
Enhanced Parallel Port BIOS Specification

Revision 3.0

March 16, 1993

This is a draft copy of the EPP BIOS specification and subject to change. This BIOS has been adopted for use by the IEEE P1284.3 standard committee. This standard is currently in draft and is not an approved IEEE standard. This document will be included as an Informative Annex of the final IEEE 1284.3 specification.

Note: Later releases of interim EPP BIOS specifications were withdrawn by the EPP Committee and are not, and will not be supported. These versions are: Rev 5.0, Rev 7.0, Version 1.0 and Version 1.1.

Updated: (6/13/96)

For more information contact Larry Stein, Chair IEEE P1284.3.

Web Site: <http://www.fapo.com> email: lstein@fapo.com

Permission is hereby given to make copies of this document in whole, to distribute to any party concerned with the development, implementation, utilization or implementation of this standard.

In Appreciation

We would like to thank the following members of the EPP committee for their contributions to this document, including:

Adaptec
Alliance Technology
Award
Chips and Technology
Compaq Computer Corp.
Databook
Disctec
FarPoint Communications
Freeport Data International
IBM
Intel Corp.
National Semiconductor
OCE Graphics
Parallel Technologies
Phoenix Technologies Ltd.
Printronic Inc.
Rainbow Technologies
Seiko Epson
Standard Microsystems Corp.
SystemSoft
Toshiba
Trantor Systems Ltd.
Wangtek
Xircom Inc.
Zenith Data Systems

For additional information concerning the EPP BIOS specification contact:

Larry Stein
FarPoint Communications
104 East Avenue K-4, Suite F.
Lancaster, California 93535
(805) 726-4420 fax: (805) 726-4438
lstein@fapo.com

Annex A

EPP BIOS

(Informative)

A.1 Background

Concurrent with the development of the original Std 1284 specification, the EPP committee, an industry group chartered to standardize and promote the use of EPP, defined a BIOS level interface to provide easy access to the EPP capability in the host computer. The BIOS interface was important for two reasons: first - there isn't a consistent architecture across various EPP implementations; and second - extensions to this BIOS provided the synchronization point between multiple devices which might be sharing the EPP port through a multiplexor or daisy chain.

A.2 Overview

The Enhanced Parallel Port (EPP) BIOS is provided as a hardware independent method of accessing an EPP port. It provides support for single I/O cycles as well as the high performance block I/O transfers. The EPP BIOS should always use the most optimum method for EPP I/O transfers. For example, if the hardware supports it, the EPP BIOS will perform EPP block I/O using 32 bit I/O.

It is important that client programs *always* use the EPP BIOS to transfer data and avoid making any direct I/O calls to the EPP port. Performing direct I/O has two serious drawbacks. First, not all EPP ports have the same hardware I/O port address map. Second, if the peripheral device is connected via an EPP multiplexor device there is no guarantee that the physical link to that device is currently selected.

A.2.1 EPP BIOS API

The EPP BIOS defines a special printer BIOS (interrupt 17 hex) call - *Installation Check* - that returns a far pointer to the EPP BIOS entry point. This pointer is called the *EPP Vector*. This vector serves as the Application Program Interface, API, for all underlying EPP services. All EPP BIOS calls are executed by making a far call using the EPP Vector. See section A.6 for a coding example. The EPP BIOS is intended as a real mode interface, however, unless otherwise noted, EPP BIOS calls can be made while the CPU is operating in real or protected mode.

The result code for all BIOS calls is always returned in the AH register. A value of zero (0) indicates that the operation was successful. All values defined in this document are in hexadecimal unless otherwise specified.

A.2.2 EPP PRODUCT ID

The EPP Product ID is a mechanism that allows a device driver to uniquely identify an EPP peripheral device. Each EPP peripheral can be assigned a unique sixteen (16) bit value. Product IDs are assigned to the manufacturer of the peripheral product in blocks of eight numbers. The EPP Product ID list is maintained by Larry Stein, Far Point Communications, 104 East Ave K-4, Suite F, Lancaster, CA 93535.

After a peripheral has been reset the first two bytes returned from two successive EPP "Address Read" I/O cycles are defined as the EPP Product ID. In a daisy-chained environment, the device returns its Product ID in response to a command packet (see the Daisy Chain section for more details). The EPP multi-port driver (e.g., the multiplexor/daisy chain device driver) automatically scans and records the Product ID for each peripheral connected to its device ports (see "Query Port" command).

Hardware support for EPP Product ID in EPP peripherals is not mandatory. Peripheral devices that do not support Product ID may provide some kind of installation time, command line, or program/runtime option for defining the Product ID. The Product ID can be set explicitly using the *Set Product ID* BIOS call (see section A.5).

The EPP Product ID is used currently by a number of devices in the marketplace, however new designs are encouraged to use the Std 1284 Device ID string for the same purpose. The Device ID is a text string that identifies Manufacturer, Model, and anything else the manufacturer wishes to include. There is no support in the EPP BIOS for Device ID, however, it is readily available from any peripheral supporting it by the means outlined in Std 1284.

A.2.3 Earlier Versions of This Document

The EPP BIOS has been implemented by various ROM BIOS vendors using revision 3 as a guide. This document closely reflects the earlier rev 3 specification. Additions and changes to the functions between rev 3 and this specification are marked with an asterisk (*) in section A.4.

A.3 Multi-Port Operation

The EPP BIOS assumes that the host's parallel port can be used to connect more than one peripheral device. The EPP BIOS supports two kinds of multi-port configurations: the multiplexor and daisy chain. Peripheral device ports are selected on a time-shared basis; only one device may be selected at a time. Device ports are numbered from one (1) through eight (8).

In both multi-port configurations - multiplexor and daisy chain - interrupts are preserved for devices which are not selected; whenever the port is released an interrupt will be generated if at least one interrupt is pending. However, there is no guaranteed maximum latency with respect to port selection. Peripheral devices must provide sufficient buffering to cope with any real-time considerations.

If both a multiplexor and a daisy chain are present, the multiplexor must be directly attached to the host's EPP port: daisy chains can be attached to the multiplexor's device ports.

A.3.1 EPP MULTIPLEXOR

An EPP multiplexor (mux) is an *external device* that permits two or more parallel port devices to share a single host parallel port. The mux has a single input port and up to eight device ports. The input port connects directly to the host's EPP parallel port. The multiplexor can select any one of its device ports. Once selected the multiplexor provides a *transparent* signaling path between the selected port and host's EPP port. For all practical purposes, when selected, the device port is directly connected to the host's EPP port. Only one device port may be selected at any given time.

An EPP multiplexor is controlled by a vendor supplied EPP multiplexor device driver. The mux driver supports a set of commands that permit a device driver to select a particular device port. The mux driver uses the same API as the EPP BIOS: multiplexor commands are presented as extensions to the EPP BIOS. When the mux device driver is run it hooks itself in front of the EPP BIOS. In this way the mux driver filters all of the BIOS calls, passing all of the standard (non-mux) commands to the underlying EPP BIOS, and executing any multiplexor commands itself. It will also virtualize other EPP BIOS functions, such as Set Mode, so that a specific device driver will not need to care if other devices are being accessed over the same port. The EPP BIOS multiplexor extensions are defined in section A.5.

A.3.2 DAISY CHAINING

The daisy chain is very similar to the multiplexor. The key difference is that the hardware for the port sharing is built into each peripheral - there is no separate external device. A daisy chain peripheral device has two ports: input and output. The device's input port is connected either to the host parallel port or the daisy chain device *in front* of itself. The output port is used to connect to the next peripheral device in the daisy chain. The last device, however, can be a conventional parallel port device - one without daisy chain support.

The daisy chain peripheral device operates in two mode, *pass-through* and *selected*. In pass-through mode, the daisy chain device appears electrically transparent to the system, allowing access to devices further down the chain. When the daisy chain device is selected, it will block (i.e., latch) the data and control lines to any following devices. Once selected, there is a transparent signaling path between the host's EPP port and the selected daisy chain peripheral device.

The daisy chain is controlled by the daisy chain manager (DChain manager). When the DChain manager is run, it hooks itself in front of the EPP BIOS (and possibly the mux

driver). In this way, the DChain manager filters all of the BIOS calls, passing all of the standard (non daisy chain) commands to the underlying EPP BIOS, executing any daisy chain commands itself. It will also virtualize other EPP BIOS functions, such as Set Mode, so that a specific device driver will not need to care if other devices are being accessed over the same port. The EPP BIOS daisy chain extensions are defined in section A.5.

A.3.3 EPP I/O CONCURRENCE

Because of its multi-port nature, the host EPP port should be thought of as a serially reusable I/O resource. EPP device drivers, or other client programs, must first lock the EPP BIOS before making any EPP I/O calls. In a multi-port environment device drivers will contend for I/O access to the EPP port. Two BIOS services - Lock and Unlock - are provided for this purpose.

Device drivers will typically lock the EPP port, perform one or more EPP I/O cycles (e.g., "Write Block"), and then unlock the port. The time interval between lock and unlock should be kept reasonably small to facilitate EPP port sharing.

In a multi-port environment, the "Lock" BIOS call is used to select a particular device port on the multiplexor or daisy chain. Multiplexors use addresses 1-8, and any device may be on any port, daisy chains are also addressed as 1-8, however, any device which does not support the daisy chain protocol should use zero (0) in the daisy chain field of the device port address.

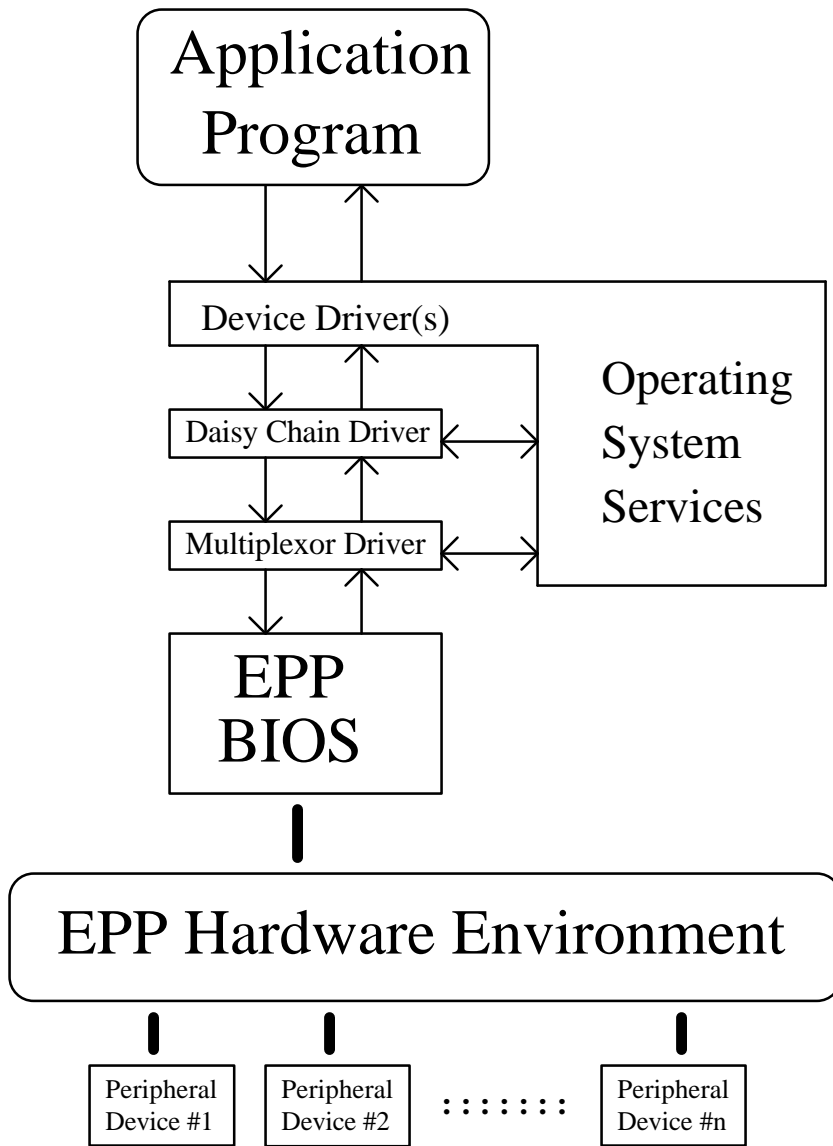


fig. A.1 System Architecture

A.4 EPP BIOS CALLS

A.4.1 Installation Check

Installation Check is used to test for the presence of an EPP port. This call returns a far pointer to the EPP BIOS entry point - the EPP Vector. For compatibility reasons, this call rides on top of the standard Printer Status BIOS call. Device drivers should call this function in initialization and remember the EPP Vector, because a network redirector or other software which intercepts printing may overwrite this call and render the EPP BIOS unrecognizable. *This call can only be made while the CPU is operating in real mode.*

API - int 17

input:	AH	= 2	
	DX	= EPP port number (0-2)	
	AL	= 0	
	CH	= 45 ('E')	
	BL	= 50 ('P')	
	BH	= 50 ('P')	
output:	AH	= 0	- if EPP present
	AL	= 45	- if EPP present
	CX	= 5050	- if EPP present

DX:BX= EPP BIOS entry point - EPP Vector

register usage: AX, BX, CX, DX

A.4.2 Query Config

Query Config returns the EPP port's configuration.

API - EPP Vector

input:	AH	= 0	
	DL	= EPP port number (0-2)	
output:	AH	= error code	
	AL	= Interrupt level of EPP port (0-15)	
		= FF - if interrupts not supported	
	BH	= EPP BIOS revision (MMMMnnnn or M.n)	
		prior to this revision of the EPP BIOS spec,	
		there was no enumeration of this value, so you	
		may not depend on this value for earlier versions	

of this BIOS.

*90 for versions supporting this document
BL = I/O Capabilities
bit 0 = Multiplexor present
bit 1 = PS/2 bi-directional capable
*bit 2 = EPP 1.9 capable
bit 3 = ECP capable
*bit 4 = Reserved
*bit 5 = Centronics FIFO capable
*bit 6 = EPP 1.7 capable
CX = SPP I/O Base address
ES:DI = EPP BIOS manufacturer's information/version text
string (zero terminated)

register usage: AX, BX, CX, DI, ES

A.4.3 Set Mode

Set Mode is used to set the operating mode of the EPP port. *This call can only be made while the CPU is operating in real mode.*

API - EPP Vector

input: AH = 1
DL = EPP port number (0-2)
AL = mode bits
bit 0 = set compatibility mode
bit 1 = set Bi-directional mode
bit 2 = set EPP mode
bit 3 = set ECP mode
*bit 4 = Reserved
*bit 5 = set Centronics FIFO mode
*bit 6 = set EPP 1.7 mode
output: AH = error code

register usage: AX, BX

A.4.4 Get Mode

Get Mode returns the current operating mode of the EPP port. *This call can only be made while the CPU is operating in real mode.*

API - EPP Vector

input: AH = 2
 DL = EPP port number (0-2)

output: AH = error code
 AL = mode bits
 bit 0 = in compatibility mode
 bit 1 = in Bi-directional PS/2 mode
 bit 2 = in EPP mode
 bit 3 = in ECP mode
 *bit 4 = Reserved
 *bit 5 = in Centronics FIFO mode
 *bit 6 = in EPP 1.7 mode
 bit 7 = EPP port interrupts enabled

register usage: AX, BX

A.4.5 Interrupt Control

Interrupt Control is used to enable or disable the interrupt associated with the EPP port. It is intended that this function will handle both the parallel port interrupt enable and the PIC. *This call can only be made while the CPU is operating in real mode.*

API - EPP Vector

input: AH = 3
 DL = EPP port number (0-2)
 AL = 0 - to disable EPP port interrupts
 = 1 - to enable EPP port interrupts

output: AH = error code

register usage: AX, BX

A.4.6 EPP Reset

EPP Reset is used to reset the peripheral device connected to the EPP port. This function will assert the EPP port INIT line for at least 50uS.

API - EPP Vector

input: AH = 4
 DL = EPP port number (0-2)

output: AH = error code

register usage: AX, BX

A.4.7 Address Write

Address Write is used to perform an address write I/O cycle.

API - EPP Vector

input: AH = 5
 DL = EPP port number (0-2)
 AL = device address

output: AH = error code

register usage: AX, BX

A.4.8 Address Read

Address Read is used to perform an address read I/O cycle.

API - EPP Vector

input: AH = 6
 DL = EPP port number (0-2)

output: AH = error code
 AL = returned address/device data

register usage: AX, BX

A.4.9 Write Byte

Write Byte is used to output a single byte via the EPP data port.

API - EPP Vector

input: AH = 7
 DL = EPP port number (0-2)
 AL = data byte

output: AH = error code

register usage: AX, BX

A.4.10 Write Block

Write Block is used to output the contents of a client-defined buffer via the EPP data port.

API - EPP Vector

input: AH = 8
 DL = EPP port number (0-2)

 CX = number of bytes to write (0 = 64K bytes)
 ES:SI = client buffer

output: AH = error code
 CX = bytes not transferred (=0 if no error)

register usage: AX, BX, CX, SI

Implementation Note: REP OUTSB with CX=0 does not move 64K bytes, it moves 0 bytes. If the EPP BIOS uses byte mode I/O, it must explicitly handle the 64K case.

A.4.11 Read Byte

Read Byte is used to input a single byte via the EPP data port.

API - EPP Vector

input: AH = 9
 DL = EPP port number (0-2)

output: AH = error code
 AL = data byte read

register usage: AX, BX

A.4.12 Read Block

Read Block is used to input a stream of bytes into a client buffer via the EPP data port.

API - EPP Vector

input: AH = 0A
 DL = EPP port number (0-2)

 CX = number of bytes to read (0 = 64K)

ES:DI = client buffer

output: AH = error code
 CX = bytes not transferred (=0 if no error)

register usage: AX, BX, CX, DI

Implementation Note: REP INSB with CX=0 does not move 64K bytes, it moves 0 bytes. If the EPP BIOS uses byte mode I/O, it must explicitly handle the 64K case.

A.4.13 Address/Byte Read

Address/Byte Read combines the *Address Write* and *Read Byte* APIs into a single call.

API - EPP Vector

input: AH = 0B
 DL = EPP port number (0-2)
 AL = device address

output: AH = error code
 AL = data byte read

register usage: AX, BX

A.4.14 Address/Byte Write

Address/Byte Write combines the *Address Write* and *Write Byte* APIs into a single call.

API - EPP Vector

input: AH = 0C
 DL = EPP port number (0-2)

 AL = device address
 DH = data byte

output: AH = error code

register usage: AX, BX

A.4.15 Address/Block Read

Address/Block Read combines the *Address Write* and *Read Block* APIs into a single call.

API - EPP Vector

input: AH = 0D
 DL = EPP port number (0-2)

 AL = device address
 CX = number of bytes to read (0 = 64K)
 ES:DI = client buffer

output: AH = error code
 CX = bytes not transferred (=0 if no error)

register usage: AX, BX, CX, DI

Implementation Note: REP INSB with CX=0 does not move 64K bytes, it moves 0 bytes. If the EPP BIOS uses byte mode I/O, it must explicitly handle the 64K case.

A.4.16 Address/Block Write

Address/Block Read combines the *Address Write* and *Write Block* APIs into a single call.

API - EPP Vector

input: AH = 0E
 DL = EPP port number (0-2)

 AL = device address
 CX = number of bytes to write (0 = 64K)
 ES:SI = client buffer

output: AH = error code
 CX = bytes not transferred (=0 if no error)

register usage: AX, BX, CX, SI

Implementation Note: REP OUTSB with CX=0 does not move 64K bytes, it moves 0 bytes. If the EPP BIOS uses byte mode I/O, it must explicitly handle the 64K case.

A.4.17 Lock Port

Lock Port is used to serialize I/O access to the EPP port. If a multi-port driver (mux or daisy chain) is not present this call will always succeed. If a multi-port driver is present this call will fail if the port is currently locked. *Lock port is used to select a multiplexor / daisy chain device port.*

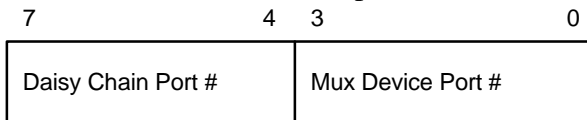
API - EPP Vector

input: AH = 0F
 DL = EPP port (0-2)

BL defined as:

Upper nibble (bits 7-4): Daisy chain port number (1-8)

Lower nibble (bits 3-0): Mux device port number (1-8)



output: AH = error code

register usage: AX, BX

Note: This call *must* be made before any EPP BIOS call is made that accesses the EPP port. Only the following calls may be made while the port is unlocked:

Installation Check	Query Device Port
Query Config	Set Product ID
Device Interrupt	Query Daisy Chain
Real Time Mode	Rescan Daisy Chain
Query Mux	

Implementation Note: This function is a stub for multi-port operation which will be intercepted by the Mux/Daisy chain driver. The EPP BIOS should always return AH=0 (no error).

A.4.18 Unlock Port

Unlock Port will release the EPP port resource for use by other EPP device drivers.

API - EPP Vector

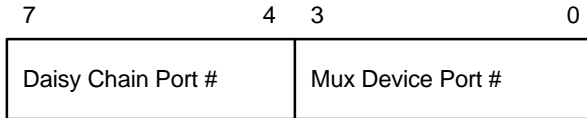
input: AH = 10

DL = EPP port (0-2)

BL defined as:

Upper nibble (bits 7-4): Daisy chain port number (1-8)

Lower nibble (bits 3-0): Mux device port number (1-8)



output: AH = error code

register usage: AX, BX

Note: A device driver must unlock the EPP port after all of its EPP transactions have been performed.

Implementation Note: This function is a stub for multi-port operation which will be intercepted by the Mux/Daisy chain driver. The EPP BIOS should always return AH=0 (no error).

A.4.19 Device Interrupt

Device Interrupt allows an EPP device driver to install an interrupt event handler, to be called whenever an EPP device interrupt occurs. The handler is called with interrupts disabled and should return via an IRET instruction after sending an EOI to the PIC. While operating under device interrupt the handler is not required to explicitly lock the EPP BIOS, the port is automatically locked before the interrupt is dispatched. *This call can only be made while the CPU is operating in real mode.*

API - EPP Vector

input: AH = 11
DL = EPP port (0-2)
AL = 0 - disable device interrupts
= 1 - enable device interrupts
= 2 - remove handler
ES:DI = far pointer to interrupt event handler

BL = EPP Daisy Chain/Mux device port

output: AH = error code

register usage: AX, BX

Note: If an EPP multiplexor is present it is the EPP Mux device driver that dispatches the handler. The EPP Mux device driver will defer calling the handler until the device port causing the interrupt can be selected/locked.

Additional Note: This function is documented to maintain compatibility with earlier revisions of this specification. It may or may not be supported under revision 9. Interrupt handling in this revision is being updated to be compatible with the IEEE P1284.3 SPI defined by IEEE P1284.3. See Function 13 - Check Int Pending.

A.4.20 Real Time Mode

Real Time Mode is used to permit a device driver to advertise whether it is operating a peripheral device that has real time requirements. By using this call, a device driver can also query if any real-time devices are currently running on a device port. This allows drivers to adjust the amount of I/O they perform while the channel is locked. If a real-time device is present then device drivers should use a *small* I/O block size, otherwise they could use a much larger block size, maximizing the channel bandwidth. The advertising device driver should remove the real-time flag if its device no longer needs low latency.

API - EPP Vector

input: AH = 12

 AL = 0 - query if any real time device present
 = 1 - add (advertise) real-time device
 = 2 - remove real-time flag

output: AH = error code

if query, then:

 AL = 0 - if no real-time devices present
 = (non zero) - if one or more real-time devices present

register usage: AX, BX

Implementation Note: This function is a stub for multi-port operation which will be intercepted by the Mux/Daisy chain driver. The EPP BIOS should always return AX=0 (no error and no real-time devices).

A.4.21 *Check INT Pending

Check INT Pending is used to determine if a parallel port interrupt originated at the specified device port. All multi-port devices using interrupts should hook the interrupt chain for the IRQ specified in the Query Config function. As the first step in the interrupt handler, the device driver should call this function to determine if it needs to service the interrupt. If the interrupt came from the specified device port, the port will be locked, and control will return to the handler. If the interrupt did not come from the specified device, the port *will not* be locked, and the handler must pass control down the chain.

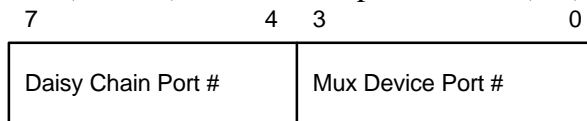
API - EPP Vector

input: AH = 13
 DL = EPP port (0-2)

BL defined as:

Upper nibble (bits 7-4): Daisy chain port number (1-8)

Lower nibble (bits 3-0): Mux device port number (1-8)



output: AH = error code
 AL = 0 - if interrupt found and port locked
 = FF - if no interrupt found

register usage: AX, BX

Implementation Note: This function is a stub for multi-port operation which will be intercepted by the Mux/Daisy chain driver. The EPP BIOS should always return AX=0 (no error and interrupt found).

A.5 MULTI-PORT EXTENSIONS

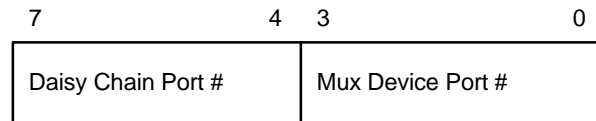
The remaining calls are fielded by either the EPP mux or daisy chain device driver. If neither of these drivers are present the EPP BIOS will return the "EPP Mux not present" or "Command not Supported" error code.

The device ports - for both multiplexor and daisy chain - are numbered one (1) through eight (8) inclusively. They are, however, both encoded in a single byte in the following manner:

Daisy chain port number: upper nibble (bits 7-4)

Multiplexor port number: lower nibble (bits 3-0)

fig 2 bit encoding for Device Port Number



Whenever the Device Port number is referenced in this section the above encoding is meant to apply. The lower nibble is always contains the mux port; the upper nibble always contains the daisy chain port number.

A.5.1 Query Mux

Query Mux is used to return state information about the EPP multiplexor.

API - EPP Vector

input: AH = 40
 DL = EPP port (0-2)

output: AH = error code
 AL = status flags
 bit 0 = channel locked
 bit 1 = interrupt pending
 BL = currently selected port
 BH = Mux Driver revision - M.m (MMMMmmmm)
 *90 for versions complying to this document
 ES:DI = pointer to ASCII string identifying driver vendor

register usage: AX, BX, ES, DI

A.5.2 Query Device Port

Query Port is used to return state information about a particular device port.

API - EPP Vector

input: AH = 41
 DL = EPP port (0-2)
 BL = Device Port

output: AH = error code
 AL = status flags

bit 0 = port selected
 bit 1 = port locked
 bit 2 = interrupts enabled
 bit 3 = interrupt pending
 CX = EPP Product/Device ID
 = 0 if undefined

register usage: AX, BX, CX

A.5.3 Set Product ID

Set Product ID is used to map a EPP Product ID onto a Device Port. This call is useful for EPP peripheral devices that do not support the Address Read/Product ID EPP bus cycles or the daisy chain Query Product ID command packet.

API - EPP Vector

input: AH = 42
 DL = EPP port (0-2)
 BL = Device Port
 CX = EPP Product ID

output: AH = error code

register usage: AX, BX

A.5.4 Query Daisy Chain

Query Daisy Chain is used to return information about the EPP daisy chain.

API - EPP Vector

input: AH = 50
 BL = Multiplexor Device Port (1-8)
 DL = EPP port (0-2)

output: AH = error code
 AL = status flags
 bit 0 = channel locked
 bit 1 = interrupt pending
 BL = currently selected device
 BH = Daisy Chain Manager revision - M.m (MMMMmmmm)
 *90 for versions complying to this document
 CL = depth of daisy chain on this port
 = 0 - if no daisy chain on this port

ES:DI = pointer to ASCII string identifying driver vendor

register: AX, BX, CL, ES, DI

A.5.5 Rescan Daisy Chain

~~Rescan Daisy Chain is used to dynamically reassign port numbers (1-8) to devices connected via the daisy chain. This function is useful for daisy chained devices which do not normally remain connected to the port; for example, a tape drive which is moved between PCs. A device driver for this type of device must issue this command before looking for its associated device. The daisy chain manager will assign the chained devices port numbers from one (1) through eight (8). The daisy chain manager may also automatically rescan the chain in response to an error—see the daisy chain section for more information.~~

~~API EPP Vector~~

~~input: AH = 51~~

~~BL = Multiplexor Device Port (1-8)~~

~~DL = EPP port (0-2)~~

~~output: AH = error code~~

~~register usage: AX, BX~~

A.6 BIOS USAGE EXAMPLE

```
EPP_VECTOR Label DWORD
EPP_OS          DW      ?
EPP_SEG        DW      ?
DEVICE_PORT    DB      0          ; store default (no mux) value
DEVICE_DATA    DB      0          ; used to store data read
                                           ; from EPP device
                :
                :
LPT1           equ     0
EPP_READ_BYTE  equ     9
EPP_LOCK_PORT  equ    0fh
EPP_UNLOCK_PORT equ   10h
EPP_QUERY_CONFIG equ   0
EPP_QUERY_DEV_PORT equ  41h

;MY_PRODUCT_ID equ    XXXX          ; peripheral's Product ID
                :
                :
                :
; Initialization - perform EPP installation check
                mov    ah, 2
                mov    dx, LPT1
                mov    ch, 'E'
                mov    bl, 'P'
                mov    bh, 'P'
                int    17h
; determine if EPP port present
                or     ah, ah
                jnz   NO_EPP
                cmp   al, 'E'
                jne   NO_EPP
                cmp   cl, 'P'
                jne   NO_EPP
                cmp   ch, 'P'
                jne   NO_EPP
; EPP port present, save EPP Vector
                mov    epp_os, bx
                mov    epp_seg, dx
; determine if EPP multiplexor present
                mov    ah, EPP_QUERY_CONFIG
                mov    dl, LPT1
                call   EPP_VECTOR
                or     bl, bl
                jz     NO_MUX
; EPP multiplexor is present
; determine to which device port my peripheral is connected
                mov    bl, 1
                mov    dh, 8
muxCheckLoop:
                push   bx
                push   dx
                mov    ah, EPP_QUERY_DEV_PORT
```

```

        call EPP_VECTOR
        pop dx
        pop bx
        cmp cx, MY_PRODUCT_ID
        je muxPortFound
        inc bl
        dec dh
        jnz muxCheckLoop
        jmp MUX_CHECK_ERROR
; store my peripheral's device port
muxPortFound:
        mov DEVICE_PORT, bl
NO_MUX:
        :
        :
        :
;
; Sample EPP BIOS invocation
; port, do "Lock", "Read Byte", then "Unlock"
        mov ah, EPP_LOCK_PORT
        mov bl, DEVICE_PORT
        mov dl, LPT1
        call EPP_VECTOR
        or ah, ah
        jnz EPP_LOCK_ERROR
        mov ah, EPP_READ_BYTE
        call EPP_VECTOR
        or ah, ah
        jnz EPP_IO_TIMEOUT
        mov DEVICE_DATA, al
        mov ah, EPP_UNLOCK_PORT
        call EPP_VECTOR
        :
        :

```

Note: If a routine is called from a "Device Interrupt" then it is unnecessary to perform the lock function.

A.7 EPP BIOS ERROR CODES

When an EPP BIOS call is made the result code is always returned in the AH register. A value of zero (0) indicates SUCCESS. All non-zero values indicates an error. The EPP BIOS error codes defined below are given in hex.

<u>Error Code</u>	<u>Description</u>
00	Successful
01	I/O timeout
02	Command/feature not supported
03	Unrecognized EPP Port Number
04	EPP BIOS Busy (BIOS is <i>not</i> re-entrant)
05	Parameter Error
10	Multiplexor currently locked (<i>obs.</i>)
20	Multiplexor not present (<i>obs.</i>)
40	Mux/Daisy Chain Manager not Present
41	EPP Port currently locked
42	EPP Port not currently locked
43	Lock Failure / Device Port out of Range