REVISION	REVISION HISTORY	DATE
1.0	Original issue distributed by Intel	9/28/92
2.0	Updated to be in snych with PCI Bus Specification Rev 2.0	7/20/93

Contact the PCI Special Interest Group office to obtain the latest revision of the specification.

Questions regarding the PCI BIOS specification or membership in the PCI Special Interest Group may be forwarded to

PCI Special Interest Group
M/S HF3-15A
5200 N.E. Elam Young Parkway
Hillsboro, Oregon 97124-6497
(503) 696-2000

Table of contents

1. Introduction

1.1. Purpose

This document describes the software interface presented by the PCI BIOS functions. This interface provides a hardware independent method of managing PCI devices in a host computer

1.2. Scope

This document is intended to provide enough information to software developers to utilize PCI devices in a host computer withut any knowledge of how the actual hardware performs the desired functions. It is also to provide enough information for an implementer to create these BIOS functions for a particular system design.

1.3. Related Documents

PCI Local Bus Specification, Revision 2.0 April 30, 1993

Standard BIOS 32-bit Service Directory Proposal, Revision 0.4 May 24, 1993
(available from Phoenix Technologies Ltd., Norwood, MA)

1.4. Terms and Abbreviations

Bus number	A number in the range 0 .. 255 that uniquely selects a PCI bus
Configuration Space	A seperate address space on PCI buses. Used for device identification and configuring devices into Memory and I/O spaces.
Device ID	A predefined field in configuration space that (along with VEndor ID) uniquely identifies the device.
Device Number	A number in the range 0 .. 31 that uniquely selects a device on a PCI bus.
Function Number	A number in the range 0 .. 7 that uniquely selects a function within a multi-function PCI device.
Multi-function PCI device	A PCI device that contains multiple functions. For instance, a single device that provides both LAN and SCSI functions, and has separate configuration space for each function is a mutli-function device.
PCI	Acronym for Peripheral Component Interconnect bus
Special Cycle	A specific PCI bus command used for broadcasting to all PCI devices on a bus.
Vendor ID	A predefined field in configuration space that (along with Device ID) uniquely identifies the device.

2. Functional Description

PCI BIOS functions provide a software interface to the hardware used to implement a PCI based system. Its primary usage is for generating operations in PCI specific address spaces (configuration space, and Special Cycles).

PCI BIOS functions are specified, and should be built, to operate in all modes of the X86 architecture. This includes real-mode, 16:16 protected mode (also known as 286 protect-mode), 16:32 protected mode (introduced with the 386), and 0:32 protected mode (also known as "flat" mode, wherein all segments start at linear address 0 and span the entire 4 Gbyte address space). Note that supporting these modes simultaneously had a significant impact on the structure of the function interface: all eferences to memory buffers have been intentionally left out, and segment registers are not part of any input/output parameter set.

Access to the PCI BIOS functions for 16-bit callers is provided through Interrupt 1Ah, 32-bit (ie. protect mode) access is provided by calling through a 32-bit protect mode entry point. The PCI BIOS function code is B1h. Specific BIOS functions are invoked using a subfunction code. A user simply sets the host processors registers for the function and subfunction desired and calls the PCI BIOS software. Status is returned using the Carry flag ([CF]) and registers specific to the functions invoked.

3. Assumptions and Constraints

3.1. ROM BIOS Location

The PCI BIOS functions are intended to be located within an IBM-PC compatible ROM BIOS. To promote this, as well as make multi-modal operation easy to implement, all references to memory buffers have been intentionally left out, and are not part of any input/output paremeter set.

3.2. Calling Conventions

The PCI BIOS funcions use the X86 CPU´s registers to pass arguments and return status. The caller must use the appropriate subfunction code.

These routines preserve all registers and flags except those used for return parameters. The CARRY FLAG [CF] will be altered as shown to indicate completion status. The calling routine will be returned to with the interrupt flag unmodified and interrupts will not be enabled during function execution. These routines are re-entrant. These routines require 1024 bytes of stack space and the stack segment must have the same size (ie. 16-bit or 32-bit) as the code segment.

The PCI BIOS provides a 16-bit real and protect mode interface and a 32-bit protect mode interface. The 16-bit interface is provided through PC/AT Int 1Ah software interrupt. The PCI BIOS Int 1Ah interface operates in either real mode, virtual-86 mode, or 16:16 protect mode. The BIOS functions may also be accessed through the industry standard entry point for INT 1Ah (physical address 000FFE6Eh) by simulating an INT instruction[1]. The INT 1Ah entry point supports 16-bit code only.

The protected mode interface supports 32-bit protect mode callers. The protected mode PCI BIOS interface is accessed by calling (not a simulated INT) through a protected mode entry point in the PCI BIOS. The entry point and information needed for building the segment descriptors are provided by the BIOS32 Service Directory (see section 3.3). 32-bit callers invoke the PCI BIOS routines using CALL FAR.

The PCI BIOS routines (for both 16-bit and 32-bit callers) must be invoked with appropriate privilege so that interrupts can be enabled/disabled and the routines can access IO space. Implementors of the PCI BIOS must assume that CS is execute-only and DS is read- only.

[1]Note that accessing the BIOS functions through the industry standard entry point will bypass any code that may have ´hooked´the INT 1Ah interrupt vector.

3.3. BIOS32 Service Directory[2]

Detecting the absence or presence of 32-bit BIOS services with 32-bit code can be problematic. Standard BIOS entry points cannot be called in 32-bit mode on all machines because the platform BIOS may not support 32-bit callers. This section describes a mechanism for detecting the presence of 32-bit BIOS services. While the mechanism supports the detection of the PCI BIOS, it is intended to be broader in scope to allow detection of any/all 32-bit BIOS services. The description of this mechanism, known as BIOS32 Service Directory, is provided in three parts; the first part specifies an algorithm for determining if the BIOS32 Service Directory exists on a platform, the second part specifies the calling interface to the BIOS32 Service Directory, and the third part describes how the BIOS32 Service Directory supports PCI BIOS detection.

3.3.1. Determining the existence of BIOS32 Service Directory

A BIOS which implements the BIOS32 Service Directory must embed a specific, contigous 16-byte data structure, beginning on a 16-byte boundary someweer in the physical address range 0E0000h - 0FFFFFh. A description of the fields in the data structure are given in Table 3.1.

Offset	Size	Description
0	4 bytes	Signature string in ASCII. The string is "_32_". This puts an ´underscore´ at offset 0, a ´3´ at offset 1, a ´2´ at offset 2, and another ´underscore´ at offset 3.
4	4 bytes	Entry point for the BIOS32 Service Directory. This is a 32-bit physical address.
8	1 byte	Revision level. This version has revision level 00h
9	1 byte	Length. This field provides the length of this data structure in paragraph (i.e. 16-byte) units. This data structure is 16-bytes long, so this field contains 01h.
0Ah	1 byte	Checksum. This field is a checksum of the complete data structure. The sum of all bytes must add up to 0.
0Bh	5 bytes	Reserved. Must be zero

Table 3.1

Clients of the BIOS 32 Service Directory should determine its existence by scanning 0E0000h to 0FFFF0h looking for the ASCII signature and a valid, checksummed data structure. If the data structure is found, the BIOS32 Service Directory can be accessed through the entry point provided in the data structure. If the data structure is not found, then the BIOS32 Service Directory (and also the PCI BIOS) is not supported by the platform.

[2]This section describes a mechanism for detecting 32-bit BIOS services. This mechanism is being proposed as an indusrty standard and is described by the document Standard BIOS 32-bit Service Directory Proposal, Revision 0.4, May 24, 1993 available from Phoenix Technologies Ltd., Norwood, MA.

3.3.2. Calling Interface for BIOS32 Service Directory

The BIOS32 Service Directory is accessed by doing a CALL FAR to the entry point provided in the Service data structure (see previous section). There are several requirements about the calling environment that must be met. The CS code segment selector and the DS data segment selector must be set up to encompass the physical page holding the entry point as well as the immediately following physical page. They must also have the same base. The SS stack segment selector must be 32-bit and provide at leat 1K of stack space. The calling environment must also allow access to IO space.

The BIOS32 Service Directory provides a single function to determine whether a particular 32-bit BIOS service is supported by the platform. All parameters to the function are passed in registers. Parameter descriptionsare provided below. If a prticular service is implemented in the platform BIOS, three values are returned. The first value is the base physical address of the BIOS service. The second value is the length of the BIOS service. These two values can be used to built the code segment selector and data segment selector for accessing the service. The third value provides the entry point to the BIOS service encoded as an offset from the base.

ENTRY

[EAX]	Service Identifier. This is a four character string used to specifically identify which 32-bit BIOS Service is being sought.
[EBX]	The low order byte ([BL]) is the BIOS32 Service Directory function selector. Currently only one function is defined (with the encoding of zero) which returns the values provided below.

EXIT:

[AL]	Return code. 00h = Service corresponding to Service Identifier is present. 80h = Service corresponding to Service Identifier is not present 81h = Unimplemented function for BIOS Service Directory (i.e. BL has an unrecognized value)
[EBX]	Physical address of the base of the BIOS service
[ECX]	Length of the BIOS service
[EDX]	Entry point into BIOS service. This is an offset from the base provided in EBX

3.3.3. Detecting the PCI BIOS

The BIOS32 Service Directory may be used to detect the presence of the PCI BIOS. The Service Identifier for the PCI BIOS is "$PCI" (049435024h).

4. Host Interface

4.1. Identifying PCI Resources

The following group of functions allow the caller to determine first, if the PCI BIOS support is installed, and second, if specific PCI devices are present in the system.

4.1.1. PCI BIOS Present

This function allows the caller to determine whether the PCI BIOS interface function set is present and what the current interface version level is. It also provides information about what hardware mechanism for accessing configuration space is supported, and whether or not the hardware supports generation if PCI Special Cycles.

ENTRY:

[AH]	PCI_FUNCTION_ID
[AL]	PCI_BIOS_PRESENT

EXIT:

[EDX]	"PCI", "P" in[DL], "C" in [DH], etc. There is a ´space´character in the upper byte.
[AH]	Present Status, 00h = BIOS present IFF EDX set properly
[AL]	Hardware mechanism
[BH]	Interface Level Major Version
[BL]	Interface Level Minor Version
[CL]	Number of last PCI bus in the system
[CF]	Present Status, set = No BIOS present, reset = BIOS Present IFF EDX set properly

If the CARRY FLAG [CF] is cleared and AH is set to 00h, it is still necessary to examine the contents of [EDX] for the presence of the string "PCI" + (trailing space) to fully validate the presence of the PCI function set. [BX] will further indicate the version level, with enough granularity to allow for incremental changes in the code that don´t affect the function interface. Version numbers are stored as binary Codes Decimal (BCD) values. For example, Version 2.10 would be returned as a 02h in the [BH] registers ans 10h in the [BL] registers. BIOS releases of the interface following this version of the specification will be Version 2.00 (BX = 0200h).

The value returned in [AL] identifies what specific HW characteristics the platform supports in relation to accessing configuration space and generating PCI Special Cycles (see Figure 1). The PCI Specification defines two HW mechanisms for accessing configuration space. Bits 0 and 1 of the value returned in [AL] specify which mechanism is upported by this platform. Bit 0 will be set (1) if Mechanism #1 is supported, and reset (0) otherwise. Bit 1 will be set (1) if Mechanism #2 is supported, and reset (0) otherwise. Bits 2,3,6 and 7 are reserved and returned as zeros.

The PCI Specification also defines HW mechanism for generating Special Cycles. Bits 4 and 5 of the value return in [AL] specify which mechanism is supported (if any). Bit 4 will be set (1) if the platform supports Special Cycle generation based on Config Mechanism #1, and reset (0) otherwise. Bit 5 will be set (1) if the platform supports Special Cycle generation based on Config Mechanism #2, and reset (0) otherwise.

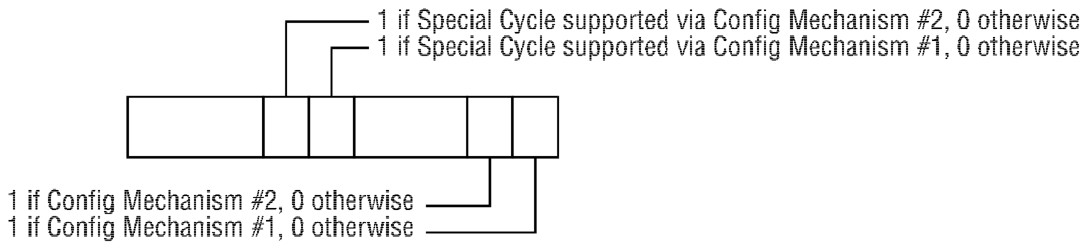
1 if Special Cycle supported via Config Mechanism #2, 0 otherwise
1 if Special Cycle supported via Config Mechanism #1, 0 otherwise
1 if Config Mechanism #2, 0 otherwise
1 if Config Mechanism #1, 0 otherwise

Figure 1. Layout of value returned in [AL]

The value returned in [CL] specifies the number of the last PCI bus in the system. PCI buses are numbered starting at zero and running up to the value specified in CL.

4.1.2. Find PCI Device

This function returns the location of PCI devices that have a specific Device ID ans Vendor ID. Given a Vendor ID, Device ID and an Index (N), the function returns the Bus Number, Device Number, and Function Number of the Nth Device/Function whose Vendor ID and Device ID match the input parameters.

ENTRY:

[AH]	PCI_FUNCTION_ID
[AL]	FIND_PCI_DEVICE
[CX]	Device ID (0...65535)
[DX]	Vendor ID (0...65534)
[SI]	Index (0...N)

EXIT:

[BH]	Bus Number (0...255)
[BL]	Device Number in upper 5 bits Function Number in bottom 3 bits
[AH]	Return Code: SUCCESSFUL DEVICE_NOT_FOUND BAD_VENDOR_ID
[CF]	Completion Status, set = error, cleared = success.

Calling software can find all devices having the same Vendor ID and Device ID by making successive calls to this functions starting with Index set to zero, and incrementing it until the return code is ´DEVICE_NOT_FOUND´. A return code of BAD_VENDOR_ID indicates that the passed in Vendor ID calue (in [DX]) had an illegal value of all 1´s.

Values returned by this function upon successful completion must be the actual values used to access the PCI device if the INT 1Ah routines are by-passed in favor of the direct I/O mechanisms described in the PCI Specification.

4.1.3. Find PCI Class Code

This function returns the location of PCI devices that have a specific Class Code. Given a Class Code and a Index (N), the function returns the Bus number, Device Number, and Function Number of the Nth Device/Function whose Class Code matches the input parameters.

ENTRY:

[AH]	PCI_FUNCTION_ID
[AL]	FIND_PCI_CLASS_CODE
[ECX]	Class Code (in lower three bytes)
[SI]	Index (0...N)

EXIT:

[BH]	Bus Number (0...255)
[BL]	Device Number in upper 5 bits Function Number in bottom 3 bits
[AH]	Return Code: SUCCESSFUL DEVICE_NOT_FOUND
[CF]	Completion Status, set = error, cleared = success.

Calling software can find all devices having the same Class Code by making succesive calls to this function starting with Index set to zero, and incrementing it until the return code is ´DEVICE_NOT_FOUND´.

4.2. PCI Bus Specific Operations

The following function set allows generation of bus specific operations on a specific PCI bus. Currently there is only one function defined.

4.2.1. Generate Special Cycle

This function allows for generation of PCI special cycles. The generated special cycle will be broadcast on a specific PCI bus in the system.

ENTRY:

[AH]	PCI_FUNCTION_ID
[AL]	GENERATE_SPECIAL_CYCLE
[BH]	Bus Number (0...255)
[EDX]	Special Cycle Data

EXIT:

[AH]	Return Code: SUCCESSFUL FUNC_NOT_SUPPORTED
[CF]	Completion Status, set = error, cleared = success.

4.3. Accessing Configuration Space

4.3.1. Read Configuration Byte

This function allows reading individual bytes from the configuration space of a specific device.

ENTRY:

[AH]	PCI_FUNCTION_ID
[AL]	READ_CONFIG_BYTE
[BH]	Bus Number (0...255)
[BL]	Device Number in upper 5 bits
	Function Number in lower 3 bits
[DI]	Register Number (0...255)

EXIT:

[CL]	Byte Read
[AH]	Return Code:
	SUCCESSFUL
	BAD_REGISTER_NUMBER
[CF]	Completion Status, set = error, cleared = success.

4.3.2. Read Configuration Word

This function allows reading individual words from the configuration space of a specific device. The Register Number parameter must be a multiple of two (i.e., bit 0 must be set to 0).

ENTRY:

[AH]	PCI_FUNCTION_ID
[AL]	READ_CONFIG_WORD
[BH]	Bus Number (0...255)
[BL]	Device Number in upper 5 bits Function Number in lower 3 bits
[DI]	Register Number (0,2,4,...254)

EXIT:

[CX]	Word Read
[AH]	Return Code: SUCCESSFUL BAD_REGISTER_NUMBER
[CF]	Completion Status, set = error, cleared = success.

4.3.3. Read Configuration Dword

This function allows reading individual bytes from the configuration space of a specific device. The Register Number parameter must be a multiple of four (i.e., bit 0 and 1 must be set to 0).

ENTRY:

[AH]	PCI_FUNCTION_ID
[AL]	READ_CONFIG_DWORD
[BH]	Bus Number (0...255)
[BL]	Device Number in upper 5 bits Function Number in lower 3 bits
[DI]	Register Number (0,4,8,...252)

EXIT:

[ECX]	Dword Read
[AH]	Return Code: SUCCESSFUL BAD_REGISTER_NUMBER
[CF]	Completion Status, set = error, cleared = success.

4.3.4. Write Configuration Byte

This function allows writing individual bytes from the configuration space of a specific device.

ENTRY:

[AH]	PCI_FUNCTION_ID
[AL]	WRITE_CONFIG_BYTE
[BH]	Bus Number (0...255)
[BL]	Device Number in upper 5 bits Function Number in lower 3 bits
[DI]	Register Number (0,4,8,...252)
[CL]	Byte Value to Write

EXIT:

[AH]	Return Code: SUCCESSFUL BAD_REGISTER_NUMBER
[CF]	Completion Status, set = error, cleared = success.

4.3.5. Write Configuration Word

This function allows writing individual words from the configuration space of a specific device. The Register Number parameter must be a multiple of two (i.e., bit 0 must be set to 0).

ENTRY:

[AH]	PCI_FUNCTION_ID
[AL]	WRITE_CONFIG_WORD
[BH]	Bus Number (0...255)
[BL]	Device Number in upper 5 bits Function Number in lower 3 bits
[DI]	Register Number (0,2,4,...254)
[CX]	Word value to write

EXIT:

[AH]	Return Code: SUCCESSFUL BAD_REGISTER_NUMBER
[CF]	Completion Status, set = error, cleared = success.

4.3.6. Write Configuration Dword

This function allows writing individual dwords from the configuration space of a specific device. The Register Number parameter must be a multiple of four (i.e., bit 0 and 1 must be set to 0).

ENTRY:

[AH]	PCI_FUNCTION_ID
[AL]	WRITE_CONFIG_DWORD
[BH]	Bus Number (0...255)
[BL]	Device Number in upper 5 bits Function Number in lower 3 bits
[DI]	Register Number (0,4,8,...252)
[ECX]	Dword value to write

EXIT:

[AH]	Return Code: SUCCESSFUL BAD_REGISTER_NUMBER
[CF]	Completion Status, set = error, cleared = success.

APPENDIX A: Function List

FUNCTION	AH	AL
PCI_FUNCTION_ID	B1h	
PCI_BIOS_PRESENT		01h
FIND_PCI_DEVICE		02h
FIND_PCI_CLASS_CODE		03h
GENERATE_SPECIAL_CYCLE		06h
READ_CONFIG_BYTE		08h
READ_CONFIG_WORD		09h
READ_CONFIG_DWORD		0Ah
WRITE_CONFIG_BYTE		0Bh
WRITE_CONFIG_WORD		0Ch
WRITE_CONFIG_DWORD		0Dh

APPENDIX A: Return Code List

RETURN CODES	AH
SUCCESFUL	00h
FUNC_NOT_SUPPORTED	81h
BAD_VENDOR_ID	83h
DEVICE_NOT_FOUND	86h
BAD_REGISTER_NUMBER	87h