Chapter 2 XSYMBIOS/ Power Management

Introduction

Because the PPT 41XX hand-held computer usually operates from battery power, support of power management services and their implementation in applications written for the PPT 41XX have a significant part in Symbol's design of system level software for the PPT 41XX.

The PPT 41XX Power Management subsystem is based on Advanced Power Management (APM) Version 1.0 and can control power to each device independently of the rest of the system. Symbol Technologies' Extended BIOS (XSYMBIOS) includes operating system level power management services that enable applications to control power to individual devices and to the system as a whole.

Advanced Power Management

Advanced Power Management (APM) consists of one or more layers of software that support power management in computers equipped with power manageable hardware. It defines the hardware-independent software interface between hardware-specific power management software and an operating system power management policy. It allows higher-level software to use APM with no knowledge of the hardware interface.

The APM software interface specification defines a layered cooperative environment in which applications, operating systems, device drivers, and the APM BIOS work together to reduce power consumption, extending the life of system batteries and increasing productivity and system availability.

Figure 2-1 provides a schematic of an Advanced Power Management System:

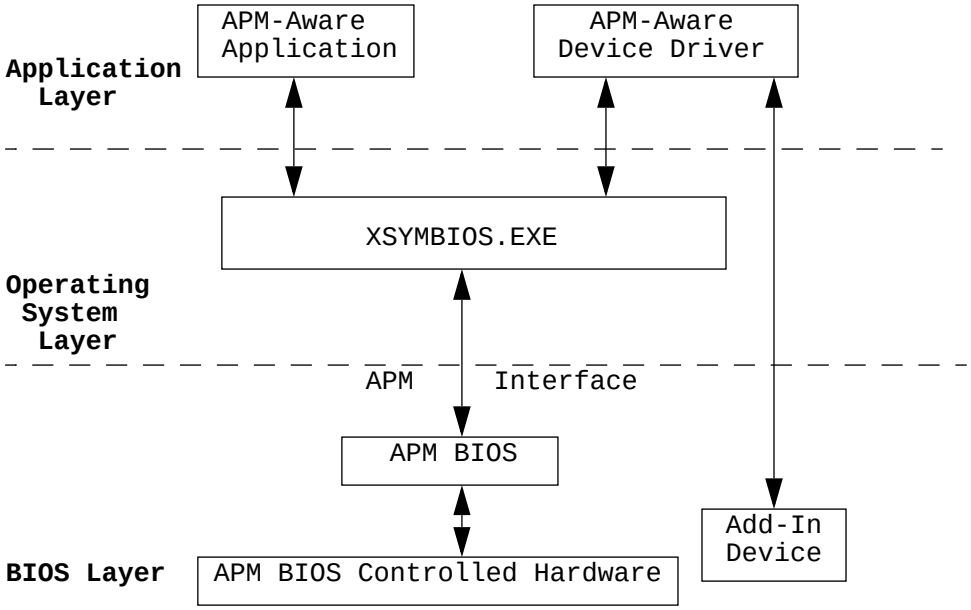

Figure 2-1. Advanced Power Management System

APM partitions power management functions into three cooperating layers shown in Figure 2-1 and standardizes the flow of information and control across these layers. The software components identified in the figure are:

- APM BIOS
  Software interface to the system hardware and its power-managed devices and components.
- APM Interface
  The interaction between XSYMBIOS and the APM BIOS.
- XSYMBIOS
  This software module connects to the APM BIOS, communicates with XSYMBIOS-aware applications, and controls power management policy.
- XSYMBIOS-aware applications
  These application programs interface with XSYMBIOS to monitor and control power management.
- XSYMBIOS-aware device drivers

These program modules provide the power management software interfaces for add-in devices (i.e., one that is not part of the primary system hardware, e.g., a device that can be inserted and removed like a PCMCIA card).

As indicated in Figure 2-1, APM operates at three layers:

- the BIOS layer
- the operating system layer (XSYMBIOS)
- the application layer

The APM BIOS is the lowest level of power management software in the system. APM is incorporated in the BIOS and is specific to the hardware platform. An APM BIOS may provide some degree of power management functionality without any support from operating system or application software. The amount of stand-alone power management functionality is minimal.

The power management functionality of a system is enhanced once an APM driver like Symbol's XSYMBIOS.EXE establishes a cooperative connection to the APM BIOS. This connection allows the firmware to communicate power management events to the APM driver and to wait for APM driver concurrence, if necessary.

XSYMBIOS functions at the operating system layer and has three primary tasks:

1. to pass calls and information between applications and the APM BIOS layer
2. to arbitrate application power management calls in a multitasking environment
3. to identify power saving opportunities not apparent at the application or BIOS layer

XSYMBIOS must regularly poll the APM BIOS to determine whether the APM BIOS has effected state changes or wants a state transition to occur. XSYMBIOS does the appropriate processing to prepare for state changes and then calls the APM BIOS to perform the appropriate power state changes.

XSYMBIOS provides an interface between the APM BIOS and power management aware applications. This interface passes application requests to the APM BIOS and sends APM BIOS power management events up to the applications.

XSYMBIOS specifies a power management-to-application interface.

XSYMBIOS-aware applications aid in power management by providing information that only the application knows or can obtain.

XSYMBIOS-aware applications may register with XSYMBIOS. XSYMBIOS notifies registered applications when system power management events occur, and the applications take suitable actions.

The rest of this chapter deals with the services Symbol has provided in XSYMBIOS to support PPT 41XX power management policy. For more details on the APM BIOS specification, refer to *Advanced Power Management, (APM), BIOS Interface Specification, Revision 1.1*, September 1993, available from either of the following sources:

Intel Corporation
Literature Distribution Center
PO BOX 7641Mt. Prospect, IL 60056-7641
Intel Order Number: 241704-001

Microsoft Corporation
Hardware Vendor Relations Group
1 Microsoft Way
Redmond, WA 98052
Microsoft Part Number: 781-110-X01

PPT 41XX Power Management Subsystem

Symbol's Extended BIOS (XSYMBIOS) provides system-level power management services for PPT 41XX terminals. XSYMBIOS.EXE controls power management for the terminal and provides an application program interface (API) for controlling power management.

This section provides a description of the power management subsystem supported by XSYMBIOS.EXE. It consists of the following subsections:

- *Power Management Overview* describes the function and purpose of the PPT 41XX power management subsystem.
- *Rules for Using the Power Management Subsystem* provides rules for using the power management subsystem in application programs for PPT 41XX terminals.

Power Management Overview

The power management subsystem manages the distribution of power to the system and to individual devices on PPT 41XX terminals. It reduces power consumption which increases battery life without significant impact on the performance of the terminal.

Figure 2-2 (see next page) depicts the PPT 41XX power management subsystem and suggests how it interfaces with DOS applications and with other TSRs (drivers) used with PPT 41XX terminals.

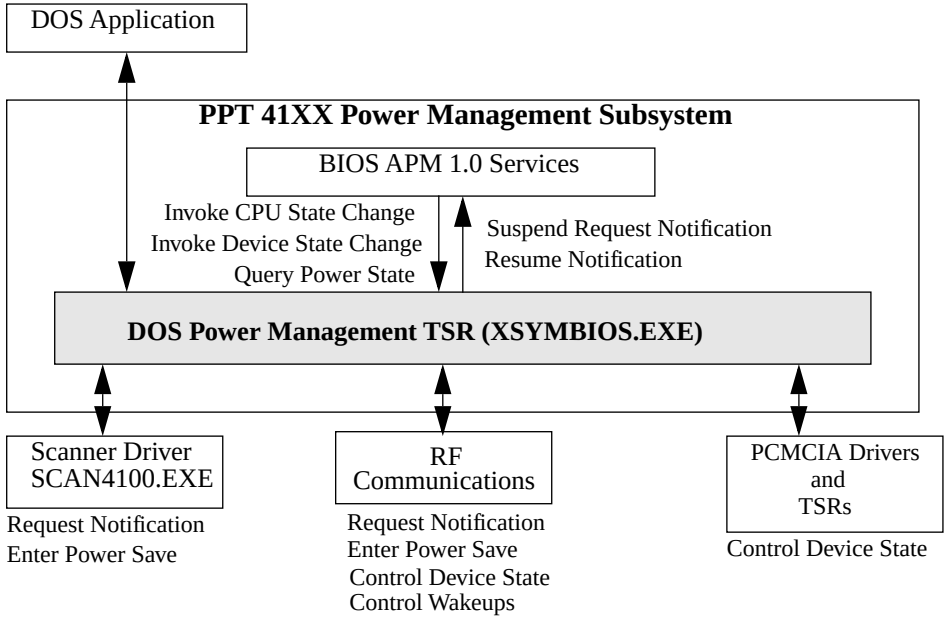

Figure 2-2. PPT 41XX Power Management Subsystem

Devices Managed by the Power Management Subsystem

The devices managed by the power management subsystem are:

- CPU
- LCD (Screen)
- COM1
- COM2
- PCMCIA
- SCAN

Except for "CPU," the names used for these devices are also the text strings used to denote the associated devices throughout the application programming interface (API). Separate API functions are used to control the CPU.

Each device has a maximum of three power consumption states:

- *Active State*
  This is the highest power consumption state for a managed device. In this state the device is fully functional and can operate at full speed. The LCD backlight is ON (unless specifically disabled).
- *Sleep State*
  This is an intermediate power consumption state which can conserve power on some devices. In this state the device can either operate with reduced performance or be quickly restored to the active state. The configuration of the device is preserved. The LCD backlight is OFF.
- *Suspend State*
  This is the lowest power consumption state and is supported by all devices. It is equivalent to the Off state. In this state the configurations of most devices are lost. For the CPU device, the state and the memory are preserved, allowing the program to resume operation when the CPU switches back to active state. For other devices the configuration must generally be restored to return to active state.

Each managed device has two *inactivity timers* associated with it:

- Sleep timer
- Suspend timer

Whenever a device is switched into active state, the inactivity timer starts for the period specified by the sleep timer. If the sleep timer expires, the device switches into sleep state, and the inactivity timer starts for the period specified by the suspend timer. If the suspend timer also expires, the device is automatically suspended.

The CPU is always active when it is executing code, even if the code is only waiting for some event. Sleep state is selected automatically when the BIOS goes into a tight loop waiting for an event (for instance, attempting to get a key from the keyboard when none is present). Any input/output activity detected by XSYMBIOS sets the CPU to active state and restarts the sleep timer. If the sleep timer expires, the CPU sleep state is selected and the suspend timer starts. If the suspend timer expires, *all* the other devices are placed in suspend state prior to suspending the CPU. If the CPU is subsequently activated, the power management subsystem restores the power state of each device in the system to what it was before the CPU was suspended. However, the configurations of devices may *not* be restored correctly. Generally, device drivers should interface with XSYMBIOS to save and restore their device states if these states were not properly preserved.

The following are some exceptions to these rules:

- If an attempt is made to switch a device into sleep state and that device does not support sleep state, the device is made active instead. The timer is processed as if sleep state did exist, i.e., the suspend timer starts.
- Any attempt to select a power consumption state when the next lower state is disabled disables the timer which would cause it to switch to the next lower state.
- Any attempt to select a disabled power consumption state automatically selects the next higher state available and disables the timer which would cause it to switch to the disabled state.

Table 2-1 lists the default states and timers for the system as a whole (CPU) and for those devices that are managed by the power management subsystem.

Table 2-1. System and Device Default States and Timers

Device	Default State	Sleep Timer	Suspend Timer
CPU (System)	Active	5 second2	See **Note 1.**
LCD	Active	See **Note 1**.	Infinite
COM1	Suspend	N / A. See **Note 2**.	Infinite
COM2	Suspend	N / A. See **Note 2**.	Infinite.
PCMCIA	Active	N / A. See **Note 2**.	Infinite
SCAN	Active. See **Note 3**.	N / A. See **Note 2**	See **Note 2**.

Note 1: This parameter is defined by Setup and obtained from CMOS.

Note 2: These devices do not support a sleep state.

Note 3: The current SCAN device does not support any power states. The Scanner Driver (SCAN4100.EXE or SCAN4122.EXE) controls power to the scanner.

In addition to the automatic processes performed by the power management subsystem, XSYMBIOS.EXE supports the following methods for controlling power management:

- Symbol Technologies proprietary power management application programming interface (API). See *Power Management API Commands (List)* and *Power Management API Commands (Descriptions)*

  This API enables / disables automatic operation at the device level and manually overrides automatic operation.
- Advanced Power Management (APM)

The API supported by XSYMBIOS.EXE allows applications to control the power to each device directly and to define how XSYMBIOS.EXE performs automatic power management when devices are inactive.

The highest level of power saving is achieved when the application switches off all unused devices and sets all used devices in the lowest possible power state, as determined by the application's requirements. However, most applications do not do this. To save power with minimal involvement from applications, XSYMBIOS.EXE monitors device activity and automatically switches inactive devices into progressively lower power states, while allowing applications to set the inactivity timers for each device and to define which power states can be selected. XSYMBIOS.EXE controls power to the system by interfacing to APM (APM version 1.0 is built into the BIOS on the PPT 41XX).

XSYMBIOS.EXE switches the system into sleep state when no activity is detected for a specified period.

YSYMBIOS.EXE has been provided for testing applications on the development PC. This program is functionally equivalent to XSYMBIOS, but ignores any commands specific to the PPT 41XX terminal. For a description of YSYMBIOS, see *YSYMBIOS–PC-Loadable Version of XSYMBIOS* in Chapter 1.

Rules for Using the Power Management Subsystem

The following rules for using the power management subsystem in application programs for PPT 41XX terminals allow application developers to achieve low power consumption with good performance. Failure to follow these rules may reduce battery life or present severe performance problems.

Rule 1. Set power management operating parameters before starting an application or as part of the application's initialization.

Identify the devices used by the application. For each device, set the required sleep and suspend timer values. If the application controls the device power states (active, sleep, or suspend) *manually*, set all devices to the lowest power state that allows the program to operate. If power management is *automatic*, set all required devices to active. Manual power management is generally more effective than automatic because it allows power to be turned on and off "just in time" rather than relying on timers.

Caution

By default, the COM1 device is set to the suspend state. Any attempt to use the COM1 device without making it active will fail since in the suspend state no power is supplied to COM1 support hardware. Accessing the serial device by going directly to

the hardware (the method usually used by PC applications) does *not* make the device active. See the descriptions of COM1ON.COM and COM1OFF.COM in the *PPT 41XX Product Reference Guide (PRG).*

Rule 2. Control power to devices and the system from within the application.

Applications are often aware of power saving opportunities before they can be detected by XSYMBIOS.EXE. Considerable power saving can be achieved by direct control from the application.

There is a producer/consumer relationship in an event-driven polled system. A consumer of events identifies opportunities for power management (since it knows when none of its desired events has yet occurred). As the primary controlling foreground process, the consumer also periodically polls power management to allow detection of asynchronous power management activities such as timeouts, power switch presses, etc.

If an event consumer is going to execute a long compute-intensive operation, it may need to indicate system activity to prevent unwanted power management from occurring until it has completed its processing. In such a case, the compute-intensive code section should also include power management polling to ensure that asynchronous power management activities are not delayed during the extended processing period. The following code fragment illustrates these points:

```
.
.
.
while (!done)
{
   event_detected = FALSE;
   for (i = 0; i < number_of_events; i++)
        if (event_occurred (i))
        {
           event_detected = TRUE;
           process_event (i);
        }
        if (!event_detected)
           save_power( );
   poll_power_management ( );
}
.
.
.
void process_event (int i)     /* For long processing events* /
{
   int old_sleep_timer = get_system_sleep_timer ( );
   set_system_sleep_timer (-1);
   make_system_active ( );
   while (more_event_work_to_do)
   {
        power_management_poll ( );
        some_event_work ( );
   }
   set_system_sleep_timer (old_sleep_timer);
}
```

where:

get_system_sleep_timer is implemented via **INT B1h Function 0x1400**

set_system_sleep_timer is implemented via **INT B1h Function 0x1402**

make_system_active is implemented via **INT B1h Function 0x09**

save_power is implemented as one of the following:

- *Multiplex Application Idle API Call*
  This API call (**INT 0x2F Function 0x1680**) indicates the calling application wishes to relinquish its execution privilege for an unspecified time duration. This is the recommended method for identifying opportunities for power management. This method does not rely on the presence of XSYMBIOS (although XSYMBIOS must be present for power savings to occur).
- *Direct XSYMBIOS Sleep Request API Call*
  This API call (**INT 0xB1 Function 0x0B**) is similar to the Multiplex Application Idle API call, but is less convenient to use as it requires a check for the presence of XSYMBIOS before it is used.

poll_power_management ( ) is implemented as one of the following:

- *DOS Idle API Call*
  This API call (**INT 0x28**) is the recommended method of polling for asynchronous power management events. This method does not rely directly on the presence of XSYMBIOS (although XSYMBIOS must be present for any events to be detected).
- *Direct XSYMBIOS Power Management Poll API Call*
  This API call (**INT 0xB1 Function 0x15**) is similar to the DOS Idle API call, but it is less convenient to use as it requires a check for the presence of XSYMBIOS before it is used.

Rule 3. Indicate system activity, especially in background interrupt service routines.

XSYMBIOS.EXE uses various hardware interrupts to control the timers and monitor system activity. If an application or a driver replaces the interrupt service routines for these interrupts, XSYMBIOS.EXE can not detect activity and may unexpectedly power down devices (or the system). XSYMBIOS.EXE also monitors software interrupts 0x10, 0x13, 0x16, 0x21, and 0x2F to detect system activity and identify power saving opportunities.

In an event-driven polled system (as described in Rule 2), a producer of an event should *prevent* unwanted power management since the production of an event indicates the need for activity.

There are three ways in which the producer of an event can indicate system activity, preventing unwanted power management. The first two methods apply primarily to producers operating in the background (i.e., producers that detect the occurrence of the events they produce as a result of interrupts). The third method applies primarily to producers operating in the foreground (i.e., producers that detect the occurrence of the events they produce as a result of polling activity performed within the main foreground polling loop of the consumer). The three methods are described as follows:

1. *Hardware Interrupt Interception and Chain On*
   XSYMBIOS intercepts many hardware interrupts and automatically indicates system activity. The producer "chains on" to the interrupt it uses rather than "taking over" the interrupt, allowing XSYMBIOS to continue to monitor the interrupt so the producer doesn't need to modified its interrupt service routine. This is the easiest solution for background producers.
2. *Reset Inactivity Timers API Call*
   This API call (**INT 0xB1 Function 0x16**) controls the power state of the CPU and the LCD.

   This method is used in cases where it is not practical to "chain on" to the interrupt (an example of such a case is when DOS or the BIOS contains a "default" interrupt handler for the IRQ being used. "Chaining on" to the interrupt allows this "default" interrupt handler to execute *after* the user interrupt handler, which may cause undesired results.)

3. *Make System Active API Call*
   This API call (**INT 0xB1 Function 0x09**) is the method recommended for a foreground producer to indicate system activity. This call must be used only by a foreground (non-interrupt service routine) process since it is non-reentrant.

Rule 4. Use caution when preventing devices or the system from sleeping or suspending.

When absolutely necessary, **INT 0xB1, Functions 0x04**, **0x05**, **0x06**, and **0x07** can disable devices or the system from entering the sleep or suspend states. Use these functions with caution. To avoid the risk of missing a critical asynchronous power management event, do not leave suspend disabled for more than a few milliseconds. Always use the disable and enable calls in pairs and ensure that a disable is always followed by an enable once the critical section of code is completed.

Rule 5. On exit from an application program, restore the original power management parameters.

Any application program that exits should restore the power management parameters to those that were in effect when the program started.

Power Management API Commands (List)

In the power management API supported by XSYMBIOS.EXE all services are accessed through **INT 0xB1** with a function (command) code supplied in the AH register and parameters (if any) passed in registers. On return from calls to these services:

- if no error has been detected, the Carry Flag is cleared
- if an error is detected, the Carry Flag is set and the AX register contains an error code

Table 2-2 lists the power management services supported by the PPT 41XX power management subsystem. The list is sorted by the hexadecimal numeral for the function code the application assigns to the AH register when it invokes the service via **INT 0xB1**. These services and their parameter values and definitions are described in the following section.

Table 2-2. Power Management API Commands (INT 0xB1)

Function Code	Power Management Service Name
0x00	*Suspend System*
0x01	*Set Wakeup Masks*
0x02	*Get Wakeup Cause*
0x03	*Get Battery Status*
0x04	*Device Sleep Disable Control*
0x05	*System Sleep Disable Control*
0x06	*Device Suspend Disable Control*
0x07	*System Suspend Disable Control*
0x08	*Activate Device*
0x09	*Activate System*
0x0A	*Sleep Device*
0x0B	*Sleep System*
0x0C	*Suspend Device*
0x0D	*Suspend System*
0x0E	*Register for Device Suspend Notification*
0x0F	*Register for System Suspend Notification*
0x10	*Register for Device Resume Notification*

Table 2-2. Power Management API Commands (INT 0xB1) (Continued)

Function Code	Power Management Service Name
0x11	*Register for System Resume Notification*
0x12	*Get Power Source*
0x13	*Get/Set Device Timer Value*
0x14	*Get/Set System Timer Value*
0x15	*Poll Power Management*
0x16	*Reset Inactivity Timers*
0x17	*Get/Set Low Battery LED Flash-On Time*
0x18	*Enable/Disable Power Management Poll on INT 0x16*
0x1A	*Get Extended Wakeup Cause*

Power Management API Commands (Descriptions)

The following descriptions of the functions in Table 2-2 are given in function code order.

Note: Prior to using any of the functions of the Power Management API, an application must verify that the XSYMBIOS.EXE TSR is installed by issuing a **Get XSYMBIOS Version Number** call **(INT 0x32, Function 0x85**) to ensure that the major and minor version numbers are not both zero (0).

Issuing an INT 0xB1 when XSYMBIOS is *not* loaded crashes the system.

For a more detailed description of the Get XSYMBIOS Version Number service, see *XSYMBIOS General System Services (INT 0x32) (List)* in Chapter 1.

Suspend System

Function: 0x00

Description

Enables the application to suspend the system and, optionally, to specify a wakeup time either as month-day-hour-minute or as a number of seconds from the current time.

If the terminal cannot power down for any reason, the routine returns an error reply (see the error codes under **Output Registers**). Otherwise, all devices managed by the power management system power down (CPU last). If the system resumes, the power down caller receives a valid reply.

The two subfunctions with alarm wakeup wake up the terminal at the specified time if it has not previously been awakened by some other cause.

Interrupt

0xB1

Input Registers

AH = 0x00

AL = subfunction code, as follows:

0x00: Power down with no alarm wakeup
0x01: Power down with alarm wakeup specified in seconds
0x02: Power down with alarm wakeup specified as date/time

If AL = 0x01

CX = number of seconds to alarm (1 - 3600)

If AL = 0x02:

DH = Month (1 - 12)
DL = Day (1 - 31)
CH = Hour (0 - 23)
CL = Minute (0 - 59)

Output Registers

If no error is detected:

Carry Flag is cleared
AX = 0x0000

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or AL, respectively
: The requested function is disallowed
0x000A: The battery level is too low to allow the terminal to resume

Notes

If the return code is 0x0002, **Function 0x07** (**System Suspend State Control**) or **Function 0x06** (**Device Suspend State Control**) has been used to disallow suspend state on the CPU or on some other device.

Function 0x0D (**Suspend System**) provides identical functionality.

Example

The following code sample illustrates the XSYMBIOS power management services below:

Suspend System (Function 0x00)
Set Wakeup Masks (Function 0x01)
Get Wakeup Cause (Function 0x02)
Get Battery Status (Function 0x03)
Device Sleep Disable Control (Function 0x04)
Device Suspend Disable Control (Function 0x06)
Activate System (Function 0x09)
Register for System Resume Notification(Function 0x11)
Get Power Source (Function 0x12)
Register for System Suspend Notification (Function 0x0F)
Get/Set Low Battery LED Flash-On Time (Function 0x17)
Enable/Disable Power Management Poll on INT 0x16 (Function 0x18)

This example is contained in the following file in the PPT 41XX Software Development Kit (SDK):

c:\SDK4100\SAMPLES\MANUAL\MANUAL\CHAP2\POWER1.C

where **c:\SDK4100** is the default installation directory.

```
/* Include Files ************************************************/

#include <stdio.h>
#include <dos.h>

/* Defines ***********************************************/

#ifndef FALSE
#  define FALSE 0
#endif

#ifndef TRUE
#  define TRUE !FALSE
#endif
```

```
#define PWR_MGMT_INT          0xB1   /* XSYMBIOS Power Management */
                                     /* Services                  */
#define XSYMBIOS_INT          0x32   /* XSYMBIOS General System   */
                                     /* Services                  */
#define PM_SYS_SPND           0x00   /* system suspend            */
#define PM_SET_WAKE_MASK      0x01   /* set wakeup masks          */
#define PM_GET_WAKE_CAUSE     0x02   /* get wakeup cause          */
#define PM_GET_BATT_STAT      0x03   /* get battery status        */
#define PM_DEV_SLEEP_CTRL     0x04   /* device sleep ctrl         */
#define PM_DEV_SUSP_CTRL      0x06   /* device suspend ctrl       */
#define PM_SET_SYS_ACTIVE     0x09   /* set system active         */
#define PM_SYS_RES_NOTIF      0x11   /* system resume notification */
#define PM_GET_PWR_SOURCE     0x12   /* get power source          */
#define PM_SYS_SUSP_NOTIF     0x0F   /* system suspend notification */
#define XB_GET_VERSION        0x85   /* get XSYMBIOS Version      *
#define XB_FLASH_ON_TIME      0x17   /* set low battery LED       */
                                     /* flash-on time             */
#define XB_DISAB_PM_POLL      0x18   /* enable/disable pm poll    */
                                     /*  on INT 0x16              */
```

```
typedef unsigned char BYTE;        /* 8 bit data type      */
typedef unsigned short WORD;       /* 16 bit data type     */
typedef unsigned long DWORD;       /* 32 bit data type     */

typedef enum {ALARM_NONE, ALARM_SECS, ALARM_DATE}
              ALARM_TYPE;
typedef enum {STATE_DSBL, STATE_ENBL, STATE_STAT}
              STATE_CTRL_TYPE;
typedef enum {GET_PWRDN, GET_TIMEOUT, SET_PWRDN, SET_TIMEOUT}
              GETSET_EXT_TYPE;

typedef struct
{
  BYTE month;
  BYTE day;
  BYTE hour;
  BYTE minute;

} DATE_TIME_TYPE;
```

```
typedef struct
{
  unsigned char data[8];
} STORAGE_TYPE;

BYTE suspended = FALSE;

BYTE resumed = FALSE;

/* Public Variables ********************************************** */

union REGS inregs;          /* input regs to int86x    */
union REGS outregs;         /* output regs from int86x  */
struct SREGS segregs;       /* seg regs to/from int86x  */

/* Local Functions Prototypes ************************************ */

WORD pm_SystemSuspend(ALARM_TYPE alarm,   /* alarm wakeup type */
                      WORD seconds,       /* seconds to wakeup */
                      DATE_TIME_TYPE far *date_time);
                                          /* date/time to wakeup */

WORD pm_SetWakeupMasks(BYTE system_mask,   /* mask for system timeout */
                       BYTE pwrdn_mask);  /* mask for normal pwr down */

WORD pm_GetWakeupCause(void);

WORD pm_GetBatteryStatus(void);

WORD pm_DevSleepCtrl(STATE_CTRL_TYPE sleep_ctrl,    /* action flag */
                     char far *device,              /* device name */
                     WORD far *dsbl_count);         /* disable count */

WORD pm_DevSuspCtrl(STATE_CTRL_TYPE susp_ctrl,   /* action flag */
                    char far *device,            /* device name */
                    WORD far *dsbl_count);       /* disable count */

WORD pm_SetSysActive(void);

WORD pm_SysSuspNotif(int action,                    /* action flag  */
                     STORAGE_TYPE far *storage,     /* storage area  */
                     void (interrupt far *routine)());   /* notif routine */
```

```
WORD pm_SysResumeNotif(int action,                          /* action flag   */
                       STORAGE_TYPE far *storage,  /* storage area  */
                       void (interrupt far *routine)()); /* notif routine */

WORD pm_GetPowerSource(void);

void xb_GetVersion(BYTE far *major,        /* major version id */
                   BYTE far *minor);       /* minor version id */

void interrupt far susp_routine();

void interrupt far res_routine();

WORD xb_SetFlashOnTime(BYTE ontime);  /*LED flash-on time*/

WORD xb_EnDisablePMPoll(int action);       /* power Management poll */

int main (void)
{
  BYTE  major;              /* XSYMBIOS major version id */
  BYTE  minor;              /* XSYMBIOS minor version id */
  WORD  batt_stat;          /* Battery status */
  WORD  power_state;        /* Power state */
  WORD  syswakeup;          /* 4100 system wakeup cause */
  DWORD devwakeup;          /* 4100 device wakeup cause */
  WORD dsbl_count;          /* sleep/suspend disable count */
  static STORAGE_TYPE susp_storage;
  static STORAGE_TYPE res_storage;

  /* Obtain and display TSR version */

  xb_GetVersion(&major, &minor);
  printf("XSYMBIOS version is: %x.%2.2x\n", major, minor);

  /* Obtain and display current power source */
  printf("Current power source is: ");
  switch (pm_GetPowerSource())
  {
    case 0:
      printf("battery\n");
      /* If battery power, display battery state */
      batt_stat = pm_GetBatteryStatus();
      printf("Battery status is: ");
```

```
        if (batt_stat & 0x01)
          printf("Power fault\n");
        else if (batt_stat & 0x02)
          printf("Low, level 2\n");
        else if (batt_stat & 0x04)
          printf("Low, level 1\n");
        else
          printf("good\n");
        break;

      case 1:
        printf("Cradle\n");
        break;

      case 2:
        printf("Charger\n");
        break;
    }

pm_DevSleepCtrl(STATE_STAT, "LCD", &dsbl_count);
printf("Current LCD sleep disable count = %i\n", dsbl_count);

/* Disable sleep state for the LCD device*/

pm_DevSleepCtrl(STATE_DSBL, "LCD", &dsbl_count);

/* Perform application-specific operations here */
/* ...                                          */
/* ...                                          */

  /* Enable sleep state */
  pm_DevSleepCtrl(STATE_ENBL, "LCD", &dsbl_count);

  pm_DevSuspCtrl(STATE_STAT, "PCMCIA1", &dsbl_count);
  printf("Current PCMCIA1 susp disable count = %i\n", dsbl_count);

  /* Disable suspend state for the PCMCIA1 device */
  pm_DevSuspCtrl(STATE_DSBL, "PCMCIA1", &dsbl_count);

  /* Perform application-specific operations here */
  /* ...                                          */
  /* ...                                          */
```

```
 /* Enable suspend state */
 pm_DevSuspCtrl(STATE_ENBL, "PCMCIA1", &dsbl_count);

 /* Reset CPU inactivity timer */
 printf("Setting CPU to active state\n");
 pm_SetSysActive();

 /* Setup wakeup mask prior to powering down */
 pm_SetWakeupMasks(0x02, 0x02);

 /* Setup for system suspend and resume notification */
 memset(susp_storage, 0, sizeof(susp_storage));
 pm_SysSuspNotif(TRUE, &susp_storage, susp_routine);
 memset(res_storage, 0, sizeof(res_storage));
 pm_SysResumeNotif(TRUE, &res_storage, res_routine);

 /* Power down unit with wakeup via alarm or trigger */
 printf("Unit will power down and wakeup in 20\n");
 printf("seconds or when left switch is depressed\n");

 printf("Press enter to continue\n");
 getchar();
 pm_SystemSuspend(ALARM_SECS, 20, NULL);

 /* Report wakeup cause */
 printf("Wakeup cause = %04X\n", pm_GetWakeupCause());

/* Report notification results */
 if (suspended)
   printf("Suspend notification occurred\n");
 else
   printf("No suspend notification occurred\n");

 if (resumed)
   printf("Resume notification occurred\n");
 else
   printf("No resume notification occurred\n");

 /* Be sure to remove notification before exiting application */
```

```
  pm_SysSuspNotif(FALSE, &susp_storage, susp_routine);
  pm_SysResumeNotif(FALSE, &res_storage, res_routine);

  return 0;

}


void interrupt far susp_routine()
{
  suspended = TRUE;
}


void interrupt far res_routine()
{
  resumed = TRUE;
}

/********** Suspend System service ***************************/

WORD pm_SystemSuspend(ALARM_TYPE alarm,   /* alarm wakeup type */
                      WORD seconds,       /* seconds to wakeup */
                      DATE_TIME_TYPE far *date_time)
                                          /* date/time to wakeup */
{
  inregs.h.ah = PM_SYS_SPND;
  inregs.h.al = alarm;

  switch (alarm)
  {
    case ALARM_SECS:
      inregs.x.cx = seconds;
      break;

    case ALARM_DATE:
      inregs.h.dh = date_time->month;
      inregs.h.dl = date_time->day;
      inregs.h.ch = date_time->hour;
      inregs.h.cl = date_time->minute;
      break;
```

```
  }

        int86(PWR_MGMT_INT, &inregs, &outregs);

  return outregs.x.ax;
}

/*************** Set Wakeup Masks service ***************************/

WORD pm_SetWakeupMasks(BYTE system_mask, /* mask for system timeout */
                       BYTE pwrdn_mask) /* mask for normal pwr down */
{
  inregs.h.ah = PM_SET_WAKE_MASK;

  inregs.h.ch = system_mask;
  inregs.h.cl = pwrdn_mask;

        int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}

/*************** Get Wakeup Cause service ***************************/

WORD pm_GetWakeupCause(void)
{
  inregs.h.ah = PM_GET_WAKE_CAUSE;

        int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}
```

```
/*************** Get Battery Status service ****************************/

WORD pm_GetBatteryStatus(void)
{
  inregs.h.ah = PM_GET_BATT_STAT;

        int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}

/*************** Device Sleep Disable Control service ******************/


WORD pm_DevSleepCtrl(STATE_CTRL_TYPE sleep_ctrl,/* action flag */
                     char far *device,          /* device name */
                     WORD far *dsbl_count)      /* disable count */
{
  WORD retval;

  inregs.h.ah = PM_DEV_SLEEP_CTRL;

  inregs.h.al = sleep_ctrl;

  segregs.ds = FP_SEG(device);
  inregs.x.dx = FP_OFF(device);

      int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  if (outregs.x.cflag)
    retval = outregs.x.ax;
  else
  {
    *dsbl_count = outregs.x.ax;
    retval = 0;
  }

  return retval;
}
```

```
/*************** Device Suspend Disable Control service ******************/

WORD pm_DevSuspCtrl(STATE_CTRL_TYPE susp_ctrl,/* action flag */
                    char far *device,          /* device name */
                    WORD far *dsbl_count)      /* disable count */
{
    WORD retval;

     inregs.h.ah = PM_DEV_SUSP_CTRL;

    inregs.h.al = susp_ctrl;

    segregs.ds = FP_SEG(device);
    inregs.x.dx = FP_OFF(device);

        int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

    if (outregs.x.cflag)
        retval = outregs.x.ax;
     else
    {
         *dsbl_count = outregs.x.ax;
        retval = 0;
    }

    return retval;
}

/********** Activate System service ******************************/

WORD pm_SetSysActive(void)
{

  inregs.h.ah = PM_SET_SYS_ACTIVE;

       int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}
```

```
/******* Register for System Suspend Notification service ******/

WORD pm_SysSuspNotif(int action,                          /* action flag   */
                     STORAGE_TYPE far *storage,           /* storage area  */
                     void (interrupt far *routine)())     /* notif routine */
{
     inregs.h.ah = PM_SYS_SUSP_NOTIF;
     inregs.h.al = (action ? 0x01 : 0x00);

    segregs.ds = FP_SEG(storage);
    inregs.x.di = FP_OFF(storage);

     segregs.es = FP_SEG(routine);
    inregs.x.bx = FP_OFF(routine);

        int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

     return outregs.x.ax;
}
```

```
/**** Register for System Resume Notification service *****/

WORD pm_SysResumeNotif(int action,                          /* action flag   */
                       STORAGE_TYPE far *storage,           /* storage area  */
                       void (interrupt far *routine)())     /* notif routine */
{
  inregs.h.ah = PM_SYS_RES_NOTIF;
  inregs.h.al = (action ? 0x01 : 0x00);

  segregs.ds = FP_SEG(storage);
  inregs.x.di = FP_OFF(storage);

  segregs.es = FP_SEG(routine);
  inregs.x.bx = FP_OFF(routine);

     int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}

/*********** Get Power Source service **************/

WORD pm_GetPowerSource(void)
{
  inregs.h.ah = PM_GET_PWR_SOURCE;

        int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}
```

```
/********** Get XSYMBIOS Version Number service ********/

void xb_GetVersion(BYTE far *major,          /* major version id */
                   BYTE far *minor)          /* minor version id */
{

     inregs.h.ah = XB_GET_VERSION;

          int86x(XSYMBIOS_INT, &inregs, &outregs, &segregs);

      *major = outregs.h.bh;
     *minor = outregs.h.bl;

      return;
}

/********** Set Low Battery LED Flash-On Time service ********/

WORD xb_SetFlashOnTime(BYTE ontime)       /* LED flash-on time */
{
  WORD retval;

  inregs.h.ah = XB_FLASH_ON_TIME;
  inregs.h.al = ontime;

  int86x(PWR_MGMT_INT, &inregs, &outregs);
  retval = outregs.h.al;

  if (outregs.x.cflag)
  {
    retval = outregs.x.ax;
    printf("Error calling Int B1 Func 17 = %x\n", retval);
   }
  else
    retval = outregs.h.al;

  return (retval & 0x00ff);
}
```

```
/**** Enable/Disable Power Management Poll on INT 0x16 service ***/

WORD xb_EnDisablePMPoll(int action)    /* enable disable power poll */
{
  WORD retval;

  inregs.h.ah = XB_DISAB_PM_POLL;
  inregs.h.al = (action ? 0x01 : 0x00);

  int86x(PWR_MGMT_INT, &inregs, &outregs);

  if (outregs.x.cflag)
    retval = outregs.x.ax;
  else
  {
    if (action == 0x80)
      retval = outregs.h.al;
  }

  return retval;
}
```

Set Wakeup Masks

Function: 0x01

Description

Enables an application to specify the wakeup events that can power up the terminal after a system time-out or a normal power down.

Interrupt

0xB1

Input Registers

AH = 0x01

CH = Wakeup mask for resume after system time-out

CL = Wakeup mask for resume after normal power down

The wakeup masks are encoded as:

Bit 7: Reserved - must be 0
Bit 6: Reserved - must be 0
Bit 5: Reserved - must be 0
Bit 4: Reserved - must be 0
Bit 3: RS232 ring
Bit 2: Pen touch
Bit 1: Left trigger
Bit 0: Right trigger

Encode a bit value of 1 for each selected wakeup event and a bit value of 0 for each event not selected.

Output Registers

If no error is detected:

Carry Flag is cleared
AX = 0x0000

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code passed in register AH
0x0002: Invalid wakeup mask specified

Example

See the **Example** for **Suspend System (0x00)** for a code sample that illustrates the **Set Wakeup Masks** service.

Get Wakeup Cause

Function: 0x02

Description

Returns the cause of the last system wakeup. See **Notes**.

Interrupt

0xB1

Input Registers

AH = 0x02

Output Registers

AX contains the wakeup cause encoded as follows:

Bit 11: Power fault resume
Bit 9: System boot
Bit 8: System resume
Bit 7: Alarm wakeup
Bit 6: Cradle removal (always enabled)
Bit 5: Cradle insertion (always enabled)
Bit 4: Power switch
Bit 3: RS232 ring
Bit 2: Pen touch
Bit 1: Left trigger
Bit 0: Right trigger

One bit is set from bits 11 through 8, and one or more bits are set from bits 7 through 0. All other bits and the Carry Flag are set to 0.

There are no error codes returned from this service.

Notes

Applications that need to monitor Spectrum24 wakeup should use **Get Extended Wakeup Cause** (**0x1A**).

Example

See the **Example** for **Suspend System** **(0x00)** for a code sample that illustrates the **Get Wakeup Cause** service.

Get Battery Status

Function: 0x03

Description

Returns the current status of the battery. See **Notes** section below.

The status conditions reported by this service are:

- *Low Battery Warning, Level 1*
  This indicates that the battery is *starting to get low*, but *not* in imminent danger of failing.
- *Low Battery Warning, Level 2*
  This indicates that the battery is very low and should be *replaced or re-charged as soon as possible*. Failure to replace or re-charge the battery within a short time results in a power fault. Five seconds after Low Battery Warning, Level 2, the unit is automatically suspended if XSYMBIOS is loaded.
- *Power Fault*
  This indicates that the battery is n*ot capable of sustaining the normal operation* of the terminal. It causes a non-maskable interrupt, so the power fault status is not visible to applications. When it detects a Power Fault condition, the power management system shuts down the terminal automatically with no notification. When the terminal resumes, the peripheral devices may need to be re-initialized.

Interrupt

0xB1

Input Registers

AH = 0x03

Output Registers

AX contains the status of the battery encoded as follows:

Bit 2 = 1 indicates a *low battery warning, level 1*.
Bit 1 = 1 indicates a *low battery warning, level 2*.
Bit 0 = 1 indicates a *power fault*.

No errors are returned by this service. The Carry Flag is always clear.

Notes

This function returns a bit mask. Therefore, more than one bit is set in the case of warning level 2.

If the power fault bit is set, the system automatically suspends to prevent data loss. The system does not resume until the power fault condition clears.

If the Low Battery Warning, Level 2 bit (Bit 1) is set, the system does not resume until Bit 1 clears.

Example

See the **Examples** provided for **Suspend System (0x00)** and **Get Power Source (0x12)** for code samples that illustrate the **Get Battery Status** service.

Device Sleep Disable Control

Function: 0x04

Description

Controls access to the sleep state for a specified device. This service does *not* apply to the CPU (see **System Sleep Disable Control**, **Function 0x05**).

This service contains three subfunctions:

- Disable Sleep State
- Enable Sleep State
- Get Current Sleep State Status

Disable Sleep State increments the count. *Enable Sleep State* decrements the count. *Get Current Sleep State Status* returns the count to the caller.

Sleep state is enabled only when the count is 0, so different tasks can disable sleep state on the device for various reasons. The device is allowed to sleep only when all tasks that have disabled sleep state re-enable it.

Sleep state is normally entered when one of the following conditions occurs:

- an application program issues **INT 0xB1, Function 0x0A**
- no I/O activity is detected on an active device for the period specified by the sleep timer

When sleep state is enabled, the inactivity timer is set to the value specified by the suspend timer, and the device is suspended if no activity is detected before the suspend timer expires.

The Disable Sleep State subfunction prevents sleep state from being selected until all disables are removed. If the device is already sleeping or sleep state is selected while the state is disabled, the device is forced to the active state with no inactivity timer.

Interrupt

0xB1

Input Registers

AH = 0x04

AL = Subfunction code as follows:

0x00: Disable Sleep State
0x01: Enable Sleep State
0x02: Get Sleep State Status

DS:DX = Address of the ASCIIZ string (LCD, COM1, COM2, PCMCIA, or SCAN) that identifies the device to which the service is being applied.

Output Registers

If no error is detected:

Carry Flag is cleared
AX = Number of sleep state disables

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or register AL, respectively
0x0004: Unable to find specified power management device
0x0005: Sleep / Suspend disable count overflow

Notes

This function is one method for disabling sleep state on a device by an application. Another method is to change the inactivity timer to -1 (see **Get/Set Device Timer Value**, **INT 0xB1**, **Function 0x13**), which disables sleep state for the device globally within the system.

In either method, keep sleep state disabled only until the important activity is completed. Using this function may be easier for applications, since an enable call reverses the effect of disable without requiring the prior value of the inactivity timer to be saved and restored. This function allows several applications to cooperate in enabling / disabling.

Applications are discouraged from disabling sleep states, except in situations where maximum performance is required, since this is likely to increase power consumption on more advanced hardware.

Example

See the **Example** for **Suspend System (0x00)** for a code sample that illustrates the **Device Sleep Disable Control** service.

System Sleep Disable Control

Function: 0x05

Description

Controls access to the sleep state for the system.

This service contains three subfunctions:

- Disable Sleep State
- Enable Sleep State
- Get Current Sleep State Status

Disable Sleep State increments a count. *Enable Sleep State* decrements the count. *Get Current Sleep State Status* returns the count to the caller.

Sleep state is enabled only when the count is 0, so different tasks can disable sleep state on the CPU for various reasons. The system is allowed to sleep only when all tasks that have disabled sleep state re-enable it.

CPU Sleep state is normally entered when one of the following conditions occurs:

- an application program issues **INT 0xB1, Function 0x0B**
- an application program issues **INT 0x2F, Function 0x1680**
- certain BIOS calls cause the BIOS to enter a tight loop waiting for an external event to occur
- no I/O activity is detected for a period determined by the CPU sleep timer

When sleep state is enabled, the inactivity timer is set to the value specified by the suspend timer, and the system is suspended if no activity is detected before the suspend timer expires.

Disable sleep state prevents sleep state from being selected until all disables are removed. If the system is already sleeping or sleep state is selected while the state is disabled, the device is forced to the active state with no inactivity timer.

Interrupt

0xB1

Input Registers

AH = 0x05

AL = Subfunction code as follows:

0x00: Disable Sleep State
0x01: Enable Sleep State
0x02: Get Sleep State Status

Output Registers

If no error is detected:

Carry Flag is cleared
AX = Number of Sleep Disable requests

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or register AL, respectively
0x0005: Sleep / Suspend disable count overflow

Notes

This function is one method for disabling system sleep state by an application. Another method is to change the inactivity timer to -1 (see **Get/Set System Timer Value**), which disables system sleep state globally within the system.

In either method, disable sleep state only until the important activity is completed. Using this function may be easier for applications, since an enable call reverses the effect of disable without requiring the prior value of the inactivity timer to be saved and restored. This function allows several applications to cooperate in enabling / disabling.

Currently, system sleep state is implemented by halting the system until an interrupt is detected. Applications that process data for excessive periods without any I/O activity can use this function to disable sleep state to prevent the inactivity timer from selecting the system sleep state. Future releases may use other techniques such as slowing the processor clock speed to implement the system sleep state.

Applications should only disable sleep states in situations where maximum performance is required, since this is likely to increase power consumption on more advanced hardware.

Example

The following code sample illustrates the XSYMBIOS power management services below:

System Sleep Disable Control (Function 0x05)
System Suspend Disable Control (Function 0x06)
Sleep System (Function 0x0B)
Poll Power Management (Function 0x15)
Reset Inactivity Timers (Function 0x16)

This example is contained in the following file in the PPT 41XX Software Development Kit (SDK):

c:\SDK4100\SAMPLES\MANUAL\MANUAL\CHAP2\POWER3.C

where **c:\SDK4100** is the default installation directory.

```
/* Include Files ************************************************/

#include <stdio.h>
#include <dos.h>

/* Defines ******************************************/

#ifndef FALSE
#  define FALSE 0
#endif

#ifndef TRUE
#  define TRUE !FALSE
#endif

#define PWR_MGMT_INT            0xB1    /* power management     */
#define PM_SYS_SLEEP_CTRL       0x05    /* system sleep ctrl    */
#define PM_SYS_SUSP_CTRL        0x07    /* system suspend ctrl  */
#define PM_SLEEP_SYS            0x0B    /* sleep system
#define PM_PWR_MGMT_POLL        0x15    /* get/set system timer value
#define PM_RESET_TIMERS         0x16    /* reset inactivity timers

typedef unsigned char BYTE;             /* 8 bit data type      */
typedef unsigned short WORD;            /* 16 bit data type     */
typedef unsigned long DWORD;            /* 32 bit data type     */

typedef enum {STATE_DSBL, STATE_ENBL, STATE_STAT} STATE_CTRL_TYPE;

/* Public Variables *********************************************/

union REGS inregs;              /* input regs to int86x    */
union REGS outregs;             /* output regs from int86x */
struct SREGS segregs;           /* seg regs to/from int86x */

/* Local Functions Prototypes ***********************************/

WORD pm_SysSleepCtrl(STATE_CTRL_TYPE sleep_ctrl,    /* action flag */
                     WORD far *dsbl_count);         /* disable count */

WORD pm_SysSuspCtrl(STATE_CTRL_TYPE susp_ctrl,      /* action flag */
                    WORD far *dsbl_count);          /* disable count */

WORD pm_SleepSys(void);
```

```
WORD pm_PowerMgmtPoll(void);

void pm_ResetTimers(int lcd_flag);        /* make lcd active flag */

*********************************************************************/

int main (void)
{
  WORD dsbl_count;          /* sleep/suspend disable count */


  pm_SysSleepCtrl(STATE_STAT, &dsbl_count);
  printf("Current system sleep disable count = %i\n", dsbl_count);

  /* Disable CPU sleep state */
  pm_SysSleepCtrl(STATE_DSBL, &dsbl_count);

  /* Perform application-specific operations here */
  /* ...                              */
  /* ...                              */

  /* Enable CPU sleep state */
  pm_SysSleepCtrl(STATE_ENBL, &dsbl_count);


  pm_SysSuspCtrl(STATE_STAT, &dsbl_count);
  printf("Current system susp disable count = %i\n", dsbl_count);

  /* Disable CPU suspend state */
  pm_SysSuspCtrl(STATE_DSBL, &dsbl_count);

  /* Perform application-specific operations here */
  /* ...                              */
  /* ...                              */

  /* Enable CPU suspend state */
  pm_SysSuspCtrl(STATE_ENBL, &dsbl_count);


  /* In order to conserve power, applications should */
  /* call pm_SleepSys within polling loops if no     */
```

```
    /* event is detected                    */
    pm_SleepSys();


    /* Applications should periodically call pm_PowerMgmtPoll */
    /* to allow any pending power management operations to    */
    /* take place                           */
    pm_PowerMgmtPoll();


    /* Although it can be called by applications, pm_ResetTimers    */
    /* is intended to be called by interrupt service routines to    */
    /* prevent the CPU, and optionally the display, from timing out */
    pm_ResetTimers(TRUE);

    return 0;
}

/********** System Sleep Disable Control service ********************/

WORD pm_SysSleepCtrl(STATE_CTRL_TYPE sleep_ctrl,    /* action flag */
                     WORD far *dsbl_count)          /* disable count */
{
    WORD retval;

    inregs.h.ah = PM_SYS_SLEEP_CTRL;

    inregs.h.al = sleep_ctrl;

       int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

    if (outregs.x.cflag)
      retval = outregs.x.ax;
    else
    {
      *dsbl_count = outregs.x.ax;
      retval = 0;
    }

    return retval;
}
```

```
/********** System Suspend Disable Control service **********/

WORD pm_SysSuspCtrl(STATE_CTRL_TYPE susp_ctrl,     /* action flag */
                    WORD far*dsbl_count)           /* disable count */
{
  WORD retval;

  inregs.h.ah = PM_SYS_SUSP_CTRL;

  inregs.h.al = susp_ctrl;

        int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  if (outregs.x.cflag)
    retval = outregs.x.ax;
  else
  {
    *dsbl_count = outregs.x.ax;
    retval = 0;
  }

  return retval;
}

/*************** Sleep System service ******************/

WORD pm_SleepSys(void)
{

  inregs.h.ah = PM_SLEEP_SYS;

     int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}




/********** Poll Power Management service ***************/
```

```
WORD pm_PowerMgmtPoll(void)
{
  inregs.h.ah = PM_PWR_MGMT_POLL;

     int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}

/******* Reset Inactivity Timers service ************************/

void pm_ResetTimers(int lcd_flag)      /* make lcd active flag */
{

  inregs.h.ah = PM_RESET_TIMERS;
  inregs.h.al = (lcd_flag ? 0x01 : 0x00);

       int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return;
}
```

Device Suspend Disable Control

Function: 0x06

Description

Controls access to the suspend state for the specified device. This service does *not* apply to the CPU (see **System Suspend Disable Control**, **Function 0x07**).

This service contains three subfunctions:

- Disable Suspend State
- Enable Suspend State
- Get Current Suspend State Status

Disable Suspend State increments a count. *Enable Suspend State* decrements the count. *Get Current Suspend State Status* returns the count to the caller.

Suspend state is enabled only when the count is 0, so different tasks can disable suspend state on a device for various reasons. The device is allowed to suspend only when all tasks that have disabled suspend state re-enable it.

Suspend state is normally entered when one of the following conditions occurs:

- an application program issues Interrupt 0xB1, Function 0x0C
- the device continues to sleep for the period specified by the suspend timer

The Disable Suspend State subfunction prevents suspend state from being selected until all disables are removed. If the device is already suspended or if suspend state is selected when the state is disabled, the device is forced to the next available higher power consumption state with no inactivity timer.

Interrupt

0xB1

Input Registers

AH = 0x06

AL = Subfunction code as follows:

0x00: Disable Suspend State
0x01: Enable Suspend State
0x02: Get Suspend State Status

DS:DX = Address of the ASCIIZ string (LCD, COM1, COM2, PCMCIA, or SCAN) that identifies the device to which the service is being applied.

Output Registers

If no error is detected:

Carry Flag is cleared
AX = Number of Suspend Disable requests

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or register AL, respectively
0x0004: Unable to find specified power management device
0x0005: Sleep / Suspend disable count overflow

Notes

This function is one method for disabling suspend state on a device by an application. Another method is to change the inactivity timer to -1 (see **Get/Set Device Timer Value**), which disables sleep state for the device globally within the system.

In either method, disable suspend state only until the important activity is completed. Using this function may be easier for applications, since an enable call reverses the effect of disable without requiring the prior value of the inactivity timer to be saved and restored. This function allows several applications to cooperate in enabling / disabling.

Using this function to prevent a device from suspending also prevents the system from being suspended since suspending the system requires that all devices be suspended first. This is acceptable only for short time periods. For this reason, an application that simply needs to prevent a device from suspending due to timeout but that does not want to prevent the device from suspending as the result of a system suspend should either set the suspend timer for the device to -1 (see **Get/Set Device Timer Value, INT 0xB1 Function 0x13**) or periodically set the device to the active state (see **Activate Device**, **INT 0xB1 Function 0x08**) to prevent the timer from timing out.

Example

See the **Example** for **Suspend System (0x00)** for a code sample that illustrates the **Device Suspend Disable Control** service.

System Suspend Disable Control

Function: 0x07

Description

Controls access to the suspend state for the system.

This service contains three subfunctions:

- Disable Suspend State
- Enable Suspend State
- Current Suspend State Status

Disable Suspend State increments a count. *Enable Suspend State* decrements the count. *Get Current Suspend State Status* returns the count to the caller.

Suspend state is enabled only when the count is 0, so different tasks can disable suspend state on the system for various reasons. The system is allowed to suspend only when all tasks that have disabled suspend state re-enable it.

System Suspend state is normally entered when one of the following conditions occurs:

- an application program issues **INT 0xB1**, **Function 0x00** or **Function 0x0D**
- no I/O activity is detected for a period determined by the system suspend timer while the system is sleeping

When suspend state is enabled, the inactivity timer is set to the value specified by the suspend timer, and the system is suspended if no activity is detected before the suspend timer expires.

Disable suspend state prevents suspend state from being selected until all disables are removed. If the system suspend state is selected while the state is disabled, then the system is forced to the next available higher power consumption state with no activity timer.

Interrupt

0xB1

Input Registers

AH = 0x07

AL = Subfunction code as follows:

0x00: Disable Suspend State
0x01: Enable Suspend State
0x02 Get Suspend State Status

Output Registers

If no error is detected:

Carry Flag is cleared
AX = Number of Suspend Disable requests

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or register AL, respectively
0x0005: Sleep / Suspend disable count overflow

Notes

This function is one method for disabling system suspend state by an application. Another method is to change the inactivity timer to -1 (see **Get/Set System Timer Value**), which disables system suspend state globally within the system.

In either method, disable suspend state only until the important activity is completed. Using this function may be easier for applications, since an enable call reverses the effect of disable without requiring the prior value of the inactivity timer to be saved and restored. This function allows several applications to cooperate in enabling / disabling.

Disable suspend state for the system only for short periods of time as it is a drastic measure which prevents the system from powering down until a power fault is detected.

If the terminal is powered by battery and a power fault condition is detected, the system suspends with no notification, even if suspends are disabled. In this condition, failing to suspend would result in unacceptable loss of data.

To prevent timeout rather than to prevent suspend, use **Get/Set System Timer Value** (**INT 0xB1 Function 0x14**) to set the system suspend timer to -1 or, periodically, to force the system to active state by using **Activate System** (**INT 0xB1**, **Function 0x09**) to prevent the timer from expiring. Refer to Rule 2 in *Rules for Using the Power Management Subsystem*.

Example

See the **Example** for **System Sleep Disable Control (0x05)** for a code sample that illustrates the **System Suspend Disable Control** service.

Activate Device

Function: 0x08

Description

Sets a device other than the CPU to the active state.

This function places the device in the highest power consumption state, making it available for use, and starts the inactivity timer at the full value specified by the sleep timer for the selected device (unless sleep state is disabled).

Interrupt

0xB1

Input Registers

AH = 0x08

DS:DX = Address of the ASCIIZ string (LCD, COM1, COM2, PCMCIA, or SCAN) that identifies the device to which the service is being applied.

Output Registers

If no error is detected:

Carry Flag is cleared
AX = 0x0000

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code passed in register AH
0x0004: Unable to find the specified power management device

Notes

This function is called automatically to make LCD active whenever a trigger or a keystroke is detected. All other devices can be switched into the active state only by explicit calls to this function from an application. Making the LCD active switches on the backlight unless the backlight has been disabled. For description of the Get/Set Backlight Brightness service, refer to *XSYMBIOS General System Services (INT 0x32) (List)* in Chapter 1.

Call this function only from the foreground (e.g., *not* from an interrupt handler), since it is *not* re-entrant.

Example

The following code sample illustrates the XSYMBIOS power management services below:

Activate Device (Function 0x08)
Sleep Device (Function 0x0A)
Suspend Device (Function 0x0C)
Suspend System (Function 0x0D)
Register for Device Suspend Notification (Function 0x0E)
Register for Device Resume Notification (Function 0x10)
Get/Set Device Timer Value (Function 0x13)
Get/Set System Timer Value (Function 0x14)

This example is contained in the following file in the PPT 41XX Software Development Kit (SDK):

c:\SDK4100\SAMPLES\MANUAL\CHAP2\POWER2.C

where **c:\SDK4100** is the default installation directory.

```
/* Include Files ************************************************** */

#include <stdio.h>
#include <dos.h>

/* Defines ****************************************** */

#ifndef FALSE
#  define FALSE 0
#endif

#ifndef TRUE
#  define TRUE !FALSE
#endif

#define PWR_MGMT_INT            0xB1    /* power management   */
#define PM_SET_DEV_ACTIVE       0x08    /* set device active  */
#define PM_SLEEP_DEV            0x0A    /* sleep device       */
#define PM_SUSP_DEV             0x0C    /* suspend device     */
#define PM_SUSP_SYS             0x0D    /* suspend system     */
#define PM_DEV_SUSP_NOTIF       0x0E    /* device suspend notification */
#define PM_DEV_RES_NOTIF        0x10    /* device resume notification */
#define PM_DEV_TIMER_VAL        0x13    /* get/set device timer value */
#define PM_SYS_TIMER_VAL        0x14    /* get/set system timer value */

typedef unsigned char BYTE;         /* 8 bit data type       */
typedef unsigned short WORD;        /* 16 bit data type      */
typedef unsigned long DWORD;        /* 32 bit data type      */

typedef enum {ALARM_NONE, ALARM_SECS, ALARM_DATE} ALARM_TYPE;
typedef enum {GET_SLEEP, GET_SUSPEND, SET_SLEEP, SET_SUSPEND}
                GETSET_ACTION_TYPE;

typedef struct
{
  BYTE month;
  BYTE day;
  BYTE hour;
  BYTE minute;
} DATE_TIME_TYPE;
```

```
typedef struct
{
  unsigned char data[8];
} STORAGE_TYPE;

/* Public Variables ******************************** */

union REGS inregs;              /* input regs to int86x    */
union REGS outregs;             /* output regs from int86x */
struct SREGS segregs;           /* seg regs to/from int86x */

BYTE suspended = FALSE;
BYTE resumed = FALSE;

/* Local Functions Prototypes ********************************** */

WORD pm_SetDevActive(char far *device);  /* device name */
WORD pm_SleepDev(char far *device);      /* device name */
WORD pm_SuspDev(char far *device);       /* device name */

WORD pm_SuspSys(ALARM_TYPE alarm,                /* alarm wakeup type */
                WORD seconds,                    /* seconds to wakeup */
                DATE_TIME_TYPE far *date_time);/* date/time to wakeup */

WORD pm_DevSuspNotif(int action,                       /* action flag  */
                     char far *device,                 /* device name  */
                     STORAGE_TYPE far *storage,        /* storage area */
                     void (interrupt far *routine)()); /* notif routine */

WORD pm_DevResumeNotif(int action,                       /* action flag  */
                       char far *device,                 /* device name  */
                       STORAGE_TYPE far *storage,        /* storage area */
                       void (interrupt far *routine)()); /* notif routine */

WORD pm_GetSetDevTimerVal(GETSET_ACTION_TYPE action, /* action flag */
                          char far *device,          /* device name */
                          WORD far *value);          /* value to set/get */

WORD pm_GetSetSysTimerVal(GETSET_ACTION_TYPE action, /* action flag */
                          WORD far *value);          /* value to set/get */

void interrupt far susp_routine();

void interrupt far res_routine();
```

```
/*********************************************************************/
int main (void)
{
  static STORAGE_TYPE susp_storage;
  static STORAGE_TYPE res_storage;
  WORD time_value;

  /* Set the device active to see the full sleep timer count */
  pm_SetDevActive("PCMCIA2");

  /* Report sleep timer value */
  pm_GetSetDevTimerVal(GET_SLEEP, "PCMCIA2", &time_value);
  printf("PCMCIA2 sleep time = %i\n", time_value);

  /* Set the device to sleep to see the full suspend timer count */
  pm_SleepDev("PCMCIA2");

  /* Report suspend timer value */
  pm_GetSetDevTimerVal(GET_SUSPEND, "PCMCIA2", &time_value);
  printf("PCMCIA2 suspend time = %i\n", time_value);

  /* Start the device out active before testing notification */
  pm_SetDevActive("PCMCIA2");

  /* Setup for PCMCIA2 device suspend and resume notification */
  memset(susp_storage, 0, sizeof(susp_storage));
  pm_DevSuspNotif(TRUE, "PCMCIA2", &susp_storage, susp_routine);
  memset(res_storage, 0, sizeof(res_storage));
  pm_DevResumeNotif(TRUE, "PCMCIA2", &res_storage, res_routine);

  /* Suspend and resume device to test notification */
  pm_SuspDev("PCMCIA2");
  pm_SetDevActive("PCMCIA2");

  /* Report notification results */
  if (suspended)
    printf("Suspend notification occurred\n");
  else
    printf("No suspend notification occurred\n");
```

```
  if (resumed)
    printf("Resume notification occurred\n");
  else
    printf("No resume notification occurred\n");

  /* Report system suspend timer value */
  pm_GetSetSysTimerVal(GET_SUSPEND, &time_value);
  printf("System suspend time = %i\n", time_value);

  /* Set CPU sleep count to -1 before long interrupt-less loop */
  /* to prevent CPU from halting during loop                   */
  time_value = -1;
  pm_GetSetSysTimerVal(SET_SLEEP, &time_value);
  /* Perform application-specific long loop here */

  /* Be sure to remove notification before exiting application */
  pm_DevSuspNotif(FALSE, "PCMCIA2", &susp_storage, susp_routine);
  pm_DevResumeNotif(FALSE, "PCMCIA2", &res_storage, res_routine);

  /* Power down with no wakeup alarm */
  printf("Unit will power down with no alarm\n");
  printf("Press enter to continue\n");
  getchar();
  pm_SuspSys(ALARM_NONE, 0, NULL);
  return 0;
}


void interrupt far susp_routine()
{
  suspended = TRUE;
}


void interrupt far res_routine()
{
  resumed = TRUE;
}
```

```
/********** Activate Device service ************************* /

WORD pm_SetDevActive(char far *device)      /* device name */
{

  inregs.h.ah = PM_SET_DEV_ACTIVE;

  segregs.ds = FP_SEG(device);
  inregs.x.dx = FP_OFF(device);

     int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}

/********** Sleep Device service ************************* /

WORD pm_SleepDev(char far *device)        /* device name */
{

  inregs.h.ah = PM_SLEEP_DEV;

  segregs.ds = FP_SEG(device);
  inregs.x.dx = FP_OFF(device);

     int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}

/************** Suspend Device Service ****************** /

WORD pm_SuspDev(char far *device)        /* device name */
{

  inregs.h.ah = PM_SUSP_DEV;

  segregs.ds = FP_SEG(device);
  inregs.x.dx = FP_OFF(device);

     int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);
```

```
  return outregs.x.ax;
}

/*************** Suspend System service ******************/

WORD pm_SuspSys(ALARM_TYPE alarm,              /* alarm wakeup type */
                WORD seconds,                  /* seconds to wakeup */
                DATE_TIME_TYPE far *date_time) /* date/time to wakeup */
{
  inregs.h.ah = PM_SUSP_SYS;
  inregs.h.al = alarm;

  switch (alarm)
  {
    case ALARM_SECS:
      inregs.x.cx = seconds;
      break;

    case ALARM_DATE:
      inregs.h.dh = date_time->month;
      inregs.h.dl = date_time->day;
      inregs.h.ch = date_time->hour;
      inregs.h.cl = date_time->minute;
      break;
  }

     int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}

/******* Register for Device Suspend Notification service ******/

WORD pm_DevSuspNotif(int action,                    /* action flag  */
                     char far *device,              /* device name*/
                     STORAGE_TYPE far *storage,     /* storage area  */
                     void (interrupt far *routine)()) /* notif routine */
{
  inregs.h.ah = PM_DEV_SUSP_NOTIF;
  inregs.h.al = (action ? 0x01 : 0x00);

  segregs.ds = FP_SEG(device);
```

```
  inregs.x.dx = FP_OFF(device);

  inregs.x.di = FP_OFF(storage);

  segregs.es = FP_SEG(routine);
  inregs.x.bx = FP_OFF(routine);

     int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}

/******* Register for Device Resume Notification service ******/

WORD pm_DevResumeNotif(int action,                          /* action flag   */
                       char far *device,                    /* device name   */
                       STORAGE_TYPE far *storage,           /* storage area  */
                       void (interrupt far *routine)())     /* notif routine */
{
  inregs.h.ah = PM_DEV_RES_NOTIF;
  inregs.h.al = (action ? 0x01 : 0x00);

  segregs.ds = FP_SEG(device);
  inregs.x.dx = FP_OFF(device);

  inregs.x.di = FP_OFF(storage);

  segregs.es = FP_SEG(routine);
  inregs.x.bx = FP_OFF(routine);

     int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  return outregs.x.ax;
}
```

```
/******* Get/Set Device Timer Value service ***************/

WORD pm_GetSetDevTimerVal(GETSET_ACTION_TYPE action,/* action flag */
                          char far *device,          /* device name */
                          WORD far *value)           /* value to set/get */
{
  WORD  retval;

  inregs.h.ah = PM_DEV_TIMER_VAL;
  inregs.h.al = action;

  inregs.x.cx = *value;
  segregs.ds = FP_SEG(device);
  inregs.x.dx = FP_OFF(device);

     int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);

  if (outregs.x.cflag)
    retval = outregs.x.ax;
  else
  {
    *value = outregs.x.ax;
    retval = 0;
  }

  return retval;
}

/******* Get/Set System Timer Value service ***************/

WORD pm_GetSetSysTimerVal(GETSET_ACTION_TYPE action,  /* action flag */
                          WORD far *value)            /* value to set/get */
{
  WORD  retval;

  inregs.h.ah = PM_SYS_TIMER_VAL;
  inregs.h.al = action;

  inregs.x.cx = *value;

     int86x(PWR_MGMT_INT, &inregs, &outregs, &segregs);
```

```
    if (outregs.x.cflag)
      retval = outregs.x.ax;
    else
    {
      *value = outregs.x.ax;
      retval = 0;
    }

    return retval;
}
```

Activate System

Function: 0x09

Description

Sets the system to the active state.

If CPU sleep state changes the CPU clock speed, it sets the clock speed to the high value and starts the inactivity timer at the full value specified by the sleep timer for the CPU (unless sleep state is disabled).

Interrupt

0xB1

Input Registers

AH = 0x09

Output Registers

Carry Flag is cleared
AX = 0x0000

Notes

Applications may periodically call this service to reset the inactivity timer if required. Refer to Rule 2 in the *Rules for Using the Power Management Subsystem* section of this chapter.

This function must only be called from the foreground (e.g., *not* from an interrupt handler), since it is *not* re-entrant.

Example

See the **Example** for **Suspend System (0x00)** for a code sample that illustrates the **Activate System** service.

Sleep Device

Function: 0x0A

Description

Places the specified device into the sleep state and processes the inactivity timer for that device. This service does *not* apply to the CPU (see **Sleep System**, **Function 0x0B**).

If sleep state is not available (either the device does not support it or it is disabled), this service makes the device active.

The inactivity timer for the specified device is processed as follows:

- If sleep state is disabled (i.e., the device is made active), the inactivity timer is disabled.
- If suspend state is disabled, the inactivity timer is disabled.
- If sleep state is newly selected, the inactivity timer is set to the full value of the suspend timer for the device. The device is suspended if no activity is detected on the device before the timer expires.
- If sleep state was previously selected, the inactivity timer is allowed to count down from its previous value.

Interrupt

0xB1

Input Registers

AH = 0x0A

DS:DX = Address of the ASCIIZ string (LCD, COM1, COM2, PCMCIA, or SCAN) that identifies the device to which the service is being applied.

Output Registers

If no error is detected:

Carry Flag is cleared
AX = 0x0000

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code passed in register AH
0x0004: Unable to find the specified power management device

Notes

This function is called automatically if the inactivity timer for an active device expires.

The function does not return an error code if sleep state is not available or has been disabled, since these are not true error conditions. In each case, the function makes the device active.

Example

See the **Example** for **Activate Device (0x08)** for a code sample that illustrates the **Sleep Device** service.

Sleep System

Function: 0x0B

Description

Places the system into sleep state and processes the system inactivity timer.

If sleep state is disabled, then this function makes the system active.

The inactivity timer for the system is processed as follows:

- If sleep state for the system is disabled (the system is made active), the inactivity timer is disabled.
- If suspend state for any device (including the system) is disabled, the inactivity timer is disabled.
- If sleep state is newly selected, the inactivity timer is set to the full value of the system suspend timer. (The system is suspended if no I/O activity is detected before the timer expires.)
- If sleep state was previously selected, the inactivity timer is allowed to count down from its previous value.

Interrupt

0xB1

Input Registers

AH = 0x0B

Output Registers

This service always returns AX = 0x0000 with the Carry Flag clear.

Notes

Calling the function sets the CPU into sleep state (if available), but the CPU automatically comes out of sleep state whenever any interrupt occurs.

This function is called automatically by XSYMBIOS at various points where XSYMBIOS goes into a tight loop waiting for an external event to occur.

Applications using a polling loop to wait for activity can make a significant contribution to power management by calling this function.

At the top of the loop, the application should clear a flag indicating that no activity has been detected. The polling loop should then be executed.

Any active processes detected in the loop should set the flag to indicate that an active process was found and set the CPU to the active state (Interrupt 0xB1, Function 0x09). This allows the process to be executed at full speed if the processor clock is slowed in sleep state.

At the bottom of the loop, the activity flag should be checked. If the flag is zero (i.e., no activity detected), the application should call the function to force the CPU into sleep state and then return to the top of the loop.

If the flag is set at the bottom of the loop, the application should return to the top of the loop immediately to check for multiple events without putting the CPU into sleep state.

Example

See the **Example** for **System Sleep Disable Control (0x05)** for a code sample that illustrates the **Sleep System** service.

Suspend Device

Function: 0x0C

Description

Suspends (switches off) the specified device and notifies any registered users that the device is about to be suspended. This service does *not* apply to the CPU (for which see **Function 0x0D**, described below).

If suspend state for the device is disabled, this function selects the lowest available power consumption state for the device (sleep or active).

Interrupt

0xB1

Input Registers

AH = 0x0C

DS:DX = Address of the ASCIIZ string (LCD, COM1, COM2, PCMCIA, or SCAN) that identifies the device to which the service is being applied.

Output Registers

If no error is detected:

Carry Flag is cleared
AX = 0x0000

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code passed in register AH
0x0004: Unable to find the specified power management device

Notes

Registered users are notified only if the device is switched from active or sleep state into suspend state. No error is returned if the device cannot be suspended because suspend state has been disabled.

Example

See the **Example** for **Activate Device (0x08)** for a code sample that illustrates the **Suspend Device** service.

Suspend System

Function: 0x0D

Description

Powers off the terminal.

If the terminal cannot be powered down for any reason, the routine immediately returns an error reply (see the error codes under **Output Registers**). Otherwise, all devices managed by the power management system power down (CPU last). If the system resumes a valid reply returns to the power down caller.

Interrupt

0xB1

Input Registers

AH = 0x0D

AL = subfunction code, as follows:

0x00: Power down with no alarm wakeup
0x01: Power down with alarm wakeup specified in seconds
0x02: Power down with alarm wakeup specified as date/time

If AL = 0x01:

CX = number of seconds to alarm (1 - 3600)

If AL = 0x02:

DH = Month (1 - 12)
DL = Day (1 - 31)
CH = Hour (0 - 23)
CL = Minute (0 - 59)

Output Registers

If no error is detected:

Carry Flag is cleared
AX = 0x0000

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or AL, respectively
0x0002: The requested function is disallowed
0x000A: The battery level is too low to allow the terminal to resume

Notes

The functionality of this service is identical to that of **Function 0x00**, described above.

If the return code is 0x0002, then **Function 0x07** (**System Suspend State Control**) or **Function 0x06** (**Device Suspend State Control**) has been used to disallow suspend state on the CPU or on some other device.

Example

See the **Example** for **Activate Device (0x08)** for a code sample that illustrates the **Suspend System** service.

Register for Device Suspend Notification

Function: 0x0E

Description

Registers or de-registers with the power management subsystem for notification when the specified device is about to be suspended. This service does *not* apply to the CPU (for which see **Function 0x0F**, described below).

Interrupt

0xB1

Input Registers

AH = 0x0E

AL = subfunction code as follows:

0x00 specifies de-register
0x01 specifies register

DS:DX = Address of the ASCIIZ string (LCD, COM1, COM2, PCMCIA, or SCAN) that identifies the device to which the service is being applied

DS:DI = Pointer to an 8-byte storage area (See ***Notes*** below)

ES:BX = Pointer to the function to be called by Subfunction 0x01 prior to suspending the specified device

Output Registers

If no error is detected:

Carry Flag is cleared
AX = 0x0000

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or AL, respectively
0x0004: Unable to find the specified power management device

0x0006: Invalid storage area specified
0x0007: User already registered for notification
0x0008: User not registered for notification

Notes

The storage area pointed to by DS:DI must be static and set to 0 before registering with this function.

After registering, the caller must not modify the storage area prior to de-registering.

When de-registering, the same storage area that was used to register must be supplied to the function.

Any registrations from an application must be de-registered prior to deleting the application.

If these rules are not followed, the system will probably crash if the device resumes.

Return codes 0x0006, 0x0007, and 0x00008 indicate that the storage area pointed to by DS:DI is invalid. If the storage area is corrupted, incorrect error codes return.

There is no limit to the number of users that may register for notification on any device.

Once registered for notification, the specified routine is called prior to suspending the device with AX containing a code indicating the reason for the suspension as follows:

Code	Explanation
0x0001	Device timed out.
0x0004	Device shut down by program command.

The notification routine should terminate with an IRET instruction after restoring all registers to the values supplied when the routine was called.

The notification routine should not make any assumptions about the amount of space available on the stack.

Example

See the **Example** for **Activate Device (0x08)** for a code sample that illustrates the **Register for Device Suspend Notification** service.

Register for System Suspend Notification

Function: 0x0F

Description

Registers or de-registers with the power management subsystem for notification when the system is about to be suspended.

Interrupt

0xB1

Input Registers

AH = 0x0F

AL = subfunction code as follows:

0x00 specifies de-register
0x01 specifies register

DS:DI = Pointer to an 8-byte storage area (See ***Notes*** below)
ES:BX = Pointer to the function to be called prior to suspending the system (Subfunction 0x01 only)

Output Registers

If no error is detected:

Carry Flag is cleared
AX = 0x0000

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or AL, respectively
0x0006: Invalid storage area specified
0x0007: User already registered for notification
0x0008: User not registered for notification

Notes

The storage area pointed to by DS:DI must be static and set to 0 before registering with this function.

After registering, the caller must not modify the storage area prior to de-registering.

When de-registering, the same storage area that was used to register must be supplied to the function.

Any registrations from an application must be de-registered prior to deleting the application.

If these rules are not followed, the system will probably crash if the device resumes.

Return codes 0x0006, 0x0007, and 0x0008 indicate that the storage area pointed to by DS:DI is invalid. If the storage area is corrupted, incorrect error codes are returned.

There is no limit to the number of users that may register for notification.

Once registered for notification, the specified routine is called prior to suspending the device with AX containing a code indicating the reason for the suspension as follows:

Code	**Explanation**
0x0002	System timed out.
0x0004	System shut down by program command.

Any necessary device suspend notifications are processed by the system suspend notification.

The notification routine should terminate with an IRET instruction after restoring all registers to the values supplied when the routine was called.

The notification routine should not make any assumptions about the amount of space available on the stack.

Example

See the **Example** for **Suspend System (0x00)** for a code sample that illustrates the **Register for System Suspend Notification** service.

Register for Device Resume Notification

Function: 0x10

Description

Registers or de-registers with the power management subsystem for notification when a power management device has resumed (i.e., been switched out of suspend state). This service does *not* apply to the CPU (see **Register for System Resume Notification**, **Function 0x11**).

Interrupt

0xB1

Input Registers

AH = 0x10

AL = subfunction code as follows:

0x00 specifies de-register
0x01 specifies register

DS:DX = Address of the ASCIIZ string (LCD, COM1, COM2, PCMCIA, or SCAN) that identifies the device to which the service is being applied

DS:DI = Pointer to an 8-byte storage area (See ***Notes*** below)

ES:BX = Pointer to the function to be called by Subfunction 0x01 after resuming (i.e., being switched out of suspend state)

Output Registers

If no error is detected:

Carry Flag is cleared
AX = 0x0000

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or AL, respectively

0x0004: Unable to find the specified power management device
0x0006: Invalid storage area specified
0x0007: User already registered for notification
0x0008: User not registered for notification

Notes

The storage area pointed to by DS:DI must be static and set to 0 before registering with this function.

After registering, the caller must not modify the storage area prior to de-registering.

When de-registering, the same storage area that was used to register must be supplied to the function.

Any registrations from an application must be de-registered prior to deleting the application.

If these rules are not followed, the system will probably crash if the device resumes.

Return codes 0x0006, 0x0007, and 0x0008 indicate that the storage area pointed to by DS:DI is invalid. If the storage area is corrupted, incorrect error codes are returned.

There is no limit to the number of users that may register for notification on any device.

Once registered for notification, the specified routine is called after the device is switched out of suspend state with AX containing a code indicating the reason for the resumption as follows:

Code	Reason for Resumption
0x0100	Device powered up by system resume.
0x0400	Device powered up by program.
0x0800	System resumed after power fault.

The notification routine should terminate with an **IRET** instruction after restoring all registers to the values supplied when the routine was called.

The notification routine should not make any assumptions about the amount of space available on the stack.

Example

See the **Example** for **Activate Device (0x08)** for a code sample that illustrates the **Register for Device Resume Notification** service.

Register for System Resume Notification

Function: 0x11

Description

Registers or de-registers with the power management subsystem for notification when the system is resumed.

Interrupt

0xB1

Input Registers

AH = 0x11

AL = subfunction code as follows:

0x00 specifies de-register
0x01 specifies register

DS:DI = Pointer to an 8-byte storage area (See ***Notes*** below)

ES:BX = Pointer to the function to be called prior to resuming the system (Subfunction 0x01 only)

Output Registers

If no error is detected:

Carry Flag is cleared
AX = 0x0000

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or AL, respectively
0x0006: Invalid storage area specified
0x0007: User already registered for notification
0x0008: User not registered for notification

Notes

The storage area pointed to by DS:DI must be static and set to 0 before registering with this function.

After registering, the caller must not modify the storage area prior to de-registering.

When de-registering, the same storage area that was used to register must be supplied to the function.

Any registrations from an application must be de-registered prior to deleting the application.

If these rules are not followed, the system will probably crash if the device resumes.

Return codes 0x0006, 0x0007, and 0x0008 indicate that the storage area pointed to by DS:DI is invalid. If the storage area is corrupted, incorrect error codes is returned.

There is no limit to the number of users that may register for notification.

Once registered for notification, the specified routine is called whenever the system resumes with AH containing a resume code and AL containing a wakeup code as indicated in the following:

Resume Code	**Reason for Resumption**
0x01	System resumed after shutdown.
0x08	System resumed after power fault.

Wakeup Code	**Cause of Wakeup**
0x00	Right Trigger
0x02	Left Trigger
0x04	Pen Touch
0x08	RS232 ring
0x10	Power switch
0x20	Cradle Insertion
0x40	Cradle Removal
0x80	Resumed on alarm

Note: This function does *not* support Spectrum24 wakeup.

If the wakeup code is 0x00, the application should call the **Get Extended PPT 4100 Wakeup Cause** service

(**INT 0xB1**, **Function 0x1A**) to determine the true wakeup cause.

System resume notifications are processed before any device resume notifications.

The notification routine should terminate with an **IRET** instruction after restoring all registers to the values supplied when the routine was called.

The notification routine should not make any assumptions about the amount of space available on the stack.

Example

See the **Example** for **Suspend System (0x00)** for a code sample that illustrates the **Register for System Resume Notification** service.

Get Power Source

Function: 0x12

Description

Returns an indicator of the current power source being used by the terminal.

Interrupt

0xB1

Input Registers

AH = 0x12

Output Registers

AX = Indicator of current power source as follows:

0x0000: Battery
0x0001: Cradle
0x0002: Charger

Example

The following code sample illustrates the XSYMBIOS power management services below:

Get Power Source (Function 0x12)
Get Battery Status (Function 0x03)

This example is contained in the following file in the PPT 41XX Software Development Kit (SDK):

c:\SDK4100\SAMPLES\MANUAL\CHAP2\POWERSRC.C

where **c:\SDK4100** is the default installation directory.

```
/* Include Files ************************************************** */

#include <dos.h>
#include <stdio.h>

/* Defines ************************************************** */

enum {BATTERY, CRADLE, CHARGER};

/*  Define the Services by Interrupt Vector number              */

#define XB_POWER_INT   0xB1

/*  Define the Functions                                        */

#define XB_GET_BATTERY_STATUS 0x03
#define XB_GET_POWER_SOURCE   0x12

/* Public Variables ************************************************** */

union REGS inregs;      /* input regs to int86       */
union REGS outregs;     /* output regs from int86    */


/************Get Power Source service ************************** */

void xb_GetPowerSource(unsigned char _far *source)
{
    inregs.h.ah = XB_GET_POWER_SOURCE;
    int86(XB_POWER_INT, &inregs, &outregs);
    *source = outregs.h.al;
}

/************Get Battery Status service ************************** */

void xb_GetBatteryStatus(unsigned char _far *level)
{
    inregs.h.ah = XB_GET_BATTERY_STATUS;
    int86(XB_POWER_INT, &inregs, &outregs);
    *level = outregs.h.al;
}
```

```
/*****************************************************************/

void main()
{
   /* Local Variables ************************************************/
   char source; /            * Power source */

   xb_GetPowerSource( (char _far *) &source );

   switch (source)
   {
        case BATTERY:
           fprintf(stdout,"Power source is battery");
           break;

        case CRADLE:
           fprintf(stdout,"Power source is cradle");
           break;

        case CHARGER:
           fprintf(stdout,"Power source is charger");
           break;

        default:
           fprintf(stdout,"Power source is unknown");
           break;
   }
}
```

Note: For an additional illustration of the use of the **Get Power Source** service, refer to the **Example** provided in the description of the **Suspend System (0x00)** service earlier in this section.

Get/Set Device Timer Value

Function: 0x13

Description

Allows the calling application to examine or to set the sleep and suspend inactivity timers for a specified device. This service does *not* apply to the CPU (for which see **Function 0x14**, described below).

Timer values are specified in seconds. Values of -1 and zero through 3600 are valid. A value of -1 indicates that a timer never expires; a value of 0 indicates a timer that expires immediately. Any value greater than 3600 (i.e., one hour) returns an error code of 0x0009 (timer out of range).

Interrupt

0xB1

Input Registers

AH = 0x13

AL = the appropriate subfunction code from the following:

0x00 specifies Get Sleep Inactivity Timer
0x01 specifies Get Suspend Inactivity Timer
0x02 specifies Set Sleep Inactivity Timer
0x03 specifies Set Suspend Inactivity Timer

CX = Required timer values for subfunction 0x02 or 0x03

DS:DX = Address of the ASCIIZ string (LCD, COM1, COM2, PCMCIA, or SCAN) that identifies the device to which the service is being applied.

Output Registers

If no error is detected:

Carry Flag is cleared
AX = Current value of the specified timer

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or AL, respectively
0x0004: Unable to find specified power management device
0x0009: The value specified for the timer is out-of-range

Example

See the **Example** for **Activate Device (0x08)** for a code sample that illustrates the **Get/Set Device Timer Value** service.

Get/Set System Timer Value

Function: 0x14

Description

Allows the calling application to examine or to set the sleep and suspend inactivity timers for the system.

Timer values are specified in seconds. Values of -1 and zero through 3600 are valid. A value of -1 indicates a timer that never expired; a value of 0 indicates a timer that expires immediately. Any value greater than 3600 (i.e., one hour) returns an error code of 0x0009 (Timer out of range).

Interrupt

0xB1

Input Registers

AH = 0x14

AL = the appropriate subfunction code from the following:

0x00 specifies Get Sleep Inactivity Timer
0x01 specifies Get Suspend Inactivity Timer
0x02 specifies Set Sleep Inactivity Timer
0x03 specifies Set Suspend Inactivity Timer

CX = Required timer values for subfunction 0x02 or 0x03

Output Registers

If no error is detected:

Carry Flag is cleared
AX =Current value of the specified timer

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or AL, respectively
0x0009: The value specified for the timer is out-of-range

Example

See the **Example** for **Activate Device (0x08)** for a code sample that illustrates the **Get/ Set System Timer Value** service.

Poll Power Management

Function: 0x15

Description

Sets all devices to the required state and performs notifications of power management events.

> **Note:** This function can also be invoked more simply by calling the **MS-DOS Idle API** (**INT 0x28**).

Interrupt

0xB1

Input Registers

AH = 0x15

Output Registers

If no error is detected:

> Carry Flag is cleared
> AX = 0x0000

This service returns no errors. AX and the Carry Flag are always clear.

Notes

The timers used to control the power management sub-system use the standard PC timer tick (**INT 0x08**). If any timer expires, the subsystem sets a flag indicating that some action is pending. It does not perform the pending action from within the interrupt service routine since this could interfere with the operation of time-critical portions of programs and cause re-entrancy problems.

The Poll routine automatically executes any pending actions. Applications should call this routine periodically to allow the timer-controlled sections of the program to activate. The Poll routine is called automatically from the DOS Idle Handler interrupt (**INT 0x28**), allowing the subsystem to operate with such programs as COMMAND.COM which call this interrupt whenever they are waiting for an event.

Example

See the **Example** for **System Sleep Disable Control (0x05)** for a code sample that illustrates the **Poll Power Management** service.

Reset Inactivity Timers

Function: 0x16

Description

Makes the system and, optionally, the LCD active. This service can safely be called from the background, e.g., an interrupt service routine.

Interrupt

0xB1

Input Registers

AH = 0x16

AL = Subfunction code, as follows:

0x00 = Make system active
0x01 = Make system and LCD active

Output Registers

Carry Flag is cleared
AX = 0x0000

Example

See the **Example** for **System Sleep Disable Control (0x05)** for a code sample that illustrates the **Reset Inactivity Timers** service.

Get/Set Low Battery LED Flash-On Time

Function: 0x17

Description

This function gets or sets the timer used to control the frequency with which the scanner LED flashes when the battery is low. This frequency is determined by the AL register setting that has been specified by the application.

The LED *flash-on* time is always fixed by the application in timer-tick units (i.e., 18.2 per sec. or 55 millisecs.).

The LED *flash-off* time is controlled by the state of the battery as specified in the following chart:

Battery Level	LED Flash-off Time
Normal	Infinite (i.e., no LED flashing at all).
Low	32 times the LED flash-on time specified in the AL register by the application.
Very Low	2 times the LED flash-on time specified in the AL register by the application.

Note: The actual flashing of the LED is controlled by the timer tick service (INT 0x08).

Interrupt

0xB1

Input Registers

AH = 0x17

AL = Subfunction code, as follows:

0x00 through 0x14: LED flash-on time in timer tick units (55 msec)
0x80: Get current LED flash-on time in timer tick units

Note: If AL = 0x00, LED flashing has been disabled.

Output Registers

If no error is detected:

Carry Flag is cleared
AL = Current value for the LED flash-on time in timer tick units (55 milliseconds)

If an error is detected:

Carry flag is set
AX = Error code as follows:

0x0001: Incorrect function code or subfunction code passed in register AH or AL, respectively

Example

See the **Example** for **Suspend System (0x00)** for a code sample that illustrates the **Get/Set Low Battery LED Flash-On Time** service.

Enable/Disable Power Management Poll on INT 0x16

Function: 0x18

Description

This function enables the application to control whether or not the INT 0x16 (keyboard) services poll power management when INT 0x16 check-for-key and key-read functions (0x00, 0x01, 0x10, and 0x11) are called.

The keyboard services interrupt (INT 0x16) performs some pre-processing before chaining on to the original routine. If the keyboard service called is 0x00, 0x01, 0x10, or 0x11, XSYMBIOS polls the power management services to set all devices to the required power state. This poll is *not* performed if polling is disabled by **Interrupt 0xB1**, **Function 0x18**, **Subfunction 0x01** (see **Input Registers**, below).

XSYMBIOS checks to see if the keyboard service is a read key request and if so if there is a key to be processed. If there is no key to satisfy the read key request, XSYMBIOS polls power management, puts the CPU to sleep until the next interrupt, and restarts at the top of the routine.

By default, polling is enabled, but if required can be disabled by this power management service. If disabled, power management polling is called only when an application tries to read a key and there is no key in the buffer.

Each disable call to this service increments a count; each enable call decrements the count. Polling is enabled only if the count is zero.

Input Registers

AH = 0x18

AL = Subfunction code, as follows:

0x00: Enable poll on INT 0x16
0x01: Disable poll on INT 0x16
0x80: Get disable count

Output Registers

If no error is detected:

Carry Flag is cleared
AL = Current number of disables

If an error is detected:

Carry flag is set.
AX = Error code as follows

0x0001: Incorrect function code passed in register AH
or register AL, respectively
0x0005: Overflow of disables count

Example

See the **Example** for **Suspend System (0x00)** for a code sample that illustrates the **Enable/Disable Power Management Poll on INT 0x16** service.

Get Extended Wakeup Cause

Function: 0x1A

Description

Returns the extended cause of the last wakeup. See **Notes** below.

Interrupt

0xB1

Input Registers

AH = 0x1A

Output Registers

DX contains extended wakeup causes encoded as follows:

Bits 15 - 0:	Note used

AX contains extended wakeup causes encoded as follows:

Bits 15 - 9:	Not used
Bit 8:	Spectrum24
Bit 7:	Not used
Bit 6:	Not used
Bit 5:	Cradle Insertion / Removal
Bit 4:	Power Switch
Bit 3:	RS - 232 Ring
Bit 2:	Pen Down
Bit 1:	Left Button
Bit 0:	Right Button

Notes

Applications that need to monitor Spectrum24 wakeup should use this service. Otherwise, use **Get Wakeup Cause** (**Function 0x02**) which provides several more standard wakeup causes.

Example

The following code sample illustrates the use of the **Get Extended Wakeup Cause** (**0x1A**) service.

```
void WakeupCause(void)
{
   union _REGS inregs, outregs;

   /*Get extended resume code */
   inregs.h.ah = 0x1a;
   _int86(0xb1, &inregs, &outregs);

   /* Display the results */
   printf("Extended Resume: %04x:%04x\n", outregs.x.dx, outregs.x.ax);
}
```

Power Management Auxiliary API Commands

As indicated in Rule 2 in *Rules for Using the Power Management Subsystem* and in the **Note** under **Poll Power Management** (**INT 0xB1 Function 0x15**), there are two services that applications may use to identify power management opportunities and operate outside of XSYMBIOS. These are:

- **Multiplex Application Idle API**, which an application invokes with **INT 0x2F** and **Function Code 0x1680**.
- **MS-DOS Idle API**, which an application invokes with **INT 0x28**.

Descriptions of these commands are provided in the following sections.

Multiplex Application Idle API (MS-DOS Idle Call)

Description

This function signals that the application has nothing to do and that any other processes can get processing time.

Interrupt

0x2F

Input Registers

AX = 0x1680

Output Registers

None. All registers are preserved.

Notes

Refer to Rule 2 in the *Rules for Using the Power Management Subsystem*. Also see **Note** in **Poll Power Management** (**Function 0x15**) in the *Power Management API Commands (Descriptions)*.

XSYMBIOS intercepts this call and enters a power saving state until the next interrupt occurs.

This function is not blocking and should be called repeatedly while the caller is idle.

MS-DOS Idle API Call (MS-DOS Idle Handler)

Description

This interrupt is invoked each time one of the DOS character input functions loops while waiting for input. TSRs can hook this interrupt if they need to perform DOS calls while the foreground program is waiting for user input.

Interrupt

0x28

Input Registers

None.

Output Registers

None, all registers preserved.

Notes

Refer to Rule 2 in the *Rules for Using the Power Management Subsystem*. Also see **Note** in **Poll Power Management** (**Function 0x15**) in the *Power Management API Commands (Descriptions)*.

This interrupt handler may invoke any **INT 0x21** function except functions **0x00** through **0x0C**.

The default handler is an **IRET** instruction.

XSYMBIOS intercepts this call and calls the **Poll Power Management** (**Function 0x15**) service. DOS applications should use this function periodically in non-time-critical sections of the program to allow XSYMBIOS to update power state for all devices.