PCI IDE Controller

Specification

Revision 1.0

3/4/94

1.0. Introduction

This document defines the necessary characteristics of a PCI-based IDE controller so that device independent software (i.e.; BIOSes) can identify and properly configure the device. The document should be viewed as a set of guidelines or rules that PCI IDE controllers should follow when they are used on PCI add-in cards. Devices that follow these guidelines are assured that they will fit cleanly in the 'Plug and Play' model of PCI, will be recognized by BIOSes on PCI machines, and will be properly configured in the system.

IDE controllers are one of a small set of 'compatibility' devices, that have well known programming interfaces at fixed addresses and typically using fixed IRQs. Most OS's contain code that deal directly with compatibility devices (IDE included) and this becomes a strong force to maintain the 'compatibility' programming models. Newer OS's allow devices to maintain the same register level programming interface (i.e.; the same meanings for register), while addressing the registers at non-standard locations and using non-standard IRQs. This helps avoid resource conflicts and allows multiple controllers of the same type to be in a system.

The guidelines in the document allow an IDE controller to be:

1. 'compatibility' only,
2. 'compatibility' or native-PCI (i.e.; fully-relocatable),
3. or native-PCI only.

The recommended implementation for add-in IDE controllers is number 2 ('compatibility' or native-PCI), or number 3 (native-PCI only).

2.0. PCI IDE Controller

The PCI IDE controller is capable of supporting upto two IDE channels (primary and secondary) with two devices per channel for a total of four IDE devices. The two channels in the IDE controller are independent and do not effect each others operation. The control registers for each channel follow the *AT Attachment (ATA) Interface Standard* with one exception defined below. The PCI IDE controller (supporting two channels) is a single PCI function. A PCI device supporting four IDE channels would be a multifunction device (actually two functions) where each function is a PCI IDE controller supporting two channels. PCI IDE controllers (like all PCI devices) must implement and respond to PCI Configuration Space.

The ATA Standard defines two sets of registers known as Control Block Registers and Command Block Registers. The Command Block Registers consist of eight IO ports providing various command/status functions for the IDE device. The Control Block registers consist of two bytes used for control/status of the IDE device. The second byte of this pair is read-only and has the interesting quirk where the top bit of this byte is shared with the floppy controller when the IDE device is mapped at 'compatibility' locations. It turns out that software controlling IDE devices (BIOS, drivers, etc.) does not use this register at all.

The exception for PCI IDE controllers to the ATA Standard is that only the first of the two bytes defined in the Control Block registers is implemented. This byte provides *Alternate Status* on reads and *Device Control* on writes. Accesses to the second byte of the Control Block registers (*Drive Address*) should be ignored by the PCI IDE controller.

The ATA Standard defines an interrupt for the IDE channel. Since the PCI IDE controller supports two IDE channels it has to deal with two interrupts. When the PCI IDE controller is operating in 'compatibility' mode it uses the standard interrupts defined for compatibility. When the controller is operating in native-PCI mode the interrupts from the channels are collapsed and shared on a single PCI interrupt line.

Global control (i.e.; enable/disable) of the PCI IDE controller is done by manipulating the 'IO enable' bit in the controller's Configuration Space Command register. Controllers must implement this bit as read-write and the bit must be set to zero (disabled) on reset. Note that this bit effects both IDE channels.

2.1. PCI IDE Controller in 'Compatibility' mode

This section defines the characteristics of a PCI IDE controller when it is operating in the 'compatibility' mode. The main characteristics are that the controller registers are hardwired to fixed IO locations and

fixed IRQs are used. Hardwired assignments are shown in the Table 1 below. When a channel is operating in 'compatibility' mode it must decode the addresses shown in the Table 1 and use the specified IRQ. When the device is disabled (using the IO Enable bit in the Command register), the device must not respond to any IO addresses, and must tristate it's IRQ connections.

Channel	Command Block Registers	Control Block Register	IRQ
Primary	1F0h - 1F7h	3F6h	14
Secondary	170h - 177h	376h	15

Table 1. Compatibility resource mappings

When a channel is operating in 'compatibility' mode and it's Control Block register is addressed with more than one byte enable asserted (i.e.; a WORD access to 3F6h or 176h) the PCI IDE controller may either return both bytes of data (with the high-order byte returning device specific data) or terminate the access with Target-Abort.

2.2. PCI IDE Controller in Native-PCI mode

This section defines the characteristics of a PCI IDE controller when it is operating in native-PCI mode. In this mode the registers of the IDE channels are completely relocatable in IO space. Base Address Registers in the PCI IDE controller's Configuration Space registers are used to map the IDE registers into IO space. Specific base address registers are used to map the different register blocks as defined in Table 2 (below). Base address registers are identified by providing their offset in configuration space.

Channel	Command Block Registers	Control Block Register
Primary	BA at offset 0x10	BA at offset 0x14
Secondary	BA at offset 0x18	BA at offset 0x1C

Table 2. Base Address Registers for Register Mapping

Base registers used to map Command Block registers must ask for 8 bytes of IO space. Base registers used to map Control Block registers must ask for 4 bytes of IO space.[1] In this four byte allocation the byte at offset 02h is where the *Alternate Status/Device Control* byte is located. Other bytes in the four byte allocation (bytes at offsets 0,1 and 3) are undefined and may be used for device specific purposes. Device independent software should only access the byte at offset 02h; Accessing other bytes may cause errors.

Interrupt signals from the IDE channels must be connected to the appropriate PCI interrupt pin and converted to the appropriate polarity. The Interrupt Pin and Interrupt Line registers must be implemented in the controller's configuration space.

2.3. PCI IDE Controller Identification and Control

The Class Code field in the controllers configuration space is used by software to determine and control the mode that PCI IDE controller is operating in. In the three byte Class Code field, the upper byte (Base Class) has the value 01h, the middle byte (Sub-Class) has the value 01h, and the low byte (Programming Interface) can have several values depending on the functionality of the controller (see Figure 1.).

In the Programming Interface byte there are two bits allocated for each IDE channel that determine and control what mode the channel is operating in. Table 3 provides the definitions for those bits.

Bit	Description

[1]This is the smallest amount of IO space that a Base Register can request.

0	Determines the mode that the primary IDE channel is operating in. Zero corresponds to 'compatibility', one means native-PCI mode. This bit is implemented as read-only if the channel supports only one mode, or read-write if both modes are supported. The powerup state for this bit (when writable) can be either 0 or 1.
1	This bit indicates whether or not the primary channel has a fixed mode of operation. If this bit is zero, the mode is fixed and is determined by the (read-only) value of bit 0. If this bit is one, the channel supports both modes and may be set to either mode by writing bit 0.
2	Determines the mode that the secondary IDE channel is operating in. Zero corresponds to 'compatibility', one means native-PCI mode. This bit is implemented as read-only if the channel supports only one mode, or read-write if both modes are supported. The powerup state for this bit (when writable) can be either 0 or 1.
3	This bit indicates whether or not the secondary channel has a fixed mode of operation. If this bit is zero, the mode is fixed and is determined by the (read-only) value of bit 0. If this bit is one, the channel supports both modes and may be set to either mode by writing bit 0.

Table 3. Bit definitions in Programming Interface byte

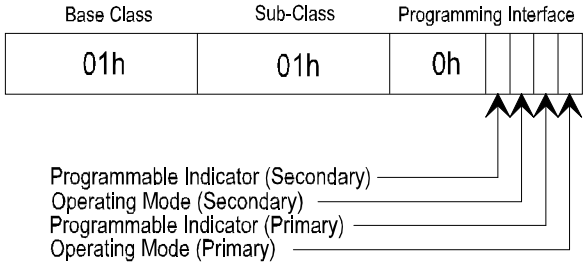

Figure 1. PCI IDE Controller Class Code

2.4. Both Modes Controller

PCI IDE controllers that support both modes ('compatibility' and native-PCI) and software configuring these devices must be aware of the following points.

- When a channel is in 'compatibility' mode, the controller does hard decodes of the compatibility addresses. Any values in the associated Base Address registers are ignored. Conversely, when a channel is in native-PCI mode decodes are done using the values in the associated Base Address registers and no 'compatibility' addresses should be used.
- When a channel is in 'compatibility' mode, the controller can either disable the first four Base Address registers (i.e.; make them not writable and return 0's when read) or leave them fully programmable. In either case the values in these registers are ignored as long as the channel is in 'compatibility' mode.
- When a channel is in compatibility mode the IRQ used by the channel must be the 'compatibility' IRQ. PCI interrupt lines must not be effected by that channel's interrupt. Conversely, when the channel is in native-PCI mode the channel's interrupt should be connected to the appropriate INTx#. Compatibility IRQs should not be effected (i.e.; they should be tristated).
- Connections of channel interrupt signals to the 'compatibility' IRQs should be disabled (i.e.; tristated) until the PCI IDE controller is enabled via the Command register in Configuration Space. The controller is enabled when a '1' is written to the IO enable bit (bit 0) in the Command register.

3.0. BIOS Implications

BIOSes are expected to recognize and properly configure PCI IDE controllers. This means examining the Class Code fields for all PCI IDE controllers to determine their capabilities (i.e.; relocatability), choosing which controllers/channels will be 'compatibility' primary and secondary, and configuring or disabling all others. The BIOS must take into account non-PCI IDE controllers when doing the configuration.

BIOSes should also consider adding the capability of booting from native-PCI IDE controllers.

Guidelines for the BIOS interpretation of the Programming Interface byte of the PCI IDE controller class code are given in Table 4. This table shows the interesting combinations of the lower four bits of the Programming Interface byte.

Lower 4 bits of Programming Interface byte	Description
0000	Indicates a 'compatibility'-only device. BIOS must assume that both channels are implemented (even if device actually supports only one). Device must be used for both 'compatibility' channels or not used at all (disabled).
0001	Primary channel is native-PCI only. Secondary channel is 'compatibility'-only. If controller is not going to be used for secondary 'compatibility' channel then both channels cannot be used.
001x	Primary channel can operate in either mode. Secondary channel is 'compatibility'-only. If controller is not going to be used for secondary 'compatibility' channel then both channels cannot be used.
0100	Secondary channel is native-PCI only. Primary channel is 'compatibility'-only. If controller is not going to be used for primary 'compatibility' channel then both channels cannot be used.
0101	Both primary and secondary channels operate in native-PCI mode only.
011x	Primary channel can operate in either mode. Secondary channel is native-PCI only.
1x00	Secondary channel can operate in either mode. Primary channel is 'compatibility'-only. If controller is not going to be used for primary 'compatibility' channel then both channels cannot be used.
1x01	Secondary channel can operate in either mode. Primary channel is native-PCI only.
1x1x	This is the most general case. Both channels support both modes.

Table 4. BIOS interpretation of Programming Interface byte.

3.1. Device Operation with Older BIOSes

PCI IDE controllers that follow this specification and are added into a system that has an older BIOS that is not aware of this specification will behave as follows:

If the device defaults to 'compatibility' mode, it will hard decode the compatibility addresses and use the compatibility IRQs. If the system already contains an IDE controller some unusual behavior will occur. The end-user will have to deal with the problem.

If the device defaults to native-PCI mode, it will be configured like any other PCI device and will operate (with the appropriate drivers) without end-user involvement.

4.0. Compatibility Interrupt Connection

This document does not define how a PCI IDE controller on an add-in card gets connected to the 'compatibility' IRQs (14 and 15) needed to operate in 'compatibility' mode. It is unlikely that these IRQs will ever be made available on the standard PCI connector. However, it is the responsibility of the add-in card to provide these connections in some manner.

5.0. Operation with PCI-to-PCI Bridges

PCI IDE controllers operating in 'compatibility' mode are not supported behind a PCI to PCI bridge. PCI to PCI bridges (and secondary Host Bus Bridges) do not support the forwarding of IDE 'compatibility' addresses to their secondary bus. To operate in 'compatibility' mode the controller must reside on the primary PCI bus of the system.

PCI IDE controllers operating in native-PCI mode can be used on any PCI bus in the system.