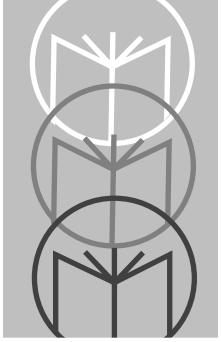

Chapter 1 XSYMBIOS/Symbol Extended BIOS

Introduction

The Symbol Extended BIOS TSR (XSYMBIOS.EXE) is a DOS level TSR (Terminate and Stay Resident) that provides BIOS type extensions for PPT 41XX terminals. It allows applications to take advantage of particular features of the PPT 41XX hardware and of extended services inherited from Series 3000 and earlier PPT 4100 platforms.

XSYMBIOS accesses the PPT 41XX gate array, controls power management for the terminal, and performs cradle insertion and removal functions. It supports ROM BIOS extensions, based on the Series 3000 BIOS, and provides an application program interface (API) for controlling power management.

YSYMBIOS.EXE is a developer's version of XSYMBIOS.EXE designed to work on a standard PC. XSYMBIOS.EXE and YSYMBIOS.EXE provide the same functions as the extended BIOS TSRs (XBIOS21T.EXE and XBIOS46T.EXE) for the PDT 2100 and the PPT 4600, respectively. These functions are:

- External Activity Status Reports (INT 0x32)
- LCD Contrast Control (INT 0x32)
- Timer Services (INT 0xAC)
- Semaphore Services (INT 0xAC)
- Sound Services (INT 0xAD)
- CRC Services (INT 0xAE)

Theory of Operation

XSYMBIOS.EXE and YSYMBIOS.EXE are loaded (usually from the AUTOEXEC.BAT file) as TSRs and provide a BIOS-like method for applications to call the supported functions. This means that all functions are accessed by performing an interrupt with parameters passed in registers. In most instances, register AH passes a function code to the routine.

On return from each function, registers (or buffers pointed to by registers) return desired output and / or completion or error codes.

Load XSYMBIOS on the terminal before the PPT 41XX scanner and radio drivers, since it contains functions that enable these drivers to interface with the terminal hardware platform.

The Cradle Handler (CRADLE.COM) TSR provides cradle support. If an installation has cradles, load Cradle Handler on the terminal *before* XSYMBIOS program is loaded to take advantage of cradle features in the operation of the terminal. If an installation does not have cradles, Cradle Handler is not required. For more information on the Cradle Handler, refer to the *Cradle Handler TSR* in this manual and to the *PPT 41XX Product Reference Guide.*

User Interface

When XSYMBIOS.EXE is successfully loaded, it displays the following banner to identify the program and version and message indicating successful installation:

Symbol BIOS Extensions Version XX.XX-XX
Copyright (C) Symbol Technologies Inc., 1994-97

If the program fails to install, it displays the following error message:

BIOS Extensions already loaded

XSYMBIOS performs tasks for applications and has no other user interface than that specified above. It is entirely passive except when asked to perform some task on behalf of an application.

Application Programming Interfaces

XSYMBIOS consists of several groups of services. Each group has its own API accessed through a different interrupt vector. Except for the power management services, the API for each group is described in this chapter. The power management services are described in *Chapter 2, XSYMBIOS/ Power Management*.

Table 1-1 contains the Series 3000 BIOS service groups supported by XSYMBIOS and the associated interrupt for each group.

Table 1-1. XSYMBIOS Service Extension Groups

PC Interrupt (Hex)	BIOS Service Group
0x14	Serial Communications
0x32	General System Services
0xAC	Timer Services
0xAD	Sound Services
0xAE	CRC Services
0xB1	Power Management Services

In each group listed in Table 1-1, executing a real mode interrupt invokes services. For interrupts **0x14**, **0x32**, **0xAC**, **0xAD**, and **0xAE**, register AH contains a function code identifying which service is to be performed.

XSYMBIOS range checks the supplied function code and dispatches to the appropriate routine if the function code is in range. If the function code is out of range, XSYMBIOS either returns an error condition to the caller (if the original interrupt vector was undefined) or passes control on to the original interrupt vector (if that vector was defined when XSYMBIOS was loaded). This allows XSYMBIOS to share interrupt vectors with other users.

XSYMBIOS Serial Communications Services (INT 0x14) (List)

Table 1-2 is a list of the Serial Communications Services supported by XSYMBIOS. The list is sorted by the hexadecimal numeral for the number (command code) the application assigns to the AH register when it invokes the service via **INT 0x14**. These services are described in the following section.

Table 1-2. Serial Communications Services (Interrupt 0x14)

Function Number	Serial Communications Service Name
0x00	Initialize Serial Port (IBM Standard)
0x01	Send One Character (IBM Standard)
0x02	Receive One Character (IBM Standard)
0x03	Get Serial Port Status (IBM Standard)
0x80	Extended Serial Port Initialization
0x81	Get Current Port Configuration
0x82	Open Serial Port
0x83	Close Serial Port
0x84	Send Block
0x85	Receive Block
0x86	Queue Status
0x87	Get System Status
0x88	Transmit Enable (Half-Duplex Line Turn Around)
0x89	Receive Enable (Half-Duplex Line Turn Around)
0x8A	Transmit Done
0x8B	Set UART Control Commands
0x8C	Clear UART Control Commands
0x8D	Allocate Communications Queues
0x8E	Purge Communications Queue
0x8F	Transmit Queue Empty Notification Control
0x91	Delete Queues
0x92	Get Queue Pointer
0x93	Version Number Check

XSYMBIOS Serial Communications Services (Descriptions)

The following descriptions of the functions in Table 1-2 are given in function code order.

Initialize Serial Port (IBM Standard)

Function: 0x00

Description

Sets the parameters for the specified serial channel and returns the same status as Function 0x03, **Get Serial Port Status (IBM Standard)**. For compatibility with IBM PC, this service reports no error if you select an invalid parameter value for the specified port.

Interrupt

0x14

Input Registers

AH = 0x00

AL = Port configuration, as follows:

Bits 7, 6, 5 = Baud, as follows:

000 = 110 baud
001 = 150 baud
010 = 300 baud
011 = 600 baud
100 = 1200 baud
101 = 2400 baud
110 = 4800 baud
111 = 9600 baud

Bits 4, 3 = Parity, as follows:

00 = None
01 = Odd
10 = None
11 = Even

Bit 2 = Stop bits, as follows:

0 = 1 bit
1 = 2 bits

Bits 1, 0 = Data size, as follows:

00 = (Not used)
01 = (Not used)
10 = 7 bits
11 = 8 bits

DX = Serial port

Output Registers

AH = Line status, as follows:

Bit 7 = Timeout error
Bit 6 = Send shift register empty
Bit 5 = Send data register empty
Bit 4 = Break detected
Bit 3 = Framing error
Bit 2 = Parity error
Bit 1 = Overrun error
Bit 0 = Data ready

AL = Modem status, as follows:

Bit 7 = Carrier detect
Bit 6 = Ring indicator
Bit 5 = Data-Set-Ready (DSR)
Bit 4 = Clear-To-Send (CTS)
Bit 3 = Delta carrier detect
Bit 2 = Trailing-edge ring detect
Bit 1 = Delta data-set-ready
Bit 0 = Delta clear-to-send

Send One Character (IBM Standard)

Function: 0x01

Description

Transmits a single character to the UART of the selected serial port or queues a single character into the output FIFO queue, depending on whether or not the port has been opened using the **Open Serial Port** service (Function 0x82). If invoked in polled mode, this function waits until the UART transmit register is empty before writing the new character. If used in interrupt mode, this function waits for space in the FIFO queue before returning. In either case, the Status Byte returned is identical to that returned by **Get Serial Port Status (IBM Standard)**, Function 0x03.

In polled mode, this function raises RTS and DTR and waits for DSR and CTS before sending the character. If either modem control line (DSR and CTS) does not go active in the current timeout value, an error is reported.

Interrupt

0x14

Input Registers

AH = 0x01

AL = Character

DX = Port number

Output Registers

AL = Last character sent

AH = Line status, as follows:

Bit 7 = Timeout error
Bit 6 = Send shift register empty
Bit 5 = Send data register empty
Bit 4 = Break detected
Bit 3 = Framing error
Bit 2 = Parity error
Bit 1 = Overrun error
Bit 0 = Data ready

Receive One Character (IBM Standard)

Function: 0x02

Description

Reads one character from the UART of the selected serial port or dequeues a character from the input FIFO queue, depending on whether or not the port was opened using the Open Serial Port service (Function 0x82). This function waits until a character is received or timeout error occurs before returning. The Status Byte returned is identical to that returned by Function 0x03, **Get Serial Port Status (IBM Standard)**.

In polled mode, this function raises DTR, drops RTS, and waits for DSR to go active before it reads the UART. If no character (or DSR) is received in the standard timeout period, an error is reported.

Interrupt

0x14

Input Registers

AH = 0x02

DX = Port number

Output Registers

AL = Character received

AH = High byte of serial port status (see Function 0x03,**Get Serial Port Status (IBM Standard)**)

Get Serial Port Status (IBM Standard)

Function: 0x03

Description

Returns the current status of the specified serial port.

Interrupt

0x14

Input Registers

AH = 0x03

DX = Port number

Output Registers

AH = Line status, as follows:

Bit 7 = Timeout error
Bit 6 = Send shift register empty
Bit 5 = Send data register empty
Bit 4 = Break detected
Bit 3 = Framing error
Bit 2 = Parity error
Bit 1 = Overrun error
Bit 0 = Data ready

AL = Modem status, as follows:

Bit 7 = Carrier detect
Bit 6 = Ring indicator
Bit 5 = Data-Set-Ready (DSR)
Bit 4 = Clear-To-Send (CTS)
Bit 3 = Delta carrier detect
Bit 2 = Trailing-edge ring detect
Bit 1 = Delta data-set-ready
Bit 0 = Delta clear-to-send

Extended Serial Port Initialization

Function: 0x80

Description

Initializes the Channel Control Block (CCB) of the specified serial port to the values of the passed parameters. If the serial port is not open, this service initializes the CCB for the next open service call. If the serial port is already open, this service also modifies the physical and logical configuration of the serial port to match the new values of any changed parameters. It reports an error for any out-of-range parameters or illegal combination specified.

The parameter block has the following format:

Physical Setup Parameters.

The following parameters control the physical configuration of the serial port. If any of these parameters change while the serial port is open, the physical configuration is changed to match the new parameters.

Byte 1 = Data rate, as follows:

0 = 150 bps
1 = 300 bps
2 = 600 bps
3 = 1200 bps
4 = 1350 bps
5 = 2400 bps
6 = 4800 bps
7 = 9600 bps
8 = 19200 bps
9 = 38400 bps

Byte 2 = Data size, as follows:

2 = 7 bits
3 = 8 bits

Byte 3 = Parity, as follows:

0 = Even
1 = Odd
2 = Mark

3 = Space
4 = None (Indicates that no parity bit is added to the character.)

Byte 4 = Stop bits, as follows:

0 = 1 stop bit
1 = 2 stop bits

Duplex Control Parameter.

This parameter specifies whether the serial port works in full or half duplex or in multi-access mode. If it is changed when the serial port is open, the serial port is reset to:

- half duplex receive if changed from full to half duplex
- full duplex send / receive if changed from half to full duplex

In either case, no data is lost from the queues.

Byte 5 = Modem duplex, as follows:

0 = Full duplex
1 = Half duplex

Half Duplex Mode Control Parameters.

The following parameters control features specific to half duplex line operations and are ignored in full duplex mode.

Bytes 6, 7 = Modem delay (in milliseconds)

Used on physical block transmit enables to control the time between the raising of RTS and the start of data transmission.

Bytes 8, 9 = Transmit carrier wait time (in milliseconds)

Used during transmit enable to control the time the terminal waits for the carrier to drop before raising RTS. The wait time is ignored if this parameter is set to zero.

Bit 15 controls error reporting if the carrier does not go active within the specified time. If this bit is set, an error is posted if the timer expires. In either case (set or reset), the line turnaround procedure continues.

Bytes 10, 11 = Receive carrier wait time (in milliseconds)

Used during receive enable to control the time the terminal waits for the carrier to go active before enabling receive. If set to zero, the wait time is ignored.

Bit 15 controls error reporting if the carrier does not go active within the specified time. If this bit is set, an error is posted if the timer expires. In either case (set or reset), the line turnaround procedure continues.

Full Duplex Mode Control Parameters.

These parameters control features specific to full duplex modems and are ignored in half duplex mode.

Bytes 12, 13 = Carrier loss detect time (in milliseconds)

Specifies the length of time the carrier must be continuously inactive before a loss of carrier error is posted.

Flow Control Parameters.

The following parameters control the pacing of data during a communications session.

Byte 14 = Control stop character

Used in software flow control mode. This character is sent when the receive queue reaches 80% of its capacity. If it is received, transmit is disabled until a control start character is received. If used, all control stop characters are filtered from the received data.

Byte 15 = Control start character

Used in software low control mode. This character is sent, if a control stop character was previously sent, when the receive queue drops to 60% of capacity. If it is received after a previous control stop, transmit is re-enabled. If used, all control start characters are filtered from the received data.

Byte 16 = Control start wait time (in seconds)

Used to specify the time the BIOS waits before reporting an error whenever a control stop character is received and a control start character is not subsequently received.

Byte 17 = CTS loss detection time (in seconds)

The time in a transmit state that the BIOS waits for CTS to go active before it reports a timeout error. If this parameter is set to zero, the BIOS waits forever.

Byte 18 = Receive character wait time (in seconds)

The time that the BIOS waits for a character on input before timing out. This is the maximum allowable time between characters when waiting for receive data (applies only to the receive block with wait set service). A timeout error occurs if the time since the last character was received is greater than the time specified.

Open Setup Parameters.

Byte 19 = DSR wait time (in seconds)

Sets the Date-Set-Ready wait time for OPEN. Reports an error if DSR is not active in the time specified. If this value is zero, there is no DSR check.

Byte 20 = CD wait time (in seconds)

Set Carrier-Detect wait time for OPEN. Reports an error if the carrier is not active in the time specified. If this value is zero, no check for CD is made. When using COM2, the CD wait time must be zero. This parameter is ignored in half duplex mode.

Byte 21 = SPACE time (in seconds)

The time that BIOS generates SPACE on the transmit data line during the open service. SPACE is generated after all other open activities (port configuration, DSR check, CD check) occur.

Byte 22 = MARK time (in seconds)

The time that BIOS generates MARK on the transmit data line during the open and close services. MARK is generated after the
SPACE on open and just before the line is shut down on close.

Logical Setup Parameters.

The following parameters select the operation mode and the flow control method, if any, for the selected channel.

Byte 23 = Line conditioning flags, as follows:

Bit 2 = CTS conditioning

Bit 1 = CD conditioning

Bit 0 = DSR conditioning

If the CTS conditioning flag is set, transmission is disabled while CTS is inactive and the CTS wait time is determined by Byte 17. Otherwise, CTS is ignored.

If the CD conditioning flag is set, reception is disabled if CD is inactive. This flag has no effect on the CD open wait time, the CD loss detection time, or the transmit and receive enable CD wait time. If using COM2, do *not* set the CD conditioning flag.

If the DSR conditioning flag is set, an error is reported if DSR is lost during the communications session. The flag has no effect on the DSR open wait time. If using COM2, do *not* set the DSR conditioning flag.

Byte 24 = Operation mode, as follows:

Bit 2 = Hardware flow control

Bit 1 = Software flow control

Bit 0 = No flow control

Selects the flow control method, if any, for the specified serial port. If flow control is selected, the remote transmitter is disabled when the input queue reaches 80% of capacity, and enabled when the input queue is reduced to 60% of capacity. Software flow control uses data characters (control start / stop) to enable / disable transmission and reception of data. Hardware flow control raises / lowers RTS to do this.

In hardware flow control, CTS line conditioning (see above) must be enabled so the remote receiver can disable transmit by dropping CTS to the PPT 41XX.

Byte 25 = Line Status Error Mask, as follows:

Bit 4 = Break detect

Bit 3 = Framing

Bit 2 = Parity

Bit 1 = Overrun

Enables or disables the handling of line status errors from the UART. If a line status error occurs and the error flag is reset, the error is ignored. See below for the action taken if the error is enabled.

Byte 26 = Error Insertion Character

This character is inserted into the data stream if an enabled line status error occurs. If this character is null, an error is reported.

Bytes 27, 28 = DTR Settling Time (in milliseconds)

The time the BIOS waits after raising DTR before checking DSR. When using COM2, set DTR settling time to 50 milliseconds.

Byte 29 = Connect Time

The number of seconds the BIOS waits after establishing a link before communications start.

Input Registers

AH = 0x80

DX = Serial port

ES:SI = Far pointer to the parameter block

Output Registers

The Carry Flag is set if an error occurs: otherwise, reset.

AX = Error status (see Function 0x87,**Get System Status**)

CX = Configuration error, if any, as follows:

Bit 6 = Invalid control start / stop characters
Bit 5 = Invalid flow control mode
Bit 4 = Unknown duplex
Bit 3 = Parity not supported
Bit 2 = Stop bits not supported
Bit 1 = Data size not supported
Bit 0 = Data rate not supported

Get Current Port Configuration

Function: 0x81

Description

Returns either the current configuration or the size of the BIOS parameter block of the specified serial port. If only the size is requested, ES:DI need not be specified. If the configuration is requested, it is copied to the buffer specified in the ES:DI register pair using the same format as Function 0x80, **Extended Serial Port Initialization**.

Interrupt

0x14

Input Registers

AH = 0x81

AL = Subfunction, as follows:

0x00 = Return parameter block and size
0x01 = Return only the parameter block size

DX = Serial port

ES:DI = Address of the parameter block buffer

Output Registers

The Carry Flag is set if an error occurs; otherwise, reset.

AX = Error status (see Function 0x87, Get System Status)

CX = Parameter block size

Open Serial Port

Function: 0x82

Description

Powers up and initializes the selected serial port to the configuration stored in the CCB. See Function 0x80, **Extended Serial Port Initialization** for information on serial port configuration.

Consecutive opens are allowed; each call performs all open tasks including purging the queues, waiting for DSR and CD, and generating the SPACE and MARK tones. DSR is not lost on subsequent opens. Initialize the queues before the first open (see Function 0x8D, **Allocate Communications Queues**).

Interrupt

0x14

Input Registers

AH = 0x82

DX = Serial port

Output Registers

The Carry Flag is set if any error occurs; otherwise, reset.

AX = Error status (see Function 0x87, **Get System Status**)

Close Serial Port

Function: 0x83

Description

Closes a communications port.

This service generates MARK tone for the time specified in the last configuration call, then terminates the communications session and powers down the selected serial port.

Interrupt

0x14

Input Registers

AH = 0x83

DX = Serial port

Output Registers

The Carry Flag is set if any error occurs; otherwise reset.

AX = Error status as follows:

Bit 15 = Invalid configuration / queue
Bit 14 = Receive queue is full
Bit 13 = Clear-To-Send (CTS) lost
Bit 12 = Control start not received
Bit 11 = Receive character timeout
Bit 10 = CD did not go inactive on transmit
Bit 9 = CD did not go active on receive
Bit 8 = CD lost during session
Bit 7 = User aborted
Bit 6 = Lost DSR while receiving
Bit 5 = Timeout waiting for DSR or CD
Bit 4 = BREAK detected
Bit 3 = Framing error
Bit 2 = Parity error
Bit 1 = Overrun error
Bit 0 = Channel not open

Send Block

Function: 0x84

Description

Moves the specified data block into the transmit queue until the queue is full or the entire data block is queued. Reports an error if the specified serial port is not open. If wait mode (0) is specified in the AL register and if a communications error occurs, this service may terminate before all data is queued. If in half-duplex receive, the line is automatically enabled for transmit (see Function 0x88,**Transmit Enable (Half-duplex Line Turn Around)**).

Interrupt

0x14

Input Registers

AH = 0x84

AL = Mode, as follows:

0x00 = Wait for all data to be queued
0x01 = Do not wait

CX = Length of data

DX = Serial port

ES:SI = Address of data block

Output Registers

The Carry Flag is set if an error occurs; otherwise, reset.

AX = Error status (see Function 0x87, Get System Status)

CX = Number of bytes not sent

DX = Number of bytes sent

ES:SI = Address of first byte not sent

Receive Block

Function: 0x85

Description

Returns the number of queued characters and the available queue space for the selected queue and serial port. If the serial port is not open, this function returns zero for both the space available and the number of queued characters.

Interrupt

0x14

Input Registers

AH = 0x85

AL = Mode, as follows:

0x00 = Wait for entire block
0x01 = Do not wait
0x02 = Wait for at least one character

CX = Maximum buffer length

DX = Serial port

ES:DI = Address of buffer

Output Registers

The Carry Flag is set if an error occurs; otherwise, reset.

AX = Error status (see Function 0x87, Get System Status)

CX = Number of bytes left to get

DX = Number of bytes received

ES:DI = Address of the next buffer location

Queue Status

Function: 0x86

Description

Returns the number of queued characters and the available queue space for the selected queue and serial port. If the serial port is not open, this function returns zero for both the space available and the number of queued characters.

Interrupt

0x14

Input Registers

AH = 0x86

AL = Queue selected, as follows:

0x00 = Input
0x01 = Output

DX = Serial port

Output Registers

The Carry Flag is set if an exception occurs; otherwise, reset.

AX = Error status (see Function 0x87, Get System Status)

CX = Queued characters

DX = Space left in queue

Get System Status

Function: 0x87

Description

Returns the current status of the selected channel. The current error status is returned in the AX register and the current state of the selected channel is returned in CX. This service monitors the status of the communications line.

Interrupt

0x14

Input Registers

AH = 0x87

DX = Serial port

ES:DI = Address of the transfer buffer

Output Registers

The Carry Flag is set if an error occurs; otherwise, reset.

AX = Communications error status, as follows:

Bit 15 = Invalid configuration / queue
Bit 14 = Receive queue full
Bit 13 = Clear-To-Send (CTS) lost
Bit 12 = Control start not received
Bit 11 = Receive character timeout
Bit 10 = CD did not go inactive on transmit
Bit 9 = CD did not go active on receive
Bit 8 = CD lost during session
Bit 7 = User aborted
Bit 6 = Lost DSR while receiving
Bit 5 = Timeout waiting for DSR or CD
Bit 4 = BREAK detected
Bit 3 = Framing error
Bit 2 = Parity error
Bit 1 = Overrun error
Bit 0 = Channel not open

CX = Current state, as follows:

Bit 6 = Full-duplex send / receive
Bit 5 = Half-duplex data submit
Bit 4 = Half-duplex modem delay
Bit 3 = Half-duplex transmit enable
Bit 2 = Half-duplex data receive
Bit 1 = Half-duplex receive enable
Bit 0 = Idle (closed)

Transmit Enable (Half-duplex Line Turn Around)

Function 0x88

Description

Causes the BIOS to perform a physical line turn around when using half-duplex modems. It is ignored when using a full-duplex modem.

This service causes the BIOS to:

1. disable Receive
2. wait a specified time for CD to drop
3. raise RTS
4. wait modem delay time
5. enable Transmit

All or most of these actions are performed by the BIOS after the call to this function returns; that is, this function call starts the line turnaround procedure but usually does not wait for the procedure to complete.

Note: If a Send Block (Function 0x84) or Send One Character (Function 0x01) call is issued while in half-duplex Receive, this function is performed automatically.

Interrupt

0x14

Input Registers

AH = 0x88

DX = Serial port

Output Registers

The Carry Flag is set if an error occurs; otherwise, reset.

AX = Error status (see Function 0x87, **Get System Status**)

Receive Enable (Half-Duplex Line Turn Around)

Function: 0x89

Description

Causes the BIOS to perform a physical line turnaround when using half-duplex modems. It is ignored when using a full-duplex modem.

This service causes the BIOS to:

1. wait until the transmit queue and the UART are empty
2. drop RTS
3. disable Transmit
4. wait a specified time for CD to go active
5. enable receive

All or most of these actions are performed by the BIOS after the call to this function returns; that is, this function call starts the line turnaround procedure but usually does not wait for the procedure to complete.

Note: If a Receive One Character (Function 0x02) or a Receive Block (Function 0x85) service is called while in half-duplex Transmit this function is performed automatically.

Interrupt

0x14

Input Registers

AH = 0x89

DX = Serial port

Output Registers

The Carry Flag is set if an error occurs; otherwise, reset.

AX = Error status (see Function 0x87, **Get System Status**)

Transmit Done

Function: 0x8A

Description

Waits until the transmit queue and the UART for the specified serial port are empty.

Interrupt

0x14

Input Registers

AH = 0x8A

DX = Serial port

Output Registers

The Carry Flag is set if any error occurs; otherwise, reset.

AX = Error status (see Function 0x87, **Get System Status**)

Set UART Control Commands

Function: 0x8B

Description

Controls serial port services and control line.

Interrupt

0x14

Input Registers

AH = 0x8B

AL = Control mask, as follows:

Bit 0 = Enable DTR
Bit 1 = Enable RTS
Bit 6 = Enable BREAK

DX = Serial port

Output Registers

The Carry Flag is set if any error occurs; otherwise, reset.

AX = Error status (see Function 0x87,**Get System Status**)

Clear UART Control Commands

Function: 0x8C

Description

Controls serial port services and control line.

Interrupt

0x14

Input Registers

AH = 0x8C

AL = Control mask as follows:

Bit 0 = Disable DTR
Bit 1 = Disable RTS
Bit 6 = Disable BREAK

DX = Serial port

Output Registers

The Carry Flag is set if any error occurs; otherwise reset.

AX = Error status as follows:

Bit 15 = Invalid configuration / queue
Bit 14 = Receive queue is full
Bit 13 = Clear-To-Send (CTS) lost
Bit 12 = Control start not received
Bit 11 = Receive character timeout
Bit 10 = CD did not go inactive on transmit
Bit 9 = CD did not go active on receive
Bit 8 = CD lost during session
Bit 7 = User aborted
Bit 6 = Lost DSR while receiving
Bit 5 = Timeout waiting for DSR or CD
Bit 4 = BREAK detected
Bit 3 = Framing error
Bit 2 = Parity error

Bit 1 = Overrun error
Bit 0 = Channel not open

Allocate Communications Queues

Function: 0x8D

Description

Sets up the FIFO transmit and receive queues for the selected serial port.

This service must be called before the initial open of the communications session or an error occurs on the subsequent open.

Queues must be paragraph-aligned and the passed segment address must point to the first location of the queue.

The actual queue size is the passed size minus 8 bytes of queue overhead. The minimum passed size is 32 bytes (queue size = 24 bytes).

Interrupt

0x14

Input Registers

AH = 0x8D

DX = Serial port

ES:SI = Queue parameter block, where:

Word 1 = Size of the input queue
Word 2 = Segment address of the input queue
Word 3 = Size of the output queue
Word 4 = Segment address of the output queue

Output Registers

The Carry Flag is set if any error occurs; otherwise reset.

AX = Error status as follows:

Bit 15 = Invalid configuration / queue
Bit 14 = Receive queue is full
Bit 13 = Clear-To-Send (CTS) lost
Bit 12 = Control start not received
Bit 11 = Receive character timeout
Bit 10 = CD did not go inactive on transmit

Bit 9 = CD did not go active on receive
Bit 8 = CD lost during session
Bit 7 = User aborted
Bit 6 = Lost DSR while receiving
Bit 5 = Timeout waiting for DSR or CD
Bit 4 = BREAK detected
Bit 3 = Framing error
Bit 2 = Parity error
Bit 1 = Overrun error
Bit 0 = Channel not open

Purge Communications Queue

Function: 0x8E

Description

Re-initializes the specified queue(s), discarding any queued data.

Interrupt

0x14

Input Registers

AH = 0x8E

AL = Queue selected, bit encoded as follows:

Bit 1 = Receive queue
Bit 0 = Transmit queue

Note: Unlike Function 0x86 (Queue Status) this function can purge either or both queues. The queue selection parameter (AL) is bit encoded, resulting in values as follows:

Value	Meaning
0x00	Clear neither queue
0x01	Clear only Transmit queue
0x02	Clear only Receive queue
0x03	Clear both queues

Do not confuse these values with the value passed to Function 0x86, which selects a *single queue only.*

DX = Serial port

Output Registers

The Carry Flag is set if any error occurs; otherwise, reset.

AX = Error status (see Function 0x87, **Get System Status**)

Transmit Queue Empty

Function: 0x8F

Description

Dispatches an event (through a FAR call) when the transmit queue empties.

The dispatch routine must be a far procedure and must save all registers it uses. The routine is not invoked immediately. Instead, a flag is set when the queue empties. The channel background timer checks every 27.45 milliseconds to see if this flag is set and that the dispatch routine is enabled. When these conditions are met, the routine is dispatched.

Interrupt

0x14

Input Registers

AH = 0x8F

DX = Serial port

AL = Enable / Disable flag, as follows:

0x00 = Disable
0x01 = Enable

If AL = 0x01, then:

ES:BX = Address of the routine

Output Registers

The Carry Flag is set if any error occurs.

AX = Error code

Delete Queues

Function: 0x91

Description

Deletes the current communications queues. Subsequent opens fail if new queues are not allocated.

Interrupt

0x14

Input Registers

AH = 0x91

DX = Serial port

Output Registers

The Carry Flag is set if an error occurs.

AX = Error code

Get Queue Pointer

Function: 0x92

Description

Returns the segment addresses and sizes of the FIFO communications queues.

Interrupt

0x14

Input Registers

AH = 0x92

DX = Serial port

ES:DI = Address of the transfer buffer

Output Registers

The Carry Flag is set and AX = 0x8000 if no queues are currently allocated for the selected port. Otherwise, the Carry Flag is reset and AX = 0x00.

If the Carry Flag is reset, the value in the ES:DI register pair points to the block containing the segment addresses and sizes of the queues as defined in Function 0x8D, **Allocate Communications Queues**.

Version Number Check

Function: 0x93

Description

Returns the major and minor version numbers for the current implementation of XSYMBIOS.

There are no major and minor versions of 0. To check if XSYMBIOS is present, call this function with BX = 0. If XSYMBIOS is present, BX is changed. If it is not present, then BX is unchanged on return.

Applications that rely on XSYMBIOS should use this function to verify that the TSR is loaded before calling any functions through the interrupt vectors that are not supported in the standard BIOS.

> **Note:** This function has been superseded by Interrupt 0x32, Function 0x85, but has not been removed from XSYMBIOS to maintain compatibility with existing applications.

Interrupt

0x14

Input Registers

AH = 0x93

Output Registers

AX = 0x00 and the Carry Flag is cleared.

BH = Major version number for XSYMBIOS

BL = Minor version number for XSYMBIOS

XSYMBIOS General System Services (INT 0x32) (List)

These services provide a hardware-independent method for applications to control standard features such as LCD contrast. No support is included for controlling the scanner LED (controlled by the scanner driver). The scanner driver API is described in detail elsewhere in this manual.

The Symbol Extended BIOS TSR supports an application programming interface to the General System Services listed in Table 1-3. All functions in this API are called by executing **INT 0x32** with an appropriate function code in register AH and (in some cases) an appropriate subfunction code in register AL. Table 1-3 lists the commands that call these functions and their associated function codes. These services are described in the following section.

Table 1-3. XBIOS21T General System Services (INT 0x32)

Function Code	General System Service Name
0x80	Get / Set Scanner LED ON / OFF
0x81	Get / Set Buzzer Volume Control
0x82	Get / Set Backlight Brightness
0x83	Get Side Switch Status (Subfunction 0x00)
0x83	Get AC Adapter Status (Subfunction 0x01)
0x83	Get Battery Present Status (Subfunction 0x02)
0x83	Get Cradle Status (Subfunction 0x03)
0x84	Get / Set Resume Mask Register
0x85	Get XSYMBIOS Version Number
0x86	Get / Set Viewing Angle

XSYMBIOS General System Services (Descriptions)

The following descriptions of the functions in Table 1-3 are given in function code order.

Get/Set Scanner LED On/Off

Function: 0x80

Description

Turns the scanner LED on or off or returns the current state of the scanner LED.

Interrupt

0x32

Input Registers

AH = 0x80

AL = Subfunction code as follows:

0x00: Turn the scanner LED *Off*
0x01: Turn the scanner LED *On*
0x80: Get the current state of the scanner LED

Output Registers

AL = Current state of the scanner LED, as follows:

0x00 = Off
0x01 = On

Example

The following code sample illustrates **Get/Set Scanner LED On/Off (Function 0x81)**.

This example is contained in the following file in the PPT 4100 Software Development Kit (SDK):

```
c:\SDK4100\SAMPLES\MANUAL\CHAP1\SCANLED.C
```

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

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

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

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

enum {OFF,          /* Scanner LED state is OFF */
      ON};          /* Scanner LED state is ON  */

/*  Define the Services by Interrupt Vector number          */

#define XB_MISC_INT     0x32        /* General System Services */

#define XB_MSSTSCAN     0x8000      /* set scan state          */
#define XB_MSGTSCAN     0x8080      /* get scan state          */

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

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

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

void sleep(clock_t wait)
{
    clock_t goal;

    goal = wait + clock();
    while(goal > clock());
}



/********* Get Scanner LED On/Off State ********************* */
```

```
void xb_GetScannerLED(unsigned char _far *status)
{
    inregs.x.ax = XB_MSGTSCAN;
    int86(XB_MISC_INT, &inregs, &outregs);
    *status = outregs.h.al;
}

/************* Set Scanner LED ON / OFF ***************************/

void xb_SetScannerLED(unsigned char status)
{
    inregs.x.ax = XB_MSSTSCAN | (status & 1);
    int86(XB_MISC_INT, &inregs, &outregs);
}

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

void main ()
{
    unsigned char status;

    xb_GetScannerLED((char _far *) &status);
    fprintf(stdout,"\nScanner status is %02x (hex)\n", status);

    xb_SetScannerLED(ON);
    fprintf(stdout,"\nScanner status is %02x (hex)\n", status);

    sleep(5000);

    xb_SetScannerLED(OFF);
    fprintf(stdout,"\nScanner status is %02x (hex)\n", status);
}
```

Get/Set Buzzer Volume Control

Function: 0x81

Description

This function supports the following services:

- Reads and returns the buzzer volume state to the application.
- Allows an application to set the buzzer volume control to LOW / HIGH.

Interrupt

0x32

Input Registers

AH = 0x81

AL = Subfunction code as follows:

0x80: Gets the current buzzer volume control setting
0x00: Sets the buzzer volume to LOW
0x01: Sets the buzzer volume to HIGH

Output Registers

AL = Current buzzer volume control setting as follows:

0x00: Current setting is LOW
0x01: Current setting is HIGH

Example

The following code sample illustrates the General System Service functions below:

Get/Set Buzzer Volume Control (Function 0x81)
Get External Activity (Trigger, Power Source, Battery-Connect, or Terminal-in-Cradle) Status (Function 0x83)
Get XSYMBIOS Version Number (Function 0x85)

This example is contained in the following file in the PPT 4100 Software Development Kit (SDK):

```
c:\SDK4100\SAMPLES\MANUAL\CHAP1\MISC1.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 XB_MISC_INT 0x32      /* interrupt vector -- general system services */

#define XB_MSBUZZER 0x81      /* buzzer volume control */
#define XB_MSGTSTAT 0x83      /* get miscellaneous status */
#define XB_TRIGST   0x00      /* trigger status subfunction */
#define XB_ADAPTST  0x01      /* adaptor status subfunction */
#define XB_BATTPST  0x02      /* battery present status subfunction */
#define XB_CRADLST  0x03      /* in cradle status subfunction */
#define XB_MSGTOVER 0x85      /* get xsymbios (old) version number */

/* Define volume levels */
#define VOL_LOW 0
#define VOL_HIGH 1


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

void xb_GetBuzzerVol(BYTE far *volume);
BYTE xb_SetBuzzerVol(BYTE volume);
void xb_GetTriggerStatus(BYTE far *left, BYTE far *right);
void xb_GetAdaptorPresent(BYTE far *present);
void xb_GetBatteryPresent(BYTE far *present);
void xb_GetInCradleStatus(BYTE far *status);
void xb_GetXSymBIOSVersion(BYTE far *major,   /* major version number */
                           BYTE far *minor); /* minor version number */
```

```
void main (void)
{
   BYTE volume;              /* buzzer volume setting */
   BYTE left;                /* left trigger status */
   BYTE right;               /* right trigger status */
   BYTE adaptor_present;     /* adaptor presence status */
   BYTE battery_present;     /* battery presence status */
   BYTE in_cradle;           /* in/out of cradle status */
   BYTE major;               /* XSymBIOS (old) major version number */
   BYTE minor;               /* XSymBIOS (old) minor version number */

   /* Set buzzer volume to low */
   xb_SetBuzzerVol(VOL_LOW);

   /* Obtain and report current buzzer volume setting */
   xb_GetBuzzerVol(&volume);
   printf("Current buzzer volume is %s\n\n", volume ? "high" : "low");

   /* Set buzzer volume to high */
   xb_SetBuzzerVol(VOL_HIGH);

   /* Obtain and report current buzzer volume setting */
   xb_GetBuzzerVol(&volume);
   printf("Current buzzer volume is %s\n\n", volume ? "high" : "low");

   /* Report trigger switch statuses */
   xb_GetTriggerStatus(&left, &right);
   printf("The left trigger switch is%s depressed\n", left? "":"not");
   printf("The right trigger switch is%s depressed\n\n", right? "":"not");

   /* Report adaptor, battery and cradle statuses */
   xb_GetAdaptorPresent(&adaptor_present);
   printf("The AC adaptor is%s present\n", adaptor_present? "":"not");

   xb_GetBatteryPresent(&battery_present);
   printf("The battery is%s present\n", battery_present ? "":"not");

   xb_GetInCradleStatus(&in_cradle);
   printf("The terminal is%s in the cradle\n\n", in_cradle ? "":"not");
```

```
    /* Obtain and report (old) XSymBIOS version numbers */
    xb_GetXSymBIOSVersion(&major, &minor);
    if ((major == 0) && (minor == 0))
        printf("The (old) XSymBIOS is reported as not installed\n");
    else
        printf("The (old) XSymBIOS version number is:%u.%2.2u\n",major, minor);

    return;
}

/************ Get Buzzer Volume ***************************/

void xb_GetBuzzerVol(BYTE far *volume)  /* current volume setting */
{
    inregs.h.ah = XB_MSBUZZER;
    inregs.h.al = 0x80;

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

    *volume = outregs.h.al;

    return;
}
```

```
/********** Set Buzzer Volume ***********************/
BYTE xb_SetBuzzerVol(BYTE volume)  /* volume setting */
{
    BYTE retval;   /* return code */

    if ((volume == VOL_LOW) || (volume == VOL_HIGH))
    {
        inregs.h.ah = XB_MSBUZZER;
        inregs.h.al = volume;

        int86x(XB_MISC_INT, &inregs, &outregs, &segregs);
        retval = FALSE;
    }
    else
        retval = TRUE;

    return retval;
}


/*********** Get Side Switch Status ****************************************/
void xb_GetTriggerStatus(BYTE far *left,       /* current left trigger status */
                         BYTE far *right)    /* current right trigger status */
{
    inregs.h.ah = XB_MSGTSTAT;
    inregs.h.al = XB_TRIGST;

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

    *right = outregs.h.al & 0x01;
    *left = outregs.h.al & 0x02;

    return;
}
```

```
/********** Get AC Adapter Status ******************************************** /
void xb_GetAdaptorPresent(BYTE far *present)  /* current power source */
{
     inregs.h.ah = XB_MSGTSTAT;
     inregs.h.al = XB_ADAPTST;

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

     *present = outregs.h.al;

     return;
}

/************** Get Battery Present Status ********************************** /
void xb_GetBatteryPresent(BYTE far *present)  /* battery presence */
{
      inregs.h.ah = XB_MSGTSTAT;
      inregs.h.al = XB_BATTPST;

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

      *present = outregs.h.al;

      return;
}


/************ Get Cradle Status ************************************************ /
void xb_GetInCradleStatus(BYTE far *status)  /* current cradle status */
{
     inregs.h.ah = XB_MSGTSTAT;
     inregs.h.al = XB_CRADLST;

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

     *status = outregs.h.al;

     return;
}
```

```
/*************** Get XSYMBIOS Version ***************************/
void xb_GetXSymBIOSVersion(BYTE far *major,  /* major version number */
                           BYTE far *minor)  /* minor version number */
{
    inregs.h.ah = XB_MSGTOVER;

    inregs.x.bx = 0;

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

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

    return;
}
```

Get/Set Backlight Brightness

Function: 0x82

Description

This function supports the following services:

- Reads and returns the setting of the backlight state control to the application.
- Allows an application to set the backlight state control to OFF / LOW / HIGH.

Interrupt

0x32

Input Registers

AH = 0x82

AL = Subfunction code as follows:

0x80: Gets the current backlight state control setting
0x00: Sets the backlight state control to OFF
0x01: Sets the backlight state control to LOW
0x02: Sets the backlight state control to HIGH

Output Registers

AL = Current backlight state control setting as follows:

0x00: Current setting is OFF
0x01: Current setting is LOW
0x02: Current setting is HIGH

Notes

This function has no effect if the terminal does not have a backlight.

If the backlight is set on (i.e., LOW or HIGH), it is turned on when the LCD is in the active state and turned off when the LCD is in the sleep or suspend states.

The activities that force the LCD back to the active state are:

- keyboard input
- pen movement or touch
- trigger switch activity

- explicit program commands

Example

The following code sample illustrates the General System Service functions below:

Get/Set Backlight Brightness (Function 0x82)
Get/Set Viewing Angle (Function 0x86)
Get XSYMBIOS Version Number (Function 0x85)

It also illustrates the use of the **Delay** service (**INT 0xAC**, **Function 0x08**), described in *XSYMBIOS Timer Services (INT 0xAC) (List)*.

This example is contained in the following file in the PPT 4100 Software Development Kit (SDK):

```
c:\SDK4100\SAMPLES\MANUAL\CHAP1\BKLIGHT.C
```

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

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

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

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

/*  Define the Services by Interrupt Vector number              */

#define XB_MISC_INT    0x32        /* General System Services */
#define XB_TIME_INT    0xAC        /* Timer Services */

#define XB_MSBAKLIT   0x82         /* backlight state control  */
#define XB_MSVWANGL   0x86         /* viewing angle (contrast) ctrl*/


#define THREE_SECONDS 3000         /*3000 ms. = 3 sec.  */
#define XB_MISC_INT 0x32           /* XSYMBIOS interrupt vector */
#define UNDEFINED   0xff

enum XB_BAKLIT {XB_BAKLIT_OFF,             /* Backlight values: OFF  */
                XB_BAKLIT_LOW,             /*  LOW  */
                XB_BAKLIT_HIGH};           /* HIGH */

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

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

/* Public Function Prototypes ***********************************/

void xb_SetBacklightState(char state)
{
    inregs.h.ah = XB_MSBAKLIT;
    inregs.h.al = state;
    int86(XB_MISC_INT, &inregs, &outregs);
}
```

```
void xb_SetViewingAngle(char contrast)
{
      inregs.h.ah = XB_MSVWANGL;
      inregs.h.al = contrast;
      int86(XB_MISC_INT, &inregs, &outregs);
}
unsigned char xb_GetXBIOS46TVersion(unsigned char _far *major,
                                                  unsigned char _far *minor)
{
      inregs.h.ah = 0x89;
      inregs.x.bx = 0xffff;
       int86(XB_MISC_INT, &inregs, &outregs);
      *major = outregs.h.bh;
      *minor = outregs.h.bl;
       return(outregs.h.al);
}
void xb_Delay(long milliseconds)
{
      inregs.h.ah = 0x08;
      inregs.x.cx = *(unsigned int *)&milliseconds;
      inregs.h.dl = *((unsigned char *)&milliseconds+2);
      int86(XB_TIME_INT, &inregs, &outregs);
}
```

```
void main ()
{
    char bin_state[] = {XB_BAKLIT_OFF,XB_BAKLIT_LOW,XB_BAKLIT_HIGH};
    char *txt_state[] = {"OFF","LOW","HIGH"};
    unsigned char major, minor;
    int i;

     long delay_in_ms = THREE_SECONDS;

    /* Check if XSYMBIOS is loaded before requesting services   */

    if (_dos_getvect(XB_MISC_INT) == NULL)
    {
        fprintf (stderr,"XSYMBIOS not loaded.\nProgram aborted.");
        exit(0);
     }

     xb_GetXSYMBIOSVersion(&major, &minor);
     if (major == UNDEFINED)
    {
        fprintf (stderr,"XSYMBIOS not loaded.\nProgram aborted.");
        exit(0);
    }
    fprintf(stderr,"XSYMBIOS reported version number as %x. %02x\n",
            major, minor);

    for (i = 0;i < sizeof(bin_state);i++)
    {
        printf ("\rSetting Backlight state to %s", txt_state[i]);

        /*  Set the backlight to the next state (e.g. OFF, LOW, HIGH   */

        xb_SetBacklightState (bin_state[i]);

         /*   Short delay between settings of the backlight    */

         xb_Delay (delay_in_ms);
    }
}
```

Get Side Switch Status

Function: 0x83; Subfunction 0x00

Description

Returns the current status of the side switches.

Interrupt

0x32

Input Registers

AH = 0x83

AL = 0x00

Output Registers

AL = current side switch status as follows:

Bit 0 = 1 indicates the right side switch is down
Bit 1 = 1 indicates the left side switch is down

Example

For a code sample that illustrates the **Get Side Switch Status (0x83/0x00)** function, see the **Example** for **Get/Set Buzzer Volume Control (0x81)**.

This code sample is also contained in the following file in the PPT 4100 Software Development Kit (SDK):

c:\SDK4100\SAMPLES\MANUAL\CHAP1\MISC1.C

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

Get AC Adapter Status

Function: 0x83; Subfunction 0x01

Description

Returns the current status of the AC adapter.

Interrupt

0x32

Input Registers

AH = 0x83

AL = 0x01

Output Registers

AL = AC adapter status as follows:

0x00: the AC adapter is *not* connected
0x01: the AC adapter is connected

Example

For a code sample that illustrates the **Get AC Adapter Status (0x83/0x01)** function, see the **Example** for **Get/Set Buzzer Volume Control (0x81)**.

This code sample is also contained in the following file in the PPT 4100 Software Development Kit (SDK):

c:\SDK4100\SAMPLES\MANUAL\CHAP1\MISC1.C

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

Get Battery Present Status

Function: 0x83; Subfunction 0x02

Description

Indicates whether or not the terminal main battery is present.

Interrupt

0x32

Input Registers

AH = 0x83

AL = 0x02

Output Registers

AL = Battery status as follows:

0x00: the battery is *not* present
0x01: the battery is present

Example

For a code sample that illustrates the **Get Battery Present Status (0x83/0x02)** function, see the **Example** for **Get/Set Buzzer Volume Control (0x81)**.

This code sample is also contained in the following file in the PPT 4100 Software Development Kit (SDK):

c:\SDK4100\SAMPLES\MANUAL\CHAP1\MISC1.C

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

Get Cradle Status

Function: 0x83; Subfunction 0x03

Description

Indicates whether or not the terminal is in a cradle.

Interrupt

0x32

Input Registers

AH = 0x83

AL = 0x03

Output Registers

AL = In-cradle status as follows:

0x00: the terminal is *not* in a cradle
0x01: the terminal is in a cradle

Example

For a code sample that illustrates the **Get Cradle Status (0x83/0x03)** function, see the **Example** for **Get/Set Buzzer Volume Control (0x81)**.

This code sample is also contained in the following file in the PPT 4100 Software Development Kit (SDK):

c:\SDK4100\SAMPLES\MANUAL\CHAP1\MISC1.C

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

Get/Set Resume Mask Register

Function: 0x84

Description

Writes a resume mask directly to the gate array or reads the current mask value.

> **Note:** This service should *not* be called by applications. An equivalent function (**Set Wakeup Masks, Interrupt 0xB1, Function 0x01**) is provided for application programs. See *Power Management API Commands (Descriptions)* in *XSYMBIOS/ Power Management* for a description of this equivalent function.

Interrupt

0x32

Input Registers

AH = 0x84

AL = Mask value encoded as a series of bits. If any bit is set, the corresponding resume source is masked. If a bit is cleared, the corresponding resume source is enabled.

The mask bit assignments are as follows:

Bit 0 - Right side switch
Bit 1 - Left side switch
Bit 2 - Pen down
Bit 3 - RS232C ring

If Bit 7 is set when the function is called, the function returns the resume source instead of writing it.

Output Registers

AL contains the resume source mask.

Example

No code sample is provided for this function. For a code sample illustrating the equivalent XSYMBIOS power management service (**Set Wakeup Masks, INT 0xB1, Function 0x01**), see *Power Management API Commands (Descriptions)* in *XSYMBIOS/ Power Management*.

Get XSYMBIOS Version Number

Function: 0x85

Description

Returns the XSYMBIOS version number.

Interrupt

0x32

Input Registers

AH = 0x85

BX = 0x00

Output Registers

BH = Major version number for XSYMBIOS

BL = Minor version number for XSYMBIOS

Example

The following code sample illustrates the **Get XSYMBIOS Version Number (0x85)** function.

This code sample is also contained in the following file in the PPT 4100 Software Development Kit (SDK):

c:\SDK4100\SAMPLES\MANUAL\CHAP1\VERSION.C

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

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

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

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

/*  Define the Services by Interrupt Vector number              */

#define XB_MISC_INT       0x32      /* General system services interrupt */

#define XB_MSGTVERS       0x85      /* get XSYMBIOS version #      */

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

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

/********** Get XSYMBIOS Version function *********************************

void xb_GetXSYMBIOSVersion(unsigned char _far *major,
                           unsigned char _far *minor)
{
    inregs.h.ah = XB_MSGTVERS;
    inregs.x.bx = 0xffff;
    int86(XB_MISC_INT, &inregs, &outregs);
    *major = outregs.h.bh;
    *minor = outregs.h.bl;
```

```
void main()
{
    /* Local Variables ************************************* /

    char major; /* major version number of XSYMBIOS* /
    char minor; /* minor version number of XSYMBIOS* /

    /* Extended BIOS call to get XSYMBIOS version number and machine type* /

    xb_GetXSYMBIOSVersion((char _far *) &major,(char _far *) &minor);

    printf("\nXSYMBIOS Version number %x.%02x\n", major, minor);

}
```

Note: See the code samples for other functions for illustrations of **Get XSYMBIOS Version Number (0x85)**.

Get/Set Viewing Angle

Function: 0x86

Description

This function supports the following services:

- Reads and returns the current LCD viewing angle (contrast).
- Allows an application to set the LCD viewing angle.

Interrupt

0x32

Input Registers

AH = 0x86

AL = Subfunction code as follows:

0x80: Gets the current LCD viewing angle
0 through 31 (decimal): Sets the LCD viewing angle

Output Registers

AL = 0 through 31 (i.e., the current setting of the LCD viewing angle)

Example

The following code sample illustrates the following General System Service functions:

Get/Set Viewing Angle (Function 0x86), described above
Get XSYMBIOS Version Number (Function 0x85), described earlier in this section

It also illustrates the **Delay** service (**INT 0xAC**, **Function 0x08**) described in *XSYMBIOS Timer Services (INT 0xAC) (List)*.

This example is contained in the following file in the PPT 4100 Software Development Kit (SDK):

c:\SDK4100\SAMPLES\MANUAL\CHAP1\VIEWANGL.C

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

/* Include Files ** /

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

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

#define ONE_SECOND 1000        /* 1000 ms. = 1 sec.      */
#define UNDEFINED  0xff

#define XB_TMRDELAY    0x08

#define XB_MSBAKLIT        0x82          /* backlight state control */
#define XB_MISC_INT        0x32          /* General System Services */
#define XB_MSVWANGL        0x86          /* viewing angle (contrast) control */
#define XB_MSGTVERS        0x85          /* XSYMBIOS version number */

#define XB_TIME_INT        0xAC        /* Timer Services interrupt vector */

#define XB_MSSTSCAN        0x8000
#define XB_MSGTSCAN        0x8080
#define XB_MSMAPKEY        0x8800
#define XB_MSGETKEY        0x8801
#define XB_MSCANKEY        0x8802

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

union REGS inregs;          /* input regs to int86        */
union REGS outregs;         /* output regs from int86     */
struct SREGS segregs;       /* segment regs from/to int86x */
```

```
/********** Get XSYMBIOS Version Number Function ************/

unsigned char xb_GetXBIOSVersion(unsigned char __far *major,
                                 unsigned char __far *minor)
{
    inregs.h.ah = XB_MSGTVERS;
    inregs.x.bx = 0xffff;
    int86(XB_MISC_INT, &inregs, &outregs);
    *major = outregs.h.bh;
    *minor = outregs.h.bl;
    return(outregs.h.al);
}


/************* Timer Services Delay Function *************************/
void xb_Delay(long milliseconds)
{
    inregs.h.ah = XB_TMRDELAY;
    inregs.x.cx = *(unsigned int *)&milliseconds;
    inregs.h.dl = *((unsigned char *)&milliseconds+2);
    int86(XB_TIME_INT, &inregs, &outregs);
}

/*************Set Viewing Angle Function ********************************/

void xb_SetViewingAngle(char contrast)
{
    inregs.h.ah = XB_MSVWANGL;
    inregs.h.al = contrast;
    int86(XB_MISC_INT, &inregs, &outregs);
}
```

```
/********** main() checks loaded status of XSYMBIOS TSR ************
************ and sets viewing angle if it is loaded ********************/
void main ()
{
    unsigned char contrast, major, minor;

    long delay_in_ms = ONE_SECOND;

    /* Check if XBIOS interrupt vector is defined                */

    if (_dos_getvect(XB_MISC_INT) == NULL)
    {
        fprintf (stderr,"XSYMBIOS not loaded.\nProgram aborted.");
        exit(0);
    }
    xb_GetXSYMBIOSVersion(&major, &minor);
    if (major == UNDEFINED)
    {
        fprintf (stderr,"XSYMBIOS not loaded.\nProgram aborted.");
        exit(0);
    }
    fprintf(stderr,"XSYMBIOS reported version number as %x.%02x\n",
            major,minor);

    for (contrast = 7;contrast < 21;contrast++)
    {
        printf ("\rSetting viewing angle (contrast) to %d", contrast);
        xb_SetViewingAngle (contrast);
        xb_Delay(delay_in_ms);
    }
}
```

XSYMBIOS Timer Services (INT 0xAC) (List)

The PPT 41XX Extended BIOS TSR (XSYMBIOS.EXE) supports the timer services listed in Table 1-4. Timers are specified in milliseconds. The minimum duration of the timer is 55 milliseconds (1 timer tick). The maximum duration of any timer is one hour.

XSYMBIOS.EXE supports an application programming interface to the Timer Services listed in Table 1-4. All functions in this API are called by executing **INT 0xAC** with an appropriate function code in register AH. Table 1-4 lists the commands that call these functions and their associated function codes. These services are described in the following section.

Table 1-4. XSYMBIOS Timer Services (INT 0xAC)

Function Code	Timer Service Name
0x00	Allocate Timer
0x01	Deallocate Timer
0x02	Start System Timer
0x03	Start Event Timer
0x04	Reset Timer
0x05	Suspend Timer Operation
0x06	Resume Timer Operation
0x07	Check Timer
0x08	Delay
0x09	Restart Timer

XSYMBIOS Timer Services (Descriptions)

The following descriptions of the functions in Table 1-4 are given in function code order.

Allocate Timer

Function: 0x00

Description

Allocates a timer from the timer pool and returns an 8-bit handle to the timer. If the pool is exhausted, it returns an error.

Interrupt

0xAC

Input Registers

AH = 0x00

Output Registers

The Carry Flag is set if no timer is available.

The Carry Flag is reset if a timer has been allocated.

AL = Timer number, if the Carry Flag is reset.

Example

The following code sample illustrates the Timer Service functions below:

Allocate Timer (Function 0x00)
Deallocate Timer (Function 0x01)
Start System Timer (Function 0x02)
Check Timer (Function 0x07)

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\TIMER1.C

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

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

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

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

#ifndef FALSE
#  define FALSE 0
#endif

#define XB_TIME_INT       0xAC    /* Timer services interrupt vector */
#define XB_TMRALLOC       0x00    /* allocate timer    */
#define XB_TMRDALLO       0x01    /* deallocate timer  */
#define XB_TMRSTSYS       0x02    /* start system timer */
#define XB_TMRCHECK       0x07    /* check timer       */

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

typedef enum {T_DEALLOC, T_ACTIVE, T_EXPIRED, T_ALLOC,
              T_PENDING, T_SUSP} TIMER_STATE;
typedef enum {TIMER, EVENT_TIMER} TIMER_TYPE;

typedef struct
{
  TIMER_STATE     state;
  TIMER_TYPE      type;
  WORD            ticks;
  void            (*routine)();
} TIMER_STATUS;

/* 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 Function Prototypes ****************************/

BYTE xb_AllocTimer(BYTE far *timer);    /* allocated timer number */
BYTE xb_DeallocTimer(BYTE timer);       /* timer number to deallocate */

BYTE xb_StartTimer(BYTE timer,                      /* timer number to set */
                   unsigned long millisecs);        /* timer count in milliseconds */

BYTE xb_CheckTimer(BYTE timer,                        /* timer number to check */
                   TIMER_STATUS *status_ptr);  /* ptr to timer status struct */
```

```
int main (void)
{

    BYTE timer;                     /* timer number provided by system */
    TIMER_STATUS status_blk;        /* timer status reported by system */

    /* First, request a timer from the system */
    if (xb_AllocTimer(&timer) != FALSE)
    {
        printf("Fatal error: No free timer available\n");
        return 1;
    }

    /* Start timer for 10 seconds */
    xb_StartTimer(timer, 10000UL);

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

    /* Get timer status */
    xb_CheckTimer(timer, &status_blk);

    /* If timer is still active, report time remaining */
    if (status_blk.state == T_ACTIVE)
        printf("Timer still has %u ticks remaining\n", status_blk.ticks);

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

    /* When done, return the timer to the system */
    xb_DeallocTimer(timer);

    return 0;
}

/************ Allocate Timer Function ***********************/
```

```
BYTE xb_AllocTimer(BYTE far *timer)    /* allocated timer number */
{
  inregs.h.ah = XB_TMRALLOC;

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

  *timer = outregs.h.al;

  return outregs.x.cflag;
}

/************ Deallocate Timer Function ********************/

BYTE xb_DeallocTimer(BYTE timer) /* timer number to deallocate */
{
  inregs.h.ah = XB_TMRDALLO;
  inregs.h.al = timer;

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

  return outregs.x.cflag;
}


/************** Start System Timer Function **********************/

BYTE xb_StartTimer(BYTE timer,                    /* timer number to start */
                   unsigned long millisecs)       /* timer value in millisecs */
{
    inregs.h.ah = XB_TMRSTSYS;
    inregs.h.al = timer;
    inregs.x.cx = *(WORD *)&millisecs;
    inregs.h.dl = *((BYTE *)&millisecs+2);

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

    return outregs.x.cflag;
}

/*************** Check Timer Function ***********************/
```

```
BYTE xb_CheckTimer(BYTE timer,                      /* timer number to check */
                   TIMER_STATUS *status_ptr)        /* ptr to timer status struct */
{
    WORD       offset_temp;

    outregs.x.cflag = 0;

    /* Note: inline assembly required due to use of the BP register */
    __asm
    {
        mov ah, XB_TMRCHECK
        mov al, timer

        int XB_TIME_INT

        jnc ok
        mov outregs.x.cflag, 1
        ok: mov offset_temp, bp
        mov outregs.x.ax, ax
        mov outregs.x.cx, cx
        mov segregs.es, es
    }

    status_ptr->state = outregs.h.al;
    status_ptr->type = outregs.h.ah;
    status_ptr->ticks = outregs.x.cx;
    status_ptr->routine = MK_FP(segregs.es, offset_temp);

    return outregs.x.cflag;
}
```

Deallocate Timer

Function: 0x01

Description

Returns a previously allocated timer to the free timer pool. If the specified timer was not previously allocated, it returns an error.

Interrupt

0xAC

Input Registers

AH = 0x01

AL = Timer number

Output Registers

The Carry Flag is set if the timer index is out of range; otherwise, it is reset.

Example

For a code sample that illustrates the **Deallocate Timer (0x01)** function, see the **Example** for **Allocate Timer (0x00)**.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\TIMER1.C

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

Start System Timer

Function: 0x02

Description

Starts a previously allocated timer to time a specified number of milliseconds (or timer ticks). When the timer expires, it sets a status bit indicating this. The **Check Timer (Function 0x07**) returns the timer status so the user can determine when the timer has expired.

Interrupt

0xAC

Input Registers

AH = 0x02

AL = Timer number

DL = MSB of time (in milliseconds)

CX = Time (in milliseconds)

Output Registers

The Carry Flag is set if the timer index is out of range; otherwise, it is reset.

Notes

If DL is set to 0xFF, then CX specifies the time in timer ticks (units of 55 milliseconds). Otherwise, time is in milliseconds.

Example

For a code sample that illustrates the **Start System Timer (0x02)** function, see the **Example** for **Allocate Timer (0x00)**.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\TIMER1.C

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

Start Event Timer

Function: 0x03

Description

Starts a previously allocated timer to time a specified number of milliseconds (or timer ticks). When the timer expires, it sets the timer status to complete and calls a specified routine in the application.

Interrupt

0xAC

Input Registers

AH = 0x03

AL = Timer number

DL = MSB of time (in milliseconds)

CX = Time (in milliseconds)

ES:BP = Address of the routine to execute

Output Registers

The Carry Flag is set if the timer index is out of range; otherwise, it is reset.

Notes

If DL is set to 0xFF, CX specifies the time in timer ticks (units of 55 milliseconds).

Since the call to the specified routine in the application is made from the interrupt service routine, the application should follow the rule for an interrupt handler, since registers are not stored / restored around the call to the event.

This service can be called from the background.

The event may occur before control returns to the calling application.

This routine is called via a FAR call and therefore must issue a FAR return.

Example

The following code sample illustrates the Timer Service functions below:

Allocate Timer (Function 0x00)
Deallocate Timer (Function 0x01)

Start Event Timer (Function 0x03)
Reset Timer (Function 0x04)
Suspend Timer Operation (Function 0x05)
Resume Timer Operation (Function 0x06)
Restart Timer (Function 0x09)

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\TIMER2.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 XB_TIME_INT       0xAC    /* Timer services interrupt vector */
#define XB_TMRALLOC       0x00    /* allocate timer    */
#define XB_TMRDALLO       0x01    /* deallocate timer  */
#define XB_TMRSTEVT       0x03    /* start event timer */
#define XB_TMRRESET       0x04    /* reset timer       */
#define XB_TMRSSPND       0x05    /* suspend timer     */
#define XB_TMRRESUM       0x06    /* resume timer      */

typedef unsigned char BYTE;       /* 8 bit data type       */
typedef unsigned short WORD;      /* 16 bit data 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 timeout = FALSE;       /* true if timeout occurred */

/* Public Function Prototypes ************************** */

BYTE xb_AllocTimer(BYTE far *timer);   /* allocated timer number */
BYTE xb_DeallocTimer(BYTE timer);      /* timer number to deallocate */

BYTE xb_StartEventTimer(BYTE timer,                /* timer number to start */
                        unsigned long millisecs,   /* timer count in milliseconds */
                        void (far *routine)());    /* address of routine */
```

```
BYTE xb_ResetTimer(BYTE timer);             /* timer number to reset */
BYTE xb_SuspendTimer(BYTE timer);           /* timer number to suspend */
BYTE xb_ResumeTimer(BYTE timer);            /* timer number to resume */

BYTE xb_RestartTimer(BYTE timer,                    /* timer number to restart */
                     unsigned long millisecs);      /* timer count in milliseconds */

/* Local Function Prototypes ******************************/

void far __loadds watchdog(void);
```

```
int main (void)
{

    BYTE        timer;        /* timer number provided by system */

    /* First, request a timer from the system */
    if (xb_AllocTimer(&timer) != FALSE)
    {
        printf("Fatal error: No free timer available\n");
        return 1;
    }

    /* Start event timer to activate watchdog in 10 seconds */
    xb_StartEventTimer(timer, 10000UL, watchdog);

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

    /* Suspend the watchdog timer while we wait for the operator */
    xb_SuspendTimer(timer);

    /* Wait for operator action */
    printf("Press any key to continue\n");
    getchar();

    /* Resume the watchdog timer */
    xb_ResumeTimer(timer);

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

    /* Restart watchdog timer operation with 5 second timeout */
    xb_RestartTimer(timer, 5000UL);

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

```
    /* When done, return the timer to the system */
    xb_DeallocTimer(timer);

    return 0;
}

/************* Example Timer Activated (Callback Routine*************
* SYNOPSIS:     void far __loadds watchdog(void)
* DESCRIPTION:  Example timer activated (callback) routine
* PARAMETERS:   None
* RETURN VALUE: None
* INPUTS:       None
* OUTPUTS:      None
*
* NOTES:        Timer activated (callback) routines must:
*
*               1) Have stack checking turned off, and
*               2) Load the DS register on entry
*
*             This example utilizes Microsoft C mechanisms to
*             meet these two requirements.
*
************************************************************** /

#pragma check_stack(off)
void far __loadds watchdog(void)
{
    timeout = TRUE;

}
#pragma check_stack(on)
```

```
/************ Allocate Timer Function ************************/

BYTE xb_AllocTimer(BYTE far *timer)        /* allocated timer number */
{
     inregs.h.ah = XB_TMRALLOC;

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

     *timer = outregs.h.al;

     return outregs.x.cflag;
}


/*************** Deallocate Timer Function ********************/

BYTE xb_DeallocTimer(BYTE timer)         /* timer number to deallocate */
{
     inregs.h.ah = XB_TMRDALLO;
     inregs.h.al = timer;

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

     return outregs.x.cflag;
}

/************** Start Event Timer Function ********************/

BYTE xb_StartEventTimer(BYTE timer,                  /* timer number to start */
                        unsigned long millisecs,  /* timer value in milliseconds */
                        void (far *routine)())       /* address of routine */
```

```
{
    WORD        routine_seg;
    WORD        routine_offset;
    static WORD bp_save;

    inregs.x.cx = *(WORD *)&millisecs;
    inregs.h.dl = *((BYTE *)&millisecs+2);
    routine_seg = FP_SEG(routine);
    routine_offset = FP_OFF(routine);

    outregs.x.cflag = 0;

    /* Note: inline assembly required due to use of the BP register */
    __asm
    {
        mov ah, XB_TMRSTEVT
        mov al, timer
        mov cx, inregs.x.cx
        mov dl, inregs.h.dl
        mov bx, routine_seg
        mov es, bx
        mov bp_save, bp
        mov bp, routine_offset

        int XB_TIME_INT

        jnc ok
        mov outregs.x.cflag, 1
        ok: mov bp, bp_save
    }

    return outregs.x.cflag;
}

/************* Reset Timer Function ************************** /
```

```
BYTE xb_ResetTimer(BYTE timer)   /* timer number to reset */
{
     inregs.h.ah = XB_TMRRESET;
     inregs.h.al = timer;

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

     return outregs.x.cflag;
}
```

```
/*************** Suspend Timer Operation Function ************************/

BYTE xb_SuspendTimer(BYTE timer)      /* timer number to suspend */
{
    inregs.h.ah = XB_TMRSSPND;
    inregs.h.al = timer;

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

    return outregs.x.cflag;
}

/**************** Resume Timer Operation Function *************/

BYTE xb_ResumeTimer(BYTE timer)      /* timer number to resume */
{
    inregs.h.ah = XB_TMRRESUM;
    inregs.h.al = timer;

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

    return outregs.x.cflag;
}

/*************** Restart Timer Function ****************************/

BYTE xb_RestartTimer(BYTE timer,                    /* timer number to restart */
                     unsigned long millisecs) /* timer count in milliseconds */

{
    inregs.h.ah = 0x09;
    inregs.h.al = timer;
    inregs.x.cx = *(WORD *)&millisecs;
    inregs.h.dl = *((BYTE *)&millisecs+2);

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

    return outregs.x.cflag;
}
```

Reset Timer

Function: 0x04

Description

Disables a previously allocated System or Event Timer. It marks the timer inactive and clears any current time and notification address.

Interrupt

0xAC

Input Registers

AH = 0x04

AL = Timer number

Output Registers

The Carry Flag is set if the timer index is out of range; otherwise, it is reset.

Notes

This service can be called from the background.

Example

For a code sample that illustrates the **Reset Timer (0x04)** function, see the **Example** for **Start Event Timer (0x03)**.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\TIMER2.C

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

Suspend Timer Operation

Function: 0x05

Description

Temporarily stops an active timer and marks the timer status as suspended. Use **Resume Timer** (**Function 0x06**) to resume the timer. **Suspend Timer** has no effect when the timer is not active.

Interrupt

0xAC

Input Registers

AH = 0x05

AL = Timer number

Output Registers

The Carry Flag is set if the timer index is out of range; otherwise, it is reset.

Notes

This service sets the timer status to “Suspended”, but does not clear the timer count or address. It simply stops decrementing the counter.

This function can be called from the background.

Example

For a code sample that illustrates the **Suspend Timer Operation (0x05)** function, see the **Example** for **Start Event Timer (0x03)**.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\TIMER2.C

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

Resume Timer Operation

Function: 0x06

Description

Restarts a previously suspended timer to resume counting from the point at which it was stopped. The function has no effect on a timer unless it was first suspended.

Interrupt

0xAC

Input Registers

AH = 0x06

AL = Timer number

Output Registers

The Carry Flag is set if the timer index is out of range; otherwise, it is reset.

Notes

This service restarts a timer stopped by **Suspend Timer** (**Function 0x05**).

This function can be called from the background.

Example

For a code sample that illustrates the **Resume Timer Operation (0x06)** function, see the **Example** for **Start Event Timer (0x03)**.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\TIMER2.C

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

Check Timer

Function: 0x07

Description

Reads the current state of a timer and returns:

- Timer type (System, Event)
- Timer status (active, suspended, expired, idle)
- Ticks to completion
- Pointer to callback function (if Event Timer)

Interrupt

0xAC

Input Registers

AH = 0x07

AL = Timer number

Output Registers

The Carry Flag is set if the timer index is out of range; otherwise, it is reset.

The Zero Flag is set if the timer expires; otherwise, it is reset.

AL = Timer status, as follows:

0x00: Deallocated
0x01: Active
0x02: Expired
0x03: Allocated (inactive)
0x04: Pending
0x05: Suspended

AH = Timer type, as follows:

0x00: Timer
0x01: Event timer

CX = Current timer ticks

ES:BP = Dispatch address (if AH = 0x01)

Notes

This function can be called from the background.

To convert Current Timer Ticks to milliseconds, multiply the value returned in CX by 55.

Example

For a code sample that illustrates the **Check Timer (0x07)** function, see the **Example** for **Allocate Timer (0x00)**.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\TIMER1.C

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

Delay

Function: 0x08

Description

Enters power saving mode for a specified time. When the time expires, it restores the power state and returns to the calling application.

Interrupt

0xAC

Input Registers

AH = 0x08

DL = MSB of delay (See **Notes** section below)

CX = Delay (in milliseconds)

Output Registers

None

Notes

If DL is set to 0xFF, CX specifies the time in timer ticks (units of 55 milliseconds). Otherwise, time is in milliseconds. The maximum time in milliseconds is 1,800,000 (30 minutes).

Example

The following code sample illustrates the **Delay** function.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\DELAY.C

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

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

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


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

#define XB_DELAY            0x08    /* delay function */
#define XB_TIME_INT         0xAC    /* Timer services interrupt vector */

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

union REGS inregs;      /* input regs to int86     */
union REGS outregs;     /* output regs from int86  */
int int_function = 0;   /* INT function number     */



/************** Delay Function ******************************

void xb_Delay(long milliseconds)
{
    inregs.h.ah = XB_DELAY;
    int_function = XB_TIME_INT;
    inregs.x.cx = *(unsigned int *)&milliseconds;
    inregs.h.dl = *((unsigned char *)&milliseconds+2);
    int86(int_function, &inregs, &outregs);
}
```

```
void main()
{
/* Local Variables ************************************************/


    static struct _dostime_t DOSTIME;
    char *tbuf={"##:##:##.#"};
    unsigned long delay_in_ms;

    /* The maximum delay is 0xffffff which is approximately      */
    /* 16777. seconds (approximately 4 hours and 39 minutes)     */

    delay_in_ms=5000;                    /* 5 sec delay  */
    _dos_gettime(&DOSTIME);
    printf("\n%02d:%02d:%02d.%02d is time before %d sec. XB_Delay",
          DOSTIME.hour, DOSTIME.minute, DOSTIME.second,
          DOSTIME.hsecond, delay_in_ms/1000);

    /* This service enters the power saving mode for a specified  */
    /* time, and restores the power state and returns to the      */
    /* caller when the time has expired.                          */

    xb_Delay(delay_in_ms);

    _dos_gettime(&DOSTIME);
    printf("\n%02d:%02d:%02d.%02d is time AFTER XB_Delay",
          DOSTIME.hour, DOSTIME.minute, DOSTIME.second,
          DOSTIME.hsecond);
}
```

Restart Timer

Function: 0x09

Description

Restarts a timer for a specified duration. This function can be used to restart a background event timer without having to re-specify the callback address.

Interrupt

0xAC

Input Registers

AH = 0x09

AL = System Timer Number

DL = MSB of time (See **Notes** section below)

CX = Time (in milliseconds)

Output Registers

The Carry Flag is set if the timer index is out of range; otherwise, it is reset.

Notes

If DL is set to 0xFF, CX specifies the time in timer ticks (units of 55 milliseconds). Otherwise, time is in milliseconds. The maximum time in milliseconds is 1,800,000 (30 minutes).

This function can be called from the background.

Example

For a code sample that illustrates the **Restart Timer (0x09)** function, see the **Example** for **Start Event Timer (0x03)**.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\TIMER2.C

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

XSYMBIOS Sound Services (INT 0xAD) (List)

The PPT 41XX Extended BIOS TSR (XSYMBIOS.EXE) supports the sound services listed in Table 1-5.

XSYMBIOS supports an application programming interface to the sound services listed in Table 1-5. All functions in this API are called by executing **INT 0xAD** with an appropriate function code in register AH. Table 1-5. lists the commands that call these functions and their associated function codes. These services are described in the following section.

Table 1-5. XSYMBIOS Sound Services (INT 0xAD)

Function Code	Sound Service Name
0x00	**Buzzer On**
0x01	**Buzzer Off**
0x02	**Beep for Duration**
0x03	**Get/Set Speaker Volume**

XSYMBIOS Sound Services (Descriptions)

The following descriptions of the functions in Table 1-5. are given in function code order.

Buzzer On

Function: 0x00

Description

Enables the buzzer and programs the buzzer frequency, causing the buzzer to emit a continuous tone until it is switched off.

Interrupt

0xAD

Input Registers

AH = 0x00

BX = Buzzer frequency (in Hertz)

> **Note:** If BX is set to zero, XSYMBIOS selects the default frequency at which the buzzer is perceptibly louder.

Output Registers

None

Example

The following code sample illustrates the sound services below:

Buzzer On (Function 0x00)
Buzzer Off (Function 0x01)

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\SOUND2.C

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

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

#include <dos.h>

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

/*  Define the Services by Interrupt Vector number              */

#define XB_TIMER_INT 0xAC       /* Timer services interrupt vector */
#define XB_DELAY     0x08       /* delay function */
#define XB_SOUND_INT 0xAD       /* Sound services interrupt vector */

enum BUZZER_CMD {BUZZER_ON, BUZZER_OFF};

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

union REGS inregs;          /* input regs to int86       */
union REGS outregs;         /* output regs from int86    */
struct SREGS segregs;       /* segment regs from/to int86x  */

void xb_Delay(long milliseconds)
{
     inregs.h.ah = XB_DELAY;
     inregs.x.cx = *(unsigned int *)&milliseconds;
     inregs.h.dl = *((unsigned char *)&milliseconds+2);
     int86(XB_TIMER_INT, &inregs, &outregs);
}

/************** Buzzer On Function ************************ */

void xb_BuzzerOn(int frequency)
{
    inregs.h.ah = BUZZER_ON;
    inregs.x.bx = frequency;
    int86(XB_SOUND_INT, &inregs, &outregs);
}
```

```
/****************** Buzzer Off Function ********************/

void xb_BuzzerOff()
{
     inregs.h.ah = BUZZER_OFF;
     int86(XB_SOUND_INT, &inregs, &outregs);
}


int main()
{
/* Local Variables ************************************/

     int frequency;                  /* Buzzer frequency in Hertz */
     frequency = 440;
     xb_BuzzerOn(frequency);         /* Turn the Buzzer On       */
     xb_Delay (1000);                /* Delay for 1 second       */
     xb_BuzzerOff();                 /* Turn the Buzzer Off      */

}
```

Buzzer Off

Function: 0x01

Description

Switches off the buzzer.

Interrupt

0xAD

Input Registers

AH = 0x01

Output Registers

None

Example

For a code sample that illustrates the **Buzzer Off** function, see the **Example** for **Buzzer On** (**Function 0x00**).

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\SOUND2.C

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

Beep for Duration

Function: 0x02

Description

Switches the buzzer on at a specified frequency for a specified time. When the time expires, the buzzer is switched off.

Beeps may be issued in two modes:

- *the foreground mode*
  The routine enters a power saving mode and does not return to the caller until the time expires.
- *the background mode*
  The routine starts the buzzer and returns immediately to the caller, switching the beep off when the timer expires.

The service routine relies on XSYMBIOS timers. If no timer is available, then the duration of the beep is 0 milliseconds.

Interrupt

0xAD

Input Registers

AH = 0x02

AL = Mode, as follows:

0x00: Foreground
0x01: Background

BX = Buzzer frequency (in Hertz)

DX = MSB of delay time (See **Notes** section below)

CX = Delay time (in milliseconds)

Output Registers

None

Notes

If DL is set to 0xFF, CX specifies the time in timer ticks (units of 55 milliseconds). Otherwise, time is in milliseconds. The maximum time in milliseconds is 1,800,000 (30 minutes).

Example

The following code sample illustrates the **Beep for Duration (Function 0x02)** service.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\SOUND3.C

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

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

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

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

#define FOREGROUND      0
#define BACKGROUND      1

#define XB_SOUND_INT    0xAD    /* Sound services interrupt vector */

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

union REGS inregs;          /* input regs to int86       */
union REGS outregs;         /* output regs from int86    */
struct SREGS segregs;       /* segment regs from/to int86x  */
```

```
/*************** Beep for Duration Function *******************************/

void xb_GenerateBeep(char fgmode,              /* foreground/background mode */
                     unsigned int frequency,/* beep freq in Hertz */
                     unsigned long delay)    /* delay in millisec */
{
    inregs.h.ah = 0x02;
    inregs.h.al = fgmode;
    inregs.x.bx = frequency;
    inregs.x.cx = *(unsigned int *)&delay;
    inregs.x.dx = *((unsigned int *)&delay+1);

    int86(XB_SOUND_INT, &inregs, &outregs);
}

void main()
{
/* Local Variables ************************************************/

    char fgmode;              /* foreground / background mode */
    unsigned int frequency;   /* beep frequency in Hertz     */
    unsigned long delay;      /* delay in milliseconds       */

    fgmode = BACKGROUND;
    frequency = 440;
    delay = 500;
    xb_GenerateBeep(fgmode, frequency, delay);
}
```

Get/Set Speaker Volume

Function: 0x03

Description

Returns the current speaker volume setting, or allows an application to select either high or low speaker volume.

Interrupt

0xAD

Input Registers

AH = 0x03

AL = Subfunction, as follows:

0x00: Get speaker volume
0x01: Set speaker volume

If AL = 0x01, then:

BL = Desired volume, as follows:

0x00 = Low
0x01 = High

Output Registers

If AL = 0x00, then:

BL = Current speaker volume, as follows:

0x00 = Low
0x01 = High

Example

The following code sample illustrates the **Get/Set Speaker Volume (Function 0x03)** service.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\SOUND1.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 XB_SOND_INT     0xAD        /* Sound services interrupt vector */
#define XB_BEEP         0x02        /* generate beep                   */
#define XB_SPKRVOL      0x03        /* get/set speaker volume          */

/* Define volume levels */
#define VOL_LOW         0
#define VOL_HIGH        1

/* Define tone modes */
#define MODE_FOREGND    0
#define MODE_BACKGND    1

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

void xb_GetBuzzerVol2 (BYTE far *volume);
void xb_SetBuzzerVol2 (BYTE volume);
BYTE xb_Beep(BYTE mode, WORD freq, DWORD duration);
```

```
void main (void)
{
    BYTE volume;          /* buzzer volume setting */

    /* Set buzzer volume to low */
    xb_SetBuzzerVol2(VOL_LOW);

    /* Obtain and report current buzzer volume setting */
    xb_GetBuzzerVol2(&volume);
    printf("Current buzzer volume is %s\n\n", volume ? "high" : "low");

    /* Generate a sample low beep (1000 Hz, 0.5 Sec) */
    xb_Beep(MODE_FOREGND, 1000, 500);

    /* Set buzzer volume to high */
    xb_SetBuzzerVol2(VOL_HIGH);

    /* Obtain and report current buzzer volume setting */
    xb_GetBuzzerVol2(&volume);
    printf("Current buzzer volume is %s\n\n", volume ? "high" : "low");

    /* Generate a sample high beep (1000 Hz, 0.5 Sec) */
    xb_Beep(MODE_FOREGND, 1000, 500);

    return;
}

/************* Get Speaker (Buzzer) Volume Function ****************/

void xb_GetBuzzerVol2(BYTE far *volume)  /* current volume setting */
{
    inregs.h.ah = XB_SPKRVOL;
    inregs.h.al = 0x00;

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

    *volume = outregs.h.bl;

    return;
}
```

```
/********** Set Speaker (Buzzer) Volume Function **********/

BYTE xb_SetBuzzerVol2(BYTE volume)  /* volume setting */
{
    BYTE retval;   /* return code */

    if ((volume == VOL_LOW) || (volume == VOL_HIGH))
    {
        inregs.h.ah = XB_SPKRVOL;
        inregs.h.al = 0x01;

        inregs.h.bl = volume;

        int86x(XB_SOND_INT, &inregs, &outregs, &segregs);
        retval = FALSE;
    }
    else
        retval = TRUE;

    return retval;

}

/************** Beep for Duration Function *************************/

BYTE xb_Beep(BYTE mode,          /* operating mode */
             WORD freq,          /* frequency in hertz */
             DWORD duration)     /* duration in milliseconds */
```

```
{
    BYTE retval;          /* return code */

    if ((mode == MODE_FOREGND) || (mode == MODE_BACKGND))
     {
        inregs.h.ah = XB_BEEP;
        inregs.h.al = mode;

        inregs.x.bx = freq;

        inregs.x.dx = (WORD)(duration >> 16);
        inregs.x.cx = (WORD)(duration);

        int86x(XB_SOND_INT, &inregs, &outregs, &segregs);
        retval = FALSE;
    }
    else
        retval = TRUE;

    return retval;
}
```

XSYMBIOS CRC Services (INT 0xAE) (List)

The PPT 41XX Extended BIOS TSR (XSYMBIOS.EXE) supports the CRC services listed in Table 1-6.

XSYMBIOS supports an application programming interface to the CRC services listed in Table 1-6. All functions in this API are called by executing **INT 0xAE** with an appropriate function code in register AH.
Table 1-6 lists the commands that call these functions along and their function codes. These services are described in the following section.

Table 1-6. XSYMBIOS CRC Services (Interrupt 0xAE)

Function Code	CRC Service Name
0x00	**Compute Running CRC-16 on a Byte**
0x01	**Compute Running CRC-16 on a Buffer**
0x02	**Compute Running CRC-32 on a Buffer**

XSYMBIOS CRC Services (Descriptions)

The following descriptions of the functions in Table 1-6 are given in function code order.

Compute Running CRC-16 on a Byte

Function: 0x00

Description

Updates a supplied CRC-16 check-sum with the appropriate value for a supplied byte and returns the modified check-sum.

Interrupt

0xAE

Input Registers

AH = 0x00

AL = Supplied byte

DX = Supplied CRC-16 check-sum

Output Registers

DX = Modified CRC-16 check-sum

Example

The following code sample illustrates the CRC services below:

Compute Running CRC-16 on a Byte (Function 0x00)
Compute Running CRC-16 on a Buffer (Function 0x01)

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\CRC16.C

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

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

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

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

#define XB_CRC_INT          0xAE    /* CRC services interrupt vector */

enum CRC_CMD {
CRC_16_BYTE,                /* CRC-16 a byte at a time  */
CRC_16,                     /* CRC-16 buffer            */
CRC_32};                    /* CRC-32 buffer            */

/* Public Variables ************************************/
union REGS inregs;          /* input regs to int86      */
union REGS outregs;         /* output regs from int86   */
struct SREGS segregs;       /* segment regs from/to int86x  */

/********** Compute Running CRC-16 on a Byte ***************/

void xb_CRC16Byte(char byte, unsigned int _far *runningcrc)
{
    inregs.h.ah = CRC_16_BYTE;
    inregs.h.al = byte;
    inregs.x.dx = *runningcrc;
    int86(XB_CRC_INT, &inregs, &outregs);
    *runningcrc = outregs.x.dx;
}

/********** Compute Running CRC-16 on a Buffer **************/
```

```
void xb_CRC16Buffer(void _far *buffer, unsigned int buffsize,
                        unsigned int runningcrc, unsigned int _far *crc)
{
    inregs.h.ah = CRC_16;
    inregs.x.cx = buffsize;
    inregs.x.dx = runningcrc;
    segregs.es = FP_SEG(buffer);
    inregs.x.si = FP_OFF(buffer);
    int86x(XB_CRC_INT, &inregs, &outregs, &segregs);
    *crc = outregs.x.dx;
}
```

```
void main()
{
    /* format of bi-sync msg is: <stx><msg len><text><etx>          */
    static char msg[] = {"\x02\x00\x05HELLO\x03"};     /* msg buffer */
    char _far *crcbuf = &msg[0];                       /* pointer to buffer */
    unsigned short i;                                  /* loop index */
    unsigned short crc16;                              /* 16-bit crc */
    unsigned short crcbufsize;                         /* buffer size */

    /* Can't use strlen() since buffer contains "\x00"             */

    crcbufsize = sizeof(msg) - 1;

    /* Do CRC calculation a byte at a time.                        */

    crc16 = 0;                                         /* Must be initialized */
    for(i = 0;i < crcbufsize; i++)
        xb_CRC16Byte(msg[i], (unsigned int _far *) &crc16);

    printf("\nCalculated CRC-16 = %04X (HEX) a byte at a time", crc16);

    /* Extended BIOS call to calculate the CRC                     */

    xb_CRC16Buffer(crcbuf, crcbufsize, 0, (unsigned int _far *) &crc16);

    printf("\nCalculated CRC-16 = %04X (HEX) for message ", crc16);

    /* Display the buffer showing non-printable characters in hex.  */
    for(i = 0;i < crcbufsize; i++)
        if (isprint(msg[i]))printf("%c", msg[i]);
        else printf("\\x%02x", msg[i]);
}
```

Compute Running CRC-16 on a Buffer

Function: 0x01

Description

Calculates a CRC-16 check-sum for a supplied buffer and returns the modified check-sum. If a buffer size of zero (0) is specified, the function exits immediately.

Interrupt

0xAE

Input Registers

AH = 0x01

CX = Size of buffer

DX = Supplied CRC-16 check-sum

ES:SI = Address of the buffer to CRC

Output Registers

DX = Modified CRC-16 check-sum

Notes

This service is more efficient than using **CRC-16 Byte** (**Function 00h**) to update each byte in the buffer individually.

Example

For sample code that illustrates the **Compute Running CRC-16 on a Buffer** service, see the **Example** for **Compute Running CRC-16 on a Byte** **(Function 0x00)**.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\CRC16.C

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

Compute Running CRC-32 on a Buffer

Function: 0x02

Description

Calculates a CRC-32 check-sum for a supplied buffer and returns the modified check-sum. If a buffer size of zero (0) is specified, the function exits immediately. See **Notes** section below.

Interrupt

0xAE

Input Registers

AH = 0x02

CX = Size of buffer in bytes. (0 size is treated as 0 and not as 64K.)

ES:SI = Address of buffer to CRC (see **Notes** below)

Output Registers

DX:AX = 32 bit CRC (DX most significant)

Notes

Traditionally when transmitting, the CRC is appended to a message (in the order DH, DL, BH, and BL) and the receiver performs a CRC on the transmitted data, including the transmitted CRC. If the data is received correctly, the residual value (calculated by performing a CRC on the entire message, including the transmitted CRC) is 0xDEBB20E3.

The CRC services normalize the supplied buffer pointer prior to processing the buffer in order to minimize problems with segment wrap-arounds. Thus the maximum buffer size is 64K - (SI & 0x0F).

Example

The following code sample illustrates the **Compute Running CRC-32 on a Buffer** function.

This example is also contained in the PPT 4100 Software Development Kit (SDK) in the following file:

c:\SDK4100\SAMPLES\MANUAL\CHAP1\CRC32.C

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

```
/* Include Files ************************************ */
#include <ctype.h>
#include <dos.h>
#include <stdio.h>

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

/*  Define the Services by Interrupt Vector number              */

#define XB_CRC_INT      0xAE      /* CRC services interrupt vector */

enum CRC_CMD {CRC_NONE, CRC_16, CRC_32};


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

union REGS inregs;          /* input regs to int86       */
union REGS outregs;         /* output regs from int86    */
struct SREGS segregs;       /* segment regs from/to int86x  */

/********* Compute Running CRC-32 on a Buffer ************** */

void xb_CRC32Buffer(void _far *buffer,
                    unsigned int buffsize,
                    unsigned long _far *crc32)
{
    inregs.h.ah = CRC_32;
    inregs.x.bx = *(unsigned int *)crc32;
    inregs.x.cx = buffsize;
    inregs.x.dx = *((unsigned int *)crc32+1);
    segregs.es = FP_SEG(buffer);
    inregs.x.si = FP_OFF(buffer);
    int86x(XB_CRC_INT, &inregs, &outregs, &segregs);
    *(unsigned int *)crc32 = outregs.x.ax;              /* crc lsb    */
    *((unsigned int *)crc32+1) = outregs.x.dx;          /* crc msb    */
}
```

```
void main()
{
    /* format of bi-sync msg is: <stx><msg len><text><etx>          */
    static char msg[] = {"\x02\x00\x05HELLO\x03"};      /* msg buffer */

    char _far *crcbuf = &msg[0];                        /* pointer to buffer */
    unsigned short i;                                   /* loop index */
    long crc32;                                         /* 32-bit crc */
    unsigned short crcbufsize;                          /* buffer size */

    /* Can't use strlen() since buffer contains "\x00"             */

    crcbufsize = sizeof(msg) - 1;

    /* Extended BIOS call to calculate the CRC-32                  */

    xb_CRC32Buffer(crcbuf, crcbufsize, &crc32);

    printf("\nCalculated CRC-32 = %08lX (HEX) for message ", crc32);

    /* Display the buffer showing non-printable characters in hex.   */

    for(i = 0;i < crcbufsize; i++)
        if (isprint(msg[i]))printf("%c", msg[i]);
        else printf("\\x%02x", msg[i]);
}
```

YSYMBIOS–PC-Loadable Version of XSYMBIOS

Overview

XSYMBIOS (Symbol Extensions to ROM BIOS services) TSR executes on the PPT 41XX terminal. YSYMBIOS is a program that is executed on a PC and provides functionality identical to that of XSYMBIOS except that it does *not* access the gate array, does *not* support power management, and *always* reports that the PC is *not* in a PPT 41XX cradle.

Where necessary, YSYMBIOS contains code to support XSYMBIOS functionality without causing problems with the normal operation of the PC on which it has been loaded. This means that application programs for the PPT 41XX can generally be executed on a PC with YSYMBIOS. However, any sections of an application that are not supported by YSYMBIOS cannot be checked.

Differences between XSYMBIOS and YSYMBIOS

Table 1-7 lists by interrupt number and function code XSYMBIOS functions that either do not exist or are disabled in YSYMBIOS, or act in YSYMBIOS differently than in XSYMBIOS.

Table 1-8 provides the same information sorted by the name of the service (XSYMBIOS API command).

Table 1-7. YSYMBIOS Differences (by Interrupt Number)

Interrupt Number	Function Number	Differences in YSYMBIOS
0x32	0x80	Implemented as a NOP.
	0x81	Implemented as a NOP.
	0x82	Implemented as a NOP.
	0x83	Implemented as a NOP.
	0x84	Should not be called by an application. See Interrupt 0xB1, Function 0x01, below, in this table.
0xAD	0x03	No effect since standard PC has no volume control.
0xB1	All functions	All of the power management code is in place but has been disabled in YSYMBIOS to prevent problems caused by accessing a non-existent PPT 41XX Gate Array.
	0x00	Dispatches any registered suspend or resume notifications and returns to the caller as if the system had resumed.
	0x01	Accepts calls to this function and performs validations on the passed parameters, but does *not* enable the wakeup causes.
	0x02	Always returns Bit 4 to indicate that the power switch caused the system to power up and Bit8 or Bit 9 to identify the action taken. (Bit 9 is set after **Function 0x00** or **Function 0x0D** is used to power down the system.)
	0x0D	Dispatches any registered suspend or resume notifications and returns to the caller as if the system had resumed.

Table 1-8. YSYMBIOS Differences (by Service Name)

XSYMBIOS API Command Name	Interrupt/ Function Code	Differences in YSYMBIOS
Get Side Switch Status	0x32 / 0x83	Implemented as a NOP.
Get Wakeup Cause	0xB1 / 0x02	Always returns Bit 4 to indicate that the power switch caused the system to power up and Bit 8 or Bit 9 to identify the action taken. (Bit 9 is set after **Function 0x00** or **Function 0x0D** is used to power down the system.)
Power Down Terminal	0xB1 / 0x00	Dispatches any registered suspend or resume notifications and returns to the caller as if the system had resumed.
Power Management Services	INT 0xB1 All functions	All of the power management code is in place but has been disabled in YSYMBIOS to prevent problems caused by accessing a non-existent PPT 41XX Gate Array.
Select Wakeup Causes	0xB1 / 0x01	Accepts calls to this function and performs validation on the passed parameters, but does *not* enable the wakeup causes.
Set Backlight Brightness	0x32 / 0x82	Implemented as a NOP.
Set Buzzer Volume	0x32 / 0x81	Implemented as a NOP.
Set Resume Mask Register	0x32 / 0x84	Should not be called by application.
Set Scanner LED	0x32 / 0x80	Implemented as a NOP.
Set Speaker Volume	0xAD / 0x03	No effect since standard PC has no volume control.
Suspend SYSTEM	0XB1 / 0X0D	Dispatches any registered suspend or resume notifications and returns to the caller as if the system had resumed.