal state of the clock is to be off (thus providing the lowest average power). However, because the floppy drive is not able to interrupt the system when a disk is inserted (and subsequently notify the system to turn the SWIM chip clock on) the system must periodically turn the clock on and check for a disk insert event. Of course, while a disk is inserted and the drive is in operation, the clock should be on.

SCC Interface

The SCC chip is a static cell CMOS design and as such has a negligible DC component to the current drawn during operation (the major contributing factor is the AC component). For the AC component, three factors determine the amount of current drawn:

- Frequency of operation
- Number of gates switching
- Capacitance to be charged when switching

The strategy for power management on the SCC chip is to control the frequency of the clock (power is left on to this device even during the sleep state). Specifically it is either on and switching at 8 MHz or it is off (DC). The state of the clock is controlled via the power manager processor through the Normandy ASIC. The normal state of the clock is to be off (thus providing the lowest average power). However, when the serial ports are in use, the clock must be on 100% of the time.

Video Display Panel Interface

The active matrix display's current drain is dependent upon the screen image. For instance a gray screen may draw up to three times the current of an all white or all black screen. Savings of about 50-70 mA are achievable mainly if the gray desktop screen is avoided. The default screen pattern has been designed to reduce this current drain.

6.4 Operating system interface (ROM calls)

You should already know what the sleep state is and how it is used (see section 6.2 “Power Manager States-Idling, Sleeping, and Waking” for information on the sleep state). Only specific parts of the system should be calling sleep. It is more likely that you want to get into the sleep queue, which is described later in this section.

Calling Sleep

Sleep request

A sleep request is called using the Sleep trap (_Sleep, or $A08A) and it takes one parameter in D0, as shown below.

```
                MOVE.L #SleepRequest,D0         ; SleepRequest = 1
                _Sleep                          ; $A08A
                BNE.S           @didnotsleep    ; Sleep indicator in the
EQ                                                      ; condition code
@didsleep       .........
```

All registers except D0 are preserved across the call.

A sleep request is only that, a request. If the system determines that there is a reason why the system should not go to sleep, the request is denied and the EQ condition code is set to not equal.

Entities that request sleep are sleeptimer, Finder, and Shutdown.

Sleep demand

A sleep demand is called using the Sleep trap (_Sleep, or $A08A) and it takes one parameter in D0, as shown below.

```
MOVE.L #SleepDemand,D0      ; SleepDemand = 2
_Sleep                      ; $A08A
```

All registers except D0 are preserved across the call.

A sleep demand is unconditional. The Macintosh Portable will go into the sleep state even if some processes may be harmed. Sleep demand is generally used to power off the machine when a critical low-power condition arises.

The entity that demands sleep is low-power alert.

The Sleep Queue

As part of its preparation for sleep, the sleep trap goes through a queue of procedures, executing each one and informing the procedure of its intentions. The sleep trap may be trying to do any of three things: request permission for sleep, alert for impending sleep, or inform the procedure that wake-up is underway.

Entities that wish to be given CPU time before or after sleep must place themselves in this queue. The sleep queue is a standard OS queue. Shown below is the sleep queue record structure.

```
TYPE   SleepQRec = RECORD
                     SleepqLink:         QElemPtr;
                     SleepqType:         Integer;
                     SleepqProc:         ProcPtr;     {Pointer to sleep routine}
                     SleepqFlags:        Integer;
               END
```

The sqFlags field contains two flags to be set by the sleep routine before installing its record into the queue.

Flag	Bit	Indication
Sleep	0	Call procedure at sleep
WakeUp	1	Call procedure at wake up

The record owner may suspend calls to its sleep procedure by clearing these bits as desired.

Installing a record

To install a record in the queue:

1. Fill the flags, type, and proc fields in the record
2. Load A0 with a pointer to this record
3. Call SlpQInstall
4. Call enqueue

Example:

```
MOVE.L MyRec(A2),A0                    ; Pointer to my record
LEA          MySleepProc,A1            ; Load pointer to my
                                              handler
MOVE.L A1,SleepqProc(A0)               ; Fill proc field
MOVE         #SlpQType,SleepqType(A0)  ; Fill type field (#16)
_SlpQInstall                           ; Add to queue ($A28A)
```

Removing a record

To remove a record from the queue:

1. Load A0 with pointer to the record
2. Call SlpQRemove

Example:

```
MOVE.L MyRec(A2),A0          ; Pointer to my record
_SlpQRemove                                ; Remove from queue ($A48A)
```

Sleep queue procedures are called with A0 pointing to the procedure's sleep queue record. Procedures in the queue may be called with any of the following values in D0. y.

1 = Sleep request
2 = Sleep demand
3 = Wake up

Sleep Queue calls

A complete sleep call to each queue procedure consists of (in the case of SleepDemand or SleepNow) a single call to each sleep procedure with D0 containing the SleepRequest value or (in the case of SleepRequest) two calls to each sleep procedure, the first passing SleepRequest (in D0) followed by another call with SleepDemand (in D0).

These calls should not be confused with the system level _Sleep calls; the sleep queue procedures are merely called with the same parameter that was passed to the Sleep trap. Sleep queue procedures only see SleepRequest, SleepDemand, or SleepWakeUp passed to them; SleepNow calls are converted to SleepDemand calls when executing sleep queue procedures.

A test of the networking services in use is performed before the sleep queue is traversed. (See “Network Services” below.)

Sleep Request

For sleep request, each queue entry gets called with D0 containing the SleepRequest parameter and A0 pointing to the procedure's sleep queue record. The sleep queue procedure may either allow or deny this request and return its answer in D0. If the procedure allows sleep, the queue is traversed and the next sleep queue procedure is executed. If any one of the procedures denies the sleep request, sleep queue traversal is halted and all procedures that had been called so far get a SleepWakeUp call indicating that the sleep request is being aborted.

It is probably best for entries not to actually do sleep preparation at the time of request, as it is possible that the request may be denied by another sleep queue entry. Instead the queue entry should wait until the sleep demand call is made. SleepRequest is used to check with all entries to see if sleep is OK and to indicate to sleep queue procedure owners that they should lock out any activity that might make it not OK.

To OK the sleep request, the sleep queue entry procedure must clear D0. To deny the sleep request, D0 must be returned as a nonzero.

Once all the sleep queue procedures are called and none of them denies the sleep request, each is called again , this time with the SleepDemand parameter passed in D0. The SleepDemand call cannot be denied.

Sleep Demand

For a SleepDemand or SleepNow system level call, the sleep queue procedures are called once. In either case the SleepDemand parameter is passed in D0 (SleepNow is converted to SleepDemand by the _Sleep trap before executing the queue procedures.) The SleepDemand call cannot be denied. All entries in the sleep queue must do the best they can to prepare for a sleep demand and cooperate fully.

Since the _Sleep trap is not called at interrupt, the procedure can do quite a bit to make sleep possible (even use the Memory mManager!). In some cases it may be desirable to ask the user for an OK or to alert the user of some problems that may occur, however this should be avoided if possible. If many sleep queue procedures decide to put up a dialog, things can get confusing. In addition, if no user is present to answer your dialog, the machine will not go to sleep.

Sleep Wake Up

The SleepWakeUp call is made to each sleep queue entry upon wake up and provides an opportunity to sleep queue procedures to restore their state. This call cannot be denied.

Remember, choosy programs choose sleep.

Network services

A SleepRequest or SleepDemand call must pass a set of network condition tests before the sleep operation can be granted. The network tests take place before sleep queue procedures are called. If the network test fails, the sleep queue procedures are not called. The following matrix illustrates what happens.

Sleep Ca

Network Services	Reque	Deman	Now
MPI	If on battery, then close drive else denied	If user OKs, then close dri else denied	Close driv
XPI	If on battery, then close drive else denied	If user OKs, th close drivers; else denied	Close driv
XPP/ Volume Mounte	Denie	If user OKs, th unmount serv and close driv else denied	Unmount ser and close dri

For the (low priority) SleepRequest, sleep is denied if any network services are actually in use. The exception here is when running on battery where the overriding concern is conserving power. In the demand case things are more interesting.

In the SleepDemand case the user is presented with one of three alerts informing him of the possible consequences if sleep is chosen. Level one alert is shown if the MPP driver or the XPP driver is open, but no servers are mounted nor is the magic chooser bit set. The language of the alert box informs the user of possible loss of networking services. If a server volume is mounted, a more harsh message appears informing of the loss of the volume, and so on. In all cases the user is given the choice of canceling the sleep demand or going ahead with it.

The SleepNow case stomps on everyone and takes no prisinors. No dialogs appear.