Developer Note

# 12″ and 7″ PC Compatibility Cards

**Developer Note**

4/17/96
Developer Press

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010



Simultaneously published in the United States and Canada.

**LIMITED WARRANTY ON MEDIA AND REPLACEMENT**

**If you discover physical defects in the manual or in the media on which a software product is distributed, APDA will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to APDA.**

**ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.**

**Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.**

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.**

**Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.**

# Contents

# Figures and Tables

# About This Note

The 12″ and 7″ PC Compatibility Cards are x86-based microprocessor cards that you can plug into any Macintosh computer that has a PCI slot. This developer note describes the features and capabilities of the cards and is intended for use by software and hardware developers.

The 12″ and 7″ PC Compatibility Cards are referred to in this developer note as the 12″ card and 7″ card, respectively, as the cards, or as the PCI cards.

To use this note, you must know how to use and program both Macintosh and PC-compatible computers. You must also be familiar with the PCI (Peripheral Component Interconnect) bus.

The user's guide for your computer provides information about setting up the computer and installing the cards in a specific computer configuration. This note does not provide that type of information.

This preface describes the contents of the note, explains visual cues and conventions, and lists other books to which you can refer.

## What This Note Contains

This note consists of four chapters and an index.

- Chapter 1, "Introduction," summarizes the hardware and software features of the cards and describes their functional and physical characteristics.
- Chapter 2, "Hardware Design," provides more detailed information about the hardware design of each card, including the microprocessor, memory, video, audio, and I/O systems.
- Chapter 3, "I/O Specifications," describes the I/O connectors that enable the cards to communicate with the host Macintosh computer, the monitor, sound devices, and video devices. It provides specifications for the connectors, including pin assignments and signal descriptions.
- Chapter 4, "Software Support," describes the interface driver that controls communication between the Macintosh host computer and the 12" and 7" cards. It provides information about initializing the driver, configuring the cards, and passing messages between the Mac OS and the cards.

# Conventions and Abbreviations

This developer note uses the following typographical conventions and abbreviations.

## Typographical Conventions

Computer-language text—any text that is literally the same as it appears in computer input or output—appears in `Courier` font.

**Note**
A note like this contains information that is interesting but not essential for an understanding of the text. ◆

**IMPORTANT**
A note like this contains important information that you should read before proceeding. ▲

▲ **WARNING**
A note like this directs your attention to something that could cause damage or result in a loss of data. ▲

## Standard Abbreviations

When unusual abbreviations appear in this developer note, the corresponding terms are also spelled out. Standard units of measure and other widely used abbreviations are not spelled out. The following abbreviations are used in this note:

| | |
|---|---|
| ADP | audio digital processor |
| API | application-programming interface |
| ASIC | application specific IC |
| BIOS | basic input/output system |
| CLUT | color lookup table |
| CPU | central processing unit |
| DAC | digital-to-analog converter |
| DAP | digital audio processor |
| DIMM | dual inline memory module |
| DOS | Disk Operating System |
| DRAM | dynamic RAM |
| EPROM | electrically programmable ROM |

| | |
|---|---|
| FM | frequency modulation |
| GIMO | Graphics Internal Monitor Out |
| Hz | hertz |
| IC | integrated circuit |
| I/O | input/output |
| IRQ | interrupt request |
| ISA | Industry Standard Architecture (in PC environment) |
| K | 1024 |
| KB | kilobyte |
| kHz | kilohertz |
| L1 | level 1 (cache)—internal to microprocessor |
| L2 | level 2 (cache)—external |
| MB | megabyte |
| MFM | modified frequency modulation |
| MHz | megahertz |
| MIDI | Musical Instrument Digital Interface |
| ns | nanosecond |
| PC | Personal Computer (usually refers to an x86-based computer) |
| PCI | Peripheral Component Interconnect |
| PLL | phased lock loop |
| RAM | random-access memory |
| RGB | red, green, blue |
| RISC | reduced instruction set computing |
| ROM | read-only memory |
| SVGA | super video graphics adapter |
| TTL | transistor-transistor logic |
| UART | universal asynchronous receiver/transmitter |
| V | volt |
| VESA | Video Electronics Standards Association |
| VGA | video graphics adapter |
| VRAM | video RAM |
| W | watt |
| XD | X data bus; standard connector in PC environment—interfaces with the ISA bus |

## Other Reference Material

This developer note assumes that you are familiar with Apple Macintosh computers and PC-compatible computers and know how to operate and program them. Additional information is available in the following publications:

- The developer notes provided for developers working with Macintosh computers.
- The user's guides shipped with Macintosh computers. These publications explain how to set up the computer and install the 12"or 7" card. The user's guide also provides basic troubleshooting information.
- *Designing PCI Cards and Drivers for Power Macintosh Computers*, part number R0650ll/A, available from Apple Computer, Inc.
- *Inside Macintosh, Volume VI*, published by Addison-Wesley Publishing Company, Inc.
- *Inside Macintosh: Devices*, published by Addison-Wesley Publishing Company, Inc.
- *PCI Local Bus Specification*, available from the PCI Special Interest Group, Intel Corporation.

## For More Information

The *Apple Developer Catalog* (ADC) is Apple Computer's worldwide source for hundreds of development tools, technical resources, training products, and information for anyone interested in developing applications on Apple computer platforms. Customers receive the *Apple Developer Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. ADC offers convenient payment and shipping options, including site licensing.

To order products or to request a complimentary copy of the *Apple Developer Catalog*, contact

Apple Developer Catalog
Apple Computer, Inc.
P.O. Box 319
Buffalo, NY 14207-0319

| | |
|---|---|
| Telephone | 1-800-282-2732 (United States)<br>1-800-637-0029 (Canada)<br>716-871-6555 (International) |
| Fax | 716-871-6511 |
| AppleLink | ORDER.ADC |
| Internet | order.adc@applelink.apple.com |

CHAPTER 1

# Introduction

The 12″ and 7″ PC Compatibility Cards are x86-based cards. You can plug the 7" card into any Macintosh computer that has a PCI (Peripheral Component Interconnect) slot. You can plug the 12" card into any Macintosh computer that supports a full-size PCI slot. The cards provide the computer with full PC compatibility and the ability to run DOS, Windows 3.1, and Windows 95. The cards perform identical functions; however, the smaller 7" card can be plugged into CPUs with limited enclosure space. In addition, the two cards have different microprocessors and PC/PCI chip sets.

**Note**

The 12" and 7" cards are referred to generically in this developer note as cards, PCI cards, or compatibility cards. The functions performed by the cards are referred to as PC system functions, as opposed to the Macintosh system functions performed by the Macintosh host computer. ◆

▲ **WARNING**

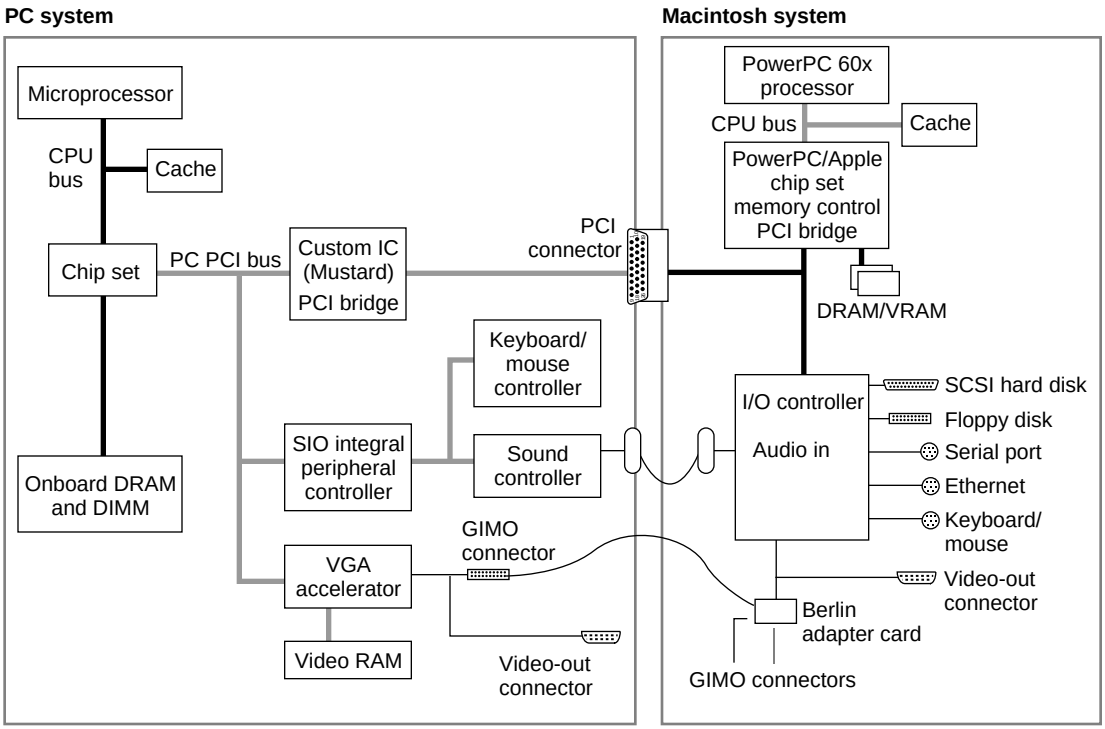
12" and 7" cards are designed to plug into PCI slots in Macintosh computers. Do not plug them into PCI slots in DOS- or Windows-based PCs. In addition, you should use only one card (12" or 7") in your Macintosh computer. It is possible to insert more than one of these cards into a Macintosh with multiple PCI slots. However, if you do this, it will compromise the functionality of the cards and cause problems with resource allocation and I/O connections. ▲

The cards differ from previous Apple cards in this category in the following ways:

- They communicate with the Macintosh computer by means of the PCI bus rather than the Apple RISC (reduced instruction set computing) bus.
- They have enhanced microprocessors.
- They have enhanced video capabilities.
- They do not share user memory; that is, there is no portion of Macintosh memory set aside for PC use. However, there is some memory sharing at a system level, where the Macintosh computer and the PC can both access certain segments of Macintosh memory. The Macintosh and PC use this facility to communicate with each other.
- They have external L2 caches as well as the L1 caches integral to the microprocessors.

This chapter summarizes the features of each card and explains the functional and physical characteristics. Figure 1-1 shows a simplified block diagram of the 12″ card and indicates how it interfaces with the Macintosh computer. The interface for the 7″ card is similar.

**Figure 1-1** Simplified block diagram of PC and Macintosh functions




# Physical Size

The 12" card is a full-sized PCI card, measuring 12.283 inches by 4.2 inches. Figure 1-2 shows an outline of the card with dimensions. The 7" card is a PCI card with smaller dimensions, measuring 6.875 inches by 4.2 inches. Figure 1-3 shows an outline of the 7" card with dimensions.

**IMPORTANT**

The measurements shown in Figure 1-2 and Figure 1-3 are intended to show the relative dimensions of the cards. They are not intended to be taken as manufacturing specifications. ▲

**Figure 1-2** 12" card dimensions

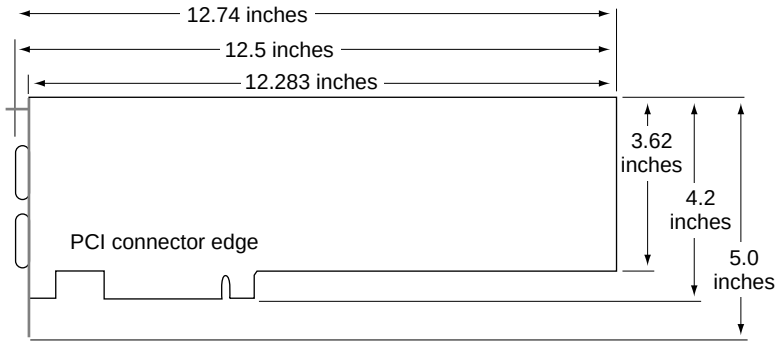
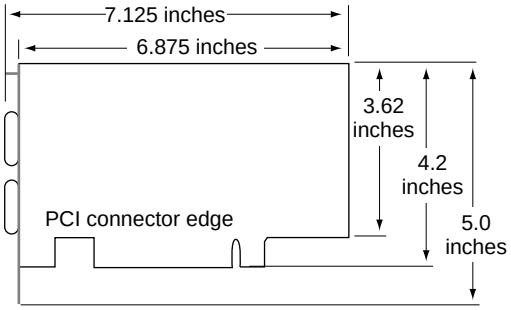



**Figure 1-3** 7" card dimensions




# Overview of Functional Capabilities

This section gives an overview of the functional capabilities of the 12" and 7" cards. The section includes information about the microprocessor; onboard DRAM and memory expansion; support for video and audio devices; support for Apple devices, such as floppy disk, hard disk, serial ports, parallel printer port, keyboard, and mouse; support for a PC-compatible joystick and MIDI (Musical Instrument Digital Interface) devices; and support for the ISA (Industry Standard Architecture) bus. Finally it provides information about the BIOS (basic input/output system). Chapter 2, “Hardware Design,” provides more detailed technical information about the components that provide this support. Chapter 3, “I/O Specifications,” describes the I/O connectors.

## Processor Capabilities

The 12" card has a Pentium-class microprocessor that operates at a clock speed of 100 MHz and supports a 64-bit data bus. The 7" card has a 5x86-class microprocessor that operates at a clock speed of 100 MHz and supports a 32-bit data bus. Both microprocessors have internal L1 caches.

Since the microprocessors are different, the PC/PCI chip sets are also different. The 12" card uses an Opti chip set, made up of the 82C556, 82C557, and 82C558 ICs. The 7" card uses the Sis 5x86-SIS chip set composed of the 85C496 and the 85C497 ICs. The BIOS is different for each card because of the different microprocessors and chip sets.

The main reason for using different microprocessors for the two cards is onboard space. The Pentium-class microprocessor with its chip set requires considerably more space than the 5x86-class microprocessor and its chip set. The Pentium-class microprocessor is therefore not used in the smaller card.

## Memory Capabilities

The 12" card has 8 MB of built-in DRAM. Memory capacity may be extended up to 64 MB using a 168-pin DIMM (dual inline memory module). The DIMM is plugged into the DIMM slot on the card and expands total memory capacity up to 72 MB in two or three 64-bit memory banks. The card also has a 256 KB cache memory.

The 7" card has 8 MB of DRAM, provided by a 168-pin DIMM, which plugs into the DIMM slot on the card. To expand memory capacity for the 7" card, you can replace the 8 MB DIMM with DIMMs of higher capacity, up to a total of 64 MB. Memory on the 7" card is organized in 32-bit memory banks, up to a total of four banks. The card also has a 128 KB cache memory.

## Video Support

An ATI 264CT VGA (video graphics adapter) controller accelerator and onboard video DRAM provide video support for fixed-frequency 12-inch to 21-inch monitors. Table 2-4 on page 23 provides a detailed list of these monitors.

The way in which the monitors are connected to the system varies, depending on the Macintosh computer in which the 12" and 7" cards are operating. “GIMO Support for Video Output” beginning on page 28 and “Loop-Back Video Support” beginning on page 30 provide detailed information on this subject.

The cards have a DB26 connector. You can use the connector and a cable to connect a second monitor that can be dedicated to the PC side of the system, while the Macintosh computer’s DB15 connector supports the Macintosh display.

The DB15 connector on the cards does not support video but allows you to connect a PC-compatible joystick or MIDI device, as described in “Joystick and MIDI Devices” on page 8.

▲ **WARNING**

Do not connect a monitor or other video device to the DB15 connector on the 12″ and 7″ cards. ▲

## Audio Support

The 12" and 7" cards provide enhanced sound capabilities that are compatible with 16-bit stereo Sound Blaster Pro capabilities. The audio circuitry, which consists of a Creative Technologies Vibra 16S 16-bit audio chip and the Yamaha OPL3 FM synthesizer chip set, enables the cards to provide audio output compatible with the Sound Blaster Pro and to mix PC sound output with the sound output of the Macintosh host computer.

**Note**

Sound Blaster Pro products developed by Creative Technologies support sound systems in PC-compatible computers. Their hardware and software specifications have become de facto standards in the industry. ◆

Each card has two audio connectors, one for input and one for output. Audio inputs from such sources as CD players are transferred to the cards via the Macintosh main circuit board. Sound outputs from the cards are transferred to the sound system on the Macintosh's main circuit board, to external speakers, or to an external audio jack. You can use these connectors with most Macintosh computers in which the 12" and 7" cards may be installed.

Some Macintosh computers do not support this type of audio connection. In this case, audio inputs and outputs may be transferred between the CPU and the cards by means of the GIMO connector and Berlin adapter card described in "GIMO Support for Video Output" beginning on page 28. Currently it is computers with smaller enclosures (for which the 7" card is designed) that use the GIMO connector to transfer audio inputs and outputs.

PC beeps generated by the PC subsystem generate a line level audio signal that is summed into the Macintosh audio subsystem or the CD-ROM audio path. You can also disable PC audio if you prefer.

## I/O Support

A PC/PCI chip set provides an integrated solution for the AT-compatible I/O. Conventional ISA expansion is not allowed; refer to "XD Connector" beginning on page 52 for further information on this subject.

The 12" card has the Opti chip set composed of the 82C556, 82C557, and 82C558 ICs. The 7" card has the Sis 5x86-SIS chip set composed of the 85C496 and 85C497 ICs. These chip sets, in conjunction with a custom IC, referred to in this document as the Mustard ASIC, provide the I/O support required by the cards for Macintosh floppy disk and hard disk drives, the Macintosh serial ports, and the Macintosh keyboard and mouse.

## Floppy Disk Drive

The Mac OS provides support that allows the cards to access the Macintosh computer's 3.5-inch internal floppy disk drive. The drive can read and write floppy disks that are DOS MFM (modified frequency modulation) formatted. If you insert a disk that is not DOS formatted when you are using the DOS side of the system, the disk will be promptly ejected from the drive.

**Note**
Modified frequency modulation (MFM) is an encoding system used to record data on magnetic surfaces such as floppy diskettes. It is used in the PC environment. ◆

## Hard Disk

The Mac OS also provides support that allows the cards to access the Macintosh computer's internal hard drive and other SCSI port(s).

## Serial Ports

The 12" and 7" cards can access the serial ports on the Macintosh host computer, and you can connect a PC serial device to the Macintosh serial port. To allow this, the Mustard ASIC provides hardware support that emulates the registers of standard serial port ICs found in most PC/AT computers. "Serial Port Support" on page 26 provides further information on this subject.

## Parallel Printer Port

The Mustard ASIC emulates a compatible parallel port interface and enables the driver software to send data to a printer through the Macintosh computer. The printer may be connected to the Macintosh, or it may be part of a network and be selected by means of the Chooser.

## Keyboard and Mouse

Both cards contain an 8242 keyboard/mouse controller. The PC's keyboard and mouse determine which interface to this controller is emulated in the Mustard ASIC, allowing the cards to access the keyboard and mouse by means of the ADB (Apple Desktop Bus).

The cards can work with other user input devices, such as a trackball. However, such devices must be connected to the Macintosh host computer by means of the ADB port.

You can define a key combination that allows you to switch the operation of the user interface devices (such as a keyboard, mouse, or a shared monitor) between the 12" or 7" card and the Macintosh host computer. You will find information on setting the key combination in the user's guide supplied with the computer.

## Joystick and MIDI Devices

You can connect a PC-compatible joystick to the DB15 (15-pin) connector on the edge of the cards. This connector is accessible on the rear panel of the Macintosh host computer. However, you need a standard joystick/media adapter to make the connection. The joystick can only be used with programs running on the PC side of the system. You can also connect MIDI devices through the same connector.

## ISA Access

Each card has an XD connector that provides limited access to the unbuffered ISA bus. This enables you to create a parallel port for third-party hardware keys. You do this by connecting an expansion card to the XD connector and then connecting a hardware key to the expansion card.

**IMPORTANT**

External cards connected to the ISA bus must have only one load if they are unbuffered. If they have more than one load, you must provide buffering for them. ▲

## System BIOS

The cards use a 128K system BIOS. The BIOS is stored in Macintosh memory and downloaded to the DRAM on the cards when the computer is booting. The BIOS is different for each card, because they have different microprocessors and chip sets.

CHAPTER 2

# Hardware Design

As described in Chapter 1, there are two versions of the PC Compatibiity Card: the 12″ version and the 7″ version. They perform the same functions; however, there are the following differences between the cards:

- The smaller 7″ card is used in Macintosh systems where space is a critical factor.
- The two cards have different microprocessors and chip sets.
- The two cards implement onboard DRAM in different ways.

This chapter

- provides a hardware overview of the cards
- describes the functional blocks of circuitry, including the microprocessor, memory system, video system, and I/O system
- provides specifications for the monitors supported

Currently, both cards may be installed in the Power Macintosh 7200, 7500, 8500, and 9500 computers. The 7" card is also designed for installation in future systems that may have mechanical size constraints.

Unless otherwise specified, the information provided in this chapter applies to both cards.

# Hardware Overview

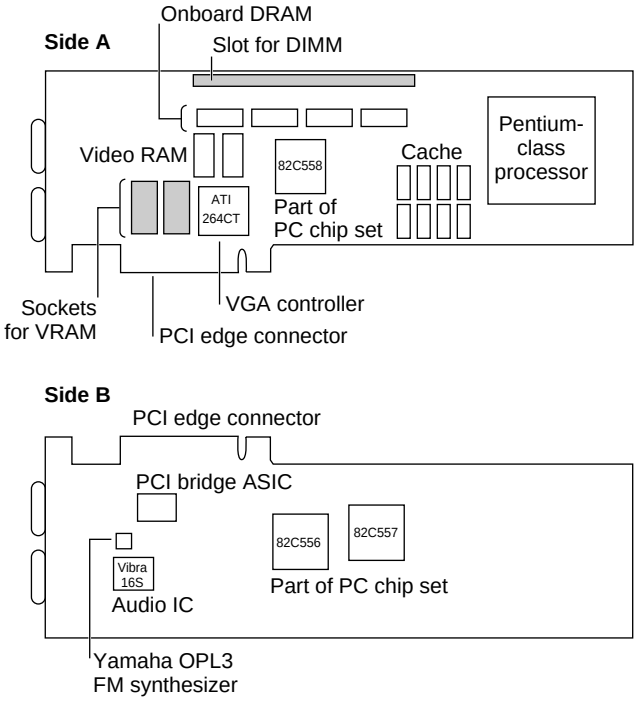
Both cards are two-sided; that is, there are components on each side of the card. Figure 2-1 on page 11 shows outlines of both sides of the 12″ card, with key ICs. Figure 2-2 on page 12 shows equivalent views of the 7" card. Figure 2-3 on page 13 is a functional block diagram of the 12″ card. Figure 2-4 on page 14 is a functional block diagram of the 7" card.

The circuitry housed on the 12″ and 7" cards includes the following:

- The microprocessor or CPU.
  - the 12″ card has a 100 MHz Pentium-class processor.
  - the 7" card has a 100 MHz 5x86-class processor.
- The memory system, including the DRAM, cache memory, and memory expansion module.
  - the 12″ card has 8 MB of onboard DRAM. It also has a 168-pin DIMM slot. It allows you to increase DRAM capacity by installing DIMMs with capacities up to 64 MB, providing a total DRAM capacity of 72 MB. The card has a 256 KB cache.
  - the 7" card has 8 MB of onboard DRAM supplied by an 8 MB DIMM housed in the card's 168-pin DIMM slot. You can expand the 7" card's DRAM capacity up to 64 MB by replacing the 8 MB DIMM with DIMMs of higher capacity. The card has a 128 KB cache.

- The PC chip set, which provides a controller for memory, for the PCI bus, and for the ISA bus.
  - the 12” card uses the Opti chip set composed of the 82C556, 82C557, and 82C558 ICs.
  - the 7" card uses the Sis 486-SIS chip set composed of the 85C496 and 85C497 ICs.
- The ATI 264CT integrated VGA (video graphics adapter) graphics accelerator IC with a built in color lookup table (CLUT) for video output. This IC supports the PCI bus I/O, and a variety of memory types and sizes, screen resolutions, and color depths.
- The Creative Technologies Vibra 16S 16-bit audio IC and the Yamaha OPL3 FM synthesizer chip set, which provide audio output compatible with the Sound Blaster Pro products.

**Figure 2-1** 12” card with featured ICs

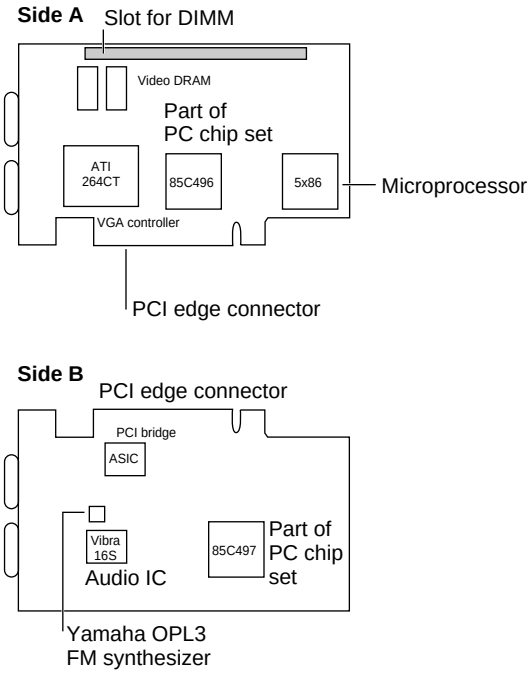



*Note:* PCI edge connector is shown to define orientation.

**Note**

Card dimensions are shown in Figure 1-2 and Figure 1-3 on page 4. Connectors are shown in Figure 3-1 on page 34 and Figure 3-2 on page 35. ◆

**Figure 2-2** 7" card with featured ICs




*Note:* PCI edge connector is shown to define orientation.

**Figure 2-3** Detailed block diagram—12” card

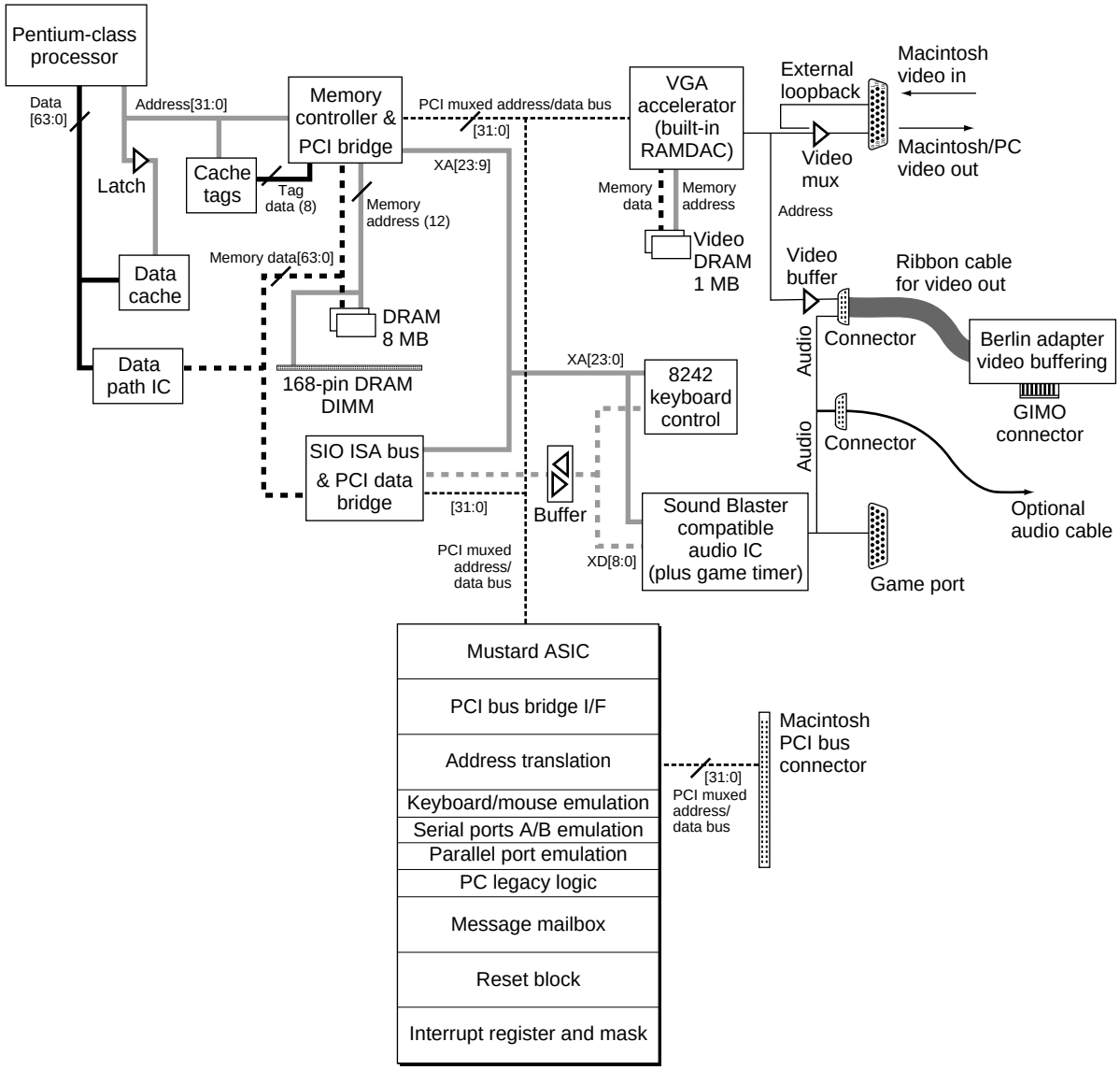
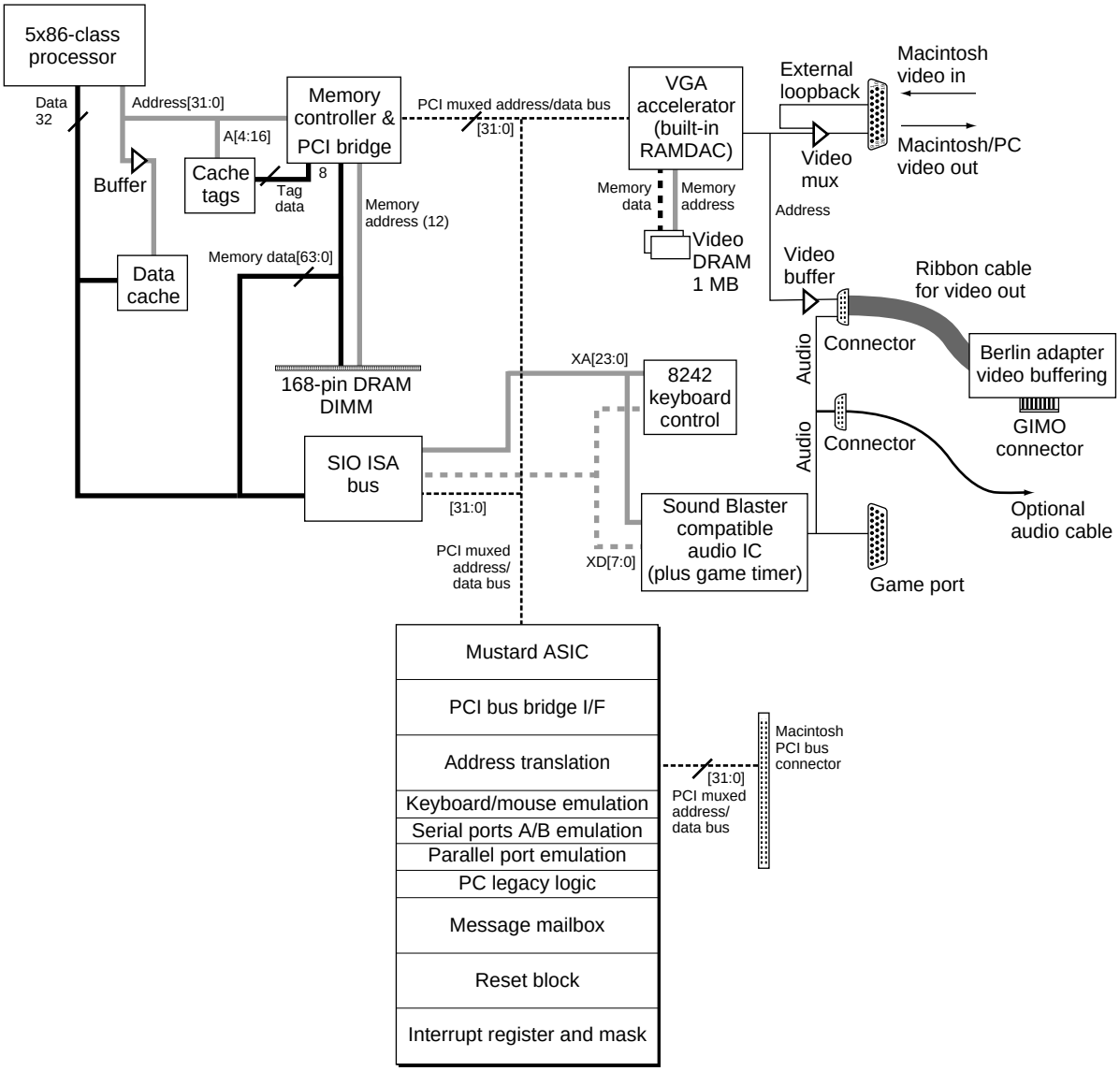


**Figure 2-4** Detailed block diagram—7"



# Hardware Features

This section describes the microprocessors on each card, PCI bus and bus devices, cache operation, byte order, interrupts, bus arbitration, memory controller, BIOS control, system clocks, and system reset.

## Microprocessors

The microprocessor on the 12” card is a Pentium-class microprocessor that runs at 100 MHz. It supports a 64-bit data path and a 32-bit address bus. The 7" card has a 100 MHz 5x86-class processor. It supports a 32-bit data path and a 32-bit address bus.

You should refer to the hardware reference manuals supplied by the microprocessor manufacturer for further information on the different microprocessors.

## PCI System Bus and Devices

The PCI bus on the PC system is a multiplexed address and data bus. The bus operates synchronously at the same clock speed as the microprocessor bus (25 MHz or 33 MHz). The bus supports burst reads and writes from the Mustard ASIC alternate bus master. The key devices attached to the bus are the memory controller/bridge IC, the VGA accelerator IC, and the Mustard ASIC. The PCI connector plugs into the PCI bus on the Macintosh side of the system by means of the PCI slot in the Macintosh host computer. “PCI Connector” beginning on page 35 provides detailed information about the PCI bus signals.

## Bus Snooping

Bus snooping is the method used by cache subsystems to monitor memory accesses performed by different bus masters. This means that the internal microprocessor cache can monitor activity on the PCI bus that is likely to change the contents of either the L1 or L2 cache memory.

The PC memory is fully cache coherent with the local PCI bus, and there are no cache coherency problems with Macintosh memory. Bus snooping required by the Macintosh host computer is done by the Macintosh and does not affect the PC system.

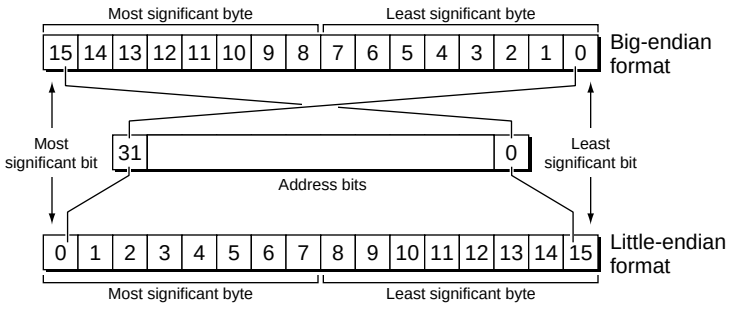
## Byte Order

There are two ways of defining the order in which bytes of data are addressed.
Big endian is a type of data formatting in which each field is addressed by referring to its most significant byte. The most significant byte is the one with the highest number. For example, if you are accessing a 4-byte, 32-bit data word, the most significant byte is byte 03, and the most significant bit is bit 31. The most significant byte is selected by the lowest address, and the least significant byte by the highest address. Macintosh computers use the big-endian data format.

Little endian is a type of data formatting in which each field is addressed by referring to its most significant byte. The most significant byte is the one with the lowest number. For example, if you are accessing a 4-byte, 32-bit word, the most significant byte is byte 00, and the most significant bit is bit 00. The most significant byte is selected by the highest address, and the least significant byte by the lowest address. Computers based on Intel architectures, such as IBM PCs, use the little-endian format.

Although the Macintosh generally uses a big-endian data format, the PCI bus by definition is little-endian on both the PC and Macintosh sides. Byte swapping, therefore, takes place on the Macintosh side before the Macintosh places data on the PCI bus.

**Figure 2-5** Example of big-endian and little-endian data formats




For further information about big-endian and little-endian addressing, refer to *Designing Cards and Drivers for Power Macintosh Computers*, Appendix B, “Big-Endian and Little-Endian Addressing.”

## Interrupts

The Mustard ASIC generates all interrupt requests that require Macintosh resources for the microprocessors on the 12” and 7" cards. The controller (the 82C558 IC on the 12” card , and the 85C497 IC on the 7" card) generates the maskable interrupt that results from the various IRQ (interrupt request) sources. In each instance, the interrupt provides

a mechanism that allows each device (keyboard, serial port, and so forth) to interrupt the CPU asynchronously. Table 2-1 defines the interrupts used in both cards.

**Table 2-1** Definition of interrupts

| Interrupt number | Description |
|---|---|
| 1 | Keyboard data ready |
| 3 | COM2 port |
| 4 | COM1 port |
| 5 | Sound Blaster |
| 6 | Message interface |
| 7 | Parallel port 1 (printer) |
| 12 | Mouse |

All transfers between the PC system and the Macintosh host computer are interrupt driven. The master interrupt register, which is part of the Mustard ASIC, contains the state of all interrupt sources on the card. Each of these sources can be individually masked by an accompanying master interrupt enable register. You can determine the precise reason for the interrupt by examining the registers for the ports associated with the interrupts listed in Table 2-1.

You can access the interrupt and status registers in the Mustard ASIC from the Macintosh environment only. In the PC environment, you will see the standard definitions for the COM1, COM2, and parallel ports.

The system interrupts for the 12″ card are controlled by the 82C558 IC. On the 7″ card, the 85C497 IC controls the interrupts.

## Bus Arbitration

On the PC side of the PCI bus there are two functional circuit blocks that can act as bus master. They are the Mustard ASIC and the PC chip set.

Each bus master must arbitrate for control of the bus using a straightforward request-grant handshake. Each bus master has a unique request signal (REQ L) that it asserts to request access to the bus. An arbitration algorithm examines the request and assigns access to the bus based on a rotating priority system. When access is granted, a grant signal (GNT L), unique to each master, is returned.

A bus error on the PC system bus causes the PC system (12″ or 7″) to hang. When this happens, Macintosh operation is not affected, and you can use the Macintosh computer to restart the PC system. You can restart from the Macintosh using Restart from the Finder menu, or by using the Cmd-Ctl-Alt-Del key sequence on the Macintosh keyboard. Note that on some keyboards, the Alt key is labeled Option or Alt Option. If the PC

keyboard is still responding, you can restart the PC system using the Ctl-Alt-Del key sequence on the PC keyboard.

The Macintosh system bus supports three bus masters: the Macintosh memory controller, the Mustard ASIC on the 12″ or 7″ card, and any other PCI card in the Macintosh.

**IMPORTANT**

Any other PCI card in the Macintosh must be of a different type from the 12″ or 7″ card. ▲

Table 2-2 summarizes the fixed arbitration device assignments and priorities.

**Table 2-2** PC arbitration priorities

| Priority | Macintosh side | PC system side |
|---|---|---|
| High | I/O IC | Memory controller (Mustard ASIC) |
| Low | Any PCI device | Any PCI device |

## Memory Controller

Memory control for the 12″ card is provided by the 82C557 IC, and by the 85C496 IC for the 7″ card. These ICs perform the following system-level functions for the related card:

- control DRAM
- control cache memory
- act as a bridge between the PC system's local bus and PCI bus

### DRAM Control

The memory controller controls all transfers between the microprocessor and the onboard DRAM and DIMM. It generates the row and column address strobe signals (RAS, CAS) and the read and write enable signals (MEMR L and MEMW L) for the DRAMs and the DIMM. It drives these signals and the memory address lines directly to the memory devices without external buffers.

The DRAM controller in the memory controller chip supports page-mode operations. For memory read operations the page hit cycles are either 3-2-2-2 or 4-2-2-2 bursts. For write operations, the page hits are single wait-state accesses. Both read and write operations are designed for DRAM devices with a 70ns access time.

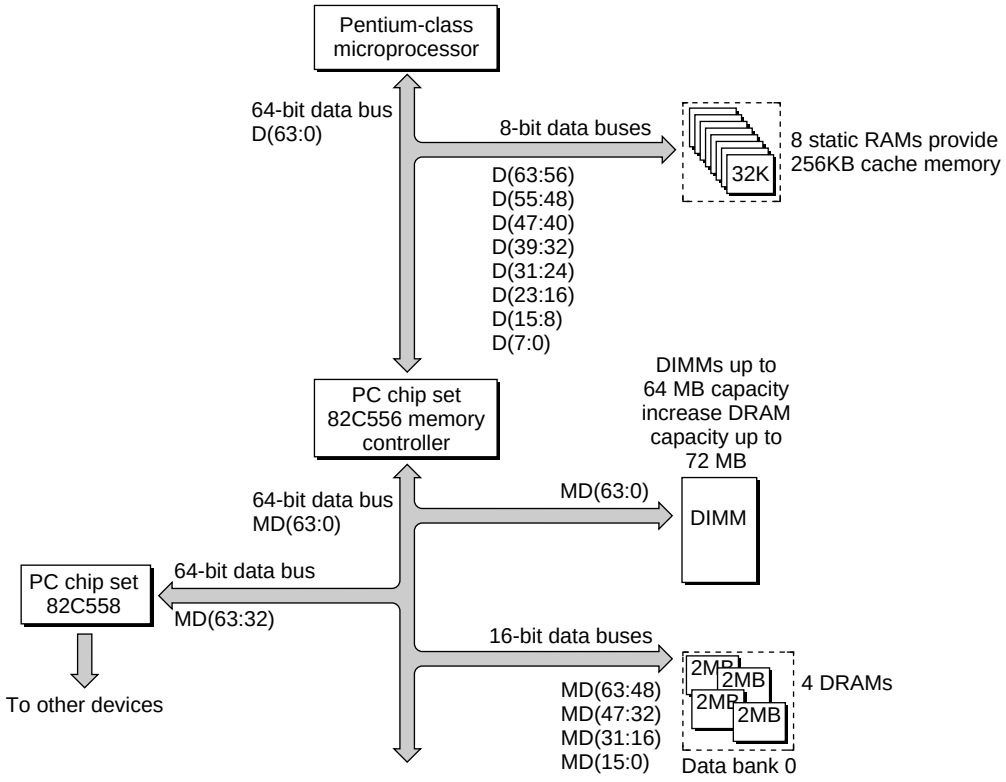
## 12” Card Memory Control—DRAM and External Cache

Basic memory for the 12” card is provided by four 2 MB DRAMs that provide 8 MB of memory. The DRAMs are organized as a 1-by-64-megabit memory bank (bank 0), as shown in Figure 2-6.

The 12” card has a 168-pin DIMM slot that can accommodate DIMMs with parity-checking capability. (The parity bits are not currently enabled.) You can use DIMMs of different capacities in the slot (8 MB, 16 MB, 32 MB, or 64 MB), expanding memory capacity up to 72 MB. This block of memory also has a 64-bit data path and is addressed as bank 1.

The 12” card has 8 static RAMs that make up the 256 KB cache memory. Each static RAM has an 8-bit data bus. The memory controller controls all transfers between the microprocessor and the external cache. It generates the cache chip select signal, CCS L 7:0, and the read/write signal, ECAWE L. It buffers the control signals and drives them out to the cache. It drives the cache address lines through external latches, which hold the last address as the microprocessor generates the next address.

**Figure 2-6** DRAM control—12” card

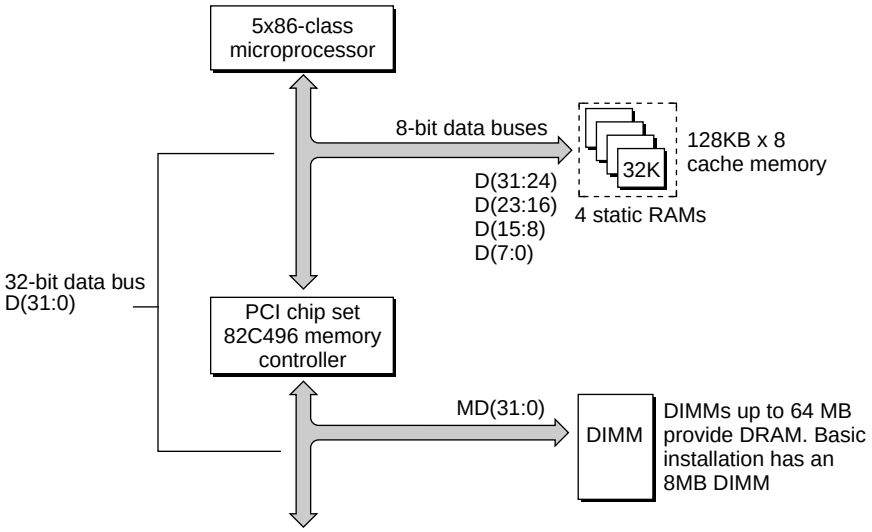


### 7” Card Memory Control—DRAM and External Cache

DRAM for the 7″ card is provided by a 168-pin DIMM that accommodates an 8 MB, 16 MB, 32 MB, or 64 MB DIMM. The 7″ card comes with an 8 MB DIMM installed. You can replace this DIMM with one of higher capacity, to provide a maximum DRAM capacity of 64 MB. As shown in Figure 2-7, the DIMM has a 32-bit data bus and is addressed as banks 1 through 4.

The 7″ card has four static RAMs that make up the 128 KB cache memory. Each static RAM has an 8-bit data bus. The memory controller controls all transfers between the microprocessor and the external cache. It generates the cache chip select signal, KCE L 3:0, and the read/write signal, KWEX. It buffers the control signals and drives them out to the cache. It drives the cache address lines through external latches that hold the last address as the microprocessor generates the next address.

**Figure 2-7** DRAM control for the 7” card




### Sensing DIMM Presence

The Mustard ASIC senses the presence of a DIMM in the DIMM slot when the system starts up and then registers that information. However, the PC BIOS determines the size of the memory using a sizing algorithm, and the BIOS programs the memory controller’s configuration registers with the starting and ending address of each memory bank.

## BIOS Control

PC computers use a BIOS. The 12″ and 7″ cards have a combined BIOS consisting of both system and VGA BIOS. The BIOS is stored in Macintosh memory and downloaded to the DRAM on the 12″ and 7″ cards when the system starts up.

**Note**
The 12″ and 7″ cards have different BIOS because they have different microprocessors and chip sets. ◆

At reset, the microprocessor on the card issues the starting reset-vector address from within the range of addressable memory. The Mustard ASIC remaps this address range down to the lower 1 MB region where the BIOS actually resides. It also performs the address translation between the BIOS address on the PC system and the corresponding addresses in memory on the Macintosh side.

## Clock Generation

The system clock is generated by a standard oscillator that produces a 14.3 MHz clock signal. This signal is transferred to a PLL (phased lock loop) device, the IMISC-464 clock divider, which creates the microprocessor bus clock (66 MHz on the 12″ card and 33 MHz on the 7″ card), the PCI clock (33 MHz), and the buffered 14.3 MHz clock. Table 2-3 provides information about clock signal distribution.

**Table 2-3** Clock signal distribution

| Clock signal | Destination | Frequency | Function |
|---|---|---|---|
| **12″ card** | | | |
| CPU CLK | Microprocessor | 66 MHz | Provides the bus timing for the microprocessor. All system timing is referenced from this clock. |
| **7″ card** | | | |
| CPU CLK | Microprocessor | 33 MHz | Provides the bus timing for the microprocessor. All system timing is referenced from this clock. |
| **12″ and 7″ cards** | | | |
| XPCI CLK | ATI-264CT video controller and Mustard ASIC | 33 MHz | Clock for the PCI bus in the PC system. |

## System Reset

The Mustard ASIC contains the reset logic that allows the Macintosh host computer to start up or reset the PC system. Hard reset is controlled by the Macintosh software. The Mustard ASIC generates a PWRONRST (power on reset) signal, which, in the 12″ card the 82C557 converts into a Reset signal for the 82C556/558 and the CPU. In the 7″ card, the 85C497 generates reset for the 85C496, the CPU, and the ISA bus. With both cards you can do a soft reset by means of the Ctl-Alt-Del keys on the PC keyboard. This type of reset is handled by the 8242 keyboard controller when the proper key code is sent to the Mustard ASIC through the keyboard port.

# Video System

The 12″ and 7″ cards in conjunction with video cards provide a complete video system to support PC video. This system consists of a VGA controller (ATI 264CT) with an integrated CLUT, a digital-to-analog converter (DAC), and a clock generator.

PC video out is supported not only on VGA monitors but also on Macintosh fixed-frequency monitors that emulate VGA modes. This means you can use your existing Macintosh monitors in a shared configuration for both Macintosh and PC video.

## Connecting the Monitor

Video output from the 12″ and 7″ cards can be displayed on a monitor that is shared with the host Macintosh computer or, with some CPUs, on a dedicated monitor. “Loop-Back Video Support” beginning on page 30 and “GIMO Support for Video Output” beginning on page 28 provide detailed information on this subject. If you are using a shared monitor, you can switch from Macintosh to PC screen using a programmable key sequence.

## Monitors Supported

The VGA controller/accelerator provides complete VGA compatibility for modes 0–7 and D–13h. Both cards come with 1 MB of video DRAM installed. The 12″ card has two additional DRAM sockets that allow you to add another 1 MB of video DRAM. The 1 MB DRAM enables a 32-bit data path, and the additional memory enables a 64-bit data path. This additional width in the data path supports additional color depths, as indicated in Table 2-4.

The ATI 264CT senses which monitor is attached whenever you use the loopback cable to connect the monitor to the cards. This is true even in shared-monitor mode. When you use the GIMO connector to connect the monitor, the Macintosh host computer determines the monitor for the PC and makes the appropriate adjustments.

**Note**

The modes listed in Table 2-4 apply to the PC display connected to the system, or to the second Macintosh display, if you have elected to use the 12″ or 7″ card as a secondary Macintosh display card. ◆

**Table 2-4** Monitors and display modes supported

| Monitor | Resolution (pixels) | Horizontal scan (kHz) | Vertical refresh (Hz) | Maximum color depth (bits per pixel) | |
|---|---|---|---|---|---|
| | | | | 1 MB video DRAM | 2 MB video DRAM |
| 21″ color | 1152 × 870 | 68.681 | 75.062 | 8 | 8 |
| 21″ 2-page (mono) | 1152 × 870 | 68.681 | 75.062 | 8 | 8 |
| 19″ RGB | 1024 × 768 | 60.241 | 74.927 | 8 | 8 |
| 16″ color | 832 × 624 | 49.725 | 74.550 | 16 | 16 |
| 15″ RGB portrait | 640 × 870 | 68.850 | 75 | 8 | 8 |
| 13″ color | 640 × 480 | 35 | 66.667 | 16 | 32 |
| 12″ mono | 640 × 480 | 35 | 66.667 | 8 | 8 |
| Multi Scan 15″ | 640 × 480 | 35 | 66.667 | 16 | 32 |
| | 832 × 624 | 49.725 | 74.550 | 16 | 16 |
| Multi Scan 17″ | 640 × 480 | 35 | 66.667 | 16 | 32 |
| | 832 × 624 | 49.725 | 74.55 | 16 | 16 |
| | 1024 × 768 | 60.241 | 74.927 | 8 | 8 |
| Multi Scan 20″ | 640 × 480 | 35 | 66.667 | 16 | 32 |
| | 832 × 624 | 49.725 | 74.55 | 16 | 16 |
| | 1024 × 768 | 60.241 | 74.927 | 8 | 8 |
| | 1152 × 870 | 68.681 | 75.062 | 8 | 8 |
| | 1280 × 1024 | 79.976 | 75.025 | 4 | 4 |
| VGA | 640 × 480 | 31.463 | 60 | 16 | 32 |
| SVGA | 800 × 600 | 35.156 | 55.98 | 16 | 32 |
| VESA | 800 × 600 | 37.879 | 60.3165 | 16 | 32 |
| | 800 × 600 | 48.077 | 72.188 | 16 | 16 |
| | 800 × 600 | 46.875 | 75 | 16 | 16 |
| | 1024 × 768 | 48.363 | 60 | 8 | 16 |
| | 1024 × 768 | 56.476 | 70 | 8 | 16 |
| | 1024 × 768 | 60.023 | 75 | 8 | 8 |
| | 1280 × 960 | 75 | 75 | 4 | 4 |
| | 1280 × 1024 | 65.625 | 60 | 4 | 8 |
| | 1280 × 1024 | 79.976 | 75.025 | 4 | 4 |

## Video Timing

To accommodate the various VGA and SVGA modes on the Macintosh monitors, the video controller must have its timing parameters changed by the BIOS. To do that, the Macintosh software reads the video sense lines and loads the appropriate values for the video BIOS before starting up the PC. The system and video BIOS reside in Macintosh system memory and can be modified by the software.

## Video ICs

Video support for the cards is provided by an ATI-264CT VGA controller and by the video DRAM.

The ATI-264CT VGA controller is a 208-pin graphics controller IC with a built-in accelerator, referred to as the coprocessor. It is compatible with the VGA display adapter and provides for modes 0–7 and D–13h. SVGA modes for 640 by 480 and 800 by 600 are also supported up to 64K colors, and SVGA modes for 1024 by 768 up to 256 colors. You should refer to the reference material provided with the device for detailed information about the controller.

Each card has 1 MB by 32 bits of installed video DRAM. The 12″ card also has sockets that allow you to expand video DRAM capacity to 2 MB, with a 64-bit data path and higher resolution support.

# Audio System

The sound system for the cards is built around a Creative Technologies Vibra 16S 16-bit audio IC and a synthesizer chip set that includes a Yamaha OPL3-L FM synthesizer and YAC516 DAC IC. These ICs provide the cards with 16-bit stereo Sound Blaster Pro capabilities.

**Note**
Sound Blaster Pro products, such as the audio IC and the synthesizer chip set, support the sound capabilities of PC systems, in this case, the 12″ and 7″ cards. Sound Blaster Pro installation software is a de facto standard set by Creative Technologies. ◆

## Audio IC

The Creative Vibra 16S IC provides 16-bit audio support. It is compatible with Sound Blaster 16 and with Roland MPU401 UART (universal asynchronous receiver/transmitter) mode. It also complies with Multimedia PC Level 2 specifications. The device also has an integrated 16-bit SigmaDelta codec that handles ADP (audio digital processor) inputs, ADP digital outputs, and DAC outputs.

The audio IC provides the following:

- a 16-bit bus interface with the ISA bus
- a digital audio processor (DAP) block that interprets Sound Blaster 16 commands and provides downward compatibility with Sound Blaster products
- a mixer block that controls volume and mixing functions
- a joystick quad timer and data buffer that allow MIDI devices to be connected to the GamePort without external TTL (transistor-transistor logic) support

For complete specifications, you should refer to the reference material provided with the audio IC.

## Sound Synthesizer Chip Set

This chip set consists of two ICs: the synthesizer and the digital-to-analog converter. The Yamaha synthesizer is a YMF262 Type 13 OPL3 device. It uses FM (frequency modulated) synthesis to generate sounds. The IC includes the following features:

- 24 operators that can be configured in four-operator mode for six channels
- 36 operators that can be configured in two-operator mode for 18 channels, or for 15 channels with five rhythm channels
- eight selectable FM source waveforms
- four channels of sound output
- hardware vibrato and tremolo effects
- two programmable times that can generate interrupt requests

Because the YMF262 outputs voice data as digital data, it is necessary to convert this data back to analog form. This function is performed by the YAC516, a delta sigma DAC with an eight times oversampling filter.

For complete specifications, you should refer to the reference material provided with these two ICs.

# I/O System

The interface between the Macintosh computer and the 12″ and 7″ cards is provided by the PCI connector, which connects the PCI bus on the PC system with the PCI bus on the Macintosh side. I/O control on the cards is provided by the Mustard ASIC and the 8242 controller, which controls the mouse and keyboard. The Mustard ASIC acts as a bridge between the two PCI buses. The ASIC also contains PC I/O emulation logic, which integrates many of the I/O functions required to support the PC. It supports the following features:

- emulation of two 16C450-compatible serial ports
- emulation of one Centronics parallel printer port
- emulation of keyboard and mouse controller that allows the cards to access the Macintosh keyboard and mouse by means of the ADB
- a 64-bit message mailbox with a 32-bit command port
- address translation
- power-on reset logic
- general-purpose I/O ports (autoconfiguration logic)
- interrupt status and mask registers

The Mustard ASIC can function either as a slave or as an alternate bus master on the Apple PCI bus or on the PC PCI bus.

## Serial Port Support

From the PC environment, you can connect a modem or other serial device to the Macintosh serial port. This is an 8-pin serial connection on the computer's back panel, identified by either a printer or modem icon.

To support serial ports, the Mustard ASIC contains two identical sets of UART emulation registers. These registers emulate the hardware of the standard 16C450 serial port ICs found in many PC/AT computers. When the microprocessor on the 12″ or the 7″ card accesses these registers, interrupts are generated in the Macintosh host computer. These interrupts signal a driver on the Macintosh computer to route the data to the Macintosh serial ports.

The Macintosh serial ports are RS-422 ports and do not support the following RS-232 signals: Carrier Detect (CD), Data Set Ready (DSR), Request to Send (RTS), and Ring Indicator (RI) signals.

An adapter cable is needed to connect a PC serial device to a Macintosh serial port. To help you design a custom cable to make the serial connection, Table 2-5 shows the signal names and pin numbers on the RS-422 8-pin connector on the Macintosh serial port, the

pins that carry these signals on PC-style DB9 and DB25 connectors, and the signal names on the RS-232 connector.

**Table 2-5** Serial port signals

| Macintosh pin number | RS-422 signal name | Description | DB9 pin number | DB25 pin number | RS-232 signal name |
|---|---|---|---|---|---|
| 1 | HSKo | Handshake signal, output | 4 | 20 | DTR |
| 2 | HSKi | Handshake signal, input | 8 | 5, 8 | CTS, DCD |
| 3 | TXD- | Transmit data inverted | 3 | 2 | TXD |
| 4 | GND | Ground | 5 | 7 | GND |
| 5 | RXD- | Receive data inverted | 2 | 3 | RXD |
| 6 | TXD+ | Transmit data | n.c. | n.c. | — |
| 7 | GPi | General-purpose input | n.c | n.c. | — |
| 8 | RXD+ | Receive data | 5 | 7 | GND |

## Printer Port Support

The Mustard ASIC contains logic that implements all the registers of the standard Centronics parallel port found on PCs. When the PC accesses these registers, interrupts are generated in the Macintosh host computer that cause the driver software in the Mac OS to send data to a print spooler file. The spooler file is then sent to whatever printer the user selects in the Macintosh environment.

**Note**

The parallel port interface does not control printer hardware signals and does not support bidirectional data transfer. ◆

## Keyboard and Mouse Controller

The Mustard ASIC contains logic that emulates in hardware the PC keyboard and mouse. It also generates the appropriate serial clock protocol and serial bit stream required to communicate with the 8242 keyboard/mouse controller. The controller is configured to support the PS2 mouse, and it makes the protocol identical for both the mouse and keyboard.

## Message Mailbox

The message-passing interface in the Mustard ASIC supports simple interrupt-driven communication between the PC and the Macintosh host computer. The message-passing interface contains two data registers and one command register. The interface uses a mechanism of arbitration and grants to control the direction of message transfer. Refer to

"Passing Messages" beginning on page 72 for a description of the software API used for passing messages.

### Autoconfiguration

The Mustard ASIC automatically configures the PC system each time the PC is reset. The following configurations are sensed and set when the system is reset:

- presence of DIMM in DIMM slot
- setup switches for audio path
- setup path for interrupts from the ISA bus

## Audio and Video I/O Support

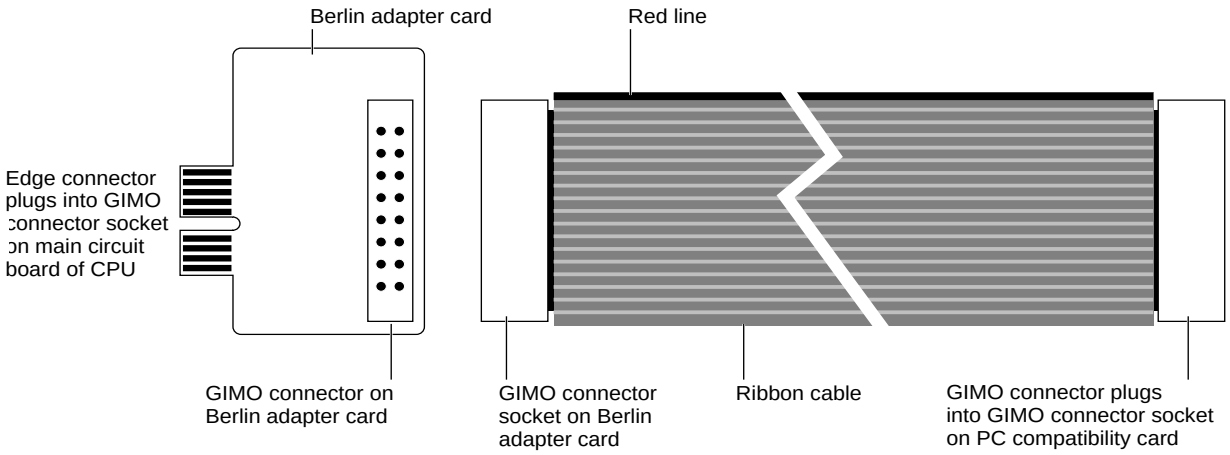
The way in which the cards support audio and video I/O depends on the Macintosh computer in which the cards are installed. This section gives an overview of the different I/O configurations supported. You should refer to the developer note and user's guide for your specific computer for more detailed information.

### GIMO Support for Video Output

The 12" and 7" card each has an onboard GIMO (Graphics Internal Monitor Out) connector socket. Figure 3-1 on page 34 and Figure 3-2 on page 35 show the position of the GIMO connector on the 12" card and the 7" card, respectively. "GIMO Connector" on page 46 provides specifications for the connector.

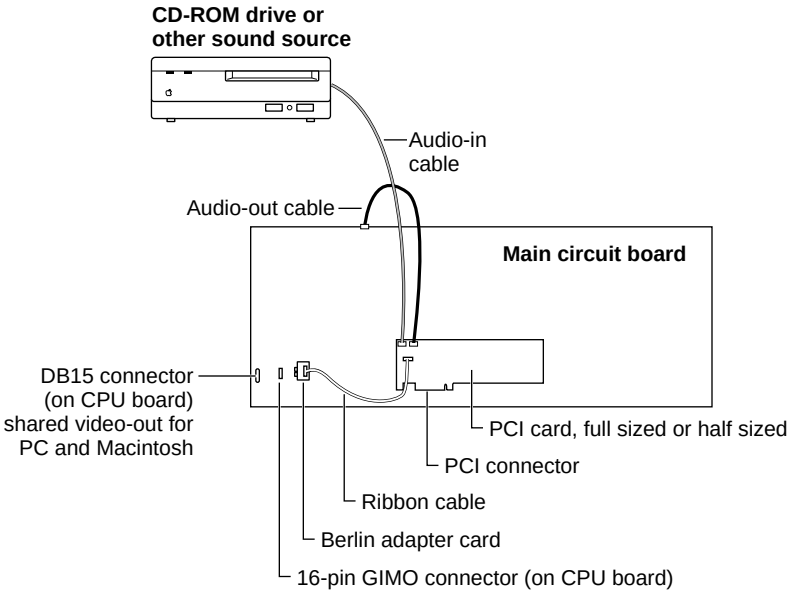
Certain CPUs, such as the Power Macintosh 7200, have equivalent GIMO connector sockets. If you are using the cards in this type of computer, you can connect the monitor using a ribbon cable and an adapter card known as the Berlin adapter card. Figure 2-8 shows a simplified view of the ribbon cable with GIMO connectors, and the Berlin adapter card with its GIMO connector socket and edge connector.

**Figure 2-8** GIMO connectors and Berlin adapter card




To make the connection, you plug the GIMO connector on the ribbon cable into the GIMO connector socket on the PCI card, as shown in Figure 2-9. You then connect the GIMO connector on the other end of the cable to the GIMO connector socket on the Berlin adapter card and then plug the Berlin adapter card into the GIMO connector socket on the CPU's main circuit board. This is the best way to integrate the PC compatibility card with the Macintosh host system, since the PC video can be alternately driven onto the built-in monitor or the external monitor without using an additional loop-back cable.

**Figure 2-9** I/O connections to Power Macintosh 7200




## Loop-Back Video Support

The Power Macintosh 7500, 8500, and 9500 CPUs do not have a GIMO connector socket to support the Berlin adapter card. The video interface with these CPUs is implemented by means of a loop-back cable, as shown in Figure 2-10. This is a special Y-shaped cable with three connectors. It connects the CPU's DB15 connector to the PCI card's DB26 connector, and it also provides a connector for a monitor.

**IMPORTANT**

The H-shaped loop-back cables used with DOS compatibility cards in Power Macintosh 6100 computers are not compatible with the 12″ and 7″ cards. You should not try to use them with these cards. ▲

You can also configure systems like the Power Macintosh 9500 with a secondary video card that allows you to connect a second Macintosh display if you wish.

If you are interfacing with a CPU that can accommodate a Berlin adapter and GIMO connector, such as the Power Macintosh 7200, you can use the loop-back cable to connect a second monitor, which may be used as a dedicated DOS/Windows screen.

Video connections for the Power Macintosh 9500 are made in a similar way, as shown in Figure 2-11 on page 32.

**Figure 2-10** I/O connections to Power Macintosh 7500 and 8500

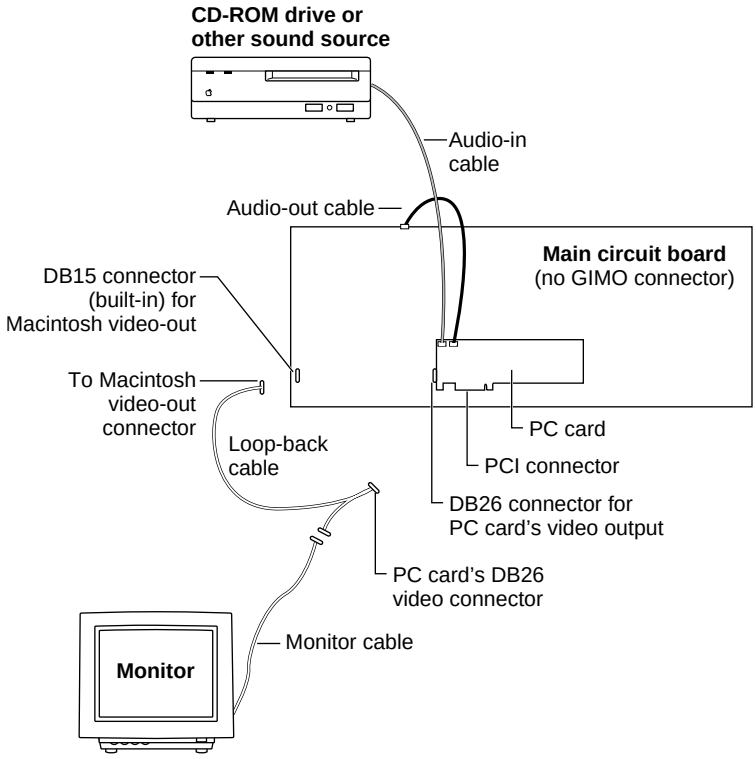
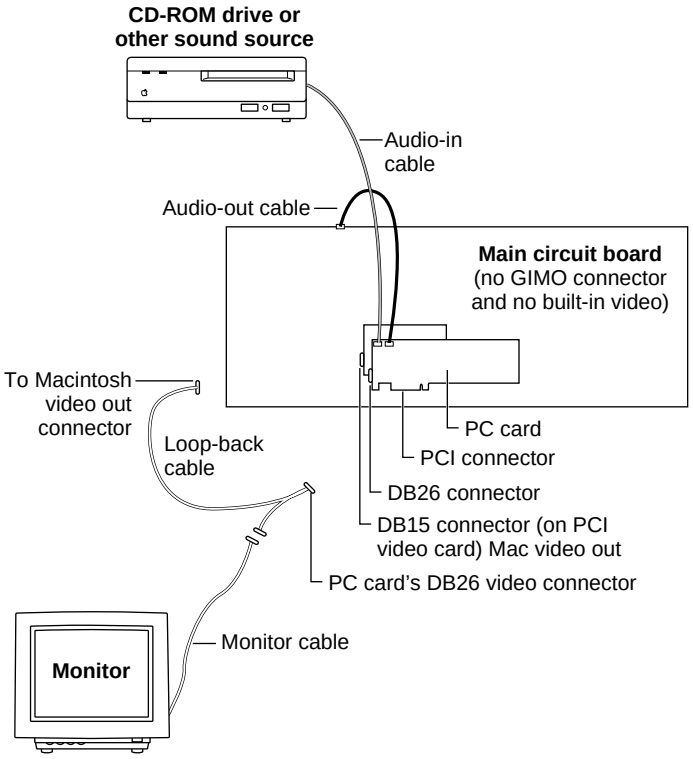


**Figure 2-11** I/O connections for Power Macintosh 9500




## Audio I/O Support

The 12″ and 7″ cards have connectors for audio I/O located on the edge of the cards, as shown in Figure 3-1 on page 34 (12″ card ) and Figure 3-2 on page 35 (7″ card). “Audio Connectors” on page 47 provides specifications for these connectors. As shown in Figure 2-9 on page 30, Figure 2-10 on page 31, and Figure 2-11, the audio output cable is connected to the audio output connector on the PCI card and then plugged into the audio out connector on the CPU’s main circuit board. The audio input cable is connected to the audio input connector on the PCI card and then plugged into an audio source such as a CD-ROM drive or other sound source.

CHAPTER 3

# I/O Specifications

The 12” and 7” PC Compatibility Cards interface with other elements in the system, such as the main CPU, the monitor, sound devices, and video devices, through a variety of connectors located on the cards. Figure 3-1 shows the physical locations of the I/O connectors on the 12” card. Figure 3-2 shows the locations of the connectors on the 7" card.

The way the interface is implemented depends on the CPU in which the card is installed. This section describes the actual physical connectors on each of the cards. For more information on the components that make up the I/O system for the two cards, see “I/O System” beginning on page 26. For more information on CPU-dependent I/O configurations, see “Audio and Video I/O Support” beginning on page 28.

This chapter describes the following connectors:

- the PCI connector, which allows you to connect the cards to the host computer’s main logic board
- the DB26 connector, which allows you to connect a monitor to the system
- the DB15 connector, which allows you to connect a mouse, keyboard, PC-compatible joystick, or a MIDI device
- the GIMO connector, which allows you to connect a monitor to the system
- two audio connectors for sound input and sound output
- a DIMM connector, which allows you to install additional memory
- an XD connector, which provides you with access to the ISA bus

**Figure 3-1** Locations of I/O connectors for the 12” card

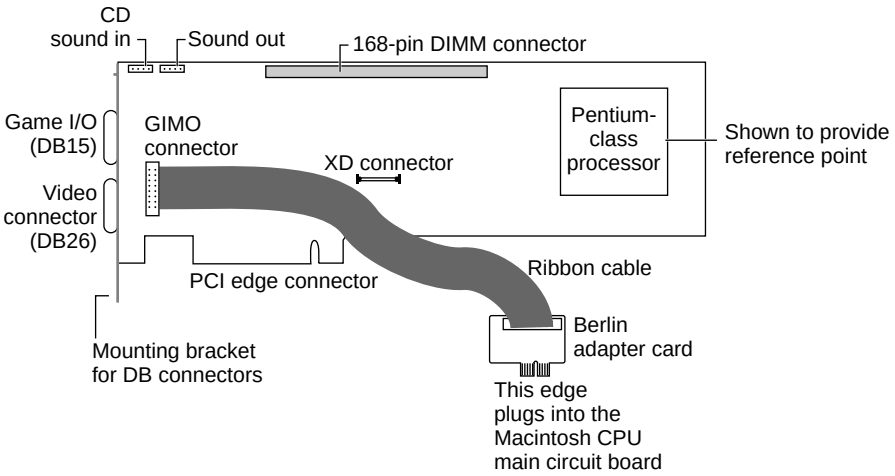
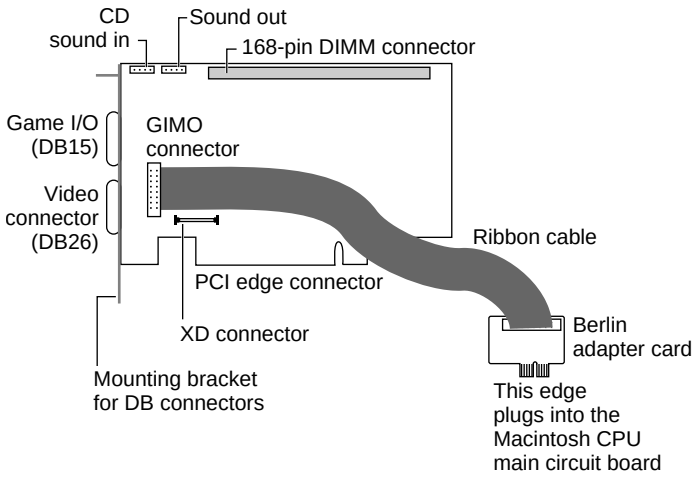


**Figure 3-2** Locations of I/O connectors for the 7” card




# PCI Connector

The PCI local bus is a bus architecture that allows you to connect cards such as the 12″ and 7″ cards to the computer’s main circuit board. Figure 3-3 shows in functional groups the PCI bus signals used on the cards. The PCI signals not used in the 12″ and 7″ cards and ground and power signals are not shown in Figure 3-3. Table 3-1 on page 36 lists some of the bus specifications.

Table 3-2 on page 38 lists and describes the PCI bus signals used on the 12″ and 7″ cards. For further information about the PCI bus, refer to *Designing PCI Cards and Drivers for Power Macintosh Computers,* published by Apple Computer, and *PCI Local Bus Specification,* published by Intel Corporation.

**Figure 3-3** PCI bus signals used in the 12” and 7” cards

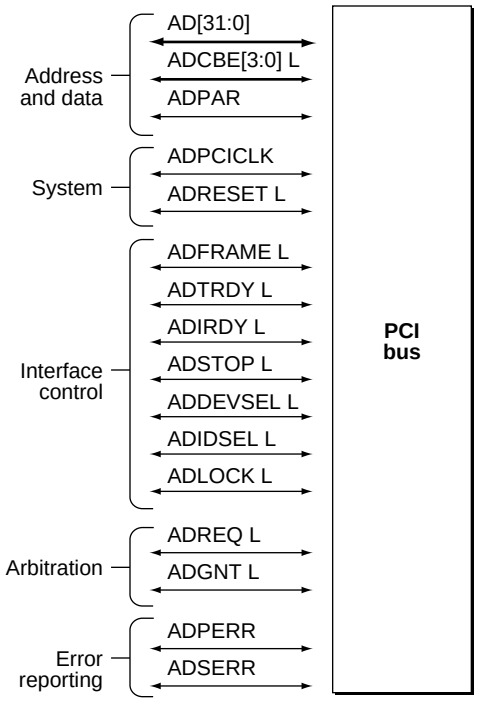



**Table 3-1** PCI bus specifications

| Feature | Description |
|---|---|
| Bus clock rate | 33 MHz |
| Addressing | Dynamic |
| Signal loading | One load per signal |
| Transaction length determination | Determined at end of transaction |
| Bus termination | Not required |
| Bus control arbitration | Centralized |
| Addressing spaces | Memory, I/O, and configuration |
| Wait-state generators | Slave and master |
| Kinds of expansion | Cards and ASIC chips |

**Table 3-1** PCI bus specifications (continued)

| Feature | Description |
|---|---|
| Timeout | 5 bus clocks |
| Burst capability | Any number of bytes |
| Power allocation | 25 W per card for 12″ card<br>15 W per card for 7″ card |

With respect to the pin assignments and signal descriptions listed in Table 3-2, you should note the following:

- An “L” used as a suffix to the signal name, for example, FRAME L, indicates a signal that is active when driven low. Signals with no suffix are active when they are driven high.
- In this context, the following terms are used to define the different components in the system:
  - *Agent* indicates any entity that operates on the PCI bus.
  - *Master* indicates any agent that initiates a bus transaction.
  - A *host bridge* is a low latency path through which the processor may directly access PCI devices that are mapped anywhere in memory, I/O, or configuration address spaces.
  - *Target* indicates any agent that responds with a positive acknowledgment (DEVSEL L) to a bus transaction initiated by the master.
- The term *tristate* is applied to signals that are capable of three states: high, low, or off. When the signal is off, it is said to be in a tristate condition or to be tristated.
- Address bits 31:00 and data bits 31:00 (AD(31:0)) are multiplexed on the same PCI pins. A bus transaction starts with the address phase followed by one or more data phases. As described in the following table, certain control signals (FRAME L, TRDY L, and IRDY L) determine the bus phase. AD(7:0) is the least significant byte, and AD(31:24) is the most significant byte.
- Groups of pins that are grounded, provide power inputs, or are not connected are listed at the end of the table.

**Table 3-2** PCI connector pin assignments

| Pin number | Signal | Description |
|---|---|---|
| A6 | IRQ L | This is a maskable interrupt request resulting from various IRQ sources. Use of interrupts is optional on the PCI bus. |
| A15 | RESET L | This is the reset signal. When it is driven low, all PCI registers, sequences, and signals are reset to a consistent state. If the signals are tristate, this means that they are returned to the tristate (off) condition. |
| A17 | GNT L | This is the grant signal. When a master device requests access to the PCI bus, this signal is asserted to indicate that access has been granted. It is a point-to-point signal, and each master has its own GNT L. |
| A20 | AD(30) | Address/data bit 30. |
| A22 | AD(28) | Address/data bit 28. |
| A23 | AD(26) | Address/data bit 26. |
| A25 | AD(24) | Address/data bit 24. |
| A26 | IDSEL L | This is the initialization device select signal. When memory chips are being initialized by configuration read and write transactions, this signal is used to select individual ICs. |
| A28 | AD(22) | Address/data bit 22. |
| A29 | AD(20) | Address/data bit 20. |
| A31 | AD(18) | Address/data bit 18. |
| A32 | AD(16) | Address/data bit 16. |
| A34 | FRAME L | This is the cycle frame signal. A bus transaction begins when this signal is asserted (driven low) by the master. The first clock cycle during which FRAME L is active is the address phase. This means that the address/data lines, AD(31:00), are carrying the physical address of the location to be accessed. Data phases on the bus are controlled by IRDY L and TRDY L. However, FRAME L remains active throughout the bus transaction. It is driven high (inactive) at the end of the final data phase. |

**Table 3-2** PCI connector pin assignments (continued)

| Pin number | Signal | Description |
|---|---|---|
| A36 | TRDY L | This is the target-ready signal. When it is asserted, it indicates that the selected device, in this context the 12″ or 7″ card, is able to complete the current data phase of the bus transaction. If a write cycle is in progress, TRDY L going active (low) indicates that valid data is present on the AD(31:0) lines. When TRDY L goes active during a read cycle, it indicates the master is prepared to accept data. A data phase is completed during any clock cycle where both TRDY L and IRDY L are active. If one signal is active before the other, wait cycles are inserted until both signals are active at the same time. |
| A38 | STOP L | When this stop signal is asserted, it means that the target device is requesting the bus master to stop the current transaction. |
| A43 | PAR | This is the parity bit for AD(31:0). All devices on the PCI bus, in this context the 12″ and 7″ cards, must generate parity. Parity is even for these devices. PAR is stable one clock after the address phase. During write transactions, this is one clock after IRDY L is asserted, and during read transactions, it is one clock after TRDY L is asserted. PAR remains valid for one clock cycle after the completion of the last data phase. The master drives PAR during address and write data phases, and the target drives PAR during read data phases. |
| A44 | AD(15) | Address/data bit 15. |
| A46 | AD(13) | Address/data bit 13. |
| A47 | AD(11) | Address/data bit 11. |
| A49 | AD(9) | Address/data bit 9. |
| A50, A51 | — | No pins with these numbers. |
| A52 | CBE(0) L | Bus command and byte enable signals (C or BE) are multiplexed on this pin. Pins B26, B33, and B44 carry the other three signals that make up this group. During the address phase of a bus transaction, CBE(3:0) L define the bus commands. Table 3-3 on page 44 shows how these signals are encoded during a bus transaction.<br><br>During the data phase, the signal(s) enable the selected byte. For example, when CBE(0) L is driven low, byte 0 is enabled. The byte enable signals remain active throughout the data phase. |

**Table 3-2** PCI connector pin assignments (continued)

| Pin number | Signal | Description |
|---|---|---|
| A54 | AD(6) | Address/data bit 6. |
| A55 | AD(4) | Address/data bit 4. |
| A57 | AD(2) | Address/data bit 2. |
| A58 | AD(0) | Address/data bit 0. |
| B16 | PCI CLK | This PCI clock input supplies timing for all transactions on the PCI bus. All other PCI signals, with the exception of RESET L and IRQ L, are sampled on the rising edge of PCI CLK. The PCI bus typically operates at 33 MHz, with a minimum frequency of 0 Hz. |
| B18 | REQ L | The master device asserts this signal to request access to the PCI bus. It is a point-to-point signal, and each master has its own REQ L. |
| B20 | AD(31) | Address/data bit 31. |
| B21 | AD(29) | Address/data bit 29. |
| B23 | AD(27) | Address/data bit 27. |
| B24 | AD(25) | Address/data bit 25. |
| B26 | CBE(3) L | Bus command and byte enable signal 3. Refer to the description for pin A52 for further information. |
| B27 | AD(23) | Address/data bit 23. |
| B29 | AD(21) | Address/data bit 21. |
| B30 | AD(19) | Address/data bit 19. |
| B32 | AD(17) | Address/data bit 17. |
| B33 | CBE(2) L | Bus command and byte enable signal 2. Refer to the description for pin A52 for further information. |
| B35 | IRDY L | This is the initiator-ready signal. When it is asserted, it indicates that the bus master, in this context the Macintosh computer, is able to complete the current data phase of the bus transaction. If a write cycle is in progress, TRDY L going active (low) indicates that valid data is present on the AD(31:0) lines. When TRDY L goes active during a read cycle, it indicates the master is prepared to accept data. A data phase is completed during any clock cycle where both TRDY L and IRDY L are active. If one signal is active before the other, wait cycles are inserted until both signals are active at the same time. |

**Table 3-2** PCI connector pin assignments (continued)

| Pin number | Signal | Description |
|---|---|---|
| B37 | DEVSEL L | This is the device select signal. When a device has decoded the device address and recognizes itself as the target of the current access, it outputs this signal. The signal is input to the host, indicating that a device on the bus has been selected. |
| B39 | LOCK L | This is the lock signal and, when it is asserted, it indicates that the operation may require multiple transactions to complete. When LOCK L is asserted, it locks the address that is currently being accessed, but transactions that are not exclusive may proceed to an address that is not currently locked. When a master device is granted access to the PCI bus, it is not guaranteed control of the LOCK L signal. Different agents may use the PCI bus, but only one master has ownership of LOCK L. If a device implements executable memory, it must also implement LOCK L and guarantee exclusive access to the memory block. The target for this sort of access must guarantee exclusive access to a minimum of 16 aligned bytes. Host bridges must also implement LOCK L. |
| B44 | CBE(1) L | Bus command and byte enable signal 1. Refer to the description for pin A52 for further information. |
| B45 | AD(14) | Address/data bit 14. |
| B47 | AD(12) | Address/data bit 12. |
| B48 | AD(10) | Address/data bit 10. |
| B50, B51 | — | No pins with these numbers. |
| B52 | AD(8) | Address/data bit 8. |
| B53 | AD(7) | Address/data bit 7. |
| B55 | AD(5) | Address/data bit 5. |
| B56 | AD(3) | Address/data bit 3. |
| B58 | AD(1) | Address/data bit 1. |
| A12, A13, A18, A24, A30, A35, A37, A42, A48, A56, B3, B12, B13, B15, B17, B22, B28, B34, B38, B46, B49, B57 | GND | On the 12″ and 7″ cards, these pins are connected to ground. They are also generally connected to ground on the PCI bus. |

**Table 3-2** PCI connector pin assignments (continued)

| Pin number | Signal | Description |
|---|---|---|
| A1 | GND (TRST L) | On the 12″ and 7″ cards, this pin is connected to ground. On the PCI bus, it may be used for Test Reset, which asynchronously initializes the TAP (Test Access Port) controller. |
| B2 | GND (TCK) | On the 12″ and 7″ cards, this pin is connected to ground. On the PCI bus, it may be used for Test Clock, which clocks state information and test data into and out of the PCI card during TAP operations. |
| B11 | GND (PRSNT2 L) | On the 12″ and 7″ cards, this pin is connected to ground. On the PCI bus, it may be used for Present 2, which is used in conjunction with Present 1 (pin B9) to indicate the presence of a PCI card in the PCI slot and to indicate the power requirements for the card. In the case of the 12″ and 7″ cards cards, Present 1 is open and Present 2 is grounded, indicating the presence of a PCI card with a 15 W maximum power requirement. |
| A5, A8, A10, A16, A59, A61, A62, B5, B6, B19, B59, B61, B62 | +5 V | On the 12″ and 7″ cards, these pins are connected to +5 V. They are also generally connected to +5 V on the PCI bus. |
| A3 | +5 V (TMS) | On the 12″ and 7″ cards, this pin is connected to +5 V. On the PCI bus, it is used for Test Mode Select, which controls the state of the TAP controller. |
| A4 | +5 V (TDI) | On the 12″ card, this pin is connected to +5 V. On the 7″ card it is tied to B4. On the PCI bus, it is used for Test Data Input, which serially shifts test data and test instructions into the PCI card during TAP operations. |
| A2 | +12 V | On the 12″ and 7″ cards, this pin is connected to the +12 V power supply. It is also generally connected to +12 V on the PCI bus. |
| B1 | –12 V | On the 12″ and 7″ cards, this pin is connected to the –12 V power supply. It is also generally connected to –12 V on the PCI bus. |
| A7 | Not connected (INTC L) | On the 12″ and 7″ cards, this pin is not connected. On the PCI bus, it is generally used for Interrupt C, which requests an interrupt and is only used on a multifunction device. |
| A9, A11, A14, A19, B10, B14 | Not connected (reserved) | On the 12″ and 7″ cards, these pins are not connected. On the PCI bus, they are reserved for future use. |

**Table 3-2** PCI connector pin assignments (continued)

| Pin number | Signal | Description |
|---|---|---|
| A21, A27, A33, A39, A45, A53, B25, B31, B36, B41, B43, B54 | Not connected (+3.3 V) | On the 12" and 7" cards, these pins are not connected. On the PCI bus, they are connected to the +3.3 V power supply. |
| A40 | Not connected (SDONE) | On the 12" and 7" cards, this pin is not connected. On the PCI bus, it is used for Snoop Done, which indicates the status of the snoop operation for the current cache access. |
| A41 | Not connected (SBO L) | On the 12" and 7" cards, this pin is not connected. On the PCI bus, it is used for Snoop Backoff, which indicates a hit to a modified cache line. |
| A60 | Not connected (REQ64 L) | On the 12" and 7" cards, this pin is not connected. On the PCI bus, it is used for Request 64-bit Transfer, which indicates the bus master's desire to transfer data in 64-bit blocks. |
| B4 | Not connected (TDO) | On the 12" and 7" cards, this pin is not connected. On the PCI bus, it is used for Test Data Output, which serially shifts test data and test instructions out of the PCI card during TAP operations. |
| B7 | Not connected (INTB L) | On the 12" and 7" cards, this pin is not connected. On the PCI bus, it is used for Interrupt B, which requests an interrupt and is only used on a multifunction device. |
| B8 | Not connected (INTD L) | On the 12" and 7" cards, this pin is not connected. On the PCI bus, it is used for Interrupt D, which requests an interrupt and is only used on a multifunction device. |
| B9 | Not connected (PRSNT1 L) | On the 12" and 7" cards, this pin is not connected. On the PCI bus, it is used for Present 1, which, in conjunction with Present 2 (pin B11), indicates the presence of a PCI card in the PCI slot and indicates the power requirements for the card. Present 2 is grounded and Present 1 is open, indicating the presence of a PCI card with a 15 W maximum power requirement. |

**Table 3-2** PCI connector pin assignments (continued)

| Pin number | Signal | Description |
|---|---|---|
| B40 | Not connected (PERR L) | On the 12" and 7" cards, this pin is not connected. On the PCI bus, it is used for Parity Error, which reports data parity errors during all PCI transactions except special cycles. |
| B42 | Not connected (SERR L) | On the 12" and 7" cards, this pin is not connected. On the PCI bus, it is used for System Error, which reports address parity errors, data parity errors during special cycles, and any catastrophic system error. |
| B60 | Not connected (ACK64 L) | On the 12" and 7" cards, this pin is not connected. On the PCI bus, Acknowledge 64-bit Transfer indicates that the target device is able to transfer data in 64-bit blocks. |

Table 3-3 shows how the bus command and byte enable (CBE(3:0)) signals are encoded.

**Table 3-3** CBE(3:0) L encoding

| CBE setting | | | | Command |
|---|---|---|---|---|
| Bit 3 | Bit 2 | Bit 1 | Bit 0 | |
| 0 | 0 | 0 | 0 | Interrupt acknowledge |
| 0 | 0 | 0 | 1 | Special cycle |
| 0 | 0 | 1 | 0 | I/O read |
| 0 | 0 | 1 | 1 | I/O write |
| 0 | 1 | 0 | 0 | Reserved |
| 0 | 1 | 0 | 1 | Reserved |
| 0 | 1 | 1 | 0 | Memory read |
| 0 | 1 | 1 | 1 | Memory write |
| 1 | 0 | 0 | 0 | Reserved |
| 1 | 0 | 0 | 1 | Reserved |
| 1 | 0 | 1 | 0 | Configuration read (used at initialization) |
| 1 | 0 | 1 | 1 | Configuration write (used at initialization) |
| 1 | 1 | 0 | 0 | Memory read multiple |
| 1 | 1 | 0 | 1 | Dual address cycle |
| 1 | 1 | 1 | 0 | Memory read line |
| 1 | 1 | 1 | 1 | Memory write and invalidate |

# DB26 Connector

The DB26 connector allows you to connect a monitor to the PC system. It is used with CPUs that do not have a GIMO connector and therefore do not support the Berlin adapter card. The DB26 connector provides a loop-back connection to the video output connector on the CPU main logic board. The loop back is implemented by a split video cable that also provides a connector for an external monitor. If the CPU does have a Berlin adapter card and GIMO connector, you can use the DB26 connector to hook up a second monitor.

Figure 3-1 on page 34 and Figure 3-2 on page 35 show the location of this connector on the 12" and 7" cards, respectively. “Loop-Back Video Support” on page 30 provides more information about this type of interface.

Table 3-4 lists the pin assignments and signal descriptions for the DB26 connector.

**Table 3-4** DB26 connector pin assignments

| Pin number | Signal | Description |
|---|---|---|
| 1 | MAC HSYNC | Macintosh horizontal synchronize |
| 2, 4, 6, 8 | GND | Logic ground |
| 3 | MAC BLUE | Macintosh blue signal |
| 5 | SENSE0 | Sense line 0 |
| 7 | BLUE OUT | Blue output from PCI card |
| 9 | HSYNC OUT | Horizontal synchronize output from PCI card |
| 10 | MAC CSYNC | Macintosh contrast synchronize |
| 11, 13, 15, 17, 21, 24 | GND | Chassis ground |
| 12 | MAC GREEN | Macintosh green signal |
| 14 | SENSE1 | Sense line 1 |
| 16 | GREEN OUT | Green output from PCI card |
| 18 | CSYNC OUT | Contrast synchronize output from PCI card |
| 19 | MAC VSYNC | Macintosh vertical synchronize |
| 20 | MAC RED | Macintosh red signal |
| 22 | CABLE DET L | Cable detect low |
| 23 | SENSE2 | Sense line 2 |
| 25 | RED OUT | Red output from PCI card |
| 26 | VSYNC OUT | Vertical synchronize output from PCI card |

# DB15 Connector (Game Port)

The DB15 connector allows you to connect MIDI devices to the card. It also allows you to connect a PC-compatible joystick, which can be used as a game controller. See Figure 3-1 on page 34 for the location of the connector on the 12" card, and Figure 3-2 on page 35 for the location of the connector on the 7″ card.The joystick can be used only with programs running on the PC side of the system. You should refer to the user's guide supplied with your Macintosh computer for information on installing a joystick or MIDI device. Table 3-5 lists the pin assignments for the connector.

**Table 3-5** DB15 connector pin assignments

| Pin number | Signal | Description |
|---|---|---|
| 1, 8, 9 | +5 V | +5 V power input |
| 2 | JOYF0 O | Joystick F0 output |
| 3 | JRC0 O | A x-axis control |
| 4, 5 | GND | Chassis ground |
| 6 | JRC1 O | A y-axis control |
| 7 | JOYF1 O | Joystick F1 output |
| 10 | JOYF2 O | Joystick F2 output |
| 11 | JRC2 O | B x-axis control |
| 12 | MIDI OUT O | MIDI output |
| 13 | JRC3 O | B y-axis control |
| 14 | JOYF3 O | Joystick F3 output |
| 15 | MIDI IN O | MIDI input output |

# GIMO Connector

The GIMO connector is a 16-pin connector on the PCI card that enables you to connect a monitor to the 12″ or 7″ card, provided that the CPU in which the card is installed has a GIMO connector and a Berlin adapter card. The video connection is made by means of a ribbon cable that plugs into the GIMO connector on the card and then plugs into the Berlin adapter card in the CPU. This cable also carries the audio input and output signals when the card is installed in systems that do not have discrete audio cables.

Figure 3-1 on page 34 and Figure 3-2 on page 35 show the location of this connector on the 12" and 7" cards, respectively. “GIMO Support for Video Output” on page 28 provides more information about this type of interface.

Table 3-6 shows the pin assignments for the GIMO 16-pin connector, Molex part number 87256-1641, or AMP part number 104338-3. The connector is at location J11 on the 12" card and J4 on the 7” card. The pin assignments are the same for each card.

**Table 3-6** GIMO connector pin assignments

| Pin number | Signal | Description |
|---|---|---|
| 1, 9, 13 | GND | Ground |
| 2 | IREF | Refresh |
| 3 | VSYNC | Vertical synchronize |
| 4 | HSYNC | Horizontal synchronize |
| 5 | RGB SEL(0) | RGB select 0 |
| 6 | VGA CSYNC | VGA contrast synchronize |
| 7 | +5 V | +5 V power supply |
| 8 | GIMO BLUE | GIMO blue |
| 10 | GIMO GREEN | GIMO green |
| 11 | GIMO DET L | GIMO detect |
| 12 | GIMO RED | GIMO red |
| 14 | GIMO OUTL | GIMO sound output left |
| 15 | GIMO GND | GIMO ground |
| 16 | GIMO OUTR | GIMO sound output right |

# Audio Connectors

In most CPUs, audio input and output are provided by two identical four-wire ribbon cables. The audio input cable routes sound inputs to the 12" or 7" card from a sound source such as a CD player. The cards have a Sound Blaster–compatible audio system, and the audio output cable routes sound to the Macintosh sound system, speakers, or an external audio jack. Figure 2-9 on page 30 shows how the audio connections are made.

**Note**
In those CPUs that do not have discrete audio connectors, audio I/O is implemented by means of a ribbon cable and the GIMO connector. ◆

lists the pin assignments and signal descriptions for the four-pin audio connectors, JST part number B-4B-PH-K. The connector at location J5 is for audio input, and the connector at J6 for audio output.

**Table 3-7** Audio connector pin assignments

| Pin number | Signal | Connector location | Description |
|---|---|---|---|
| 1 | n.c. | J5 | Not connected |
| 2 | CD INR | J5 | CD input right channel |
| 3 | CD GND | J5 | Ground |
| 4 | CD INL | J5 | CD input left channel |
| 1 | n.c. | J6 | Not connected |
| 2 | SND OUTR | J6 | Sound output right channel |
| 3 | SND GND | J6 | Ground |
| 4 | SND OUTL | J6 | Sound output left channel |

# DIMM Connector

If you want to increase memory capacity, you can plug a DIMM into the DIMM slot provided on the 12" card, or remove the 8 MB DIMM from the 7” card and replace it with a DIMM of higher capacity. See “Memory Capabilities” on page 5 for further information about DIMM configurations.

The DIMM connector is a 160-pin connector, Berg part number 95566-11102. Table 3-8 lists the pin assignments for this connector. The pins that are connected to ground, the +5 V power supply, and pins that are not connected are listed at the end of the table.

**Table 3-8** DIMM connector pin assignments

| Pin number | 12” card | | 7” card | |
|---|---|---|---|---|
| | Signal | Description | Signal | Description |
| 2 | MD(0) | Data bit 0 | D(0) | Data bit 0 |
| 3 | MD(1) | Data bit 1 | D(1) | Data bit 1 |
| 4 | MD(2) | Data bit 2 | D(2) | Data bit 2 |
| 5 | MD(3) | Data bit 3 | D(3) | Data bit 3 |
| 7 | MD(4) | Data bit 4 | D(4) | Data bit 4 |

**Table 3-8** DIMM connector pin assignments (continued)

| Pin number | 12” card | | 7” card | |
|---|---|---|---|---|
| | Signal | Description | Signal | Description |
| 8 | MD(5) | Data bit 5 | D(5) | Data bit 5 |
| 9 | MD(6) | Data bit 6 | D(6) | Data bit 6 |
| 10 | MD(7) | Data bit 7 | D(7) | Data bit 7 |
| 13 | MD(8) | Data bit 8 | D(8) | Data bit 8 |
| 14 | MD(9) | Data bit 9 | D(9) | Data bit 9 |
| 15 | MD(10) | Data bit 10 | D(10) | Data bit 10 |
| 16 | MD(11) | Data bit 11 | D(11) | Data bit 11 |
| 17 | MD(12) | Data bit 12 | D(12) | Data bit 12 |
| 19 | MD(13) | Data bit 13 | D(13) | Data bit 13 |
| 20 | MD(14) | Data bit 14 | D(14) | Data bit 14 |
| 21 | MD(15) | Data bit 15 | D(15) | Data bit 15 |
| 27 | DWE L | Write enable | BMWE | Write enable |
| 28 | CAS(0) | Column address strobe byte 0 | BCAS(0) | Column address strobe byte 0 |
| 29 | CAS(2) | Column address strobe byte 2 | BCAS(2) | Column address strobe byte 2 |
| 30 | RAS(2) | Row address strobe byte 2 | DRAS(1) | Row address strobe byte 1 |
| 33 | MA(0) | Address bit 0 | BMA(11) | Address bit 11 |
| 34 | MA(2) | Address bit 2 | BMA(9) | Address bit 9 |
| 35 | MA(4) | Address bit 4 | BMA(7) | Address bit 7 |
| 36 | MA(6) | Address bit 6 | BMA(5) | Address bit 5 |
| 37 | MA(8) | Address bit 8 | BMA(3) | Address bit 3 |
| 38 | MA(10) | Address bit 10 | BMA(1) | Address bit 1 |
| 45 | RAS(2) | Row address strobe byte 2 | DRAS(2) | Row address strobe byte 2 |
| 46 | CAS(4) | Column address strobe byte 4 | BCAS(0) | Column address strobe byte 0 |
| 47 | CAS(6) | Column address strobe byte 6 | BCAS(2) | Column address strobe byte 2 |
| 48 | DWE L | Write enable | BMWE | Write enable |
| 52 | MD(16) | Data bit 16 | D(16) | Data bit 16 |
| 53 | MD(17) | Data bit 17 | D(17) | Data bit 17 |

**Table 3-8** DIMM connector pin assignments (continued)

| Pin number | 12” card | | 7” card | |
|---|---|---|---|---|
| | **Signal** | **Description** | **Signal** | **Description** |
| 55 | MD(18) | Data bit 18 | D(18) | Data bit 18 |
| 56 | MD(19) | Data bit 19 | D(19) | Data bit 19 |
| 57 | MD(20) | Data bit 20 | D(20) | Data bit 20 |
| 58 | MD(21) | Data bit 21 | D(21) | Data bit 21 |
| 60 | MD(22) | Data bit 22 | D(22) | Data bit 22 |
| 65 | MD(23) | Data bit 23 | D(23) | Data bit 23 |
| 67 | MD(24) | Data bit 24 | D(24) | Data bit 24 |
| 69 | MD(25) | Data bit 25 | D(25) | Data bit 25 |
| 70 | MD(26) | Data bit 26 | D(26) | Data bit 26 |
| 71 | MD(27) | Data bit 27 | D(27) | Data bit 27 |
| 72 | MD(28) | Data bit 28 | D(28) | Data bit 28 |
| 74 | MD(29) | Data bit 29 | D(29) | Data bit 29 |
| 75 | MD(30) | Data bit 30 | D(30) | Data bit 30 |
| 76 | MD(31) | Data bit 31 | D(31) | Data bit 31 |
| 86 | MD(32) | Data bit 32 | D(0) | Data bit 0 |
| 87 | MD(33) | Data bit 33 | D(1) | Data bit 1 |
| 88 | MD(34) | Data bit 34 | D(2) | Data bit 2 |
| 89 | MD(35) | Data bit 35 | D(3) | Data bit 3 |
| 91 | MD(36) | Data bit 36 | D(4) | Data bit 4 |
| 92 | MD(37) | Data bit 37 | D(5) | Data bit 5 |
| 93 | MD(38) | Data bit 38 | D(6) | Data bit 6 |
| 94 | MD(39) | Data bit 39 | D(7) | Data bit 7 |
| 97 | MD(40) | Data bit 40 | D(8) | Data bit 8 |
| 98 | MD(41) | Data bit 41 | D(9) | Data bit 9 |
| 99 | MD(42) | Data bit 42 | D(10) | Data bit 10 |
| 100 | MD(43) | Data bit 43 | D(11) | Data bit 11 |
| 101 | MD(44) | Data bit 44 | D(12) | Data bit 12 |
| 103 | MD(45) | Data bit 45 | D(13) | Data bit 13 |
| 104 | MD(46) | Data bit 46 | D(14) | Data bit 14 |
| 105 | MD(47) | Data bit 47 | D(15) | Data bit 15 |

**Table 3-8** DIMM connector pin assignments (continued)

| Pin number | 12” card | | 7” card | |
|---|---|---|---|---|
| | **Signal** | **Description** | **Signal** | **Description** |
| 112 | CAS(1) | Column address strobe byte 1 | BCAS(1) | Column address strobe byte 1 |
| 113 | CAS(3) | Column address strobe byte 3 | BCAS(3) | Column address strobe byte 3 |
| 114 | RAS(3) | Row address strobe byte 3 | DRAS(3) | Row address strobe byte 3 |
| 117 | MA(1) | Address bit 1 | BMA(10) | Address bit 10 |
| 118 | MA(3) | Address bit 3 | BMA(8) | Address bit 8 |
| 119 | MA(5) | Address bit 5 | BMA(6) | Address bit 6 |
| 120 | MA(7) | Address bit 7 | BMA(4) | Address bit 4 |
| 121 | MA(9) | Address bit 9 | BMA(2) | Address bit 2 |
| 122 | MA(11) | Address bit 11 | BMA(0) | Address bit 0 |
| 124 | DIMM DET | Detect DIMM | DIMM DET | Detect DIMM |
| 126 | MA(0) | Address bit 0 | BMA(11) | Address bit 11 |
| 129 | RAS(3) | Row address strobe byte 3 | DRAS(1) | Row address strobe byte 1 |
| 130 | CAS(5) | Column address strobe byte 5 | CAS(1) | Column address strobe byte 1 |
| 131 | CAS(7) | Column address strobe byte 7 | n.c. | Not connected |
| 136 | MD(48) | Data bit 48 | D(16) | Data bit 16 |
| 137 | MD(49) | Data bit 49 | D(17) | Data bit 17 |
| 139 | MD(50) | Data bit 50 | D(18) | Data bit 18 |
| 140 | MD(51) | Data bit 51 | D(19) | Data bit 19 |
| 141 | MD(52) | Data bit 52 | D(20) | Data bit 20 |
| 142 | MD(53) | Data bit 53 | D(21) | Data bit 21 |
| 144 | MD(54) | Data bit 54 | D(22) | Data bit 22 |
| 149 | MD(55) | Data bit 55 | D(23) | Data bit 23 |
| 151 | MD(56) | Data bit 56 | D(24) | Data bit 24 |
| 153 | MD(57) | Data bit 57 | D(25) | Data bit 25 |
| 154 | MD(58) | Data bit 58 | D(26) | Data bit 26 |
| 155 | MD(59) | Data bit 59 | D(27) | Data bit 27 |

**Table 3-8** DIMM connector pin assignments (continued)

| Pin number | 12” card | | 7” card | |
|---|---|---|---|---|
| | Signal | Description | Signal | Description |
| 156 | MD(60) | Data bit 60 | D(28) | Data bit 28 |
| 158 | MD(61) | Data bit 61 | D(29) | Data bit 29 |
| 159 | MD(62) | Data bit 62 | D(30) | Data bit 30 |
| 160 | MD(63) | Data bit 63 | D(31) | Data bit 31 |
| 1, 12, 23, 31, 32, 43, 44, 54, 68, 78, 85, 96, 107, 116, 127, 138, 152, 162 | GND | Ground | GND | Ground |
| 6, 18, 26, 40, 49, 59, 73, 84, 90, 102, 110, 133, 143, 157, 168 | +5 V | +5 V power supply | +5 V | +5 V power supply |
| 11, 22, 24, 25, 39, 41, 42, 50, 51, 61, 62, 63, 64, 66, 77, 79, 80, 81, 82, 83, 95, 106, 108, 109, 111, 115, 123, 125, 128, 132, 134, 135, 145, 146, 147, 148, 150, 161, 163, 164, 165, 166, 167 | n.c. | Not connected | n.c. | Not connected |

# XD Connector

A 50-pin connector on the cards provides limited unbuffered access to the ISA bus. This enables you to create a parallel port for third-party dongles, otherwise known as hardware keys. You do this by connecting an expansion card to the XD connector and

then connecting a hardware key to the expansion card. The XD connector may also be used for a sound expansion card.

The XD connector is a 50-pin connector, JAE part number KX15-50K3E9. Table 3-9 shows the pin assignments for this connector.

**Table 3-9** XD connector pin assignments

| Pin number | Signal | Description |
|---|---|---|
| 1 | ISA IRQ(3) | ISA interrupt request 3 |
| 2 | SA(10) | S address bit 10 |
| 3 | ISA IRQ(4) | ISA interrupt request 4 |
| 4 | SA(9) | S address bit 9 |
| 5 | ISA IRQ(7) | ISA interrupt request 7 |
| 6 | SMEMW | S memory write |
| 7 | IOCHK | I/O check |
| 8 | IRQ(9) | Interrupt request 9 |
| 9 | DREQ(6) | DMA request 6 |
| 10 | MINUS 12 V | –12 V power supply |
| 11 | DACK | DMA acknowledge |
| 12 | SMEMR | S memory read |
| 13 | REFRESH | Refresh |
| 14 | +12 V | +12 V power supply |
| 15 | IOR | I/O read |
| 16 | AEN | Address enable |
| 17 | DREQ(3) | DMA request 3 |
| 18 | OSC | Oscillator |
| 19 | RESETDRV | Reset drive |
| 20 | +5 V | +5 V power supply |
| 21 | SA(7) | S address bit 7 |
| 22 | XD(7) | Buffered data bit 7 |
| 23 | SA(4) | S address bit 4 |
| 24 | 0WS | 0 write strobe |
| 25, 29, 45 | GND | Ground |
| 26 | IOW | I/O write |
| 27 | ISA Detect | Detect external device on ISA bus |

**Table 3-9** XD connector pin assignments (continued)

| Pin number | Signal | Description |
|---|---|---|
| 28 | SA(1) | S address bit 1 |
| 30 | SA(0) | S address bit 0 |
| 31 | DACK(3) | DMA acknowledge 3 |
| 32 | TC | Terminate count |
| 33 | BALE | Bus address latch enable |
| 34 | SA(2) | S address bit 2 |
| 35 | IOCHRDY | I/O check ready |
| 36 | SA(8) | S address bit 8 |
| 37 | SA(5) | S address bit 5 |
| 38 | IRQ(10) | Interrupt request 10 |
| 39 | ATCLK | AT clock |
| 40 | XD(0) | Buffered data bit 0 |
| 41 | SA(3) | S address bit 3 |
| 42 | +5 V | +5 V power supply |
| 43 | SA(6) | S address bit 6 |
| 44 | XD(1) | Buffered data bit 1 |
| 46 | XD(2) | Buffered data bit 2 |
| 47 | XD(5) | Buffered data bit 5 |
| 48 | XD(3) | Buffered data bit 3 |
| 49 | XD(6) | Buffered data bit 6 |
| 50 | XD(4) | Buffered data bit 4 |

CHAPTER 4

# Software Support

The interface driver for the 12″ and 7″ PC Compatibility Cards controls communication between them and the Mac OS (Macintosh Operating System). Applications running on the Mac OS can use the driver to configure and control the cards. Applications running in both environments can use the driver to exchange messages.

This chapter describes the routines that allow you to initialize the driver, configure the PC system, make control and status calls, and pass messages between the Mac OS and the PC driver. You should also refer to *Inside Macintosh: Devices* for information about opening and closing the interface driver.

**Note**
When installed in a Macintosh computer, the 12″ and 7″ cards implement PC functions, that is, the functions of a Pentium-based or 5x86-based computer. For the sake of brevity, these functions are referred to in this chapter as the PC system. Figure 1-1 on page 3 summarizes the PC system functions implemented by the cards. ◆

# Initializing the Interface Driver

Before you can use the interface driver, your application must initialize it by calling an open routine. When initialization is complete, the application calls a close routine. You can open and close the driver only from the Mac OS. The name of the driver is `"\p.Symbiosis"`.

## Opening the Driver

The driver must be open before your application can communicate with it. As described in *Inside Macintosh: Devices*, there are various ways of opening drivers. The two principal ways are `OpenDriver` and `PBOpen`. In this case, `PBOpen` is the preferred way.

When you call the `PBOpen` routine, it opens the driver specified by the name parameter, `"\p.Symbiosis"`. The routine allocates and initializes the driver's memory, installs the interrupt handler, and makes patches to the system as needed by the driver. The `PBOpen` routine initializes all devices to the null device and puts the PC system into a reset state. The `PBOpen` routine fails if the driver cannot allocate enough memory or if it cannot find the 12″ or the 7″ card.

## Closing the Driver

When you have finished communicating with the driver, you can close it using either the `CloseDriver` or `PBClose` routine. It is preferable to use `PBClose`. When you call the `PBClose` routine, it releases all memory allocated to the interface driver, removes the driver's interrupt handler, removes any patches installed by the open routine, and puts the PC system into a reset state.

# Configuring the PC System

An application running on the Mac OS can use the interface driver to configure the PC system. You can also use the driver to perform the following operations:

- configure the disk drives available to the cards
- set and read the status of the network driver
- configure the communications port
- configure the parallel port
- define the key combination that switches focus between the PC system and the Macintosh

This section describes the routines that perform these configuration tasks. Each routine lists the parameter block settings associated with that particular routine.

## rsSetDriveConfig

You can use the `rsSetDriveConfig` control routine to configure each of the PC system's fixed drives (A:, B:, C:, and D:). These drives can be configured as a floppy drive, Macintosh file, SCSI partition, or have no corresponding driver.

The status code for this routine is `rsSetDriveConfig = 500`.

The application running on the Macintosh computer should call `rsSetDriveConfig` at least once before starting the PC system. If you want to change the drive configuration after the PC system has been started, you can also call this routine. When you do this, the drive configuration does not change until the PC system is restarted.

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsSetDriveConfig`. |
| —> | `csParam+0` | `long` | Pointer to `RSFixedDriveConfig`. |

The RSFixedDriveConfig data structure pointed to by csParam is shown below.

```
        typedef struct{
short   type;             // Type of device this drive is
short   vRefNum;          // Volume refNum or SCSI ID
long    dirID;            // Directory ID or starting sector
                          number on SCSI drive
long    fileNamePtr;      // Filename or number of sectors on
                          SCSI drive
        } RSFixedDriveConfig[4], *RSFixedDriveConfigPtr;
```

**Field descriptions**

`RSFixedDriveConfig[0]`
Contains the configuration for drive A:.

`RSFixedDriveConfig[1]`
Contains the configuration for drive B:.

`RSFixedDriveConfig[2]`
Contains the configuration for drive C:.

`RSFixedDriveConfig[3]`
Contains the configuration for drive D:.

`type` field — Specifies what type of drive is configured: `rsFloppyDrive`, `rsFileDrive`, `rsPartitionDrive`, or `rsNULLDrive`.

If the value is `rsNULLDrive = 0`, the corresponding drive does not exist to the PC system and no other fields need to be filled in.

If the value is `rsFloppyDrive = 1`, the corresponding drive is an Apple SuperDrive connected to one of the Macintosh computer's floppy drive connectors.

If the value is `rsFileDrive = 2`, the corresponding drive is connected to a Macintosh file system file. The `vRefNum` field contains the volume the file is on, `dirID` contains the directory ID of the file, and `fileNamePtr` contains a pointer to the filename. The driver opens and closes the file as needed.

If the value is `rsPartitionDrive = 3`, the corresponding drive is connected to a SCSI drive partition. The `vRefNum` field contains the SCSI ID, `dirID` contains the starting sector number of the partition, and `fileNamePtr` contains the number of sectors in the partition.

If the value is set to `rsIgnore`, the configuration of the corresponding drive is not changed.

## rsGetNetDriveConfig

You can use the `rsGetNetDriveConfig` control routine to obtain configuration data about the drives. This routine returns a pointer to an array of 22 `RSNetDriveConfig` data structures, one for each drive letter from E through Z.

The status code for this routine is `rsGetNetDriveConfig = 650`.

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsGetNetDriveConfig.` |
| <— | `csParam+0` | `long` | Pointer to `RSNetDriveConfig.` |

The `RSNetDriveConfig` data structure pointed to by `csParam` is shown below.

```
        typedef struct{
char    status;     // 0 = unused, -1 = in use, 1 = cannot be used
char    changed;    // Used by the driver, do not use
short   vRefNum;    // Reference number of volume containing shared drive
long    dirID;      // Directory ID
        } RSNetDriveConfig[26], *RSNetDriveConfigPtr;
```

The `RSNetDriveConfig` data structure contains the current configuration for folder sharing for each PC system drive letter. If the PC system has its `LASTDRIVE` parameter set to less than Z or if other block device drivers are loaded on the PC system, not all drive letters will be available. The data structures for drives that are not available have their status parameters set to 1 by the interface driver.

The caller can use the returned pointer to modify an entry in the `RSNetDriveConfig` data structure and then call the `rsSetNetDriveConfig` control call.

## rsSetNetDriveConfig

You can use the `rsSetNetDriveConfig` control routine to establish links between Macintosh directories and PC system drive letters. The call simply notifies the interface driver that an entry in the `RSNetDriveConfig` data structure has been modified.

The control code for this routine is `rsSetNetDriveConfig = 600`.

#### Parameter block

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsSetNetDriveConfig`. |
| —> | `csParam+0` | `long` | Entry number of `RSNetDriveConfig (0=E)`. |

## rsSetComPortConfig

You can use the `rsSetComPortConfig` control routine to set the configurations of the two PC system communications ports, COM1 and COM2. Each communications port can have a virtual connection to either the modem port, the printer port, a communication toolbox port, a spool file, or the null device.

The `rsSetComPortConfig` routine should be called at least once before the PC system is started up. It can also be called after the PC system has been started, in which case, the change in configuration takes effect immediately without a restart.

The control code for this routine is `rsSetComPortConfig = 300`.

The device types for the PC communications ports are

- `rsModemComPort = 1`
- `rsPrinterComPort = 2`
- `rsSpoolComPort = 3`

#### Parameter block

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |

| | | | |
|---|---|---|---|
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsSetComPortConfig.` |
| —> | `csParam+0` | `long` | Pointer to `RSComConfig.` |

A pointer to an `RSComConfig` data structure is passed in the `csParam` field.

```
typedef struct{
short   type;          // Port type (rsModemComPort,
                       rsPrinterComPort, etc.)
short   vRefNum;       // Volume reference number for serial
                       spool file
long    dirID;         // Directory ID
long    fileNamePtr    // Pointer to the filename
        ;
        } RSComConfig[2], *RSComConfigPtr;
```

#### Field descriptions

`RSComFig[0]` Contains the configuration for COM1.

`RSComConfig[1]` Contains the configuration for COM2.

`type` field Specifies what type of connection to make: `rsNULLComPort`, `rsModemComPort`, `rsPrinterComPort`, `rsSpoolComPort`, `rsComToolBoxComPort`, or `rsIgnore`.

`vRefNum` parameter
The value of this parameter is the volume reference number, `dirID` is the directory ID, `type` is the port type (`ModemComPort`, and so on), and `fileNamePtr` is the pointer to the name of the spool file.

PC port connected to the null device
Any output from the PC system is ignored.

PC port connected to the modem or printer port
PC system controls the port by means of the UART emulation in hardware on the 12″ or 7″ card. For example, when the PC system sets the baud rate divisor in the UART emulation register, the interface driver intercepts the operation and translates the action to a control call to the driver for the modem or printer port.

PC port connected to a spool file
All output from the PC system is captured and written to the specified file. The driver opens and closes the file as needed.

`type` field is set to `rsIgnore`
The port's configuration does not change.

## rsSetParallelPortConfig

You can use the `rsSetParallelPortConfig` control routine to set the configuration of the parallel port emulation. The parallel port is used for a printer. A pointer to an `RSParallelConfig` data structure is passed in `csParam`.

When a print job has been completed, the driver notifies the application by means of the `rsSetNotificationProc` procedure, defined in "rsSetNotificationProc" on page 70. The driver also notifies the application if it has trouble saving the spool data.

The control call for this routine is `rsSetParallelPortConfig = 400`.

#### Parameter block

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsSetParallelPortConfig`. |
| —> | `csParam+0` | `long` | Pointer to `RSParallelConfig`. |

The `RSParallelConfig` data structure in the `csParam` field is defined below.

```
        typedef struct{
short   eojTimeOut;   // End of job after n seconds of no data
short   vRefNum;      // RefNum of the Macintosh volume the directory is on
long    spoolDirID;   // RefNum for spool directory
        } RSParallelConfig, *RSParallelConfigPtr;
```

#### Field descriptions

| | |
|---|---|
| `eojTimeOut` field | Specifies the number of seconds the parallel port may be inactive before the driver will force an end of job timeout. If this field is set to `0`, the driver does not force the end of job based on time. |
| `vRefNum` field | Contains the reference number of the volume that contains the directory. |
| `spoolDirID` field | The ID of the directory where the spool files will be stored. |

## rsSetDeactivateKey

You can use the `rsSetDeactivateKey` control routine to set the deactivate key along with its modifiers and a user-defined task. When the PC system has control of the

keyboard, the driver monitors the keyboard input data for the deactivate key combination and calls the user-defined task when that key combination occurs.

The status code for this routine is `rsDeactivateKey = 104`.

The user-defined task is called during `NeedTime`, which is the period after the deactivate key and modifiers are pressed. (If a driver has the `dNeedTime` flag set, it gets called in round-robin fashion at System Task time.)

If the user-defined task is null, no task is called. The modifiers are specified as they appear in `KeyMap+6`. The value of the deactivate key is the Macintosh key code of the desired key. `KeyMap` refers to the variable type referred to by the `GetKeys` call. This is documented in *Inside Macintosh: Macintosh Toolbox Essentials*, Chapter 2, "Event Manager."

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsSetDeactivateKey`. |
| —> | `csParam+0` | `long` | Pointer to user-defined task. |
| —> | `csParam+4` | `word` | Modifiers. |
| —> | `csParam+6` | `word` | The deactivate key. |

Upon return, the parameter block is set as follows.

| | | | |
|---|---|---|---|
| <— | `csParam+0` | `long` | Pointer to the previous user-defined task. |
| <— | `csParam+4` | `word` | The previous modifiers. |
| <— | `csParam+6` | `word` | The previous deactivate key. |

# Control and Status Calls

An application running on the Mac OS can use the interface driver to make control and status calls to the PC system. You can use the routines in this section to do the following:

- get the status of the PC system
- enable and disable the PC system's video display
- enable and disable disk mounting on the PC system
- activate and deactivate keyboard operation by the PC system
- activate and deactivate mouse tracking by the PC system

- terminate print spooling from the PC system

In some instances, for example, in the case of bits 0, 1, and 7, it is better to use the Gestalt function. Refer to "Gestalt Selector" on page 78 for further information.

**Note**
Applications can call Gestalt to get information about the operating environment. The Gestalt function then calls other selector functions. ◆

## rsPCStatus

You can use the `rsPCStatus` status routine to get information about the state of the PC hardware. This routine returns the current state of the PC system.

The status code for this routine is `rsPCStatus = 701`.

**Parameter block**
—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsPCStatus`. |
| <— | `csParam+4` | `long` | The status word. |

The status word is a 32-bit word. When the bits are set (1), they indicate different conditions in the PC system, for example, whether it is running, if the screen is enabled, and so forth. Table 4-1 lists the meaning of the bits in the status word.

**Note**
In the case of bits 0, 1, and 7, it is better to use the Gestalt function to get the required information. ◆

**Table 4-1** PC status word

| Bit | Meaning |
|---|---|
| 0 | 1 = PC running (`rsBooted`) |
| 1 | 1 = VGA screen enabled (`rsVGA Enabled`) |
| 2 | 1 = keyboard enabled (`rsKeyboardEnabled`) |
| 3 | 1 = mouse enabled (`rsMouseEnabled`) |

**Table 4-1** PC status word (continued)

| Bit | Meaning |
|---|---|
| 4 | 1 = disk mounting enabled (`rsDiskMountEnabled`) |
| 5 | 1 = shared memory enabled (`rsSharedEnabled`) |
| 6 | 1 = DMA enabled (`rsDMAEnabled`) |
| 7 | 1 = video cable enabled (`rsCableInstalled`) |
| 8 | 1 = modem port is used by COM1 |
| 9 | 1 = printer port is used by COM1 |
| 10 | 1 = modem port is used by COM2 |
| 11 | 1 = printer port is used by COM2 |
| 24–31 | Reserved |

## rsEnableVideo

You can use the `rsEnableVideo` control routine to enable the VGA display output. You use this routine when you have a shared video monitor and want to switch it from the Mac OS to the PC system.

The control call for this routine is `rsEnableVideo = 705`.

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsEnableVideo`. |

## rsDisableVideo

You can use the `rsDisableVideo` control routine to disable the VGA display output when the Macintosh video output is connected to the video connector. If the Macintosh video is not connected, this call does nothing.

The control call for this routine is `rsDisableVideo = 706`.

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsDisableVideo`. |

## rsMountDisks

You can use the `rsMountDisks` control routine to enable PC disks to be mounted and unmounted. After the call has been made, the interface driver monitors all disk-insertion events, looking for PC-formatted disks. If the disk inserted is not a Macintosh-formatted disk, it is considered to be a PC disk, and it is made available to the PC system if the PC system is active. PC disks are mounted and unmounted automatically, and the `rsMountDisks` call merely enables the process.

The status code for this routine is `rsMountDisks = 501`.

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsMountDisks`. |

## rsDontMountDisks

You can use the `rsDontMountDisks` control routine to stop the interface driver from monitoring disk-insertion events. If the interface driver has already mounted a PC disk before you make this call, the PC disk remains in the drive and available to the PC system.

The status code for this routine is `rsDontMountDisks = 502`.

#### Parameter block

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsDontMountDisks`. |

## rsActivateKB

You can use the `rsActivateKB` control routine to direct data from the Macintosh computer keyboard to the PC system. All keys except the Command key are trapped. This means that instead of being passed to the Macintosh, key codes (keystrokes) are translated and transmitted to the PC system.

The status code for this routine is `rsActivateKB = 102`.

#### Parameter block

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsActivateKB`. |

## rsDeactivateKB

You can use the `rsDeactivateKB` control routine to stop transmission of keyboard data to the PC system and direct the keyboard data to the Mac OS.

The status code for this routine is `rsDeactivateKB = 103`.

#### Parameter block

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsDeactivateKB`. |

## rsBeginMouseTracking

You can use the `rsbeginMouseTracking` control routine to direct mouse movements and button presses to the PC system. This routine also causes the driver to hide the Macintosh cursor.

The status code for this routine is `rsBeginMouseTracking = 201`.

#### Parameter block

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsBeginMouseTracking`. |

## rsEndMouseTracking

You can use the `rsEndMouseTracking` control routine to direct the mouse movements and button presses to the Mac OS. This routine also causes the driver to show the Macintosh cursor.

The status code for this routine is `rsEndMouseTracking = 202`.

#### Parameter block

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsEndMouseTracking`. |

## rsEndPrintJob

You can use the `rsEndPrintJob` control routine to end the current print job and, if there is one, close the spool file. Any subsequent data transferred from the PC system to the parallel port starts a new spool file.

The control call for this routine is `rsEndPrintJob = 401`.

#### Parameter block

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsEndPrintJob`. |

# Detecting Errors

Applications running on the Mac OS can use the routines described in this section to detect error conditions or other special events on the PC system. Table 4-2 summarizes the return codes used to indicate PC system printing or serial communication errors.

**Table 4-2** Return codes for PC printing or serial communication errors

| Code | Description | Error type |
|---|---|---|
| `rsPrintSpoolErr = 0x7F00` | Print spool file open or a write error | Printing |
| `rsPSFileReady = 0x7F01` | Ready to print to spool file | Printing |
| `rsCOM1SpoolErr = 0x7F01` | Serial spool file open or a write error | Serial communication |
| `rsCOM2SpoolErr = 0x7F02` | Serial spool file open or a write error | Serial communication |

## rsSetNotificationProc

You can use the `rsSetNotificationProc` control routine to install a user-defined procedure that is called whenever a special event happens within the driver. The procedure can be called at interrupt time and puts off handling the event until a noninterrupt time.

The control call to set the address of the notification procedure is `rsSetNotificationProc = 900`.

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsSetNotificationProc`. |
| —> | `csParam+0` | `long` | Pointer to the notification procedure. |
| —> | `csParam+4` | `long` | `A1Param` value. |

Upon return, the parameters are set as follows:

| | | | |
|---|---|---|---|
| <— | `csParam+0` | `long` | Pointer to the previous notification procedure. |
| <— | `csParam+4` | `long` | Previous `A1Param` value. |

The caller passes a pointer to the user-defined procedure and to a parameter that is passed to that procedure in the `A1Param` value. The control routine returns the previous values. Calling `rsSetNotificationProc` with a `NULL` pointer disables the notification procedure.

When the user-defined procedure is called, the D0.w register contains the event and the A1 register contains the `A1Param` value. The procedure can use registers D0–2 and A0–1.

The events are listed inTable 4-3.

**Table 4-3** Special events

| Event | Description |
|---|---|
| `rsPrintSpoolErr` | Problem opening or writing to a print spool file |
| `rsCOM1SpoolErr` | Problem opening or writing to the COM1 spool file |
| `rsCOM2SpoolErr` | Problem opening or writing to the COM2 spool file |
| `rsDiskFileErr` | Problem reading the disk file |

## rsLastError

You can use the `rsLastError` status routine to obtain the last nonzero error code returned.

The status code for this routine is `rsLastError = 702`.

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsLastError`. |
| <— | `csParam+4` | `long` | Pointer to the last error routine. |

# Passing Messages

Applications running on the Mac OS and the PC system can send messages to each other by calling the interface driver. Applications can also install a receive procedure for receiving messages. When the interface driver receives a message that is intended for your application, the driver calls your receive procedure. Your procedure then decides whether or not to accept the data in the message and, if it accepts the data, where to store it.

The registers referenced in the following sections are registers in the Pentium and 5x86 processors on the 12" and 7" PC Compatibility Cards. Registers AX, BX, CX, DX, DI, SI, ES, and DS are to x86 processors what the D0-7 and A0-7 registers are to 68K processors.

## Message Conventions

Before communication can take place, an application running on the Mac OS and an application running on the PC system must have the same definitions of the messages they transfer. A message consists of a message parameter block containing up to 64 KB of data. The parameters and the data can consist of any data in any format. The command must be a unique value recognized by the applications on the Mac OS and the PC system that are sending and receiving messages. These applications must request command numbers from the interface driver before sending messages.

### Macintosh Interface

Applications running on the Mac OS communicate with the interface driver through driver calls. Your application should first open the driver using the `open` call and then use the control routines described in the following sections to register, send, and receive messages.

### PC System Interface

Applications running on the PC system communicate with the interface driver through a software interrupt interface. The application loads registers with appropriate values, including a function selector in register AH, and then calls the interface driver with an `INT 5Fh` call. PC system applications can determine whether the interface driver is available by calling `INT 5Fh` with register AH = 0. If the interface driver is installed, it returns `0A5h` in register AH and the highest implement function code (currently 4) in register AL.

## Registering Messages

For an application on the Mac OS to send messages to an application on the PC system, both applications must register their messages with the interface driver. This is done by

calling the driver with a 32-bit selector that is defined in both applications and with a count of the number of messages to be used by the applications. The interface driver allocates a range of messages for that selector and returns the base command number to the caller. The interface driver makes sure that both the PC application and the Macintosh application registering messages under the same selector will receive the same base command number.

### Registering Messages From the Mac OS

To register your messages from a Macintosh application, make an `rsRegisterMessage` control call with the message selector in `csParam+0` and the number of message commands to allocate in `csParam+4`.

The routine that registers message type is `rsRegisterMessage = 803`.

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver
<—> indicates bidirectional transfer (input and output)

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsRegisterMessage`. |
| <—> | `csParam+0` | `long` | 32-bit message selector. |
| —> | `csParam+4` | `long` | Number of message commands to allocate. |

The interface driver returns the base command number in `csParam+0`. If the interface driver cannot allocate the messages, an error code is returned in `ioResult`.

### Registering Messages From the PC System

To register your messages from a PC application, load the 32-bit selector into register EBX and the message count into register CX. Then call `INT 5Fh` with AH = 4. The interface driver returns the base command number in register BX. Register AH contains an error code if the message could not be allocated.

## Sending a Message

To send a message, you must pass a message parameter block (`MsgPBlk`) to the interface driver. The `rsSendMessage` routine is always asynchronous. The message control code for sending a message is `rsSendMessage = 800`. This routine simply queues the message parameter block and returns to the caller. The `msgResult` field is set to 1 (busy) until the message has been sent.

After the message has been sent, the `msgResult` field is set to 0 (no error) or -3 (`MsgTimeout`). The `msgActCount` field contains the number of bytes actually sent. If you have specified a completion routine, it is then called.

## Sending a Message From the Mac OS

To send a message on the Mac OS, build a `MsgPBlk` data structure and then pass the pointer to the interface driver in an `rsSendMessage` control call.
The `MsgPBlk` data structure for applications on the Mac OS has the following format:

```
MsgPBlk          RECORD    0
msgQLink         DS.1      1    ; Next queue element
msgQType         DS.w      1    ; Queue flag
msgCmd           DS.w      1    ; The message type or command
msgParam1        DS.1      1    ; Message parameter 1
msgParam2        DS.1      1    ; Message parameter 2
msgBuffer        DS.1      1    ; Pointer to the message data buffer
msgReqCount      DS.1      1    ; Requested data length
msgActCount      DS.1      1    ; Actual data length
msgCompletion    DS.1      1    ; Pointer to completion routine or
                                  NULL
msgResult        DS.w      1    ; The result of any message operation
msgFlags         DS.w      1    ; Message flags (Swap, and Shared);
                                  set to zero!
msgUserData      DS.1      1    ; For the caller's use
MsgPBlkcSize     Equ       *    ; Size of record
                 ENDR
```

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsSendMessage`. |
| —> | `csParam+0` | `long` | Pointer to `MsgBlk`. |

Your completion routine is called at deferred time and can use registers D0–D2 and A0–A2. You must save all other registers. Upon return, A0 contains a pointer to the `MsgPBlk` structure.

**Note**
Deferred time refers to deferred task time. Hardware and software interrupts occur all the time. The system processes these interrupts. After doing so, but before returning control to the interrupt process, the system looks for any registered deferred tasks and executes them. Refer to *Inside Macintosh: Processes* for further information on this subject. ◆

## Sending a Message From the PC system

To send a message on the PC system, build a `MsgPBlk` structure and call the interface driver with AH = 1 (`rsSendMessage`) and ES:BX = the pointer to the `MsgPBlk` structure. When you execute an `INT 5Fh`, the message is queued, `msgResult` is set to 1 (busy), and control returns to your application.

Your completion routine is called with a FAR call, and it should return with an `RETF`. Your routine may also use registers AX, BX, CX, DX, DI, SI, ES, and DS. When your completion routine is called, ES:BX is a pointer to the `MsgPBlk` structure.

The `MsgPBlk` data structure on the PC system has the following format:

```
MsgPBlk         STRUCT
link            DWORD    ; Link to next queue element
msgCmd          WORD     ; The message command or type
msgParam1       DWORD    ; Param 1
msgParam2       DWORD    ; Param 2
msgCompletion   DWORD    ; Pointer to completion routine or NULL
msgBuffer       DWORD    ; Pointer to the data buffer
msgReqCount     DWORD    ; Length of the data
msgActCount     DWORD    ; # of bytes actually transferred
msgResult       BYTE     ; The result of message operation
msgFlags        BYTE     ; Message flags (Shared and Swapped);
                           set to zero
msgUserData     DWORD    ; For use by caller
MsgVXD          DWORD    ; For use by ApplePC VxD
MsgPBlk         ENDS
```

**Note**
The sizes of some fields are different from the Mac OS equivalent. ◆

# Installing a Message Handler

Before you can receive messages, you must install a message handler, using the `rsInstallMsgHandler` routine. The interface driver calls the message handler when the driver receives a message with a command value greater than or equal to `recCmdBase` and less than `recCmdBase+recCmdCount` in the `MsgRecElem` data

structure. The driver passes the message's 16-bit command and the two 32-bit parameters to your message handler. Refer to "Installing a Message Handler on the Mac OS" on page 76 and "Installing a Message Handler on the PC System" on page 77 for information about about the format of the `MsgRecElem` data structure.

The message handler examines the command and parameters and determines whether there is any data to be received. If there is, the handler passes back a pointer to the `MsgBlk` data structure. The interface driver then receives the data and puts it into the buffer pointed to by `msgBuffer`. The driver then updates `msgActCount` with the number of bytes of data received and sets `msgResult` to 0 (no error), –1 (`MsgOverrun`), –2 (`MsgUnderrun`), or –3 (`MsgTimeout`). The driver then calls your completion routine, if there is one.

A message handler is described by a `MsgRecElem` record. The `recProc` field points to the handler procedure. The values of `recBaseCmd` and `recCmdCount` are the values allocated by `rsRegisterMessage`.

**IMPORTANT**

Before your application terminates, you must remove your message handler so that the interface driver will not call it after the application has terminated. See "Removing a Message Handler" on page 78 for more information. ▲

## Installing a Message Handler on the Mac OS

To install a message handler on the Mac OS, build a `MsgRecElem` record and pass a pointer to it in a control call to the interface driver. The `MsgRecElem` data structure for applications on the Mac OS has the following format:

```
MsgRecElem       RECORD   0
recQLink         DS.1     1    ; Next queue element
recQType         DS.w     1    ; Queue flags
recFlags         DS.w     1    ; Not used. Set to zero
recProc          DS.1     1    ; Pointer to the receive procedure
recCmdBase       DS.w     1    ; First command received by this
                                  procedure
recCmdCount      DS.w     1    ; Number of commands allocated
                               for   this procedure
recUserData      DS.1     1    ; For caller's use
recVXD                         ; For use by Apple PC VxD
MsgRecElemSize   Equ      *
                 ENDR
```

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsInstallMesHandler`. |
| —> | `csParam+0` | `long` | Pointer to `MsgRecElem`. |

When your message handler procedure is called, register D0.w contains the message command, register D1.1 contains the `msgParam1` value, register D2.1 contains the `msgParam2` value, and register A1 contains a pointer to the `MsgRecElem` record. Your routine must pass back a pointer to a `MsgPBlk` structure in A0 if you wish to receive the message data. Otherwise, return 0 in A0. The handler procedure is called at interrupt time with interrupts masked at the slot interrupt level. It can use registers D0–D2 and A0–A1.

The completion routine for the `MsgPBlk` structure returned by the receive procedure is called at deferred time and can use registers D0–D2 and A0–A1. You must save all other registers. Upon return, A0 contains a pointer to the `MsgPBlk` structure.

## Installing a Message Handler on the PC System

To install a message handler on the PC system, build a `MsgRecElem` record and call `INT 5Fh` with AH = 2 and ES:BX containing a pointer to the `MsgRecElem` structure. For an application running on the PC system, the `MsgRecElem` data structure has the following format:

```
MsgRecElem    STRUCT
Link          DWORD     ; Pointer to next link
Code          DWORD     ; Pointer to the code for this link
cmdBase       WORD      ; Base message number for this   procedure
cmdCount      WORD      ; Number of message numbers for this
                          procedure
userData      DWORD     ; For caller's use
msgVXD        DWORD     ; Reserved for driver use
MsgRecElem    ENDS
```

When your message handler is called, AX contains the message command, ECX contains `msgParam1`, EDX contains `msgParam2`, and ES:DI contains a pointer to the `MsgRecElem` record. Your application must pass a pointer to a `MsgPBlk` structure in ES:BX if you wish to receive the message data. Otherwise, return 0 in BX. The handler is

called at interrupt time with interrupts turned off. It can use registers AX, BX, CX, DX, DI, SI, ES, and DS.

The completion routine for the `MsgPBlk` structure returned by the receive procedure is called at interrupt time and can use registers AX, BX, CX, DX, DI, SI, ES, and DS. You must save all other registers. ES:BX must also contain a pointer to the `MsgPBlk` structure.

## Removing a Message Handler

Message handlers can be called until they are removed. Before your application terminates, you must remove the handler so that the interface driver will not call it after the application has terminated.

### Removing a Message Handler on the Mac OS

To remove a message handler on the Mac OS, your application makes an appropriate control call to the interface driver and passes the driver a pointer to the handler. The message control code is `rsRemoveMsgHandler = 802`.

**Parameter block**

—> indicates input to the driver
<— indicates output from the driver

| | | | |
|---|---|---|---|
| —> | `ioCompletion` | `long` | Pointer to the completion routine. |
| <— | `ioResult` | `word` | Device driver's result code. |
| —> | `ioRefNum` | `word` | Device driver's reference number. |
| —> | `csCode` | `word` | Equals `rsRemoveMsgHandler`. |
| —> | `csParam+0` | `long` | Pointer to `MsgRecElem`. |

### Removing a Message Handler on the PC System

To remove a message handler on the PC system, your application makes a call to `INT 5Fh` with AH = 3 and with a pointer to the `MsgRecElem` record in registers ES:BX.

# Gestalt Selector

The Gestalt function enables you to obtain information about software or hardware components available on your system. The original Gestalt Manager consists of three functions—Gestalt, New Gestalt, and Replace Gestalt. These functions are described in *Inside Macintosh, Volume VI*, in the section "Using the Gestalt Manager."

12" and 7" cards have an added Gestalt function. This is a Public Gestalt selector that enables you to determine the state of certain features associated with the PC system. The longword bit field result is a 32-bit four-character selector. The four characters are

pc (pc followed by two spaces). Only 4 bits of the word are currently in use, as shown below. The remaining 28 bits are undefined.

```
gestaltDOSCompatibleState = pc

enum {
   gestaltPCInstalled,
   gestaltPCRunning,
   gestaltPCHasTakenOver,
   gestaltPCSharingMonitor
};
```

#### Field descriptions

`gestaltPCInstalled`
This is bit 0. If it is high, the PC Setup Init has been run and is installed.

`gestaltPCRunning`
This is bit 1. It tells the caller whether or not the PC system is running. If the bit is high, the PC system is running.

`gestaltPCHasTakenOver`
This is bit 2. It tells the caller whether or not the PC system is in the foreground. If the bit is high, the PC system is in the foreground, and you are running applications on the PC system rather than the Mac OS. You may see physical signs of this condition. For example, if you have a shared monitor, the PC screen will replace the Macintosh screen. If you have two monitors, the Macintosh screen will dim when the PC system is in the foreground.

`gestaltPCSharingMonitor`
This is bit 3. It tells the caller whether or not the system is set up with a dedicated monitor for the PC system. If the bit is high, there is no dedicated monitor for the PC system.

# Summary of Constants

Table 4-4 provides a summary of the constants associated with the routines described in this chapter.

**Table 4-4** Summary of constants

| Call and constant | Code/ call type | Description |
|---|---|---|
| **Setting up shared folders** | | |
| `rsSetNetDriveConfig = 600` | Control | Sets shared drive configuration |
| `rsGetNetDriveConfig = 650` | Status | Gets drive letter information |
| **Setting up parallel ports** | | |
| `rsSetParallelPortConfig = 400` | Control | Sets configuration for parallel port |
| `rsEndPrintJob = 401` | Control | Forces end of current print job |
| **Setting up communication ports** | | |
| `rsModemComPort = 1` | Device | Sets up modem port |
| `rsPrinterComPort = 2` | Device | Sets up printer port |
| `rsSpoolComPort = 3` | Device | Data from communication port dumped into a file |
| `rsSetComPortConfig = 300` | Control | Sets the connection for the communication port |
| **PC hardware on/off switch—Macintosh <—> PC swap** | | |
| `rsEnableVideo = 705` | Control | Enables VGA output |
| `rsDisableVideo = 706` | Control | Disables VGA output |
| **PC error detection—printing and serial communication errors** | | |
| `rsPrintSpoolErr = 0x7F00` | Notification | There is a problem opening or writing the print spool files |
| `rsPSFileReady = 0x7F01` | Notification | At least one spool file is ready to be printed |
| `rsCOM1SpoolErr = 0X7F01` | Notification | Serial spool file is open or there is a write error on COM1 |
| `rsCOM2SpoolErr = 0X7F02` | Notification | Serial spool file is open or there is a write error on COM2 |

**Table 4-4** Summary of constants (continued)

| Call and constant | Code/ call type | Description |
|---|---|---|
| **PC event detection—notification procedure or event notification** | | |
| `rsSetNotificationProc = 900` | Notification | Sets address of the notification procedure |
| **Messaging** | | |
| `rsRegisterMessage = 803` | Control | Registers message type |
| `rsInstallMsgHandler = 801` | Control | Installs message handler |
| `rsSendMessage = 800` | Control | Sends a message |
| `rsRemoveMsgHandler = 802` | Control | Removes message handler |
| `msgNoError = 0` | Result | No errors detected, message completed |
| `msgOverrun = -1` | Result | More data was available |
| `msgUnderrun = -2` | Result | Less data was available |
| `msgTimeout = -3` | Result | A timeout error has occurred |
| `Mac MsgBlk ≠ PC MsgBlk` | Result | The Macintosh message block is not the same size as the PC message block |

# Messaging Code Samples

This section summarizes the code associated with messages being passed from the Macintosh to the PC system.

## Registering Owner Type

The following parameter block is used to define the owner of the message:

```
pbp.csCode = rsRegisterMessage = 803

// owner == Application creator
pbp.csPtr = (void*)owner;

// cmdCount == Whatever is needed
pbp.csData = cmdCount;

err - PBControl((ParmBlkPtr)&pbp);
cmdBase = (int)pbp.csPtr;
```

### Supplementary Information

The value `cmdCount` is needed to register a message. It contains the number of different types of messages an application would like to support. For example, if an application wants to send two types of messages (“send DOS command” and “receive error code”) between the Macintosh computer and the PC system, it would set this value to `2`.

## Installing Command Receiver

The following parameter block is used to install the command receiver:

```
pRecElem–>recFlags = 0;
// Receive message proc.
pRecElem->recProc = (ProcPtr)RcvMsg;
pRecElem->recCmdBase = cmdBase;
pRecElem->recCmdCount = cmdCount;
// User-defined data
precElem->recUserData = nil;
pbp.csCode = rsInstallMsgHandler; = 801
pbp.csPtr = pRecElem;
err = PBControl ((ParmBlkPtr)&pbp);
```

### Supplementary Information

Interpretation of messages is done by the `RcvMsg` procedure. As part of the application that asks for a message block, it has to install a message receive procedure.

The result returned from registering a message is the base address of the messages that your application will get back, `cmdBase`. If the application asks for 2 cmd and the base address returned is `C000`, then all messages that have the address of `C000` and `C001` belong to that application and are interpreted however the application wishes to interpret them.

The value `cmdCount` is needed to register a message. It contains the number of different types of messages an application would like to support. For example, if an application wants to send two types of messages (“send DOS command” and “receive error code”) between the Macintosh computer and the PC system, it would set this value to `2`.

## Sending a Message to the PC System

The following code is used to pass a message to the PC system:

```
pMsg->msgCompletion = (ProcPtr)WrComp;
pMsg->msgUserData = nil;
pMsg->msgFlags = 0;
RSParamBlockRec Params;
pbp.ioCRefNum = gRSDrvrRefNum;
```

```
pbp.csCode = rsSendMessage; = 800
// The message you wish to pass
pbp.csPtr = pMsg;
err = PBControl((ParmBlkPtr)&pbp);
```

Remove message = 802

## Supplementary Information

The `WrComp` procedure is used by the message system to inform the application that the message is complete. This allows the application to do whatever other processing is needed based on the completion of the message being passed.

The value of `gRSDrvrRefNum` is the driver reference number returned by the `PBOpen` call.

# Index

## Numerals



## A



## B



## C

## M, N, O



## P, Q



## R

## S



## T, U



## U



## V, W



## X, Y, Z

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Proof pages and final pages were created on an Apple LaserWriter Pro printer. Line art was created using Adobe Illustrator™ and Adobe Photoshop™. PostScript™, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

WRITER
Joyce D. Mann

DEVELOPMENTAL EDITORs
Wendy Krafft and Beverley McGuire

ILLUSTRATOR
Sandee Karr

PRODUCTION EDITOR
Alex Solinski

Special thanks to Dan Miranda, Dave Townsley, Rich Collyer, Rich Kubota, and Damon Schaefer