Developer Note

Power Macintosh 5400 Computer

Developer Note

4/12/96
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

This developer note describes the Apple Power Macintosh 5400 computer and emphasizes features that are new or different from previous Macintosh models. It is intended to help experienced Macintosh hardware and software developers design compatible products. If you are unfamiliar with Macintosh computers or would simply like more technical information, you may wish to read the related technical manuals listed in the section "Supplemental Reference Documents."

Contents of This Note

The information is arranged in seven chapters and an index:

- Chapter 1, "Introduction," gives a summary of the features of the Power Macintosh 5400 computer, describes the physical appearance, and lists the available configurations and options.
- Chapter 2, "Architecture," describes the internal organization of the computer. It includes a block diagram and descriptions of the main components of the logic board.
- Chapter 3, "I/O Features," describes the built-in input/output (I/O) devices and the external I/O ports. It also describes the built-in monitor configuration of the Power Macintosh 5400 and external video monitors that can be used with the computer.
- Chapter 4, "Expansion Features," describes the expansion slots of the Power Macintosh 5400 computer. This chapter provides guidelines for designing cards for the I/O expansion slot and brief descriptions of the expansion modules for the other slots.
- Chapter 5, "Software Features," summarizes the new features of the ROM software and the system software that accompany the Power Macintosh 5400 computer.
- Chapter 6, "Large Volume Support," describes the large volume file system, and defines the modifications to the application programming interface to the hierarchical file system that support volume sizes greater that 4 GB.
- Chapter 7, "Software for the ATA (IDE) Hard Disk," gives the program interface for the system software and the driver that support the internal IDE hard disk drive.

This developer note also contains an appendix listing abbreviations and an index.

Supplemental Reference Documents

To supplement the information in this developer note, developers should have copies of the appropriate Motorola reference books for the PowerPC™ 603e microprocessor. Software developers should have a copy of Motorola's *PowerPC Programmer's Reference Manual*. Hardware developers should have copies of Motorola's *PowerPC 603 RISC Microprocessor User's Manual*.

For additional information about the digital data format used in the video input module, refer to *Macintosh DAV Interface for NuBus Expansion Cards*, part of *Macintosh Developer Note Number 8*, APDA catalog number R0566LL/A. For information about the digital video interface, refer to the *SAA7140 Philips Desktop Video Handbook*.

Developers may also need copies of the appropriate Apple reference books. You should have the relevant books of the *Inside Macintos*h series. You should also have *Guide to the Macintosh Family Hardware*, second edition, *Designing Cards and Drivers for the Macintosh Family*, third edition, and *Designing PCI Cards and Drivers for Power Macintosh Computers*. These books are available in technical bookstores and through APDA.

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 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) 1-800-637-0029 (Canada) 716-871-6555 (International)
Fax	716-871-6511
AppleLink	ORDER.ADC
Internet	order.adc@applelink.apple.com

Conventions and Abbreviations

This developer note uses the following typographical conventions and abbreviations.

Typographical Conventions

New terms appear in **boldface** where they are first defined.

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

Hexadecimal numbers are preceded by a dollar sign ($). For example, the hexadecimal equivalent of decimal 16 is written as $10.

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. ▲

Sidebar

Sidebars are for digressions—information that is not part of the main discussion. A sidebar may contain background information that is interesting to know, information about a related subject, or technical details that are not required reading.

Standard Abbreviations

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

A	amperes	mA	milliamperes
dB	decibels	µA	microamperes
GB	gigabytes	MB	megabytes
Hz	hertz	MHz	megahertz
in.	inches	mm	millimeters
k	1000	ms	milliseconds
K	1024	µs	microseconds
KB	kilobytes	ns	nanoseconds
kg	kilograms	Ω	ohms
kHz	kilohertz	sec.	seconds
kΩ	kilohms	V	volts
lb.	pounds	W	watts

Here are other abbreviations used in this developer note:

$*n*	hexadecimal value *n*
AC	alternating current
ADB	Apple Desktop Bus
AV	audiovisual
AWACS	audio waveform amplifier and converter for sound
CD-ROM	compact-disk read-only memory
CLUT	color lookup table
DAV	digital audio video
DESC	digital video decoder and scaler
DIMM	dual inline memory module
DMA	dynamic memory access
DRAM	dynamic random-access memory
DVA	digital video application
EMI	electromagnetic interference
FPU	floating-point unit
IC	integrated circuit
IDE	integrated device electronics
IIC	inter-integrated circuit (an internal control bus)
I/O	input/output

IR	infrared
LS TTL	low-power Schottky TTL (a standard type of device)
MMU	memory management unit
MOS	metal-oxide semiconductor
NTSC	National Television Standards Committee (the standard system used for broadcast TV in North America and Japan)
NMI	nonmaskable interrupt
PAL	Phase Alternating Line system (the standard for broadcast TV in most of Europe, Africa, South America, and southern Asia)
PCI	Peripheral Component Interconnect
PDS	processor-direct slot
PWM	pulse-width modulation
RAM	random-access memory
RGB	a video signal format with separate red, green, and blue components
RISC	reduced instruction set computing
RMS	root-mean-square
ROM	read-only memory
SANE	Standard Apple Numerics Environment
SCSI	Small Computer System Interface
SCC	serial communications controller
SECAM	the standard system used for broadcast TV in France and the former Soviet countries
SIMM	single inline memory module
S-video	a type of video connector that keeps luminance and chrominance separate; also called a Y/C connector
SWIM	Super Woz Integrated Machine, a custom IC that controls the floppy disk interface
TTL	transistor-transistor logic (a standard type of device)
VCR	video-cassette recorder
VLSI	very large scale integration
VRAM	video RAM; used for display buffers
Y/C	a type of video connector that keeps luminance and chrominance separate; also called an S-video connector
YUV	a video signal format with separate luminance and chrominance components

CHAPTER 1

Introduction

The Power Macintosh 5400 computer is a new Macintosh model that incorporate a PowerPC™ 603e microprocessor running at 100 and 120 MHz, a Level 2 cache expansion slot, a Peripheral Component Interconnect (PCI) card expansion slot, enhanced AV features (audio and video input and output), and a new PCI-based communications slot. The Power Macintosh 5400 computer is housed in an all-in-one enclosure, like that of the Power Macintosh 5200, featuring a 15-inch monitor with tilt and swivel capability and stereo speakers.

Summary of Features

Here is a summary of the hardware features of the Power Macintosh 5400 computer. Each feature is described more fully later in this note.

- Microprocessor: PowerPC 603e microprocessor running at 100 MHz and 120 MHz.
- RAM: 8 MB soldered to the main logic board; expandable to 136 MB using 168-pin JEDEC-standard DIMM devices. Two DIMM slots are provided for DRAM expansion.
- ROM: 4 MB soldered on main logic board; 64-bit ROM data bus width.
- Cache: 256 KB Level-2 cache on a 160-pin DIMM card (optional)
- Video configuration: internal video supports built-in 15-inch multiscan monitor; 1 MB DRAM frame buffer on the main logic board.
- Video modes supported: 640 by 480 and 800 by 600 @ 16 bits per pixel, and 832 by 624 and @ 8 bits per pixel.
- Video input: 60 pin DAV connector supports an optional video card for real-time video display, capture, and overlay. An adapter cable provides backward compatibility with DVA cards designed for the Power Macintosh 5200 computer.
- Video output: video mirror feature allows an external monitor to be connected to a Power Macintosh 5400 computer using an optional video buffer board connected to the video mirror connector.
- Sound: 16 bits/channel SRS® stereo surround sound input and output, external jack for sound in, front jack for headphones, rear jack for stereophonic speakers, two built-in stereo speakers. The Power Macintosh 5400 also includes a built-in microphone.
- TV receiver: optional internal TV tuner.
- Remote control: infrared
- Hard disks: one internal 3.5-inch IDE hard disk with 1.2 GB or larger capacity; external SCSI port for additional SCSI devices. PIO, Singleword DMA, and Multiword DMA data transfers are supported.
- Floppy disk: one internal 1.4 MB Apple SuperDrive.
- 4X-speed CD-ROM drive: internal SCSI connection for optional AppleCD 600 CD-ROM drive
- Processor bus: 64-bit wide, 40MHz, supporting split address and data tenures.

- Standard Macintosh I/O ports: two serial ports, sound input and output jacks, a SCSI port, and an ADB port.
- GeoPort: supported on both the modem and printer port.
- PCI-based communications slot: 112-pin connector accepts an optional modem or Ethernet interface. This is the first entry-level Power Macintosh with this type of communications slot.
- Expansion slot: accepts one 7-inch PCI card.
- Power switch: soft power controlled from keyboard and remote control.
- Case design: Power Macintosh 5400 has an all-in-one enclosure similar to the Power Macintosh 5200 featuring a tilt and swivel monitor and built-in stereo speakers.
- Fan speed control: The speed of the fan is thermally controlled and is automatically set to the lowest possible speed to minimize noise. The fan speed varies according to the temperature inside the enclosure.
- Energy saving: sleep, startup, and shutdown scheduling can be controlled with an Energy Saver control panel.

Comparison With Power Macintosh 5200 Computer

The Power Macintosh 5400 computer is electrically similar to the Power Macintosh 5200. Table 1-1 compares the features of these computers.

Table 1-1 Comparison with the Power Macintosh 5200 series computer

Features	Power Macintosh 5200	Power Macintosh 5400
Processor type	PowerPC 603	PowerPC 603e
Processor speed	75 MHz	100 MHz and 120 MHz
Cache	256 KB level-2 cache	256 KB level-2 cache (optional)
Amount of RAM	8 MB–64 MB	8 MB–136 MB
RAM expansion	2 SIMMs	2 168-pin DIMMs
Memory bus	32 bits, 37.5 MHz	64 bits, 40 MHz
Video RAM	1 MB (DRAM)	1 MB (DRAM)
Video input	optional card for video input, capture, and overlay	optional card for video input, capture, and overlay

continued

Table 1-1 Comparison with the Power Macintosh 5200 series computer (continued)

Features	Power Macintosh 5200	Power Macintosh 5400
Video output	optional mirror connector supports external monitor operating in mirror mode;	optional mirror connector supports external monitor operating in mirror mode; built-in video supports up to 832-by-624 pixel resolution at 8 bits per pixel
Sound capabilities	8 or 16 bits/channel; mono in, stereo out	8 or 16 bits/channel; stereo in, stereo record, stereo out; SRS surround sound mode
Remote control	built-in infrared (IR) receiver	built-in IR receiver
Floppy disk drive	1, internal	1, internal
ADB ports	1	1
Internal hard disk	1 (IDE)	1 (IDE)
Internal CD-ROM	optional	optional
External SCSI ports	1	1
Communications slot	1, for optional modem or Ethernet interface (68040 bus configuration)	1, for optional modem or Ethernet interface (PCI bus configuration)
Expansion slot	1 I/O slot (accepts PDS card for Macintosh LC series)	1 PCI I/O slot for 7-inch card
DMA I/O	None	10 DMA channels
Serial ports	2, modem and printer, LocalTalk supported	2, LocalTalk and GeoPort supported

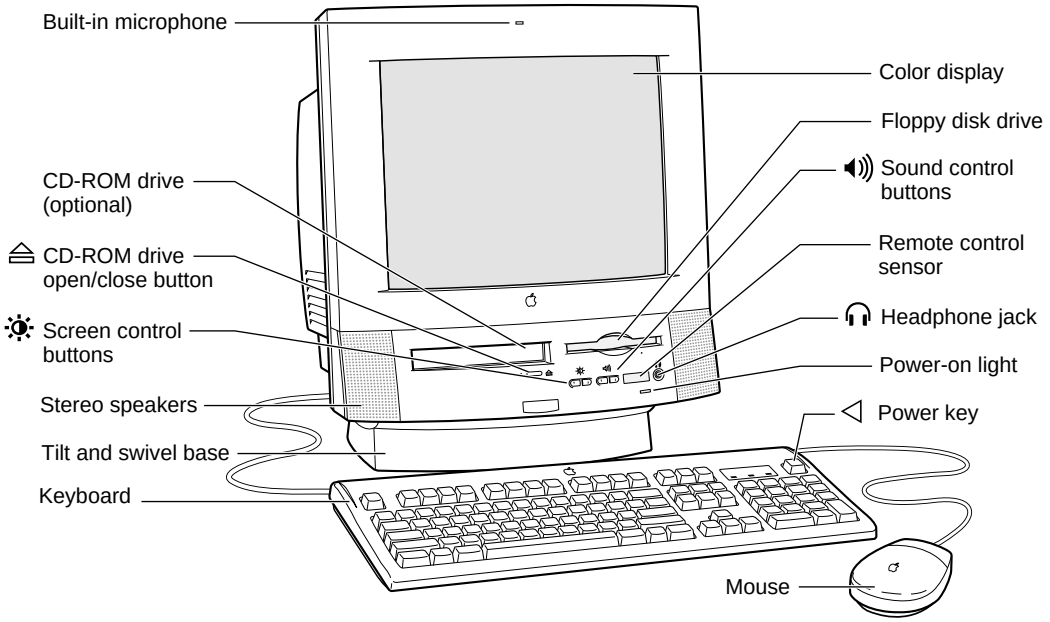
External Features

The Power Macintosh 5400 computer has an integrated design featuring a built-in 15-inch multiscan color monitor with both tilt and swivel capabilities, stereo speakers, front panel stereo headphone jack, and front panel push button controls for audio and video.

Front View

Figure 1-1 is a front view of a Power Macintosh 5400 computer. The front view shows the display screen, the built-in microphone and stereo speakers, the openings for the floppy disk and optional CD-ROM drive, the CD-ROM open and close button, the headphone jack, the power-on light, the IR sensor for the remote control, and the push buttons that control the screen intensity and sound level.

Figure 1-1 Front view of the computer

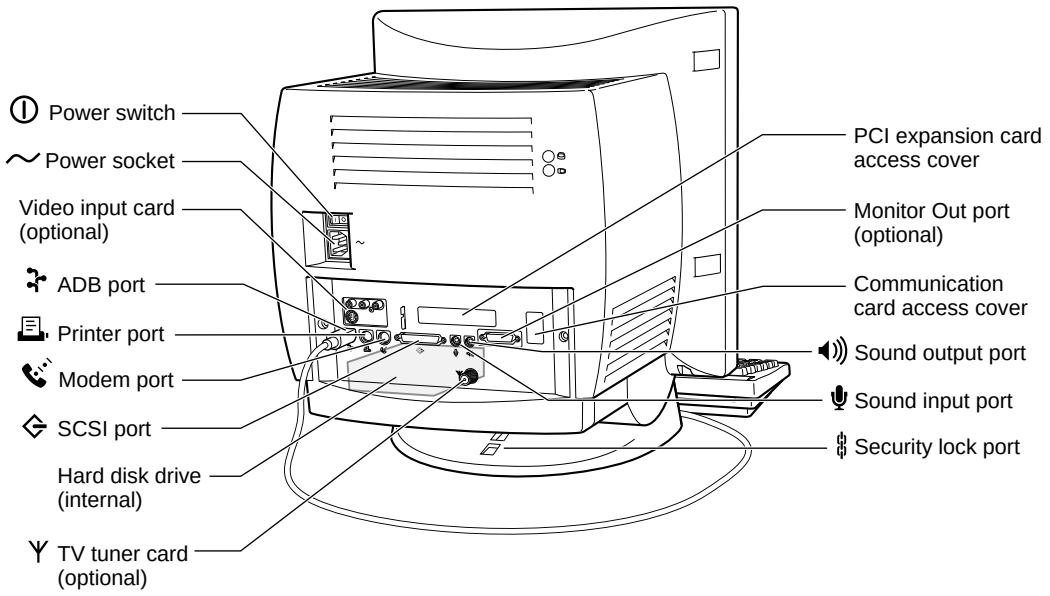


Back View

The back panel includes the power socket, the reset button, the I/O ports, and openings for I/O access to the expansion modules: the I/O expansion card, the communications card, and the video input card.

Figure 1-2 shows the back view of a Power Macintosh 5400 computer.

Figure 1-2 Back view of the computer



Access to the Logic Board

The logic board can be removed from the case so that the user can add expansion RAM, Level 2 cache, or plug in an I/O expansion card. To get access to the logic board, you must first remove the back panel. It is secured by two screws on either side of the I/O connectors. After removing the screws, you can pull gently on the two latches on the underside of the computer's case and the back panel will slip out. You can then grasp the logic board handle and pull the board straight out the back of the case.

Front Panel Push Buttons

The Power Macintosh 5400 computer has two pairs of push buttons on the front panel. The pair on the left controls the intensity of the screen; the pair on the right controls the sound level.

Power On and Off

The user can turn the power off and on by pressing one of two buttons:

- the Power key on the keyboard
- the Power key on the remote control

If files are still open when the user attempts to turn off the computer by using either one of the Power keys or the Shut Down menu item, the system displays an alert box warning the user that files are open and should be closed to avoid loss of data.

Optional Features

Several features of the Power Macintosh 5400 computer are implemented as plug-in modules available either as a configuration option at the time of purchase or as a later upgrade. The modules are designed so that they can be installed by the user.

TV Tuner

The TV tuner module turns the computer into a television receiver, complete with remote control. The features of the TV tuner module are similar to those of the TV tuner in the Power Macintosh 5200 and 6200 computers. The TV picture is in its own window on the desktop, and the TV signal is carried in YUV format for improved picture clarity.

The features of the TV tuner module are

- ability to tune 181 broadcast and cable channels (U.S. version)
- coaxial connector for TV antenna or cable input (F-type connector in U.S. and Japanese versions; IEC-type connector in Europe)
- TV picture in a resizable and movable window
- YUV format for improved clarity (see sidebar)
- support for closed captioning and teletext
- software password protection
- automatic and manual channel programming
- single remote control for TV and for playback of audio CDs

The TV tuner module is available in versions for NTSC, PAL, and SECAM television systems.

The TV picture appears in its own window. The default size of the window is 320 by 240 pixels. The user can resize the TV window up to a maximum size of 640 by 480 pixels or down to a minimum size of 160 by 120 pixels. The resolution of the TV picture does not

Why YUV Looks Clearer

You may be wondering how the digital YUV format used in the Power Macintosh 5400 computer provides a clearer TV picture than the RGB format used in the Macintosh TV computer—after all, picture information can be freely converted between the two formats. The difference is due to the way the bits are allocated. The RGB format used in the Macintosh TV is a 16-bit format using 5 bits each for red, green, and blue, with the remaining bit unused. The YUV format used in the Power Macintosh 5400 computer is also a 16-bit format, with 8 bits for the Y (luminance) channel and 8 bits for the U and V (chrominance) channels to share by multiplexing. The YUV format looks clearer because the YUV format carries more levels of luminance information.

increase at the larger window sizes; instead, the image is expanded by either doubling the size of the pixels or two-dimensional linear interpolation.

The TV tuner module works in conjunction with the video input module, which converts the video data into digital YUV format and stores it in the display buffer.

The TV tuner comes with a remote control device similar to the one used with the Macintosh TV computer. The user can switch channels either by using the remote control or by typing the channel numbers on the keyboard. The user can toggle between the current and previous channel by pressing the Tab key on the keyboard. Each time the channel changes, the computer displays the channel name (assigned by the user) on the picture in the video window.

The user can customize the operation of the TV tuner by adding or removing TV channels that are unused or unwanted. The computer can program the channels automatically, scanning through all available channels and disabling those that do not have a valid signal. When the user then scans for the next channel by using the remote control or the Tab key on the keyboard, the tuner skips the disabled channels.

The software that supports the TV tuner module is an application called Apple Video Player. The application includes password protection for the disabled channels. Parents might use this feature to prevent children from watching undesirable channels.

The software allows the user to capture or freeze a single frame of video or record a segment of video as a QuickTime Movie. The TV window cannot be resized while the computer is recording a movie.

Video Input

The video input card accepts video from an external source and displays it in a window on the computer's display. The features of the video input card are

- acceptance of video input in NTSC, PAL, or SECAM format
- connectors for stereo sound, composite video, and S-video (Y/C)
- video display in a 320-by-240-pixel window
- pixel expansion for 640-by-480-pixel maximum display
- video overlay capability
- YUV format for digital video input
- a digital video connector (DAV) for adding a video processor on an expansion card

The video input card provides AV features similar to those of the Macintosh Quadra 660AV, with one key improvement. Whereas the Macintosh Quadra 660AV digitizes color video using a 16-bit RGB format, the video input card uses a digital YUV format. Because a standard television signal has more information in its chrominance channel than in its luminance channels, digitizing the video signal as YUV format results in a clearer picture.

The video input card can accept video input from either an external device such as a VCR or camcorder, or from the internal TV tuner module. The external device can be

connected to the video input card either through the composite video connector or the S-video connector.

The default window size is 320 by 240 pixels; the user can resize the window up to 640 by 480 pixels—the full screen on a 14-inch monitor. The large image uses pixel expansion of the 320 by 240 pixel image.

Note
The video input card does not work on all video monitors. It will work with 800-by-600-pixel monitors that have a 60 Hz refresh rate, but not with that size monitor at a 72 Hz refresh rate. In addition, 60 Hz monitors at 800-by-600-pixels must be set to 8-bits per pixel or less. ◆

The video input card plugs into a dedicated slot on the main logic board. The slot connector is a 60-pin microchannel connector. The module fits only its proper slot and only in the proper orientation so that the user can safely install the video input card.

The video input card has a separate connector called the DAV (digital video application) connector. The DAV connector makes the digitized video data available to a card in the I /O expansion slot. Such a card can contain a hardware video compressor or other video processor. For more information, see the section “The DAV Connector” beginning on page 53.

Video Display Mirror Out

The Power Macintosh 5400 computer supports a feature called video display mirror output that allows a second monitor to be connected to the computer through a video buffer card. The video buffer card plugs into a 22-pin connector on the computer’s main logic board.

In the video display mirror out mode, the image on the second monitor’s screen is the same as that on the screen of the built-in monitor. That mode of operation is appropriate, for example, for presentations, so that the audience and the presenter can see the same displays.

Communications

The main logic board in the Power Macintosh 5400 computer has a communications slot that allows the computer to support a communications module without occupying the PCI expansion slot. A communications card can be installed by either the user or the dealer.

The communications slot in the Power Macintosh 5400 computer uses a PCI bus, rather than the 680xx bus. The following cards are supported:

- the 10BaseT (twisted pair) ethernet card
- the 10Base2 (thin coax) ethernet card
- the AAUI (Apple standard) ethernet card
- the 14.4 bps fax/data modem card

Compatibility Issues

The Power Macintosh 5400 computer incorporates several changes from earlier desktop models. This section describes key issues you should be aware of to ensure that your hardware and software work properly with this new model. Some of the topics described here are covered in more detail in later parts of this developer note.

Microprocessor Differences

Applications developers must be aware of certain differences between the PowerPC 603 and the PowerPC 601 microprocessors that can affect the way code is executed. Because of these differences, programs that execute correctly on the PowerPC 601 microprocessor may cause compatibility and performance problems on the PowerPC 603 microprocessor.

POWER-Clean Code

The first generation Power Macintosh computers used the PowerPC 601 microprocessor, a traditional microprocessor that bridged the new PowerPC architecture with the POWER architecture from which it descended. The PowerPC 601 implemented most of the old POWER instruction set as well as the newer PowerPC instruction set.

Later versions of the microprocessor, namely the PowerPC 603 and 604, implement only the PowerPC instruction set, hence the term *power clean.* Because of the differences in instruction set implementation, a possibility exists for incompatibility and poor performance, particularly in the area of compilers.

Newer compilers, designed for the PowerPC instruction set, do not generate the old POWER instructions. However, compilers designed for the POWER instruction set are also being used to compile programs for the PowerPC. Most of those compilers have the option to suppress the generation of offending instructions. For example, the IBM xLc C compiler and the xLCC++ compiler have the option `-garch=ppc`. Developers using these compilers should verify that the options are in effect for all parts of their code. To be on the safe side, you should contact your compiler vendor to make sure that the compiler you are using does not generate POWER instructions.

Completion Serialized Instructions

Several types of instructions can interfere with instruction pipelining and degrade system performance. Most noticeable are completion serialized instructions such as load and store string and load and store multiple. These instructions are referred to as completion serialized instructions because they cannot be executed until all prior instructions have been completed.

Representatives of Apple Computer, Inc., are working with compiler developers to establish guidelines for the appropriate use of these instructions.

Split Cache

Unlike the PowerPC 601, which has a unified cache, the PowerPC 603 has separate caches for instructions and data. This can lead to cache coherency problems in applications that mix code and data.

In the Macintosh Operating System, the Code Fragment Manager loads almost all native code to ensure that the code is suitable for execution. If your code is loaded by the Code Fragment Manager, you don't have to worry about cache coherency.

If, however, your application generates code in memory for execution, problems can arise. Examples include compilers that generate code for immediate execution and interpreters that translate code in memory for execution. For such cases, you can use the call `MakeDataExecutable` to notify the Macintosh Operating System that data is subject to execution. This call is defined in OSUtils.h.

Data Alignment

In PowerPC systems, data is normally aligned on 32-bit boundaries, whereas data for the 680x0 is typically aligned on 16-bit boundaries. Even though the PowerPC was designed to support the 680x0 type of data alignment, misaligned data can cause some loss of performance. Furthermore, performance with misaligned data varies across the different implementations of the PowerPC microprocessor.

While it is essential to use 16-bit alignment for data that is being shared with 680x0 code, you should use PowerPC alignment for all other kinds of data. In particular, you should not use global 680x0 alignment when compiling your PowerPC applications; instead use alignment pragmas to turn on 680x0 alignment only when necessary.

Communications Slot

The communications slot in the Power Macintosh 5400 computer is a PCI bus compatible slot and is in general not compatible with communication cards for the Macintosh LC family of computers, the Macintosh Quadra 630 computer, or cards that operate in the communication slot in Power Macintosh 5200 and 6200 computers. The exception is that cards which do not use the bus, such as serial modem cards, can be designed to work in either type of comm slot. For more information about designing serial modem cards that are compatible with both communications slots, see "The PCI-Bus Communications Slot" beginning on page 58.

DAV Slot

The digital audio video (DAV) slot in the Power Macintosh 5400 computer is compatible with TNT, Nitro, and Tsunami computers. However, it is not compatible with the DAV slot in the Quadra 660AV, Quadra 840AV, Power Macintosh 6100, 7100, and 8100 computers, nor is it plug-in compatible with the DVA slot in the Power Macintosh 5200 and 6200 computers. It is a 60-pin slot with additional signals and capabilities. For additional information about the DAV slot, see "The DAV Connector" beginning on page 53.

Expansion Slot

The I/O expansion slot in the Power Macintosh 5400 computer is a PCI expansion slot and is not compatible with PDS expansion cards for the Macintosh LC family of computers, the Macintosh Quadra 630 computer, or with cards that operate in the I/O expansion slot in Power Macintosh 5200 and 6200 computers.

Cards that are incompatible with the I/O expansion slot include

- cards with drivers that include incompatible code. Some drivers that do not follow Apple Computer, Inc.'s programming guidelines won't work on machines that use the PowerPC 603 microprocessor. For example, some of those drivers write directly to the cache control register in an MC68030. Such code won't work on a PowerPC 603 microprocessor.
- cards with drivers that include code to check the `gestaltMachineType` value and refuse to run on a newer CPU. The idea is to protect users by refusing to run on a machine that the cards haven't been tested on. Such cards have compatibility problems with all new Macintosh models.

RAM Expansion

The Power Macintosh 5400 computer uses JEDEC-standard 168-pin DIMMs (dual inline memory module) DRAM cards rather than the 72-pin SIMM DRAM cards used in the Power Macintosh 5200 and 6200 computers. Information about DRAM DIMM configurations supported on the Power Macintosh 5400 computer, see "RAM DIMMs" beginning on page 42 in Chapter 4, "Expansion Features."

DRAM DIMM developers should note that the PSX memory controller on the main logic board of the Power Macintosh 5400 computer does not provide support for 4 M by 4 bits (12 by 10 addressing) or 1 M by 16 bits (12 by 8 addressing) DRAM devices.

RAM DIMM Dimensions

Apple Computer has made the following change to the mechanical specification for the RAM DIMM.

IMPORTANT

The JEDEC MO-161 specification shows three possible heights for the 8-byte DIMM. For Power Macintosh computers, developers should use only the shortest of the three: 1.100 inches. Taller DIMMs put excessive pressure on the DIMM sockets due to mechanical interference inside the case. ▲

Cache Expansion

On the Power Macintosh 5400 computer, the optional 256K level-2 cache includes an integrated cache controller. Apple does not support development of third-party cache cards for these computer models. The 160-pin cache expansion slot is not compatible with cache cards for previously released Power Macintosh computer models.

ATA (IDE) Hard Disk

The internal hard disk in the Power Macintosh 5400 computer is an ATA (IDE) drive, not a SCSI drive. This could cause compatibility problems for hard disk utility programs. The system software release for the Power Macintosh 5400 computer includes version 3.0 of the ATA Manager and supports PIO, Singleword DMA, and Multiword DMA data transfers. For more information about the software that controls the ATA drive, see Chapter 7, “Software for the ATA (IDE) Hard Disk.”

CHAPTER 2

Architecture

This chapter describes the architecture of the Power Macintosh 5400 computer. It describes the major components of the main logic board: the microprocessor, the custom ICs, and the display RAM. It also includes a simplified address map.

Block Diagram and Main ICs

The architecture of the Power Macintosh 5400 computer is based on the PowerPC 603e. Figure 2-1 shows the system block diagram. The architecture of the Power Macintosh 5400 computer is based on two buses: the processor bus and the PCI bus. The processor bus connects the microprocessor, video, cache, and memory; the PCI bus connect the expansion slots and the I/O devices.

PowerPC 603e Microprocessor

The Power Macintosh 5400 computer uses a PowerPC 603e microprocessor running at 100 MHz and 120 MHz. The principle features of the PowerPC 603e microprocessor include

- full RISC processing architecture
- parallel processing units: two integer and one floating-point
- a branch manager that can usually implement branches by reloading the incoming instruction queue without using any processing time
- an internal memory management unit (MMU)
- 32 KB of on-chip cache memory (16 KB each for data and instructions)

For complete technical details, see the Motorola *PowerPC 603 RISC Microprocessor User's Manual.* This book is listed in "Supplemental Reference Documents," in the preface.

Memory Subsystem

The memory subsystem of the Power Macintosh 5400 computer consists of ROM and an optional second-level (L2) cache, in addition to the internal cache memory of the PowerPC 603e microprocessor. The PSX custom IC provides burst mode control to the cache and ROM.

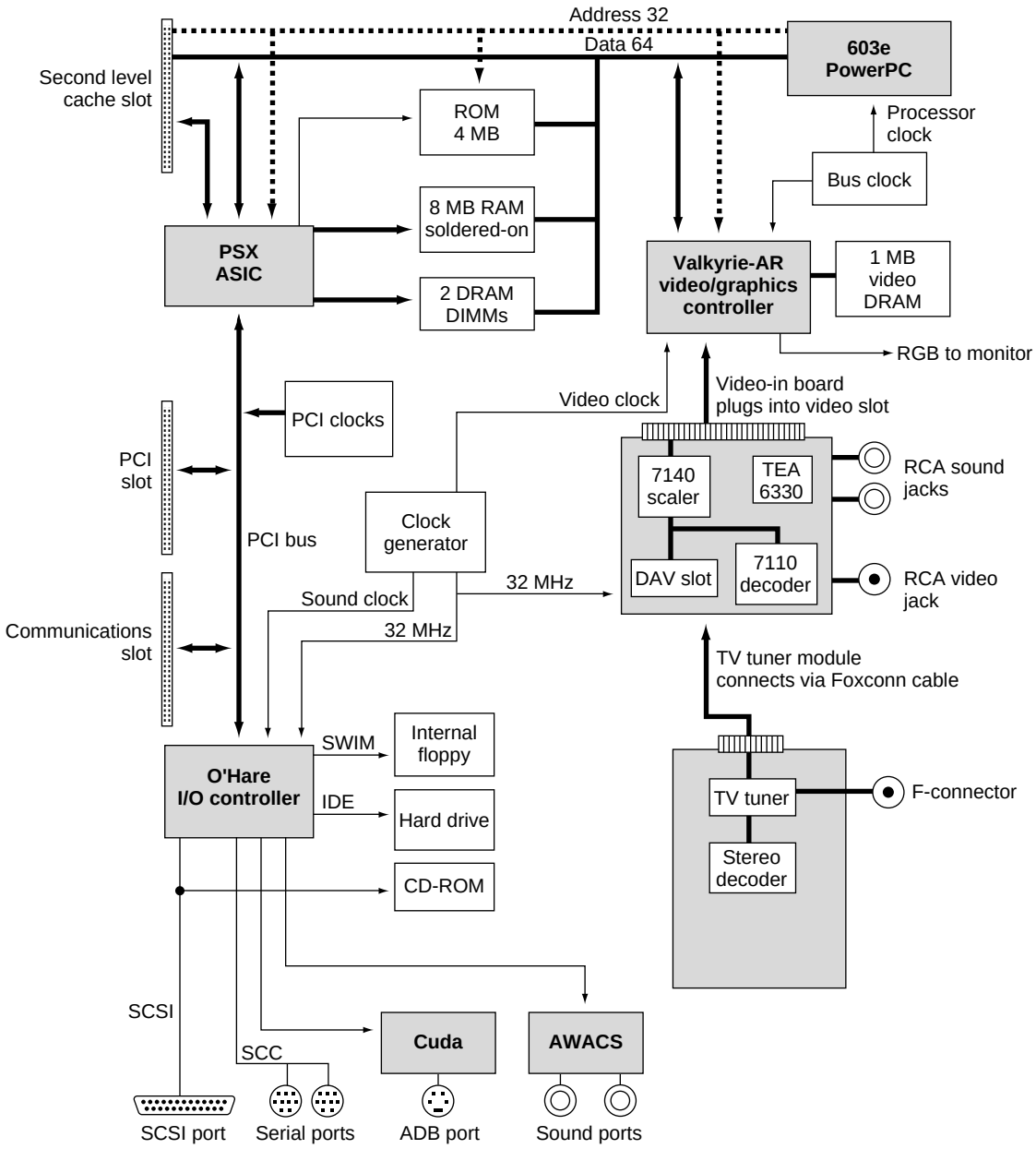
ROM

The ROM consists of 4 MB of masked ROM soldered to the main logic board.

Second Level Cache (Optional)

The optional second-level (L2) cache consists of 256 KB of high-speed RAM on a 160-pin DIMM card, which is plugged into a 160-pin edge connector on the main logic board.

Figure 2-1 System block diagram



System RAM

The Power Macintosh 5400 computer has 8 MB of DRAM memory soldered on the main logic board. All RAM expansion is provided by DRAM devices on 8-byte JEDEC-standard DIMMs (dual inline memory modules). Two 168-pin DIMM sockets are used for memory expansion. Available DIMM sizes are 8, 16, 32, and 64 MB. The DIMM sockets support both single- and double-sided DRAM modules. The PSX custom IC provides memory control for the system RAM.

Custom ICs

The architecture of the Power Macintosh 5400 computer is designed around five large custom integrated circuits:

- the PSX memory controller and PCI bridge
- the O'Hare I/O subsystem and DMA engine
- the AWACS sound processor
- the Cuda ADB controller
- the Valkyrie-AR video subsystem

The computer also uses several standard ICs that are used in other Macintosh computers. This section describes only the custom ICs.

PSX IC

The PSX IC functions as the bridge between the PowerPC 603e microprocessor and the PCI bus. It provides buffering and address translation from one bus to the other.

The PSX IC also provides the control and timing signals for system cache, ROM, and RAM. The memory control logic supports byte, word, long word, and burst accesses to the system memory. If an access is not aligned to the appropriate address boundary, PSX generates multiple data transfers on the bus.

O'Hare IC

The O'Hare IC is based on the Grand Central IC present in the Power Macintosh 7500 computer. It is a an I/O controller and DMA engine for Power Macintosh computers using the PCI bus architecture. It provides power-management control functions for Energy Star–compliant features included in the Power Macintosh 5400 computer. The O'Hare IC is connected to the PCI bus and uses the 32 MHz PCI bus clock.

The O'Hare IC includes circuitry equivalent to the IDE, SCC, SCSI, sound, SWIM3, and VIA controller ICs. The functional blocks in the O'Hare IC include the following:

- support for descriptor-based DMA for I/O devices
- system-wide interrupt handling
- a SWIM3 floppy drive controller

- SCSI controller (MESH based)
- SCC serial I/O controller
- IDE hard disk interface controller
- sound control logic and buffers

The O'Hare IC provides bus interfaces for the following I/O devices:

- Cuda ADB controller IC (VIA1 and VIA2 registers)
- AWACS sound input and output IC
- 8 KB non-volatile RAM control
- PWM outputs for brightness and contrast control on the Power Macintosh 5400

The SCSI controller in the O'Hare IC is a MESH controller. DMA channels in the O'Hare IC are used to support data transfers. In the Power Macintosh 5400 computer, the clock signal to the SCSI controller is 45 MHz.

The O'Hare IC also contains the sound control logic and the sound input and output buffers. There are two DMA data buffers—one for sound input and one for sound output—so the computer can record sound input and process sound output simultaneously. The data buffer contains interleaved right and left channel data for support of stereo sound.

The SCC circuitry in the O'Hare IC is an 8-bit device. The `PCLK` signal to the SCC is an 16 MHz clock. The SCC circuitry supports GeoPort and LocalTalk protocols.

AWACS Sound IC

The audio waveform amplifier and converter (AWACS) is a custom IC that combines a waveform amplifier with a 16-bit digital sound encoder and decoder (codec). It conforms to the IT&T *ASCO 2300 Audio-Stereo Codec Specification* and furnishes high-quality sound input and output. For information about the operation of the AWACS IC, see Chapter 3 of *Developer Note: Power Macintosh Computers*, available on the developer CD-ROM and as part of *Macintosh Developer Note Number 8.*

Cuda IC

The Cuda IC is a custom version of the Motorola MC68HC05 microcontroller. It provides several system functions, including

- the ADB interface
- management of system resets
- maintenance of parameter RAM
- management of the real-time clock
- on/off control of the power supply (soft power)
- the programming interface to devices on the IIC (interintegrated circuit) bus

The devices on the IIC bus include the AWACS sound IC, the digital video decoder and scaler (DESC) on the video input module, and the Cyclops IC, which is the controller for the remote control receiver. The computer reads and writes status and control information to those devices by commands to the Cuda IC.

Valkyrie-AR IC

The Valkyrie-AR IC is a custom IC containing the logic for the video display. It includes the following functions:

- display memory controller
- video CLUT (color lookup table)
- video DAC (digital-to-analog converter)

A separate data bus handles data transfers between the Valkyrie-AR IC and the display memory. The display memory data bus is 32 bits wide, and all data transfers consist of 32 bits at a time. The Valkyrie-AR IC breaks each 32-bit data transfer into several pixels of the appropriate size for the current display mode—4, 8, or 16 bits per pixel. The Valkyrie-AR IC does not support 24 bits per pixel.

To keep up with the large amount of data that must be transferred into and out of the display memory, the Valkyrie-AR IC has several internal buffers. Besides input and output buffers for display data, the Valkyrie-AR IC also has a buffer for both addresses and data being sent from the main processor to the display. That buffer can hold up to four transactions, allowing the main processor to complete a write instruction to the display memory and continue processing without waiting for some other transaction that might be taking place on the display memory bus.

The CLUT in the Valkyrie-AR custom IC provides color palettes for 4-bit and 8-bit display modes. In 16-bit display mode, the CLUT is used to provide gamma correction for the stored color values. With a black-and-white or monochrome display mode, all three color components (R, G, and B) are the same.

The Valkyrie-AR IC uses several clocks. Its transactions with the CPU are synchronized to the system bus clock. Data transfers from the frame-buffer DRAM are clocked by the MEM_CLK signal, which runs at 60 MHz. Data transfers to the CLUT and the video output are clocked by the dot clock, which has a different rate for different display monitors.

For more information about the interaction between the Valkyrie-AR IC, the display memory, and the main processor, see the section “Display RAM” later in this chapter.

Display RAM

The display memory in the Power Macintosh 5400 computer is separate from the main memory. To reduce the cost of the computer, the display memory is implemented with DRAM devices instead of more expensive VRAM devices. The display memory consists of 1 MB of 60 nanosecond (ns) DRAM devices configured to make a 32-bit data bus. The display memory cannot be expanded.

The display memory contains three separate frame buffers. The first frame buffer holds the graphics data—the display that is generated by the computer. The other two frame buffers hold video data from the video input module. The video data frame buffers are used alternately: while one is supplying data to be sent to the video monitor, the other is receiving the next frame of video input.

The display data generated by the computer can have pixel depths of 4, 8, or 16 bits for monitors up to 800 by 600 pixels and 4 or 8 bits for larger monitors up to 832 by 624 pixels. Data from the video input module is always stored and transferred at 16 bits per pixel. The video frame buffers support live video in a 320 by 240-pixel frame at 30 frames per second.

Note
The Power Macintosh 5400 computer cannot display live video from the video-in module on monitor sizes larger than 800 by 600 pixels. Apple Computer, Inc., does not recommend the use of such monitors for these applications. ◆

The Power Macintosh 5400 computer can display video in a window inside the computer graphics display. The Valkyrie-AR IC has registers that contain the starting location of the video window within the display, the starting address of the video data in the video buffer, and the size of the video window.

CHAPTER 3

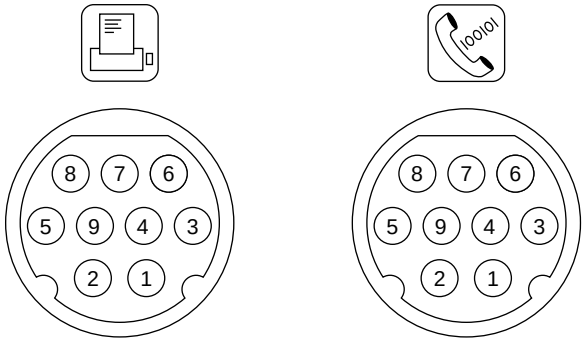
I/O Features

This chapter describes both the built-in I/O devices and the interfaces for external I/O devices. It also describes the types of external video monitors that can be used with the Power Macintosh 5400 computer.

Serial I/O Ports

The Power Macintosh 5400 computer has two serial ports, one for a printer and one for a modem. Both serial ports have 9-pin mini-DIN sockets that accept either 8-pin or 9-pin plugs. The modem port supports the GeoPort serial protocol. Figure 3-1 shows the mechanical arrangement of the pins on the serial port sockets; Table 3-1 shows the signal assignments.

Figure 3-1 Serial port sockets



Printer Modem

Table 3-1 Serial port signals

Pin	Name	Signal description
1	HSKo	Handshake output
2	HSKi	Handshake input (external clock on modem port)
3	TxD–	Transmit data –
4	Gnd	Ground
5	RxD–	Receive data –
6	TxD+	Transmit data +
7	GPi	General-purpose input (wakeup CPU or perform DMA handshake)
8	RxD+	Receive data +
9	+5V	+5 volts to external device (100 mA maximum)

Pin 9 on each serial connector provides +5 V power from the ADB power supply. An external device should draw no more than 100 mA from that pin. The total current available for all devices connected to the +5 V supply for the ADB and the serial ports is 500 mA. Excessive current drain will cause a fuse to interrupt the +5 V supply; the fuse automatically resets when the load returns to normal.

Both serial ports include the GPi (general-purpose input) signal on pin 7. The GPi signal for each port connects to the corresponding data carrier detect input on the SCC portion of the O'Hare custom IC, described in Chapter 2. On serial port A (the modem port), the GPi line can be connected to the receive/transmit clock (RTxCA) signal on the SCC. That connection supports devices that provide separate transmit and receive data clocks, such as synchronous modems. For more information about the serial ports, see *Guide to the Macintosh Family Hardware,* second edition.

ADB Port

The Apple Desktop Bus (ADB) port on the Power Macintosh 5400 computer is functionally the same as on other Macintosh computers.

The ADB is a single-master, multiple-slave serial communications bus that uses an asynchronous protocol and connects keyboards, graphics tablets, mouse devices, and other devices to the computer. The custom ADB microcontroller drives the bus and reads status from the selected external device. A 4-pin mini-DIN connector connects the ADB to the external devices. Table 3-2 lists the ADB connector pin assignments. For more information about the ADB, see *Guide to the Macintosh Family Hardware,* second edition.

Table 3-2 ADB connector pin assignments

Pin number	Name	Description
1	ADB	Bidirectional data bus used for input and output. It is an open-collector signal pulled up to +5 volts through a 470-ohm resistor on the main logic board.
2	PSW	Power-on signal that generates reset and interrupt key combinations.
3	+5V	+5 volts from the computer.
4	GND	Ground from the computer.

Note
The total current available for all devices connected to the +5 V pins on the ADB and the modem port is 500 mA. Each device should use no more than 100 mA. ◆

Disk Drives

The Power Macintosh 5400 computer has one internal high-density floppy disk drive and one internal ATA (IDE) hard disk drive. Some models also have an internal CD-ROM drive.

Floppy Disk Drive

The Power Macintosh 5400 computer has one internal high-density floppy disk drive (Apple SuperDrive). The drive is connected to a 20-pin connector on a cable that is connected to the main logic board by the internal chassis connector. Table 3-3 shows the pin assignments on the floppy disk connector.

Table 3-3 Pin assignments on the floppy disk connector

Pin number	Signal name	Signal description
1	GND	Ground
2	PH0	Phase 0: state control line
3	GND	Ground
4	PH1	Phase 1: state control line
5	GND	Ground
6	PH2	Phase 2: state control line
7	GND	Ground
8	PH3	Phase 3: register write strobe
9	+5V	+5 volts
10	/WRREQ	Write data request
11	+5V	+5 volts
12	SEL	Head select
13	+12V	+12 volts
14	/ENBL	Drive enable
15	+12V	+12 volts
16	RD	Read data

continued

Table 3-3 Pin assignments on the floppy disk connector (continued)

17	+12V	+12 volts
18	WR	Write data
19	+12V	+12 volts
20	n.c.	Not connected

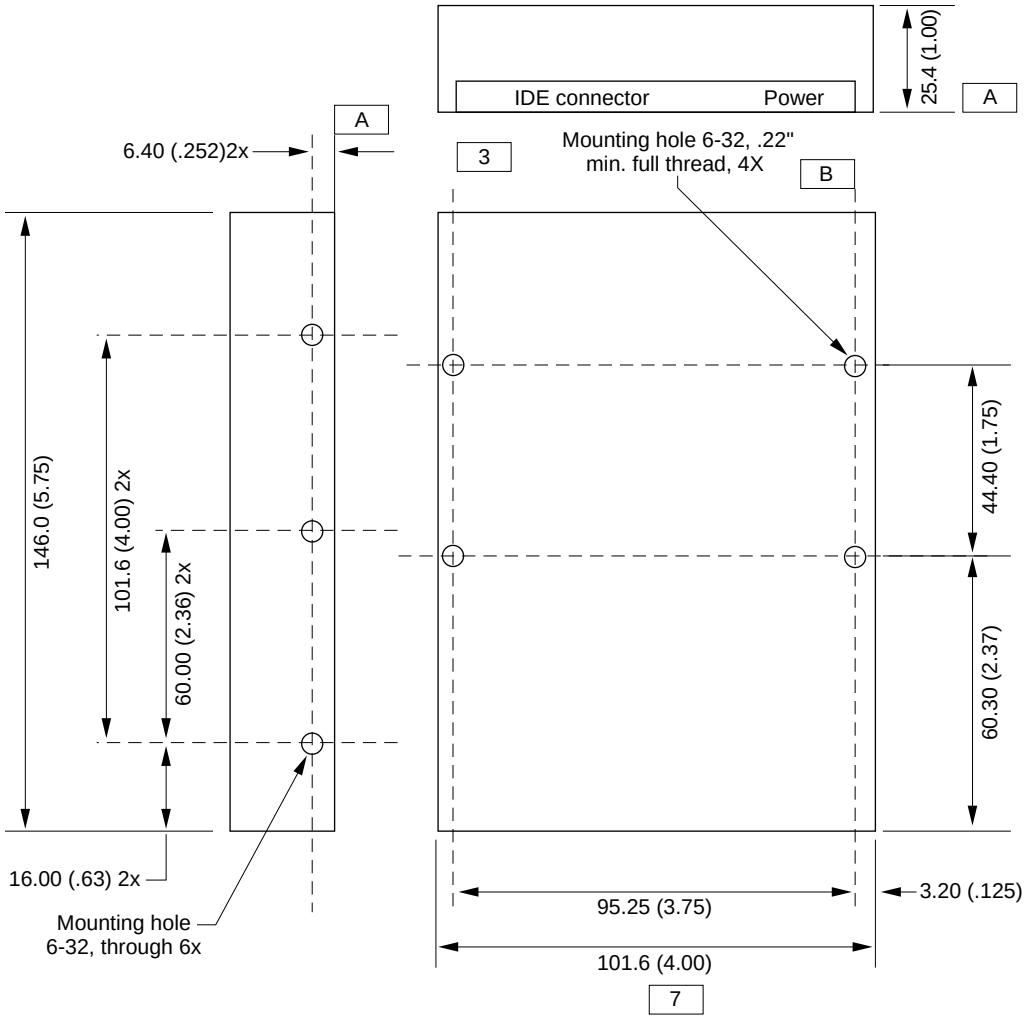
ATA (IDE) Hard Disk

The Power Macintosh 5400 computer has an internal hard disk that uses the standard ATA-2 interface. This interface, used for ATA drives on IBM AT–compatible computers, is also referred to as the IDE interface. The implementation of the ATA interface on the Power Macintosh 5400 computer is a subset of the ATA interface specification, ANSI proposal X3T9.2/90-143, Revision 3.1.

Hard Disk Specifications

Figure 3-2 shows the maximum dimensions of the hard disk and the location of the mounting holes. As the figure shows, the minimum clearance between conductive components and the bottom of the mounting envelope is 0.5 mm.

Figure 3-2 Maximum dimensions of the hard disk



Notes:

1 A Defined by plane of bottom mount holes

2 B Defined by center line of bottom mount holes

3 40-pin IDE and 4-pin power connector placement must not be reversed

4 Dimensions are in millimeters (inches)

5 Drawing not to scale

6 Tolerances .X = ±0.50, .XX = ±0.25

7 Dimension to be measured at center line of side-mount holes

8 Minimum 0.5 MM clearance from any conductive PCB components to A

Hard Disk Connectors

The internal hard disk has a standard 40-pin ATA connector and a separate 4-pin power connector. The 40-pin connector cable is part of the cable harness attached to the main logic board by the internal chassis connector. The power cable is attached directly to the power supply.

The exact locations of the ATA connector and the power connector are not specified, but the relative positions must be as shown in Figure 3-2 so that the cables and connectors will fit.

Pin Assignments

Table 3-4 shows the pin assignments on the 40-pin ATA (IDE) hard disk connector. A slash (/) at the beginning of a signal name indicates an active-low signal.

Table 3-4 Pin assignments on the ATA (IDE) hard disk connector

Pin number	Signal name	Pin number	Signal name
1	/RESET	2	GROUND
3	DD7	4	DD8
5	DD6	6	DD9
7	DD5	8	DD10
9	DD4	10	DD11
11	DD3	12	DD12
13	DD2	14	DD13
15	DD1	16	DD14
17	DD0	18	DD15
19	GROUND	20	Key
21	Reserved	22	GROUND
23	DIOW	24	GROUND
25	DIOR	26	GROUND
27	/IORDY	28	Reserved
29	Reserved	30	GROUND
31	INTRQ	32	/IOCS16
33	DA1	34	/PDIAG
35	DA0	36	DA2
37	/CS0	38	/CS1
39	/DASP	40	GROUND

ATA (IDE) Signal Descriptions

Table 3-5 describes the signals on the ATA (IDE) hard disk connector.

Table 3-5 Signals on the ATA (IDE) hard disk connector

Signal name	Signal description
DA(0–2)	ATA device address; used by the computer to select one of the registers in the ATA drive. For more information, see the descriptions of the CS0 and CS1 signals.
DD(0–15)	ATA data bus; buffered from IOD(16–31) of the computer's I/O bus. DD(0–15) are used to transfer 16-bit data to and from the drive buffer. DD(8–15) are used to transfer data to and from the internal registers of the drive, with DD(0–7) driven high when writing.
/CS0	ATA register select signal. It is asserted high to select the additional control and status registers on the ATA drive.
/CS1	ATA register select signal. It is asserted high to select the main task file registers. The task file registers indicate the command, the sector address, and the sector count.
/IORDY	ATA I/O ready; when driven low by the drive, signals the CPU to insert wait states into the I/O read or write cycles.
/IOCS16	ATA I/O channel select; asserted low for an access to the data port. The computer uses this signal to indicate a 16-bit data transfer.
DIOR	ATA I/O data read strobe.
DIOW	ATA I/O data write strobe.
INTRQ	ATA interrupt request. This active high signal is used to inform the computer that a data transfer is requested or that a command has terminated.
/RESET	Hardware reset to the drive; an active low signal.
Key	This pin is the key for the connector.

CD-ROM Drive

Some configurations of the Power Macintosh 5400 computer have an internal CD-ROM drive, an AppleCD 600i. The AppleCD 600i supports the worldwide standards and specifications for CD-ROM and CD-digital audio discs described in the Sony/Philips Yellow Book and Red Book. The drive can read CD-ROM, CD-ROM XA, CD-I, and PhotoCD discs as well as play standard audio discs.

The AppleCD 600i CD-ROM drive has a sliding tray to hold the disc. The drive features a quadruple-speed mechanism that supports sustained data transfer rates of 600 KB per

second and a data buffer that further enhances performance. Table 3-6 is a summary of the specifications of the CD-ROM drive.

Table 3-6 Specifications of the AppleCD 600i CD-ROM drive

Feature	Specification
Rotation speed	Approximately 920 to 2120 rpm
Average access time	Less than 200 ms
Sustained transfer rate	600 KB per second
SCSI burst rate	More than 3 MB per second

SCSI Bus

The Power Macintosh 5400 computer has a SCSI bus for the internal CD-ROM device and one or more external SCSI devices. The CD-ROM device receives power directly from the power supply.

SCSI Connectors

The SCSI connector for the internal CD-ROM drive is a 50-pin connector with the standard SCSI pin assignments. It attaches to a cable that is connected to the main logic board by the internal chassis connector. The external SCSI connector is a 25-pin D-type connector with the same pin assignments as other Apple SCSI devices. Table 3-7 shows the pin assignments on the internal and external SCSI connectors.

Table 3-7 Pin assignments for the SCSI connectors

Pin number (internal 50-pin)	Pin number (external 25-pin)	Signal name	Signal description
2	8	/DB0	Bit 0 of SCSI data bus
4	21	/DB1	Bit 1 of SCSI data bus
6	22	/DB2	Bit 2 of SCSI data bus
8	10	/DB3	Bit 3 of SCSI data bus
10	23	/DB4	Bit 4 of SCSI data bus
12	11	/DB5	Bit 5 of SCSI data bus
14	12	/DB6	Bit 6 of SCSI data bus

continued

Table 3-7 Pin assignments for the SCSI connectors (continued)

Pin number (internal 50-pin)	Pin number (external 25-pin)	Signal name	Signal description
16	13	/DB7	Bit 7 of SCSI data bus
18	20	/DBP	Parity bit of SCSI data bus
25	–	n.c.	Not connected
26	25	TPWR	+5 V terminator power
32	17	/ATN	Attention
36	6	/BSY	Bus busy
38	5	/ACK	Handshake acknowledge
40	4	/RST	Bus reset
42	2	/MSG	Message phase
44	19	/SEL	Select
46	15	/C/D	Control or data
48	1	/REQ	Handshake request
50	3	/I/O	Input or output
20, 22, 24, 28, 30, 34, and all odd pins except pin 25	7, 9, 14, 16, 18, and 24	GND	Ground

SCSI Bus Termination

The internal end of the SCSI bus is terminated by an active terminator. The terminator is located on the main logic board near the portion of the internal chassis connector that contains the signals for the internal CD-ROM drive. On enclosures with only one internal SCSI device located close to the logic board, the active termination is automatically enabled. On enclosures with multiple SCSI devices, the active termination is disabled, and a positive terminator is located at the end of the internal bus.

Sound

The sound system supports both 8-bit and 16-bit stereo sound output and input. Like other Macintosh computers, the Power Macintosh 5400 computer can create sounds digitally and play the sounds through the internal speakers or send the sound signals out through the sound output jacks. They can also record sound from several sources: the built-in microphone, a microphone connected to the sound input jack, the video input module, or a compact disc in the CD-ROM player.

Sound Output

The Power Macintosh 5400 computer has two built-in speakers and two sound output jacks, one on the front and one on the back. Both output jacks are connected to the sound amplifier; the jack on the front is intended for ease of access when connected to a pair of headphones. Inserting a plug into either jack disconnects the internal speakers. The rear jack is intended for use with external speakers and it is muted when headphones are plugged in the front jack. (Options in the Monitors and Sound control panel can be used to determine the interaction between the sound input and output devices.)

Sound output is controlled by the O'Hare IC. The AWACS IC provides the stereo sound output to both the internal speakers and the sound output jacks.

Sound Input

The Power Macintosh 5400 computer has a stereo sound input jack on the back for connecting an external microphone or other sound source. The sound input jack accepts a standard 1/8-inch stereophonic phone plug (two signals plus ground).

The sound input jack accepts either the Apple PlainTalk line-level microphone or a pair of line-level signals.

Note
The Apple PlainTalk microphone requires power from the main computer, which it obtains by way of an extra-long, 4-conductor plug that makes contact with a 5-volt pin inside the sound input jack. ◆

IMPORTANT
The microphone for the Macintosh LC and LC II does not work with the Power Macintosh 5400 computer; it requires the line-level signal provided by the Apple PlainTalk microphone. ▲

Sound Input Specifications

The sound input jack has the following electrical characteristics:

- input impedance: 15k ohms
- maximum input level: 1.06 V RMS

Routing of the Sound Signals

All audio sources are routed to the AWACS custom IC, which can enable them in three groups: internal microphone, sound input jack, and (CD-ROM, TV tuner, modem, DAV card, cross-platform card).

Digitizing Sound

The Power Macintosh 5400 computer digitizes and records sound as 16-bit samples. The computer can use either of two sampling rates: 11k samples per second, 22k samples per second, and 44k samples per second.

The sound system plays samples at the sampling rate specified in the control panel for sound.

Sound Modes

The sound mode is selected by a call to the Sound Manager. The sound circuitry normally operates in one of three modes:

- Sound playback: computer-generated sound is sent to the speaker and the sound output jacks.
- Sound playback with playthrough: computer sound and sound input are mixed and sent to the speakers and the sound output jacks.
- Sound record with playthrough: input sound is recorded and also fed through to the speakers and the sound output jacks.

When recording from a microphone, applications should reduce the playthrough volume to prevent possible feedback from the speakers to the microphone.

The O'Hare IC provides separate sound buffers for input and for stereo output, so the computer can record and send digitized sound to the sound outputs simultaneously.

Keyboard

The keyboard has a Power key, identified by the symbol ⏻. When the user chooses Shut Down from the Special menu, the computer either shuts down or a dialog appears asking if you really want to shut down. The user can also turn off the power by pressing the Power key.

There are no programmer's switches, so the user invokes the reset and nonmaskable interrupt (NMI) functions by pressing Command key combinations while holding down the Power key, as shown in Table 3-8. The Command key is identified by the symbols and ⌘.

Note
The user must hold down a key combination for at least 1 second to allow the ADB microcontroller enough time to respond to the NMI or hard-reset signal. ◆

Note
The NMI function can always be activated from the keyboard. This is a change from the Macintosh LC computer, where keyboard activation of the NMI function can be disabled by the software. ◆

Table 3-8 Reset and NMI key combinations

Key combination	Function
Command-Power (⌘-◁)	NMI (always active)
Control-Command-Power (Control-⌘-◁)	Reset

Built-in Video

The Power Macintosh 5400 computer has a built-in 15-inch multiscan monitor. The built-in video circuitry support pixel display sizes of 512 × 384, 640 × 480, 800 × 600, and 832 × 624. When power is applied, the monitor is initially set for a display size of 640 × 480 pixels. The user can switch the monitor resolution on the fly from the computer's Control Panels menu.

Optional Video Display Mirror Output Feature

The Power Macintosh 5400 uses a feature, called video display mirror output, to make the video information on its built-in monitor available to an external monitor. This means that the information displayed on an external monitor is exactly the same as that displayed on the built-in monitor. This feature is implemented by plugging an optional video buffer board into the 22-pin Video Mirror connector on the main logic board. The Video Mirror connector's pin assignments are shown in Table 3-9.

Note
The external monitor must support the same video mode selected on the built-in monitor, and no attempt is made to read the MONID lines of the external monitor to determine what monitor is attached. ◆

The optional video buffer board includes a ribbon cable with a DB-15 connector. This connector attaches to a large opening in the upper part of the computer's back panel, identified in Figure 1-2 on page 6, as the Monitor Out port. The cable from an external video monitor plugs into this DB-15 connector to allow the external monitor to display the same image as the built-in monitor.

Table 3-9 Video mirror connector pin assignments

Pin	Signal name	Description
1	VID GND	Video ground
2	RED	Red signal
3	GREEN	Green signal

continued

Table 3-9 Video mirror connector pin assignments (continued)

Pin	Signal name	Description
4	VID GND	Video ground
5	VID GND	Video ground
6	BLUE	Blue signal
7	CSYNC	C sync
8	VSYNC	Vertical sync
9	MLB.SYNC.EN.L	Not used (reserved)
10	HSYNC	Horizontal sync
11	DAC.ISET.1	Not used(reserved)
12	DAC.ISET.2	Not used (reserved)
13	SND GND	Not used (reserved)
14	SND RIGHT	Not used (reserved)
15	SND LEFT	Not used (reserved)
16	+5V	+5 volts
17	GND	Ground
18	SDAT	Not used (reserved)
19	SCLK	Not used (reserved)
20	+12V	+12 volts
21	–12V	–12 volts
22	Dot Clock	Scaled dot clock (scaled to 10 percent)

External Video Monitors

The computer can work with several sizes of external video monitors; however, you can connect an external monitor to the Power Macintosh 5400 only if the optional video display mirror out feature is implemented on that computer, and then it can display only the same video as the internal monitor. Table 3-10 shows the monitor types supported and the maximum pixel depths available. The pixel depth determines the maximum number of colors that can be displayed. The maximum pixel depth available depends on the size of the monitor's screen.

For more information about the video monitors, see "Video Timing Parameters" on page 37.

Table 3-10 Maximum pixel depths for video monitors

Monitor type	Screen size, in pixels	Maximum pixel depth, in bits per pixel	Maximum number of colors displayed
12-inch color*	512 by 384	16	32,768
14-inch color	640 by 480	16	32,768
15-inch multiscan	800 by 600	16	32,768
VGA	640 by 480	8	256
SVGA	800 by 600	16	256
16-inch color	832 by 624	8	256

* The Power Macintosh 5400 computer does not support screen sizes of 512 by 384 pixels.

Video Timing Parameters

The Power Macintosh 5400 computer supports several different types of monitors and screen sizes, as listed in Table 3-10.

Table 3-11 Monitors supported

Monitor type	Screen size (pixels)
12-inch color	512 by 384
14-inch color	640 by 480
15-inch multiscan	800 by 600
VGA	640 by 480
SVGA	800 by 600
16-inch color	832 by 624

Figure 3-3 shows simplified timing diagrams and identifies the horizontal and vertical timing parameters in a video signal. Table 3-13 and Table 3-13 list the values of those parameters for the different types of monitors.

Figure 3-3 Video timing diagram

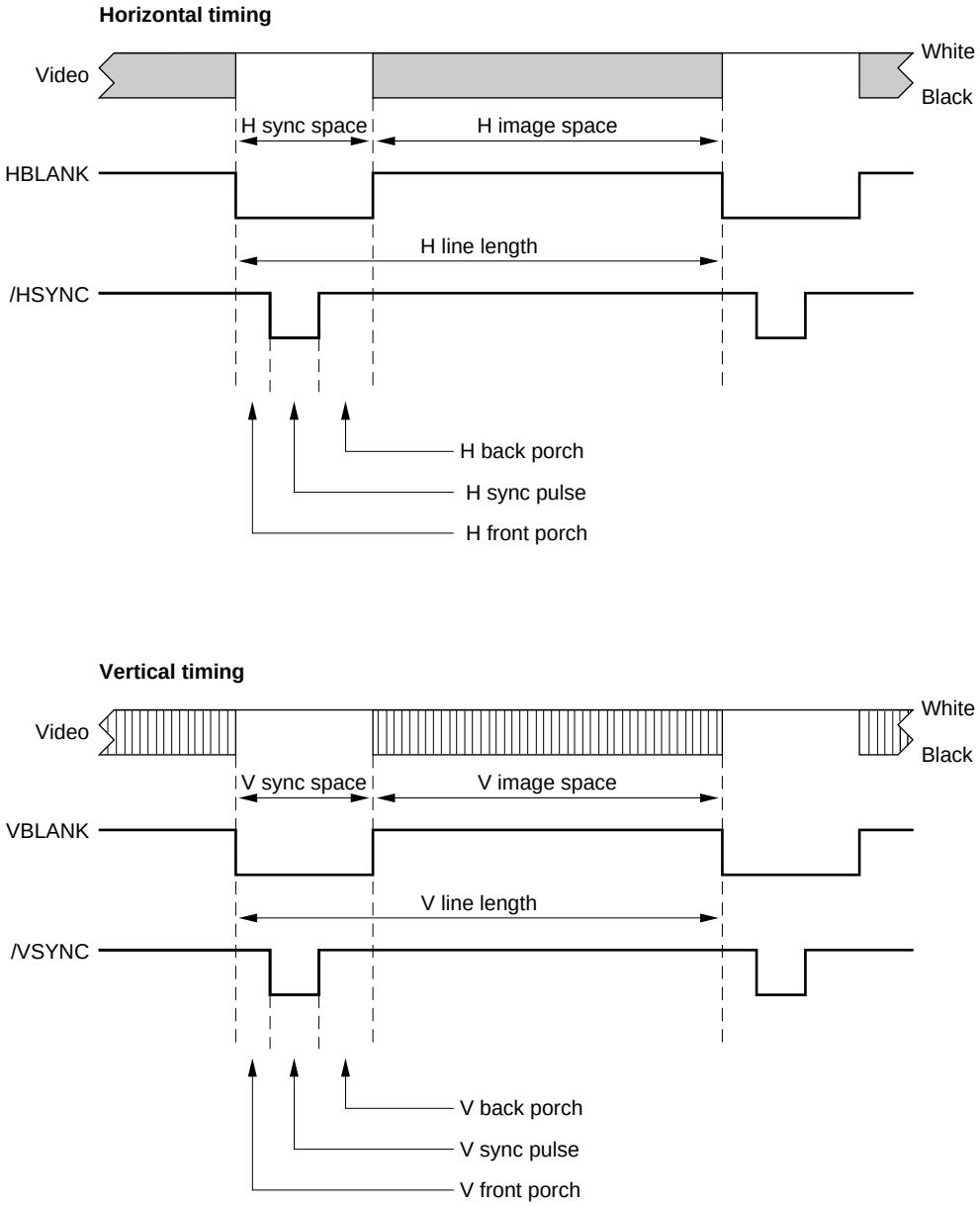


Table 3-13 lists the timing parameters for the smaller monitors listed: the 12-inch color monitor, the 14-inch color monitor, and a standard VGA monitor.

Table 3-12 Video timing parameters for smaller monitors

	Monitor type and dimensions		
Parameter	12-inch color (512 by 384)	14-inch color (640 by 480)	VGA (640 by 480)
Dot clock	15.67 MHz	30.24 MHz	25.18 MHz
Dot time	63.83 ns	33.07 ns	39.72 ns
Line rate	24.48 kHz	35.00 kHz	31.47 kHz
Line time	40.85 µs (640 dots)	28.57 µs (864 dots)	31.78 µs (800 dots)
Horizontal active video	512 dots	640 dots	640 dots
Horizontal blanking	128 dots	224 dots	160 dots
Horizontal front porch	16 dots	64 dots	16 dots
Horizontal sync pulse	32 dots	64 dots	96 dots
Horizontal back porch	80 dots	96 dots	48 dots
Frame rate	60.15 Hz	66.67 Hz	59.94 Hz
Frame time	16.63 ms (407 lines)	15.01 ms (525 lines)	16.68 ms (525 lines)
Vertical active video	384 lines	480 lines	480 lines
Vertical blanking	23 lines	45 lines	45 lines
Vertical front porch	1 line	3 lines	10 lines
Vertical sync pulse	3 lines	3 lines	2 lines
Vertical back porch	19 lines	39 lines	33 lines

Table 3-13 lists the timing parameters for SVGA monitors running at 60 and 72 frames per second, and for the 16-inch color monitor.

Table 3-13 Video timing parameters for larger monitors

	Monitor type and dimensions		
Parameter	SVGA (800 by 600 at 60 frames per second)	SVGA (800 by 600 at 72 frames per second)	16-inch color (832 x 624)
Dot clock	40.00 MHz	50.00 MHz	57.2832 MHz
Dot time	25.00 ns	20.00 ns	17.46 ns
Line rate	37.88 kHz	48.08 kHz	49.725 kHz
Line time	26.4 µs (1056 dots)	20.80 µs (1040 dots)	20.11 µs (1152 dots)
Horizontal active video	800 dots	800 dots	832 dots
Horizontal blanking	256 dots	240 dots	320 dots
Horizontal front porch	40 dots	56 dots	32 dots
Horizontal sync pulse	128 dots	120 dots	64 dots
Horizontal back porch	88 dots	64 dots	224 dots
Frame rate	60.31 Hz	72.18 Hz	74.55 Hz
Frame time	16.58 ms (628 lines)	13.85 ms (666 lines)	13.41 ms (667 lines)
Vertical active video	600 lines	600 lines	624 lines
Vertical blanking	28 lines	66 lines	43 lines
Vertical front porch	1 line	37 lines	1 line
Vertical sync pulse	4 lines	6 lines	3 lines
Vertical back porch	23 lines	23 lines	39 lines

CHAPTER 4

Expansion Features

This chapter describes the expansion features of the Power Macintosh 5400 computer: the RAM expansion slot, the L2 cache expansion slot, the PCI expansion slot, the DAV connector on the video input module, and the communications slot.

Note
Apple does not support development of third-party cards for the video input slot, nor does Apple support development of third-party level-2 (L2) cache cards, because the L2 cache controller is integrated into the design of the cache card. ◆

RAM DIMMs

The Power Macintosh 5400 computer has two RAM expansion slots. The RAM expansion slots accept a new type of memory module: the 8-byte DIMM (dual inline memory module). As its name implies, the 8-byte DIMM has a 64-bit-wide data bus.

The 8-byte DIMM is a new industry standard. Its mechanical design is defined by the MO-161 specification published by the JEDEC JC-11 committee; its electrical characteristics are defined by the JEDEC Standard No. 21-C. The 8-byte DIMM connector used in the Power Macintosh 5400 computer is Burndy Corporation's part number ELF168E5GC-3Z50 or equivalent.

The minimum bank size supported by the PSX IC is 4 MB and the largest is 32 MB; the largest DIMM supported is a two-bank DIMM holding 64 MB. Table 4-1 shows the single-bank DIMM configurations and sizes for a range of DRAM device sizes that are supported on the Power Macintosh 5400 computer.

Table 4-1 Memory sizes and configurations

Device size	DIMM configuration	DIMM size	Maximum memory with 2 DIMMs installed
4 Mbit	512K by 64	4 MB	16 MB
4 Mbit	1 Mbit by 64	8 MB	24 MB
16 Mbit	1 Mbit by 64	8 MB	24 MB
16 Mbit	2 Mbits by 64	16 MB	40 MB
16 Mbit	4 Mbits by 64	32 MB	72 MB

The 8-byte DIMMs can be installed one or more at a time. The Power Macintosh 5400 computer supports only linear memory organization, therefore no performance gains are seen when two DIMMs of the same size are installed. Any size DIMM can be installed in either DIMM slot, and the combined memory of all of the DIMMs installed will be configured as a contiguous memory space.

RAM DIMM Connectors

Table 4-2 gives the pin assignments for the RAM DIMM connectors.

Table 4-2 Pin assignments on the RAM DIMM connectors

Pin number	Signal name	Pin number	Signal name
1	VSS	85	VSS
2	DQ(0)	86	DQ(32)
3	DQ(1)	87	DQ(33)
4	DQ(2)	88	DQ(34)
5	DQ(3)	89	DQ(35)
6	VCC	90	VCC
7	DQ(4)	91	DQ(36)
8	DQ(5)	92	DQ(37)
9	DQ(6)	93	DQ(38)
10	DQ(7)	94	DQ(39)
11	Reserved	95	Reserved
12	VSS	96	VSS
13	DQ(8)	97	DQ(40)
14	DQ(9)	98	DQ(41)
15	DQ(10)	99	DQ(42)
16	DQ(11)	100	DQ(43)
17	DQ(12)	101	DQ(44)
18	VCC	102	VCC
19	DQ(13)	103	DQ(45)
20	DQ(14)	104	DQ(46)
21	DQ(15)	105	DQ(47)
22	Reserved	106	Reserved
23	VSS	107	VSS

continued

Table 4-2 Pin assignments on the RAM DIMM connectors (continued)

Pin number	Signal name	Pin number	Signal name
24	Reserved	108	Reserved
25	Reserved	109	Reserved
26	VCC	110	VCC
27	/WE(0)	111	Reserved
28	/CAS(0)	112	/CAS(1)
29	/CAS(2)	113	/CAS(3)
30	/RAS(0)	114	/RAS(1)
31	/OE(0)	115	Reserved
32	VSS	116	VSS
33	A(0)	117	A(1)
34	A(2)	118	A(3)
35	A(4)	119	A(5)
36	A(6)	120	A(7)
37	A(8)	121	A(9)
38	A(10)	122	A(11)
39	Not connected	123	Not connected
40	VCC	124	VCC
41	Reserved	125	Reserved
42	Reserved	126	B(0)
43	VSS	127	VSS
44	/OE(2)	128	Reserved
45	/RAS(2)	129	/RAS(3)
46	/CAS(4)	130	/CAS(5)
47	/CAS(6)	131	/CAS(7)
48	/WE(2)	132	/PDE
49	VCC	133	VCC
50	Reserved	134	Reserved
51	Reserved	135	Reserved
52	DQ(16)	136	DQ(48)
53	DQ(17)	137	DQ(49)
54	VSS	138	VSS

continued

Table 4-2 Pin assignments on the RAM DIMM connectors (continued)

Pin number	Signal name	Pin number	Signal name
55	DQ(18)	139	DQ(50)
56	DQ(19)	140	DQ(51)
57	DQ(20)	141	DQ(52)
58	DQ(21)	142	DQ(53)
59	VCC	143	VCC
60	DQ(22)	144	DQ(54)
61	Reserved	145	Reserved
62	Reserved	146	Reserved
63	Reserved	147	Reserved
64	Reserved	148	Reserved
65	DQ(23)	149	DQ(55)
66	Reserved	150	Reserved
67	DQ(24)	151	DQ(56)
68	VSS	152	VSS
69	DQ(25)	153	DQ(57)
70	DQ(26)	154	DQ(58)
71	DQ(27)	155	DQ(59)
72	DQ(28)	156	DQ(60)
73	VCC	157	VCC
74	DQ(29)	158	DQ(61)
75	DQ(30)	159	DQ(62)
76	DQ(31)	160	DQ(63)
77	Reserved	161	Reserved
78	VSS	162	VSS
79	PD(1)	163	PD(2)
80	PD(3)	164	PD(4)
81	PD(5)	165	PD(6)
82	PD(7)	166	PD(8)
83	ID(0)	167	ID(1)
84	VCC	168	VCC

Table 4-3 describes the signals on the RAM DIMM connector.

Table 4-3 RAM DIMM signals

Signal name	Description
A(0–11)	Address inputs
/CAS(0–7)	Column address strobe signals
DQ(0–63)	Data input and output signals
ID(0–1)	Memory module identification (not used)
/OE(0, 2)	Output enable signals
PD(1–8)	Presence detect signals
/PDE	Presence detect enable signal (not used)
/RAS(0-3)	Row address strobe signals
Reserved	Reserved, don't use.
VCC	+5 V power
VSS	Ground
/WE(0, 2)	Read/write input signals

RAM Address Multiplexing

Signals A[0–11] on each RAM DIMM make up a 12-bit multiplexed address bus that can support several different types of DRAM devices. Table 4-4 shows the address multiplexing modes used with several types of DRAM devices. The devices are characterized by their bit dimensions: for example, a 256K by 4-bit device has 256K addresses and stores 4 bits at a time.

Table 4-4 Address multiplexing modes for various DRAM devices

Device size	Device type	Size of row address	Size of column address
4 Mbits	512 K by 8 bits	10	9
4 Mbits	1 M by 4 bits	10	10
16 Mbits	1 M by 16 bits	10	10
16 Mbits	2 M by 8 bits	11	10
16 Mbits	2 M by 8 bits	12	9
16 Mbits	4 M by 4 bits	11	11

Table shows how the address signals to the RAM devices are multiplexed during the row and column address phases for noninterleaved banks.

Table 4-5 Address multiplexing in noninterleaved banks

	Individual signals on the DRAM_ADDR bus											
	A(11)	**A(10)**	**A(9)**	**A(8)**	**A(7)**	**A(6)**	**A(5)**	**A(4)**	**A(3)**	**A(2)**	**A(1)**	**A(0)**
Row address	22	23	21	20	19	18	17	16	15	14	13	12
Column address		24	22	11	10	9	8	7	6	5	4	3

IMPORTANT

The PSX DRAM controller on the main logic board of the Power Macintosh 5400 computer does not provide support for 4 M by 4 bits (12 by 10 addressing) or 1 M by 16 bits (12 by 8 addressing) DRAM devices. ▲

RAM Devices

The memory controller in the PSX IC supports 1 MB, 4 MB, and 16 MB DRAM devices. The access time (T_{RAS}) of the DRAM devices is 70 ns or faster.

Note

The computer supplies +5 volts at VCC on the RAM expansion slot for DRAM DIMMs. Power for DRAM devices that require 3.3 volts is not supplied on the RAM expansion slot. ◆

RAM Refresh

The PSX IC provides a CAS-before-RAS refresh cycle every 15.6 µs. DRAM devices must be compatible with this refresh cycle; for example, this cycle will refresh 2K-refresh parts within 32 milliseconds.

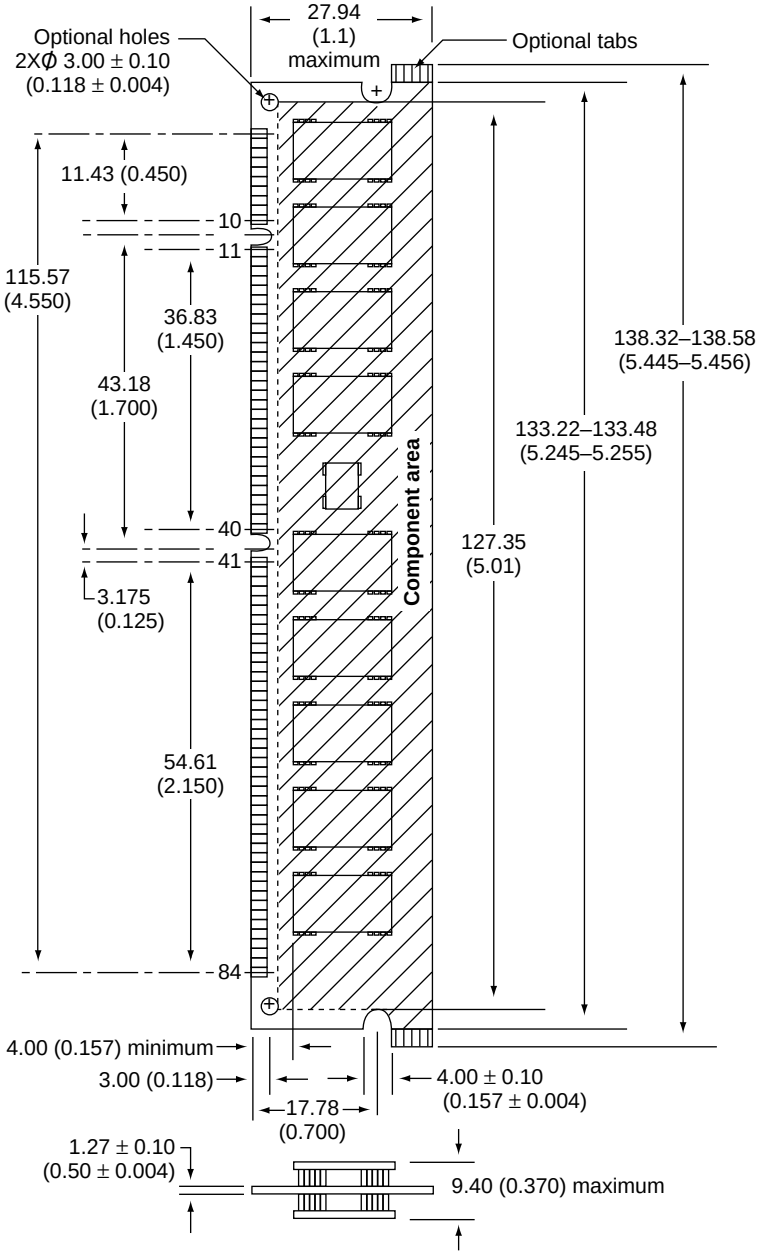
RAM DIMM Dimensions

Figure 4-1 shows the dimensions of the RAM DIMM.

IMPORTANT

The JEDEC MO-161 specification shows three possible heights for the 8-byte DIMM. For Power Macintosh computers, developers should use only the shortest of the three: 1.100 inches. Taller DIMMs put excessive pressure on the DIMM sockets due to possible mechanical interference inside the case. ▲

Figure 4-1 Dimensions of the RAM DIMM



Note: dimensions are in millimeters (inches)

Level-2 Cache DIMM

The Power Macintosh 5400 computer has a slot for a level-2 (L2) cache on a DIMM.

The L2 cache DIMM contains the cache controller, tag, and data store memory. It is a lookaside cache, which is connected to the PowerPC processor bus. Several signals are also included to control cache operation. These signals include: `/L2_DIS`, `/MEM_INHIBIT`, `/L2_BR`, `/L2_BG`, and `L2_PRSNT`.

Table 4-6 shows the pin and signal assignments on the L2 cache DIMM connector.

Table 4-6 Pin and signal assignments for level-2 cache DIMM connector

Pin	Signal name	Pin	Signal name	Pin	Signal name	Pin	Signal name
1	+5 V	41	A15	81	D63 (LSB)	121	A16
2	D31	42	A13	82	D62	122	A14
3	D30	43	+3.3 V	83	D61	123	A12
4	D29	44	A11	84	GND	124	A10
5	D28	45	A9	85	D60	125	A8
6	D27	46	A7	86	D59	126	GND
7	+5 V	47	A5	87	D58	127	A6
8	D26	48	A3	88	D57	128	A4
9	D25	49	+3.3 V	89	D56	129	A2
10	D24	50	A1	90	GND	130	A0 (MSB)
11	D23	51	/WT	91	D55	131	/DBB
12	D22	52	/GBL	92	D54	132	GND
13	+5 V	53	reserved	93	D53	133	/CPU_BG
14	D21	54	/SRESET	94	D52	134	/CPU_BR
15	D20	55	+3.3 V	95	D51	135	L2_PRSNT
16	D19	56	TTYPE0	96	GND	136	reserved
17	D18	57	TTYP1	97	D50	137	TSIZ0
18	D17	58	TTYPE2	98	D49	138	GND
19	+5 V	59	TTYPE3	99	D48	139	TSIZ1
20	D16	60	TTYPE4	100	/L2_DIS	140	TSIZ2
21	/L2_BR	61	+3.3 V	101	/TBST	141	SHD

continued

Table 4-6 Pin and signal assignments for level-2 cache DIMM connector (continued)

Pin	Signal name	Pin	Signal name	Pin	Signal name	Pin	Signal name
22	/L2_BG	62	D15	102	GND	142	D47
23	TC0	63	D14	103	/CI	143	D46
24	TC1	64	D13	104	/RSRV	144	GND
25	+3.3 V	65	D12	105	reserved	145	D45
26	/HRESET	66	D11	106	/MEM_INHIBIT	146	D44
27	/TEA	67	+5 V	107	/AACK	147	D43
28	/TS	68	D10	108	GND	148	D42
29	GND	69	D9	109	/TA	149	D41
30	SYS_CLK	70	D8	110	/ARTRY	150	GND
31	+3.3 V	71	D7	111	/ABB	151	D40
32	A31 (LSB)	72	D6	112	A30	152	D39
33	A29	73	+5 V	113	A28	153	D38
34	A27	74	D5	114	GND	154	D37
35	A25	75	D4	115	A26	155	D36
36	A23	76	D3	116	A24	156	GND
37	+3.3 V	77	D2	117	A22	157	D35
38	A21	78	D1	118	A20	158	D34
39	A19	79	+5 V	119	A18	159	D33
40	A17	80	D0 (MSB)	120	GND	160	D32

Table 4-7 defines the signals on the level-2 cache DIMM connector.

Table 4-7 Signal descriptions for level-2 cache DIMM connector

Signal name	Description
+5 V	Power supply voltage of +5 volts for tag RAM (5% tolerance)
+ 3.3 V	Power supply voltage of +3.3 volts for data RAM (5% tolerance)
GND	Ground
A(0-31)	Processor address bus signals 0 through 31
D(0-63)	Processor data bus signals 0 through 63; sampled on the rising edge of the CLK signal during a write cycle
/AACK	Address acknowledge, same as AACK_ signal on PowerPC 603

continued

Table 4-7 Signal descriptions for level-2 cache DIMM connector (continued)

Signal name	Description
/ARTRY	Address retry, same as ARTRY_ signal on PowerPC 603
/ABB	Address bus busy, same as ABB_ signal on PowerPC 603
/CI	Cache inhibit, same as CI_ signal on PowerPC 603
/CPU_BG	Bus transaction granted, same as BG_ signal on PowerPC 603
/CPU_BR	Bus transaction requested, same as BR_ signal on PowerPC 603
/DBB	Data bus busy, same as DBB_ signal on PowerPC 603
/GBL	Global transaction
/HRESET	Main logic board hardware reset
/L2_BG	Bus grant to L2 cache; used only in copyback mode
/L2_BR	Bus request from L2 cache; used only in copyback mode
/L2_DIS	Disables cache when low; contents are invalidated
L2_PRSNT	L2 cache present; tied directly to power rail on cache DIMM
/MEM_INHIBIT	Indicates L2 cache will source the data for the current cycle. Inhibits main logic board memory controller.
/RSRV	Reservation signal, same as RSRV_ signal on PowerPC 603
reserved	DO NOT USE
SHD	Share
/SRESET	Soft reset, same as SRESET_ signal on PowerPC 603
SYS_CLK	System clock, same as SYSCLOCK signal on PowerPC 603
/TA	Transfer acknowledge, same as TA_ signal on PowerPC 603
/TBST	Transfer burst in progress, same as TBST_ signal on PowerPC 603
TC(0-1)	Transfer code, same as TC signal on PowerPC 603
/TEA	Transfer error acknowledge, same as TEA_ signal on PowerPC 603
/TS	Transfer start signal, same as TS_ signal on PowerPC 603
TSIZ (0-2)	Transfer size for the data transaction
TTYPE(0-4)	Transfer type, same as TT signal on PowerPC 603
/WT	Write-thru, same as WT_ signal on PowerPC 603

PCI Expansion Slot

The Power Macintosh 5400 computer uses the industry-standard peripheral component interconnect (PCI) bus for an I/O expansion bus. The PCI bus is a 32-bit multiplexed address and data bus. The PCI expansion slot has a 33.33 MHz system clock.

PCI I/O expansion cards are mounted horizontally in a 90-degree straight-through adapter board, which is installed in the PCI expansion slot on the main logic board.

A total of 15 watts of power is provided for the PCI expansion slot. Both 5 volts and 3.3 volts are supplied; the total power consumed by both voltages must not exceed the 15-watts maximum.

The Power Macintosh 5400 computer requires that PCI cards use the 5-volts signaling standard described in the *PCI Local Bus Specification*, Revision 2.0.

The Power Macintosh 5400 computer accepts standard 6.88-inch PCI cards as defined by the *PCI Local Bus Specification*, Revision 2.0. The cards are required to use the standard ISA fence described in the specification.

The PCI slots support all the required PCI signals and certain optional PCI signals. The supported PCI signals are listed in Table 4-3.

Table 4-8 PCI signals

Signal name	Description
AD [0–31]	Address and data, multiplexed
C/BE[0–3]	Bus command and byte enable signals, multiplexed
PAR	Parity; used with AD and C/BE signals
FRAME#	Cycle frame; asserted to indicate a bus transaction
TRDY#	Target ready; selected device is able to complete the current phase
IRDY#	Initiator ready; master device is able to complete the current phase
STOP#	Stop; indicates the current target device is requesting the master to stop the current transaction
DEVSEL#	Device select; indicates that the driving device has decoded its address as the target of the current access
IDSEL	Initialization device select; used during configuration
REQ#	Request; indicates to the arbiter that the asserting agent requires use of the bus
GNT#	Grant; indicates to the agent that access to the bus has been granted
CLK	Clock; rising edge provides timing for all transactions

continued

Table 4-8 PCI signals (continued)

Signal name	Description
RST#	Reset; used to bring registers and signals to a known state
INTA#, INTB#, INTC#, INTD#	Interrupt request pins; wired together on each slot
LOCK#	Lock; indicates an operation that may require multiple transactions to complete.
PERR#	Parity error; used to report data parity errors during PCI transactions excluding a Special Cycle transaction.
SERR#	System error; used to report address parity errors, data parity errors during a Special Cycle, or any other system error that will be catastrophic.

The PCI slot in the Power Macintosh 5400 computer does not support the optional 64-bit bus extension signals or cache support signals.

For more information about the PCI expansion slot, refer to *Designing PCI Cards and Drivers for Power Macintosh Computers.*

The DAV Connector

The optional video input card has a separate connector called the DAV (digital audio video) connector. The DAV connector provides access to the video input card's 4:2:2 unscaled YUV video input data bus and associated control signals. By means of a cable to the DAV connector, a PCI expansion card can gain access to the digital video bus on the video input card and use it to transfer real-time video data to the computer. Such a PCI expansion card can contain a hardware video compressor or other video processor.

The DAV connector is a 60-pin flat ribbon connector located at the top edge of the video input card. Figure 4-2 is a view of the main logic board showing the PCI expansion card and the location of the DAV connector on the video input card.

Note
The interface of the 60-pin DAV connector is a superset of the interface on the 34-pin DVA connector on the Power Macintosh 5200, Power Macintosh 6200, and Quadra 630 computers. An adapter cable is provided with the Power Macintosh 5400 video-in cards to connect 34-pin DVA compatible cards developed for the Power Macintosh 5200 and 6200 computers to the new 60-pin DAV connector. ◆

The DAV connector accepts YUV video and analog sound from the expansion card but does not itself generate YUV video output or audio output signals.

Figure 4-2 Location of the DAV connector

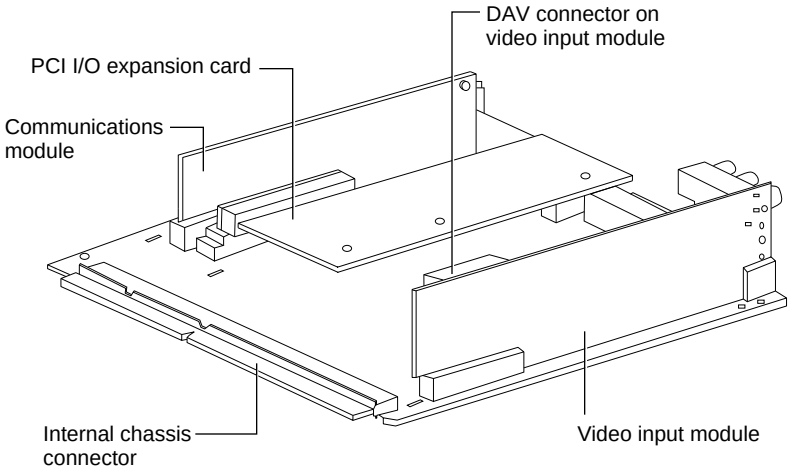
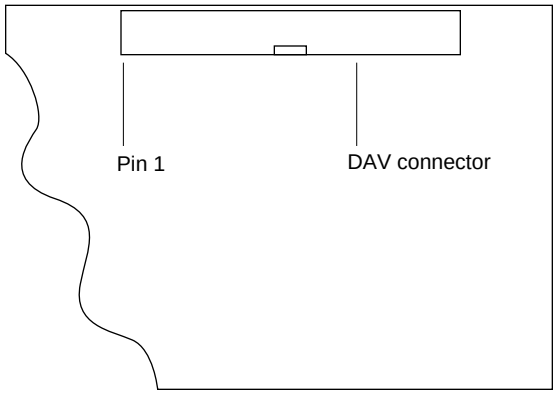


Figure 4-3 shows the orientation of the DAV connector on the video input module.

Figure 4-3 Orientation of the DAV connector



IMPORTANT

The DAV connector on the video input card provides some of the functionality of the DAV connectors found on the Power Macintosh Power Macintosh 7100 and 8100 models, and the Macintosh Quadra AV models, but it is not compatible with any of those connectors. Refer to *Macintosh DAV Interface for NuBus Expansion Cards* in *Developer Note Number 8* for more information.

AV cards designed for the DVA connector in the Power Macintosh 5200 and 6200 computers are compatible with the 60-pin DAV connector when an adapter cable is used. ▲

Pin Assignments

The DAV connector on the video-in card for the Power Macintosh 5400 computer is a 60-pin dual-row type with 0.100-inch pin spacing. The pin assignments on the DAV connector are shown in Table 4-9. The pins are numbered as shown in the table, with pin 1 across from pin 31 and pin 30 across from pin 30.

Table 4-9 Pin assignments on the DAV connector

Pin number	Signal name	Pin number	Signal name
1	Ground	2	GEOPORT_CLK
3	Ground	4	LLC_OUT
5	Ground	6	PXQ_OUT
7	Ground	8	VS_OUT
9	Ground	10	HS_OUT
11	UV bit 7	12	UV bit 6
13	UV bit 5	14	UV bit 4
15	UV bit 3	16	UV bit 2
17	UV bit 1	18	UV bit 0
19	Y bit 7	20	Y bit 6
21	Y bit 5	22	Y bit 4
23	Y bit 3	24	Y bit 2
25	Y bit 1	26	Y bit 0
27	Ground	28	LLC_IN
29	Ground	30	PXQ_IN
31	Ground	32	VS_IN
33	Ground	34	HS_IN

continued

Table 4-9 Pin assignments on the DAV connector (continued)

Pin number	Signal name	Pin number	Signal name
35	Ground	36	HREF_IN
37	Ground	38	FLD
39	IIC_DATA	40	IIC_CLK
41	Ground	42	SND_L
43	SND_RET	44	SND_R
45	Ground	46	AUDIO_SDIN
47	Ground	48	AUDIO_SDOUT
49	Ground	50	AUDIO_BITCLK
51	Ground	52	AUDIO_SYNC
53	Ground	54	NC
55	VID_RET	56	NC
57	VID_RET	58	NC
59	NC	60	NC

Signal Descriptions

Table 4-10 gives descriptions of the signals on the DAV connector.

Table 4-10 Descriptions of the signals on the DAV connector

Signal name	Signal description
LLC_OUT	Clock reference signal
FLD	YUV directional signal
HS_IN	Horizontal reference signal
HS_OUT	Horizontal sync signal
LLC_IN	Line-locked clock signal
UV(bits 0–7)	Digital chrominance data bus
VS_OUT	Vertical sync signal
Y(bits 0–7)	Digital luminance data bus

Using the YUV Bus

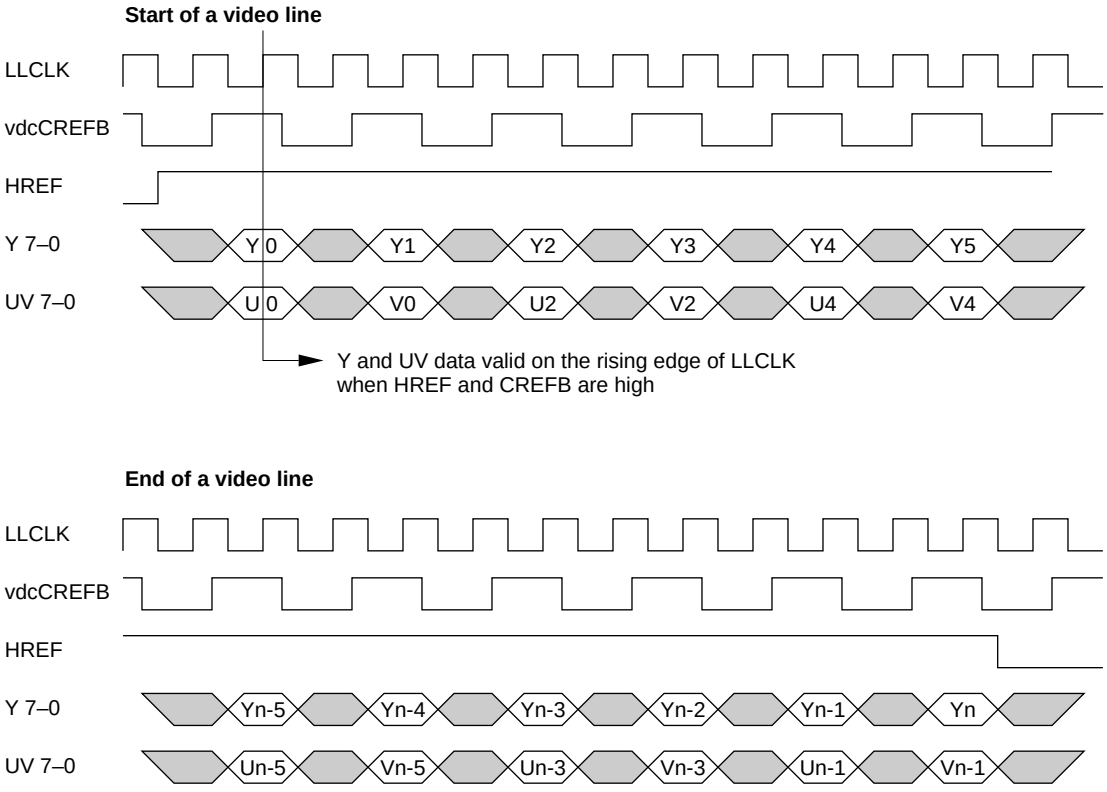
The video input module contains a digital video decoder and scaler (DESC), the Philips SAA7140 IC. Logic on the video input card uses the CVBS port on the DESC and pulls the FLD signal low, disabling the YUV bus. For an expansion card to use the YUV bus,

the software associated with the card must set the FLD signal high so that the DESC will accept data on the YUV bus. To do that, the software can use the Cuda Dispatch Manager to issue a IIC command to write to register $E of the DESC. For information about using the registers in the DESC IC, please refer to the *SAA7140 Philips Desktop Video Handbook.*

Video Data Format

Digital video data is transmitted as lines and fields. Each line consists of an even number of samples on the Y and UV buses as shown in Figure 4-4. HREF is high during a video line and low during the horizontal blanking interval. The falling edge of the VS signal indicates the beginning of a video field. For more information about digital video data in YUV format, see *Macintosh DAV Interface for NuBus Expansion Cards* in *Developer Note Number 8.*

Figure 4-4 Video data timing



The PCI-Bus Communications Slot

The main logic board has a separate slot for an optional communications card. The communications slot on Power Macintosh 5400 is a PCI-bus based communications slot rather than a processor direct PDS-based communications slot like that found on the Power Macintosh 5200 and 6200 computers.

The electrical interface of the communications slot is a parallel bus, the SCC lines, and lines for supporting modem audio. The PSX custom IC provides bus conversion from the host PowerPC 603e bus to the PCI parallel bus. Cards that use the communications slot are memory mapped into the I/O space of the Power Macintosh 5400 computer via the parallel bus. The communications slot supports SCC port A (modem port) for a universal modem card that is compatible with both the communications slot in the Power Macintosh 5200 and 6200 computers and PCI communications slot in the Power Macintosh 5400 computer.

PCI-Bus Communications Slot Connector

The PCI-bus based communications slot connector is a 112-pin half-height microchannel connector. A communications card mounts vertically in the connector and its I/O connector is accessed through the communications port access hole on the right hand side of the back panel. The size constraints of a communications card are 1.57 inches (40 mm) wide by 6 inches (152 mm) long.

A maximum of 2.5 watts of power is allocated to the communications slot. The maximum possible current ratings for each power line are:

Voltage	Current
+5 V	500 mA
+12 V	100 mA
Trickle +5 V	5 mA
–5 V	20 mA

Table 4-11 lists the pin assignments of the PCI-bus communications slot.

Table 4-11 Pin assignments for the PCI-bus communications slot connector

Odd-numbered pins	Function	Even-numbered pins	Function
1	/DCD	2	/DTR
3	/CTS	4	/RTS
5	RxD	6	TxD

continued

Table 4-11 Pin assignments for the PCI-bus communications slot connector (continued)

Odd-numbered pins	Function	Even-numbered pins	Function
7	IN_SENSE	8	SCC_ENAB
9	INT_MIC	10	MIC_SENSE
11	MIC_RET	12	EXT_AUD_L
13	Reserved	14	EXT_AUD_RET
15	GND	16	+12V
17	-5V	18	+12V
19	SYS_WAKEUP	20	Trickle +5
21	GND	22	GND
23	A1	24	A0
25	A3	26	A2
27	+3.3V	28	+3.3V
29	A5	30	A4
31	A7	32	A6
33	+5V	34	+5V
35	A8	36	C/BE(0)~
37	A10	38	A9
39	GND	40	GND
41	A12	42	A11
43	A14	44	A13
45	C/BE(1)~	46	A15
47	GND	48	Gnd
49	SERR~	50	PAR
51	PERR~	52	SBO~
53	LOCK~	54	SDONE
55	+3.3V	56	+3.3V
57	DEVSEL~	58	STOP~
59	IRDY~	60	TRDY~
61	+5V	62	+5V
63	C/BE(2)~	64	FRAME~
65	A17	66	A16
67	GND	68	GND

continued

Table 4-11 Pin assignments for the PCI-bus communications slot connector (continued)

Odd-numbered pins	Function	Even-numbered pins	Function
69	A19	70	A18
71	A21	72	A20
73	A23	74	A22
75	GND	76	GND
77	C/BE(3)~	78	IDSEL
79	A25	80	A24
81	A27	82	A26
83	+3.3V	84	+3.3V
85	A29	86	A28
87	A31	88	A30
89	+5V	90	+5V
91	REQ~	92	GNT~
93	+5V	94	+5V
95	INT~	96	Reserved
97	Reserved	98	RST~
99	GND	100	Reserved‘
101	CLK	102	Reserved
103	GND	104	Reserved
105	Reserved	106	Reserved
107	Reserved	108	Reserved
109	CommGnd	110	RefGnd
111	AudToSlot	112	AudFromSlot

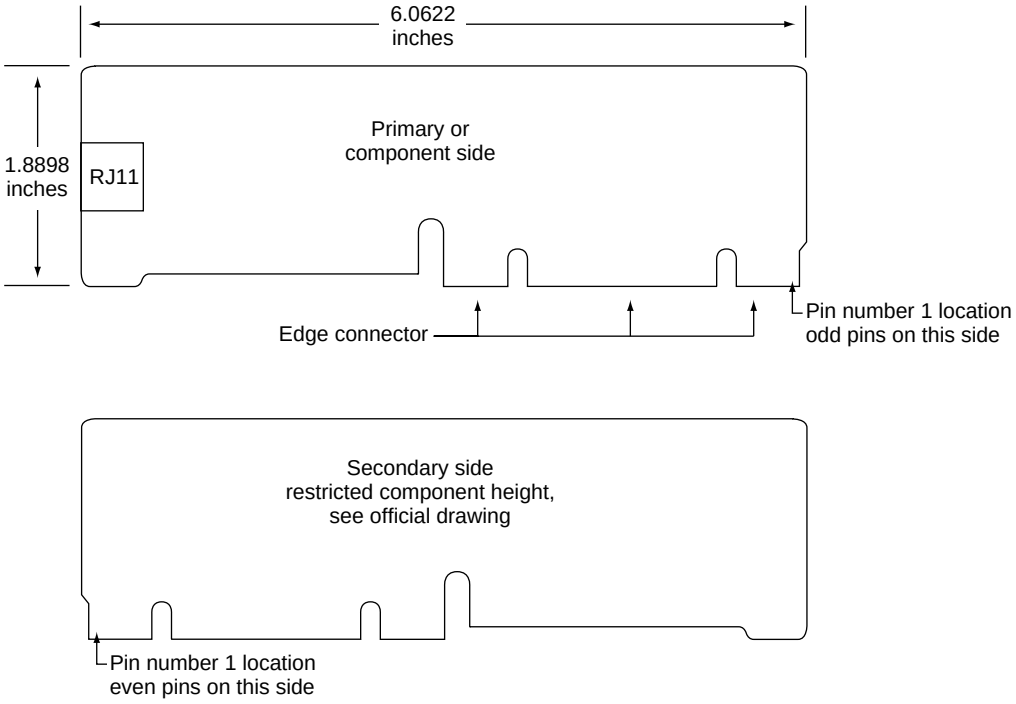
Universal Serial Modem Card

The PCI-bus communications slot of the Power Macintosh 5400 computer is not compatible with cards designed for the 68030-bus communications slot of the Power Macintosh 5200 and 6200 computers. Such cards will not physically fit into the communications slot of the Power Macintosh 5400 computer, because the communications slot is keyed at the opposite end. In addition, the two communications slots are not fully signal compatible.

However, the signals on the communications slot in the Power Macintosh 5400 computer are configured to make it possible to design a universal communications slot card that works in the 68030-style PDS comm slot if access to the parallel bus is not required (for example, a serial modem card). The power, ground, serial, and audio signals of the

communications slot in the Power Macintosh 5400 computer are located on the connector in such a way that a dual-keyed (cutout) modem card will fit and operate in the Power Macintosh 5200 and 6200, LC575, LC 630 and Power Macintosh 5400 communications slot. A simplified design diagram for a dual-keyed universal modem card is shown in Figure 4-5.

Figure 4-5 Universal modem card for communications slot



IMPORTANT

Serial modem cards designed for universal operation must not attempt to access the parallel bus of either the 68030-bus or the PCI-bus communications slots to be compatible in both configurations. ▲

Table 4-12 lists the pin assignments on a universal serial modem card that operates in either the 68030-bus or PCI-bus communications slots. The assignments are the same as

those on the PCI-bus communications slot, listed in Table 4-11, with the PCI bus signals removed and the addition of the key slot location.

Table 4-12 Pin assignments for a universal serial modem card

Odd-numbered pins	Function	Even-numbered pins	Function
1	/DCD	2	/DTR
3	/CTS	4	/RTS
5	RxD	6	TxD
7	IN_SENSE	8	SCC_ENAB
9	INT_MIC	10	MIC_SENSE
11	MIC_RET	12	EXT_AUD_L
13	Not connected	14	EXT_AUD_RET
15	Gnd	16	+12V
17	–5V	18	Not connected
19	SYS_WAKEUP	20	Trickle+5
21	GND	22	Not connected
key (pin 23)	key slot (cutout)	key (pin 24)	key slot (cutout)
key (pin 25)	key slot (cutout)	key (pin 26)	key slot (cutout)
27	Not connected	28	Not connected
29	Not connected	30	Not connected
31	Not connected	32	Not connected
33	Not connected	34	+5V
35	Not connected	36	Not connected
37	Not connected	38	Not connected
39	Not connected	40	Not connected
41	Not connected	42	Not connected
43	Not connected	44	Not connected
45	Not connected	46	Not connected
47	GND	48	Not connected
49	Not connected	50	Not connected
51	Not connected	52	Not connected
53	Not connected	54	Not connected
55	Not connected	56	Not connected

continued

Table 4-12 Pin assignments for a universal serial modem card (continued)

Odd-numbered pins	Function	Even-numbered pins	Function
57	Not connected	58	Not connected
59	Not connected	60	Not connected
61	Not connected	62	+5V
63	Not connected	64	Not connected
65	Not connected	66	Not connected
67	Not connected	68	Not connected
69	Not connected	70	Not connected
71	Not connected	72	Not connected
73	Not connected	74	Not connected
75	GND	76	Not connected
77	Not connected	78	Not connected
79	Not connected	80	Not connected
81	Not connected	82	Not connected
83	Not connected	84	Not connected
85	Not connected	86	Not connected
87	Not connected	88	Not connected
89	Not connected	90	+5V
key	key slot (cutout)	key	key slot (cutout)
key	key slot (cutout)	key	key slot (cutout)
91	Not connected	92	Not connected
93	Not connected	94	Not connected
95	Not connected	96	Reserved
97	Reserved	98	RST~
99	GND	100	Reserved
101	Not connected	102	Not connected
103	GND	104	Reserved
105	Reserved	106	Reserved
107	Reserved	108	Reserved
109	CommGnd	110	RefGnd
111	AudFromSlot	112	AudToSlot

CHAPTER 5

Software Features

The first part of this chapter describes the software in the ROM of the Power Macintosh 5400 computer. The second part describes the system software that supports the new features of these computers.

For a description of the system software for the internal IDE hard disk, see Chapter 7, "Software for the ATA (IDE) Hard Disk."

ROM Software

The ROM is based on the ROM used in the current Power Macintosh models with the necessary changes to support machine-specific hardware.

The following is a list of the most significant ROM changes:

- Hardware initialization now includes support for MMU programming and other PowerPC 603 microprocessor functions, addition of new diagnostics, and removal of the 68040 check/support code.
- The nanokernel has been modified to support the PowerPC microprocessor.
- The software no longer supports 1- or 2- bit video modes.
- The software supports 16-bit sound.

Machine Identification

The ROM includes new tables and code for identifying the two computers.

Applications can find out which computer they are running on by using the Gestalt Manager routines; see *Inside Macintosh: Overview.* The `gestaltMachineType` value returned by the Power Macintosh 5400 computer is ?? (hexadecimal ??).

System Software

The Power Macintosh 5400 computer is shipped with a version of System 7.5 software preinstalled. The disk labeled "Install Me First" includes a *system enable*r file that contains the resources the system needs to start up and initialize the computer.

As soon as the system software on disk takes over the startup process, it searches for all system enablers that can start up the particular machine. Each system enabler contains a resource that specifies which computers it is able to start up and the time and date of its creation. If the system software finds more than one enabler for the particular computer, it passes control to the one with the most recent time and date.

In general, the system enabler included in each reference release of system software is able to start up all previous computers. The enablers for computers introduced after a reference release may be independent or may use resources from the previous reference release.

The system enabler includes modifications to the video digitizer allowing it to run in native mode to improve video capture performance.

New Features

The system software for the Power Macintosh 5400 computer includes the following new features:

- large volume support
- Drive Setup (replaces HDSC setup)
- transport-independent networking (Open Transport)
- Open Firmware startup
- Monitors & Sound (audio and video management)
- Energy Saver software

Large Volume Support

The largest disk volume or partition supported by System 7.5 is 4 GB. The new system software extends that limit to 2 terabytes.

IMPORTANT
The largest possible file is still just under 2 GB. ▲

The changes necessary to support the larger volume size affect many parts of the system software. The affected software includes system-level and application-level components.

64-Bit Volume Addresses

The current disk driver application programming interface (API) has a 32-bit volume address limitation. This limitation has been circumvented by the addition of a new 64-bit extended volume API (`PBXGetVolInfo`) and 64-bit data types (`UnsignedWide`, `Wide`, `XVolumeParam`, and `XIOParam`).

For the definitions of the new API and data types, please see "The API Modifications" in Chapter 6, "Large Volume Support."

System-Level Software

Several system components have been modified to use the 64-bit APIs to correctly calculate true volume sizes and read and write data to and from large disks. The modified system components are

- virtual memory code
- Disk Init file
- FSM file

- Apple disk drivers
- Hierarchical file system (HFS) ROM code

Application-Level Software

Current applications do not require modification to gain access to disk space beyond the traditional 4 GB limit as long as they do not require the true size of the large partition. Applications that need to obtain the true partition size must be modified to use the new 64-bit API and data structures. Typical applications include utilities for disk formatting, partitioning, initialization, and backup.

The following application-level components of the system software have been modified to use the 64-bit APIs:

- Finder
- Finder extensions (AppleScript, AOCE Mailbox, and Catalogs)
- Drive Setup
- Disk First Aid

In the past, the sum of the sizes of the files and folders selected in the Finder was limited to the largest value that could be stored in a 32-bit number—that is, 4 GB. By using the new 64-bit APIs and data structures, the Finder can now operate on selections whose total size exceeds that limit. Even with very large volumes, the Finder can display accurate information in Folder and Get Info windows and to obtain the true volume size for calculating available space when copying.

The Finder extensions AppleScript, AOCE Mailbox, and Catalogs have been modified in the same way as the Finder because their copy-engine code is similar to that in the Finder.

The modified HDSC Drive Setup application is described in “Drive Setup” beginning on page 69. The Disk First Aid application is not described in this developer note.

Limitations

The software modifications that support large partition sizes do not solve all the problems associated with the use of large volumes. In particular, the modifications do not address the following:

- HFS file sizes are still limited to 2 GB or less.
- Large allocation block sizes cause inefficient storage. On a 2 GB volume, the minimum file size is 32 KB; on a 2 terabyte volume, the minimum file size is 32 MB.
- Drives with the new large volume device driver will not mount on computers running older versions of the Macintosh Operating System.

Drive Setup

The software for the Power Macintosh 5400 computer includes a new disk setup utility named Drive Setup. In addition to the ability to support large volumes, the Drive Setup utility has several other enhancements over the older HDSC Setup utility, including

- an improved user interface
- support for multiple partitions
- the ability to mount volumes from applications
- the ability to start up (boot) from any HFS partition
- support for removable media drives

Open Transport

Open Transport is the new communications and networking architecture that will become the standard for Macintosh networking and communications. Open Transport provides a mechanism for communications applications to operate independently from underlying networks such as AppleTalk, TCP, or IPX. Open Transport provides a code base and architecture that supports network stacks while eliminating many of the interrupt latency problems associated with AppleTalk.

Note
Open Transport runs native on the PowerPC microprocessors. ◆

Open Transport has two major aspects: the client interfaces and the environment for developing protocols and communications modules. The Open Transport client interfaces are a superset of the XTI interface from X/Open, a consortium of UNIX® vendors. XTI is a superset of TLI, a UNIX standard interface. By using the Open Transport interfaces, applications (called *clients*) can operate independently of the transport layer.

The environment for developing protocols and communications modules for Open Transport also uses industry standards. These standards are the UNIX standard Streams, and two other standards, Transport Provider Interface (TPI) and Data Link Provider Interface (DLPI).

Open Transport does not use the conventional .ENET style drivers; instead it uses Streams-based DLPI drivers that are more appropriate for use with PCI devices. In addition to being consistent with industry standards, Streams-based DLPI drivers provide higher performance than .ENET style drivers.

Apple Computer, Inc.'s, Open Transport software includes new stack implementations for AppleTalk and MacTCP. Apple expects that third parties will provide implementations of DECnet, IPX, and other network protocols.

The Open Transport implementation of AppleTalk has a significant feature not found in the classic AppleTalk implementation for Macintosh computers. The Open Transport implementation supports multihoming (sometimes called *multiporting*), which makes it

possible for AppleTalk to be active on more than one network port on the machine at a time.

The Open Transport implementation of TCP/IP is a replacement for MacTCP. It is designed for use under the Open Transport software interface.

New Features of Open Transport

The new features of Open Transport include

- a new API
- dynamic loading and shared code
- multihoming (multiporting)
- an optional static node number (AppleTalk)
- an optional NBP-to-catalog server (AppleTalk)
- IP multicasting (MacTCP)
- dynamic retransmission timers (MacTCP)

Compatibility

Open Transport is compatible with existing AppleTalk networks and supports existing .ENET clients such as Soft Windows and DECnet.

Open Transport provides compatibility with 680x0-based computers by means of the following features:

- environment options
- 680x0-based APIs and stacks
- Open Transport APIs and stacks
- API compatibility glue
- use of parameter-block APIs with Open Transport stacks for 680x0-based applications

Open Transport provides compatibility with Power Macintosh computers by means of the following features:

- environment options
- 680x0-based APIs and stacks run in emulation mode
- Open Transport APIs and stacks run in native mode
- API compatibility glue runs in mixed mode
- 680x0-based applications can use parameter-block APIs with Open Transport stacks
- 680x0-based applications can use Open Transport APIs and stacks
- native applications can use parameter block APIs with 680x0-based stacks
- native applications can use parameter block APIs with Open Transport stacks

Open Firmware Startup

The **Open Firmware startup process** in PCI-compatible Macintosh computers conforms to the IEEE Standard 1275 for Boot Firmware and the *PCI Bus Binding to IEEE 1275-1994* specification. These specifications are listed in "Supplemental Reference Documents," in the preface.

The Open Firmware startup process is driven by **startup firmware** (also called *boot firmware*) stored in the Macintosh ROM and in PCI card expansion ROMs. While the startup firmware is running, the Macintosh computer starts up and configures its hardware (including peripheral devices) independently of any operating system. The computer then finds an operating system in ROM or on a mass storage device, loads it into RAM, and terminates the Open Firmware startup process by giving the operating system control of the PowerPC main processor. The operating system may be the Macintosh Operating System or a different system, provided it uses the PowerPC instruction set.

The Open Firmware startup process includes these specific features:

- Startup firmware is written in the Forth language, as defined by the IEEE Standard 1275. Firmware code is stored in a tokenized representation called **FCode**, an abbreviated version of Forth in which most Forth words are replaced by single bytes or 2-byte groups. The startup firmware in the Power Macintosh ROM includes an FCode loader that installs FCode in system RAM so that drivers can run on the PowerPC main processor. Expansion card firmware can modify the Open Firmware startup process by supplying FCode that the computer's startup firmware loads and runs before launching an operating system.
- The startup firmware creates a data structure of nodes called a **device tree,** in which each PCI device is described by a **property list.** The device tree is stored in system RAM. The operating system that is ultimately installed and launched can search the device tree to determine what devices are available.
- Device drivers required during system startup (called **boot drivers**) are also stored in the expansion ROM on the PCI card. Plug-in expansion cards must contain all the driver code required during startup. The boot drivers are native drivers and are embedded in the FCode in the expansion ROM. The startup firmware in the Power Macintosh ROM installs the boot drivers in system RAM and lets them run on the PowerPC main processor.
- The startup firmware in the Power Macintosh ROM contains debugging facilities for both FCode and some kinds of operating-system code. These facilities can help expansion card designers develop the firmware for new peripheral devices compatible with Macintosh computers.

You can write PCI expansion ROM code in standard Forth words and then reduce the result to FCode by using an **FCode tokenizer,** a program that translates Forth words into FCodes. The Forth vocabulary that you can use is presented in IEEE Standard 1275.

The burden on developers to provide Forth boot drivers need not be heavy. Developers can choose the level of support that they provide. The following are the three possible levels of support:

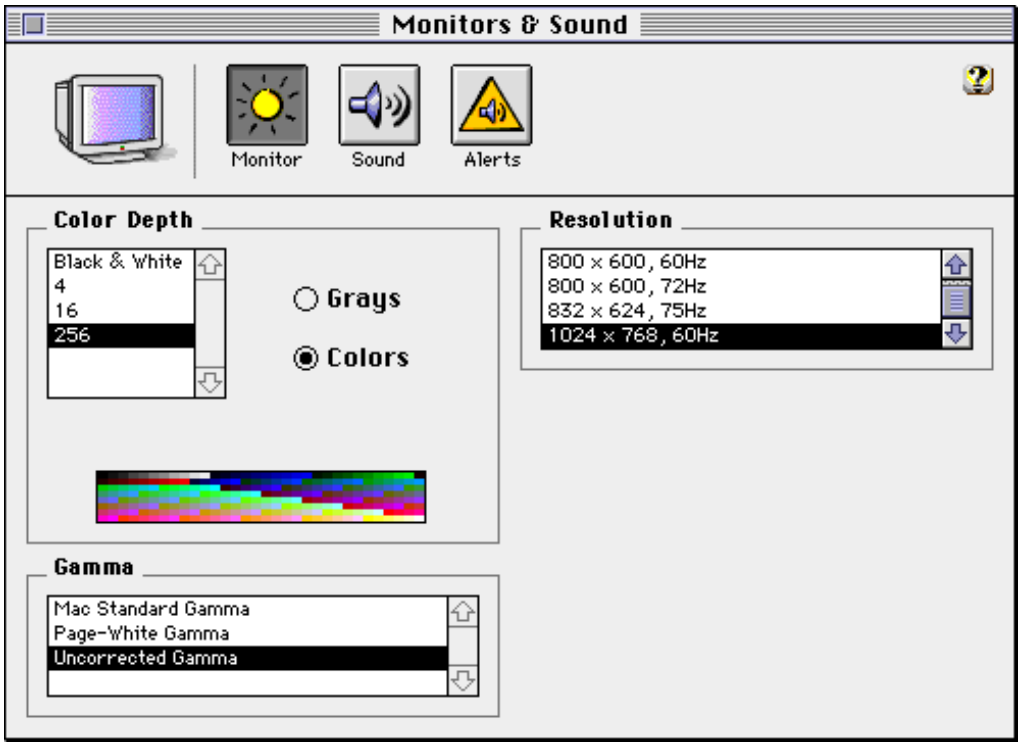
- **No driver.** The expansion ROM contains minimal FCode. The Open Firmware startup process recognizes the card and installs a node in the device tree, but no driver code is loaded and no device initialization occurs.
- **Run-time driver.** Only a small amount of Forth code is required to install an OS-dependent run-time driver in the device's property list. Sample code is provided in *Designing PCI Cards and Drivers for Power Macintosh Computers.*
- **Boot driver.** Expansion cards that need to be used at startup time must contain a boot driver with the required methods for the type of device (typically Open, Close, Read, and Write). Sample code is provided in *Designing PCI Cards and Drivers for Power Macintosh Computers*.

Monitors & Sound Control Panel

The Monitors & Sound control panel software is a new all in one control panel for configuring the audio and video input and output features of the Power Macintosh 5400 computer. The control panel combines the functions of three control panels used up until now: the Monitors and Sound control panels found on all Macintosh computers and the Video control panel used with A/V Macintosh computers.

Figure 5-1 shows the main window of the Monitors & Sound control panel

Figure 5-1 Main window of the Monitors & Sound control panel



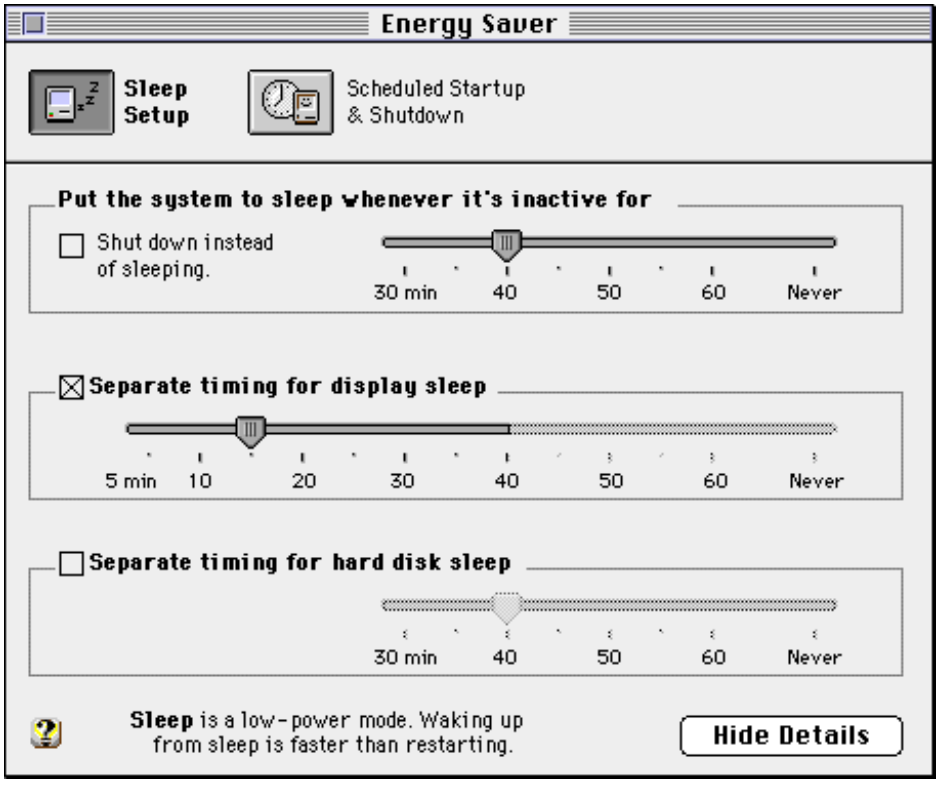
The Monitors & Sound control panel is a QuickTime component control panel

Energy Saver Software

The system software for the Power Macintosh 5400 computer supports a superset of the Macintosh Energy Saver power-management application software. The enhanced Energy Saver application fully employs the capabilities of Energy-Star compliant features in Macintosh computers and peripherals, such as hard disk drives and displays.

The Energy Saver application, shown expanded for Sleep preferences in Figure 5-2, provides desktop computer users access to power-saving features previously available only in portable computer power-management software. The user experience provided by the human interface to the new power management software is consistent across portable and desktop Macintosh computers.

Figure 5-2 Energy Saver application dialog box



Features of the New Energy Saver Application

The Energy Saver power management software allows control of these features:

- idle-time energy savings
  - computer sleep time

- display sleep time
- hard disk spin down time

- scheduled energy savings
  - scheduling of startup
  - scheduling of shutdown including document auto-save
- wakeup preferences
  - blink power-on light when waking up
  - play user defined sound when waking up
  - wake up when modem detects a telephone ring

The Energy Saver software provides users with a single control panel to configure all of the energy management features of their Macintosh computer system. The improved single control panel interface presents a clearer conceptual model of what energy saving features of the Macintosh computer can be managed. It also delivers a user experience that is as consistent as possible across desktop Macintosh computers and portable Macintosh computers. In addition, the new Energy Saver look and feel is appropriate for existing and future Macintosh system software models. Last but not least, Energy Saver satisfies the EPA requirements for Energy Star compliance.

To find out more about how to use all of the capabilities if the Energy Saver control panel, see the User Guide and Macintosh Guide help files that accompany the shipping versions of the Power Macintosh 5400 computer.

Performance Enhancements

The system software for the Power Macintosh 5400 computer includes the following performance enhancements:

- a new Dynamic Recompilation Emulator
- a Resource Manager completely in native code
- an improved math library
- new BlockMove extensions

Dynamic Recompilation Emulator

The Dynamic Recompilation Emulator (or DR Emulator) is an extension to the current interpretive emulator providing on-the-fly translation of 680x0 instructions into PowerPC instructions for increased performance. The DR Emulator operates as an enhancement to a modified version of the existing interpretive emulator.

The design of the DR Emulator mimics a hardware instruction cache and employs a variable size translation cache. Each compiled 680x0 instruction requires on average fewer than 20 PowerPC instructions. In operation, the DR Emulator depends on locality of execution to make up for the extra cycles used in translating the code.

The DR Emulator provides a high degree of compatibility for 680x0 code. One area where compatibility will be less than that of the current interpretive emulator is for self-modifying code that does not call the cache flushing routines. Such code also has compatibility problems on Macintosh Quadra models with the cache enabled.

Resource Manager in Native Code

The Resource Manager in the software for the Power Macintosh 5400 computer is similar to the one in the earlier Power Macintosh computers except that it is completely in native PowerPC code. Because the Resource Manager is intensively used both by system software and by applications, the native version provides an improvement in system performance.

The Process Manager has been modified to remove patches it formerly made to the Resource Manager.

Math Library

The new math library (MathLib) is an enhanced version of the floating-point library included in the ROM in the first generation of Power Macintosh computers.

The new math library is bit compatible in both results and floating-point exceptions with the math library in the first-generation ROM. The only difference is in the speed of computation.

The new math library has been improved to better exploit the floating-point features of the PowerPC microprocessor. The math library now includes enhancements that assist the compiler in carrying out its register allocation, branch prediction, and overlapping of integer and floating-point operations.

Compared with the previous version, the new math library provides much improved performance without compromising its accuracy or robustness. It provides performance gains for often-used functions of up to 15 times.

The application interface and header files for the math library have not been changed.

New BlockMove Extensions

The system software for the Tsunami computer includes new extensions to the `BlockMove` routine. The extensions provide improved performance for programs running in native mode.

The new `BlockMove` extensions provide several benefits for developers.

- They're optimized for the PowerPC 603 processor, rather than the PowerPC 601.
- They're compatible with the new dynamic recompilation emulator.
- They provide a way to handle cache-inhibited address spaces.
- They include new high-speed routines for setting memory to zero.

Note
The new `BlockMove` extensions do not use the string instructions, which are fast on the PowerPC 601 but slow on other PowerPC implementations.

Some of the new `BlockMove` extensions can be called only from native code.

Except for `BlockZero` and `BlockZeroUncached`, the new `BlockMove` extensions use the same parameters as `BlockMove`. Calls to `BlockZero` and `BlockZeroUncached` have only two parameters, a pointer and a length, the same as the second and third parameters of `BlockMove`.

Table 5-1 summarizes the `BlockMove` routines according to three criteria: whether the routine can be called from 680x0 code, whether it is okay to use for moving 680x0 code, and whether it is okay to use with buffers or other uncacheable destination locations.

Table 5-1 Summary of `BlockMove` routines

`BlockMove` version	Can be called from 680x0 code	Okay to use for moving 680x0 code	Okay to use with buffers
`BlockMove`	Yes	Yes	No
`BlockMoveData`	Yes	No	No
`BlockMoveDataUncached`	No	No	Yes
`BlockMoveUncached`	No	Yes	Yes
`BlockZero`	No	—	No
`BlockZeroUncached`	No	—	Yes

The fastest way to move data is to use the `BlockMoveData` routine. It is the recommended method whenever you are certain that the data is cacheable and does not contain executable 680x0 code.

The `BlockMove` routine is slower than the `BlockMoveData` routine only because it has to clear out the software cache used by the DR Emulator. If the DR Emulator is not in use, the `BlockMove` routine and the `BlockMoveData` routine are the same.

IMPORTANT
The versions of `BlockMove` for cacheable data use the `dcbz` instruction to avoid unnecessary prefetching of destination cache blocks. For uncacheable data, you should avoid using those routines because the `dcbz` instruction faults on uncacheable or write-through locations, making execution extremely slow. ▲

IMPORTANT
Driver software cannot call the `BlockMove` routines directly. Instead, drivers must use the `BlockCopy` routine, which is part of the Driver Services Library. The `BlockCopy` routine is an abstraction that allows you to postpone binding the specific type of `BlockMove` operation until implementation time. ▲

The Driver Services Library is a collection of useful routines that Apple Computer provides for developers working with the new Power Macintosh computers. For more information, please refer to *Designing PCI Cards and Drivers for Power Macintosh Computers.*

Hardware Support Features

The system software for the Power Macintosh 5400 computer includes the following features to support the hardware:

- PCI bus support
- POWER-clean native code
- POWER emulation (for PowerPC 601 compatibility)
- Display Manager
- support of native drivers
- IDE hard drive support

PCI Bus Support

The Power Macintosh 5400 computer does not use NuBus™ or Macintosh LC PDS slots for hardware expansion, but instead use the industry standard PCI bus architecture. To support these computers as well as future Macintosh models that do not use the NuBus architecture, new system software includes a bus-neutral expansion architecture that is used by system software in place of Slot Manager calls that are specific to NuBus.

Removal of Slot Manager Dependencies

The system software that controls NuBus cards in current Macintosh models has many explicit dependencies on the Slot Manager. The system software for models that use PCI bus slots requires changes to each of those dependencies so that PCI cards can operate with the system in the same fashion as NuBus cards.

The system software that formerly called the Slot Manager has been modified to use other services. The new Display Manager provides the means of obtaining video-specific information that was previously obtained by way of the Slot Manager. For example, QuickDraw currently calls the Slot Manager at startup time to check the consistency of the `'scrn'` resource. In the software for the Power Macintosh 5400 computer, QuickDraw calls the new Display Manager to check this consistency.

The following components formerly used the Slot Manager; they have been modified to use the services of the Display Manager:

- the Monitors Control Panel
- QuickDraw
- Palette Manager
- Device Manager

PCI Compatibility

To support a third party NuBus-to-PCI bridge product for PCI-based computers, it is important to retain Slot Manager capability. Also, several important applications (such as DECnet™ and SoftWindows™) rely on Slot Manager calls to indicate the presence of networking cards. For compatibility, the new expansion architecture will support existing PCI-based cards by way of particular Slot Manager calls.

For more information about PCI expansion cards, please refer to *Designing PCI Cards and Drivers for Power Macintosh Computers.*

POWER-Clean Native Code

The instruction set of the PowerPC 601 microprocessor includes some of the same instructions as those found in the instruction set of the POWER processor, and the compiler used to generate native code for the system software in the first generation of Power Macintosh models generated some of those POWER-only instructions. However, the PowerPC 603e microprocessor used in the Power Macintosh 5400 computes does not support the POWER-only instructions, so a new POWER-clean version of the compiler is being used to compile the native code fragments.

Note
The term *POWER-clean* refers to code that is free of the POWER instructions that would prevent it from running correctly on a PowerPC 603 or PowerPC 604 microprocessor. ◆

Here is a list of the POWER-clean native code fragments in the system software.

- interface library
- private interface library
- native QuickDraw
- MathLib
- Mixed Mode Manager
- Code Fragment Manager
- Font Dispatch
- Memory Manager
- standard text

- the `FMSwapFont` function
- standard C library

POWER Emulation

The first generation of Power Macintosh computers included emulation for certain PowerPC 601 instructions that would otherwise cause an exception. The emulation code dealt with memory reference instructions to handle alignment and data storage exceptions. It also handled illegal instruction exceptions caused by some PowerPC instructions that were not implemented in the PowerPC 601. In the second generation of Power Macintosh computers, the emulation code has been revised to include the POWER instructions that are implemented on the PowerPC 601 but not on the PowerPC 603.

Note
Although the term *POWER emulation* is often used, a more appropriate name for this feature is *PowerPC 601 compatibility.* Rather than supporting the entire POWER architecture, the goal is to support those features of the POWER architecture that are available to programs running in user mode on the PowerPC 601-based Power Macintosh computers. ◆

POWER-Clean Code

Because the emulation of the POWER-only instructions degrades performance, Apple Computer encourages developers to revise any applications that use those instructions to conform with the PowerPC architecture. Emulation works, but performance is degraded; POWER-clean code is better.

Limitations of PowerPC 601 Compatibility

The emulation code in the second-generation Power Macintosh computers allows programs compiled for the PowerPC 601 to execute without halting on an exception whenever they use a POWER-only feature. For most of those features, the emulation matches the results that are obtained on a Power Macintosh computer with a PowerPC 601. However, there are a few cases where the emulation is not an exact match; those cases are summarized here.

- **MQ register.** Emulation does not match the undefined state of this register after multiply and divide instructions.
- **`div` and `divo` instructions.** Emulation does not match undefined results after an overflow.
- **Real-time clock registers.** Emulation matches the 0.27 percent speed discrepancy of the Power Macintosh models that use the PowerPC 601 microprocessor, but the values of the low-order 7 bits are not 0.
- **POWER version of `dec` register.** Emulation includes the POWER version, but decrementing at a rate determined by the time base clock, not by the real-time clock.

- **Cache line compute size (`clcs`) instruction.** Emulation returns values appropriate for the type of PowerPC microprocessor.
- **Undefined SPR encodings.** Emulation does not ignore SPR encodings higher than 32.
- **Invalid forms.** Invalid combinations of register operands with certain instructions may produce results that do not match those of the PowerPC 601.
- **Floating-point status and control register (FPSCR).** The FPSCR in the PowerPC 601 does not fully conform to the PowerPC architecture, but the newer PowerPC processors do.

Emulation and Exception Handling

When an exception occurs, the emulation code first checks to see whether the instruction encoding is supported by emulation. If it is not, the code passes the original cause of the exception (illegal instruction or privileged instruction) to the application as a native exception.

If the instruction is supported by emulation, the code then checks a flag bit in the emulator's context block definition to see whether emulation has been enabled. If emulation is not enabled at the time, the emulator generates an illegal instruction exception.

Note
There is a mechanism to turn off POWER emulation so that developers can verify that they have removed the POWER-only instructions from their applications. ◆

In the first generation of Power Macintosh computers, emulation also allowed access to the performance monitor control registers, which are privileged SPRs. In the second-generation machines, access to privileged SPRs is emulated only for the performance monitor registers and only if the processor supports the performance monitor; otherwise, the emulator generates a privileged instruction exception. Notice that the emulator generates a privileged instruction exception even if the original exception was an illegal instruction exception. It does this to provide exception reporting that is consistent with that of the first-generation Power Macintosh computers.

Code Fragments and Cache Coherency

Whereas the PowerPC 601 microprocessor has a single cache for both instructions and data, the PowerPC 603 has separate instruction and data caches. As long as applications deal with executable code by using the Code Fragment Manager, cache coherency is maintained. Applications that bypass the Code Fragment Manager and generate executable code in memory, and that do not use the proper cache synchronization instructions or CFM calls, are likely to encounter problems when running on the PowerPC 603.

IMPORTANT

The emulation software in the Power Macintosh 5400 computer cannot make the separate caches in the PowerPC 603 behave like the combined cache in the PowerPC 601. Applications that generate executable code in memory must be modified to use the Code Fragment Manager or maintain proper cache synchronization by other means. ▲

Display Manager

Until now, system software has used the NuBus-specific Slot Manager to get and set information about display cards and drivers. New system software removes this explicit software dependency on the architecture of the expansion bus. The Display Manager provides a uniform API for display devices regardless of the implementation details of the devices.

In a computer that uses PCI expansion cards, the Slot Manager is generally not available to provide information about display cards; instead, the Expansion Manager must be used. The Display Manager makes the actual calls to either the Slot Manager or the Expansion Manager, as appropriate, thus isolating the bus-specific calls to a single component and avoiding the need to change additional system software in the future. See the section "Removal of Slot Manager Dependencies" on page 77.

Support of Native Drivers

The Power Macintosh 5400 computer uses a new native-driver model for system software and device driver developers. Several components of system software are being modified to support native drivers. The modified components are

- Device Manager
- interrupt services
- driver loader library
- driver support library
- Slot Manager stubs
- Macintosh startup code
- interface libraries
- system registry

For more information, refer to *Designing PCI Cards and Drivers for Power Macintosh Computers.*

CHAPTER 6

Large Volume Support

This chapter describes the large volume file system for the Power Macintosh 5400 computer. The large volume file system is a version of the hierarchical file system (HFS) that has been modified to support volume sizes larger than the current 4 GB limit. It incorporates only the changes required to achieve that goal.

Overview of the Large Volume File System

The large volume file system includes

- modifications to the HFS ROM code, Disk First Aid application, and Disk Init file
- a new extended API that allows reporting of volume size information beyond the current 4 GB limit
- new device drivers and changes to the Device Manager API to support devices that are greater than 4 GB
- a new version of Drive Setup that supports large volumes and chainable drivers (Chainable drivers are needed to support booting large volumes on earlier Macintosh models.)

API Changes

The system software on the Power Macintosh 5400 computer allows all current applications to work without modifications. Unmodified applications that call the file system still receive incorrect values for large volume sizes. The Finder and other utility programs that need to know the actual size of a volume have been modified to use the new `PBXGetVolInfo` function to obtain the correct value. (`PBXGetVolInfo` is an extended version of the `PBHGetVInfo` function.)

The existing low-level driver interface does not support I/O to a device with a range of addresses greater than 4 GB because the positioning offset (in bytes) for a read or write operation is a 32-bit value. To correct this problem, a new extended I/O parameter block record has been defined. This extended parameter block has a 64-bit positioning offset. The new parameter block and the extended `PBXGetVolInfo` function are described in "The API Modifications" beginning on page 85.

Allocation Block Size

The format of HFS volumes has not changed. What has changed is the way the HFS software handles the allocation block size. Existing HFS code treats the allocation block as a 16-bit integer. The large volume file system uses the full 32 bits of the allocation block size parameter. In addition, any software that deals directly with the allocation block size from the volume control block must now treat it as a true 32-bit value.

Even for the larger volume sizes, the number of allocation blocks is still defined by a 16-bit integer. As the volume size increases, the size of the allocation block also increases. For a 2 GB volume, the allocation block size is 32 KB and therefore the smallest file on

that disk will occupy at least 32 KB of disk space. This inefficient use of disk space is not addressed by the large volume file system.

The maximum number of files will continue to be less than 65,000. This limit is directly related to the fixed number of allocation blocks.

File Size Limits

The HFS has a maximum file size of 2 GB. The large volume file system does not remove that limit because doing so would require a more extensive change to the current API and would incur more compatibility problems.

Compatibility Requirements

The large volume file system requires at least a 68020 microprocessor or a Power Macintosh model that emulates it. In addition, the file system requires a Macintosh IIci or more recent model. On a computer that does not meet both those requirements, the large volume file system driver will not load.

The large volume file system requires System 7.5 or higher and a new Finder that supports volumes larger than 4 GB (using the new extended `PBXGetVolInfo` function).

The API Modifications

The HFS API has been modified to support volume sizes larger than 4 GB. The modifications consist of two extended data structures and a new extended `PBXGetVolInfo` function.

Data Structures

This section describes the two modified data structures used by the large volume file system:

- the extended volume parameter block
- the extended I/O parameter block

Extended Volume Parameter Block

In the current volume parameter record (`volumeParam` instance of `HParamBlockRec`), volume size information is clipped at 2 GB. Because HFS volumes can now exceed 4 GB, a new extended volume parameter block is needed in order to report the larger size information. The `XVolumeParam` record contains 64-bit integers for reporting the total bytes on the volume and the number of free bytes available (field names `ioVTotalBytes` and `ioVFreeBytes`). In addition, several of the fields that were

previously signed are now unsigned (field names `ioVAtrb`, `ioVBitMap`, `ioAllocPtr`, `ioVAlBlkSiz`, `ioVClpSiz`, `ioAlBlSt`, `ioVNxtCNID`, `ioVWrCnt`, `ioVFilCnt`, and `ioVDirCnt`).

```
struct XVolumeParam {
   ParamBlockHeader
   unsigned long     ioXVersion;     // XVolumeParam version == 0
   short             ioVolIndex;     // volume index
   unsigned long     ioVCrDate;      // date & time of creation
   unsigned long     ioVLsMod;       // date & time of last modification
   unsigned short    ioVAtrb;        // volume attributes
   unsigned short    ioVNmFls;       // number of files in root directory
   unsigned short    ioVBitMap;      // first block of volume bitmap
   unsigned short    ioAllocPtr;     // first block of next new file
   unsigned short    ioVNmAlBlks;    // number of allocation blocks
   unsigned long     ioVAlBlkSiz;    // size of allocation blocks
   unsigned long     ioVClpSiz;      // default clump size
   unsigned short    ioAlBlSt;       // first block in volume map
   unsigned long     ioVNxtCNID;     // next unused node ID
   unsigned short    ioVFrBlk;       // number of free allocation blocks
   unsigned short    ioVSigWord;     // volume signature
   short             ioVDrvInfo;     // drive number
   short             ioVDRefNum;     // driver reference number
   short             ioVFSID;        // file-system identifier
   unsigned long     ioVBkUp;        // date & time of last backup
   unsigned short    ioVSeqNum;      // used internally
   unsigned long     ioVWrCnt;       // volume write count
   unsigned long     ioVFilCnt;      // number of files on volume
   unsigned long     ioVDirCnt;      // number of directories on volume
   long              ioVFndrInfo[8]; // information used by the Finder
   uint64            ioVTotalBytes;  // total number of bytes on volume
   uint64            ioVFreeBytes;   // number of free bytes on volume
};
```

Field descriptions

`ioVolIndex`	An index for use with the `PBHGetVInfo` function.
`ioVCrDate`	The date and time of volume initialization.
`ioVLsMod`	The date and time the volume information was last modified. (This field is not changed when information is written to a file and does not necessarily indicate when the volume was flushed.)
`ioVAtrb`	The volume attributes.
`ioVNmFls`	The number of files in the root directory.
`ioVBitMap`	The first block of the volume bitmap.
`ioAllocPtr`	The block at which the next new file starts. Used internally.

`ioVNmAlBlks`	The number of allocation blocks.
`ioVAlBlkSiz`	The size of allocation blocks.
`ioVClpSiz`	The clump size.
`ioAlBlSt`	The first block in the volume map.
`ioVNxtCNID`	The next unused catalog node ID.
`ioVFrBlk`	The number of unused allocation blocks.
`ioVSigWord`	A signature word identifying the type of volume; it's $D2D7 for MFS volumes and $4244 for volumes that support HFS calls.
`ioVDrvInfo`	The drive number of the drive containing the volume.
`ioVDRefNum`	For online volumes, the reference number of the I/O driver for the drive identified by `ioVDrvInfo`.
`ioVFSID`	The file-system identifier. It indicates which file system is servicing the volume; it's zero for File Manager volumes and nonzero for volumes handled by an external file system.
`ioVBkUp`	The date and time the volume was last backed up (it's 0 if never backed up).
`ioVSeqNum`	Used internally.
`ioVWrCnt`	The volume write count.
`ioVFilCnt`	The total number of files on the volume.
`ioVDirCnt`	The total number of directories (not including the root directory) on the volume.
`ioVFndrInfo`	Information used by the Finder.

Extended I/O Parameter Block

The extended I/O parameter block is needed for low-level access to disk addresses beyond 4 GB. It is used exclusively by `PBRead` and `PBWrite` calls when performing I/O operations at offsets greater than 4 GB. To indicate that you are using an `XIOParam` record, you should set the `kUseWidePositioning` bit in the `ioPosMode` field.

Because file sizes are limited to 2 GB, the regular `IOParam` record should always be used when performing file level I/O operations. The extended parameter block is intended only for Device Manager I/O operations to large block devices at offsets greater than 4 GB.

The only change in the parameter block is the parameter `ioWPosOffset`, which is of type `int64`.

```
struct XIOParam {
   QElemPtr      qLink;       // next queue entry
   short         qType;       // queue type
   short         ioTrap;      // routine trap
   Ptr           ioCmdAddr;   // routine address
   ProcPtr       ioCompletion;// pointer to completion routine
   OSErr         ioResult;    // result code
   StringPtr     ioNamePtr;   // pointer to pathname
```

```
   short         ioVRefNum;   // volume specification
   short         ioRefNum;    // file reference number
   char          ioVersNum;   // not used
   char          ioPermssn;   // read/write permission
   Ptr           ioMisc;      // miscellaneous
   Ptr           ioBuffer;    // data buffer
   unsigned long ioReqCount;  // requested number of bytes
   unsigned long ioActCount;  // actual number of bytes
   short         ioPosMode;   // positioning mode (wide mode set)
   int64         ioWPosOffset;// wide positioning offset
};
```

Field descriptions

`ioRefNum` The file reference number of an open file.

`ioVersNum` A version number. This field is no longer used and you should always set it to 0.

`ioPermssn` The access mode.

`ioMisc` Depends on the routine called. This field contains either a new logical end-of-file, a new version number, a pointer to an access path buffer, or a pointer to a new pathname. Because `ioMisc` is of type `Ptr`, you'll need to perform type coercion to interpret the value of `ioMisc` correctly when it contains an end-of-file (a `LongInt` value) or version number (a `SignedByte` value).

`ioBuffer` A pointer to a data buffer into which data is written by `_Read` calls and from which data is read by `_Write` calls.

`ioReqCount` The requested number of bytes to be read, written, or allocated.

`ioActCount` The number of bytes actually read, written, or allocated.

`ioPosMode` The positioning mode for setting the mark. Bits 0 and 1 of this field indicate how to position the mark; you can use the following predefined constants to set or test their value:

```
CONST
fsAtMark = 0;  {at current mark}
fsFromStart = 1;{from beginning of file}
fsFromLEOF = 2;{from logical end-of-file}
fsFromMark = 3;{relative to current mark}
```

You can set bit 4 of the `ioPosMode` field to request that the data be cached, and you can set bit 5 to request that the data not be cached. You can set bit 6 to request that any data written be immediately read; this ensures that the data written to a volume exactly matches the data in memory. To request a read-verify operation, add the following constant to the positioning mode:

```
CONST
rdVerify = 64; {use read-verify mode}
```

	You can set bit 7 to read a continuous stream of bytes, and place the ASCII code of a newline character in the high-order byte to terminate a read operation at the end of a line.
`ioPosOffset`	The offset to be used in conjunction with the positioning mode.

New Extended Function

This section describes the extended `PBXGetVolInfo` function that provides volume size information for volumes greater than 4 GB.

Before using the new extended call, you should check for availability by calling the `Gestalt` function. Make your call to `Gestalt` with the `gestaltFSAttr` selector to check for new File Manager features. The response parameter has the `gestaltFSSupports2TBVolumes` bit set if the File Manager supports large volumes and the new extended function is available.

PBXGetVolInfo

You can use the `PBXGetVolInfo` function to get detailed information about a volume. It can report volume size information for volumes up to 2 terabytes.

```
pascal OSErr PBXGetVolInfo (XVolumeParam paramBlock, Boolean
async);
```

`paramBlock`	A pointer to an extended volume parameter block.
`async`	A Boolean value that specifies asynchronous (true) or synchronous (false) execution.

An arrow preceding a parameter indicates whether the parameter is an input parameter, an output parameter, or both:

Arrow	Meaning
→	Input
←	Output
↔	Both

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result code of the function.
↔	`ioNamePtr`	`StringPtr`	Pointer to the volume's name.
↔	`ioVRefNum`	`short`	On input, a volume specification; on output, the volume reference number.

→	ioXVersion	unsigned long	Version of XVolumeParam (value = 0).
→	ioVolIndex	short	Index used for indexing through all mounted volumes.
←	ioVCrDate	unsigned long	Date and time of initialization.
←	ioVLsMod	unsigned long	Date and time of last modification.
←	ioVAtrb	unsigned short	Volume attributes.
←	ioVNmFls	unsigned short	Number of files in the root directory.
←	ioVBitMap	unsigned short	First block of the volume bitmap.
←	ioVAllocPtr	unsigned short	Block where the next new file starts.
←	ioVNmAlBlks	unsigned short	Number of allocation blocks.
←	ioVAlBlkSiz	unsigned long	Size of allocation blocks.
←	ioVClpSiz	unsigned long	Default clump size.
←	ioAlBlSt	unsigned short	First block in the volume block map.
←	ioVNxtCNID	unsigned long	Next unused catalog node ID.
←	ioVFrBlk	unsigned short	Number of unused allocation blocks.
←	ioVSigWord	unsigned short	Volume signature.
←	ioVDrvInfo	short	Drive number.
←	ioVDRefNum	short	Driver reference number.
←	ioVFSID	short	File system handling this volume.
←	ioVBkUp	unsigned long	Date and time of last backup.
←	ioVSeqNum	unsigned short	Used internally.
←	ioVWrCnt	unsigned long	Volume write count.
←	ioVFilCnt	unsigned long	Number of files on the volume.
←	ioVDirCnt	unsigned long	Number of directories on the volume.
←	ioVFndrInfo[8]	long	Used by the Finder.
←	ioVTotalBytes	uint64	Total number of bytes on the volume.
←	ioVFreeBytes	uint64	Number of free bytes on the volume.

DESCRIPTION

The PBXGetVolInfo function returns information about the specified volume. It is similar to the PBHGetVInfo function described in *Inside Macintosh: Files* except that it returns additional volume space information in 64-bit integers.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for `PBXGetVolInfo` are

Trap macro	Selector
`_HFSDispatch`	$0012

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsvErr`	–35	No such volume
`paramErr`	–50	No default volume

CHAPTER 7

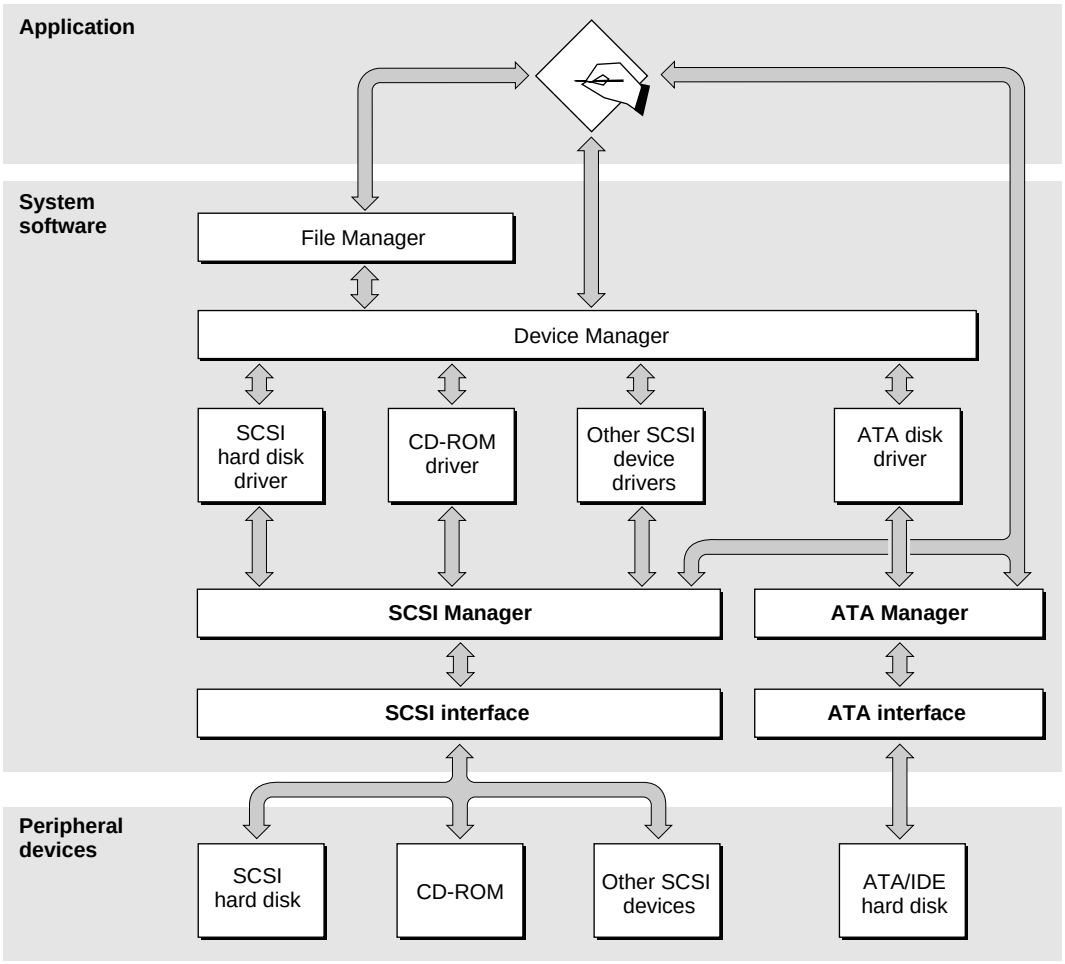
Software for the ATA (IDE) Hard Disk

This chapter describes the system software that controls ATA (AT-Attachment) devices, such as ATA hard disk drives (sometimes referred to as integrated drive electronics (IDE) drives) installed in a Macintosh computer. To use the information in this chapter, you should already be familiar with writing programs for the Macintosh computer that call device drivers to manipulate devices directly. You should also be familiar with the ANSI specification X3.279-1996 "AT Attachment Interface with Extensions (ATA-2)".

Introduction to ATA Software

Support for ATA disk drives is incorporated in the ROM software. System software for controlling ATA disk drives is provided by the ATA disk driver, which is loaded into RAM from the drives media by the ATA Manager. The relationship of the ATA disk driver and the ATA Manager is shown in Figure 7-1.

Figure 7-1 Relationship of the ATA Manager to the Macintosh system architecture



At the system level, the ATA disk driver and ATA Manager work in the same way that the SCSI Manager and associated SCSI device drivers work. The ATA disk driver provides drive partition, data management, and error-handling services for the Macintosh Operating System as well as support for determining device capacity and controlling device-specific features. The ATA Manager provides an interface to the ATA disk drive for the ATA disk driver.

ATA disk drives appear on the desktop the same way SCSI disk drives currently do. Except for applications that perform low-level services, such as formatting and partitioning utilities, applications interact with the ATA disk drives in a device-independent manner through the File Manager.

ATA Disk Driver

The ATA disk driver has the following features:

- Uses the ATA Manager for system and bus independence.
- Supports multiple partitions (volumes).
- Recognizes both Macintosh partitioned and non-partitioned media.
- Adheres to the New Driver Rules described in *Designing PCI Cards and Drivers for the Macintosh Family.*
- Supports both synchronous and asynchronous requests from the file system.

The ATA disk driver supports all ATA drives that adhere to the ANSI ATA specification X3.279-1996.

The ATA disk driver relies on the services of the ATA Manager, which provides the ATA protocol engine and relieves the driver of system and bus dependencies. The main functions of the driver are managing the media and monitoring the status of the drive.

The ATA disk driver is responsible for providing block-oriented access to the storage media. The file systems treat the media as one or more logical partitions or volumes in which data at any address can be read or written indefinitely.

The ATA disk driver provides operating system–dependent services through a set of driver routines required to interface with the Macintosh Operating System. In addition, it provides additional control and status functions that are specific to this implementation of the ATA disk driver. The required disk driver routines, as specified in *Inside Macintosh: Devices*, are `open`, `close`, `prime`, `control`, and `status`.

There are two versions of the ATA disk driver, a RAM-based version which is installed on the drive media by the Drive Setup application, and a ROM-resident version. At system startup time, if a RAM-based driver is not found on the ATA drive media by the ATA Manager, the ATA disk driver in the ROM is selected as the driver for the drive. Note that this is different from the SCSI driver loading sequence, which always requires a RAM-based driver be installed on the media. The ATA disk driver in ROM is a subset of the ATA disk driver on the media and should not be used for normal operation. It provides emergency access to the ATA drive. The ATA disk driver installed on the media by the Drive Setup application provides the latest features and optimal performance.

The RAM-based ATA disk driver supports all modes of PIO and DMA operations as defined in the ANSI ATA-2 Specification. When the driver is opened for an ATA drive, the ATA disk driver configures the ATA Manager and the drive for optimal performance based upon both the system and drive capabilities. Typically, DMA modes are selected over PIO modes.

The ATA disk driver supports conservation of system power by spinning down the disk drive to reduce power consumption. Spinning down the drive also flushes the drive write cache to prevent data loss. The ATA disk driver spins down the disk drive in response to a Sleep demand, the "Set Power Mode" control call (csCode 70), system shutdown and restart, and when no access has been made to the drive within the time specified in the Energy Saver Control Panel.

The ATA disk driver usually has a driver reference number of –54 (decimal), but may also have a different reference number if -54 is taken when the driver is loaded. The driver name is `.ATDISK`. Like all Macintosh device drivers, the ATA disk driver can be called by using either the driver reference number or the driver name `.ATDISK`.

The ATA disk driver does not provide request queuing. All driver requests are either completed immediately or are passed to the ATA Manager for further processing. However, the driver does process asynchronous requests using the ATA Manager to notify it when an operation has completed.

ATA Manager

The ATA Manager manages the ATA controller and its protocol. It provides data transport services between ATA devices and the system, directing commands to the appropriate device and handling interrupts from the devices.

The ATA Manager schedules I/O requests from the ATA disk driver, the operating system, and applications. It is also responsible for managing the hardware interface to the ATA controller electronics.

When making calls to the ATA Manager you have to pass and retrieve parameter information through a parameter block. The size and content of the parameter block depends on the function being called. However, all calls to the ATA Manager have a common parameter block header structure. The structure of the `ataPBHdr` parameter block is common to all ATA parameter block data types. Several additional ATA parameter block data types have been defined for the various functions of the ATA Manager. The additional parameter block data types, which are specific to the function being called, are described in "ATA Manager Reference" beginning on page 110.

ATA Disk Driver Reference

This section describes the Macintosh device driver routines provided by the ATA disk driver. The information in this section assumes that you are already familiar with how to use device driver services on the Macintosh computer. If you are not familiar with

Macintosh device drivers, refer to the chapter "Device Manager" in *Inside Macintosh: Devices* for additional information.

High-Level Device Manager Routines

The ATA disk driver supports the required set of routines for handling requests from the Device Manager, as defined in the chapter "Device Manager" of *Inside Macintosh: Devices*. Those routines are briefly defined here for convenience. Additional control functions supported in the ATA disk driver are defined in "ATA Disk Driver Control and Status Functions" beginning on page 99.

Open Routine

The `open` routine should not be called to open the ATA disk driver. The ATA disk driver requires a physical drive ID from the ATA Manager, and is called by the ATA Manager after being loaded from the drive media. An open call to the ATA disk driver returns a result of `openErr` if it has not been opened previously, and returns a result of `noErr` and does not reopen if it is already open.

When opened, the ATA disk driver initializes itself for the drive specified and registers itself for control of the drive with the ATA Manager. The driver installs itself in the system Unit Table and installs a system Drive Queue entry for each file system partition (volume) found on the media. After opening the ATA disk driver is able to respond to all other Close, Prime, Status, and Control calls.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`openErr`	–23	Could not open the driver
`DRVRCantAllocate`	–1793	Global memory allocation error
`ATABufFail`	–1796	Device buffer test failed

Close Routine

The `close` routine instructs the ATA disk driver to terminate execution. The driver deregisters for control of the drive with the ATA Manager, removes the Drive Queue entries for each volume associated with the drive, and deallocates all memory used during operation. The driver does not remove itself from the Unit Table.

RESULT CODES

`noErr`	0	Successful completion, no error occurred

Prime Routine

The `prime` routine performs logical block read and write operations to a specified volume with automatic retries on errors. The driver accepts either the standard 32-bit address or a 64-bit large volume address, both of which must be aligned on a 512 byte boundary representing a logical block address on the volume. The `prime` routine performs either a `read` or `write` command as specified by the caller.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`ioErr`	–36	I/O error
`paramErr`	–50	Invalid parameter specified
`nsDrvErr`	–56	No such drive installed

Status Routine

The `status` routine returns status information about the ATA disk driver. The type of information returned is specified in the `csCode` field and the information itself is pointed to by the `csParamPtr` field.

The status functions supported by the ATA disk driver are shown in Table 7-1.

Table 7-1 Status functions supported by the ATA disk driver

Value of `csCode`	Definition
8	Return drive status information
43	Return driver Gestalt information
44	Return boot partition
45	Return partition mounting status
46	Return partition write protect status
70	Power mode status information

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`statusErr`	–18	Unimplemented status function; could not complete requested operation
`nsDrvErr`	–56	No such drive installed

Control Routine

The ATA driver implements many of the control functions supported by the SCSI hard disk driver and defined in *Inside Macintosh: Devices*. The ATA disk driver also implements several new functions defined in *Designing PCI Cards and Drivers for Power Macintosh Computers*. The control functions are listed below and described in "ATA Disk Driver Control and Status Functions" beginning on page 99.

Table 7-2 Control function supported by the ATA disk driver

Value of csCode	Definition
5	Verify media
6	Format media
7	Eject media
21	Return drive icon
22	Return media icon
23	Return drive characteristics
44	Enable partition as startup partition
45	Enable partition to be mounted
46	Set partition write protected
48	Disable partition mounting
49	Disable partition write protection
60	Mount volume
70	Set power mode

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	No such drive installed

ATA Disk Driver Control and Status Functions

The ATA disk driver supports a standard set of control functions for ATA disk drive devices. The functions are used for control, status, and power management.

verify

The `verify` control function requests a read verification of the data on the ATA hard drive media. This function performs no operation and returns `noErr` if the logical drive number is valid.

An arrow preceding a parameter indicates whether the parameter is an input parameter, an output parameter, or both.

Arrow	Meaning
→	Input
←	Output
↔	Both

Parameter block

→	`csCode`	A value of 5.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

format

The `format` control function initializes the hard drive for use by the operating system. Because ATA hard drives are low-level formatted at the factory, this function does not perform any operation. The driver always returns `noErr` if the logical drive number is valid.

Parameter block

→	`csCode`	A value of 6.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

eject

The `eject` control function is used by the driver to determine when a volume becomes unmounted. If the unmounted volume was the last mounted volume of the drive, the drive is placed in a low power mode to conserve power. If the drive is also ejectable (PCMCIA, for example), a drive ejection is initiated.

Parameter block

→	`csCode`	A value of 7.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

return drive icon

The `return drive icon` control function returns a pointer to the device icon and the device location string. The drive icon for ATA hard disk devices is shown in Figure 7-2.

Figure 7-2 ATA hard disk drive icon

Parameter block

→	`csCode`	A value of 21.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`csParam[0–1]`	Address of drive icon and location string (information is in `ICN#` format).
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

return media icon

The `return media icon` control function returns a pointer to the media icon and the location string. The media icon will differ depending on the media (hard drive, CD-ROM drive, or PCMCIA drive).

Parameter block

→	`csCode`	A value of 22.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`csParam[0-1]`	Address of drive icon and location string (information is in `ICN#` format).
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

return drive characteristics

The `return drive characteristics` function returns information about the characteristics of the specified drive as defined in *Inside Macintosh,* Volume V.

Parameter block

→	`csCode`	A value of 23.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`csParam[0-1]`	Drive information. $0601 = primary, fixed, SCSI, internal. $0201 = primary, removable, SCSI, internal.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

enable startup partition

The `enable startup partition` control function enables the specified partition to be the startup (boot) partition. The partition is specified either by its logical drive number or its block address on the media. The current entry for the boot partition is

cleared. A `controlErr` is returned if the partition does not have a partition map entry on the media or could not be enabled as the startup partition.

Parameter block

→	`csCode`	A value of 44.
→	`ioVRefNum`	The logical drive number or 0 if using partition block address
→	`csParam[]`	The partition block address (long) if `ioVRefNum` param = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

enable partition mounting

The `enable partition mounting` control function enables the specified partition to be mounted when the drive is recognized. The partition is specified either by its logical drive number or its block address on the media. A `controlErr` is returned if the partition does not have a partition map entry on the media or could not be enabled for mounting.

Parameter block

→	`csCode`	A value of 45.
→	`ioVRefNum`	The logical drive number or 0 if using partition block address
→	`csParam[]`	The partition block address (long) if `ioVRefNum` param = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

enable partition write protect

The `enable partition write protect` control function enables software write protection on the specified partition. The partition is specified either by its logical drive number or its block address on the media. A `controlErr` is returned if the partition does not have a partition map entry on the media or write protection could not be enabled for the partition.

Parameter block

→	`csCode`	A value of 46.
→	`ioVRefNum`	The logical drive number or 0 if using partition block address
→	`csParam[]`	The partition block address (long) if `ioVRefNum` param = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

clear partition mounting

The `clear partition mounting` control function prevents a partition from being mounted when the drive is recognized. The partition is specified either by its logical drive number or its block address on the media. A `controlErr` is returned if the partition does not have a partition map entry on the media or partition mounting could not be cleared.

Parameter block

→	`csCode`	A value of 48.
→	`ioVRefNum`	The logical drive number or 0 if using partition block address
→	`csParam[]`	The partition block address (long) if `ioVRefNum` param = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

clear partition write protect

The `clear partition write protect` control function disables software write protection for the specified partition. The partition is specified either by its logical drive number or its block address on the media. A `controlErr` is returned if the partition does not have a partition map entry on the media or write protection could not be cleared.

Parameter block

→	`csCode`	A value of 49.
→	`ioVRefNum`	The logical drive number or 0 if using partition block address
→	`csParam[]`	The partition block address (long) if `ioVRefNum` param = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

mount volume

The `mount volume` control function instructs the drive to post a Disk Inserted event for the specified partition. The partition is specified either by its logical drive number or its block address on the media.

Parameter block

→	`csCode`	A value of 60.
→	`ioVRefNum`	The logical drive number or 0 if using partition block address
→	`csParam[]`	The partition block address (long) if `ioVRefNum` param = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

set power mode

The `set power mode` control function changes the drive power mode to one of four modes: active, standby, idle, and sleep. It can be used to reduce drive power consumption.

In the idle mode, the nonessential electronics on the ATA hard drive are disabled. For example, the read and write channels are disabled during the idle state. The spindle motor remains enabled during the idle state, so the drive still responds immediately to any commands requesting media access.

In the standby mode, the head is parked and the spindle motor is disabled. The drive interface remains active and is still capable of responding to commands. However, it can

take several seconds to respond to media access commands, because the drive's spindle motor must return to full speed before media access can take place.

In the sleep mode, both the drive interface and the spindle motor are disabled. The driver must reset and reconfigure the drive before another access to the drive can be made. Since many drives do not support the sleep mode, and because there is little power savings difference between standby and sleep modes, the ATA disk driver may put the drive in standby mode instead.

Parameter block

→	`csCode`	A value of 70.
→	`ioVRefNum`	The logical drive number.
→	`csParam[0]`	The most significant byte contains one of the following codes: 0 = enable the active mode 1 = enable the standby mode 2 = enable the idle mode 3 = enable the sleep mode
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred.
`controlErr`	–17	The power management information couldn't be returned due to a manager error.
`nsDrvErr`	–56	The specified logical drive number does not exist.

drive info

The ATA disk driver provides a drive status function for retrieving status information from the drive. The `drive info` status function returns the same type of information that disk drivers are required to return for the `status` function, as described in the chapter "Device Manager" in *Inside Macintosh: Devices*.

Parameter block

→	`csCode`	A value of 8.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	The `csParam` field contains status information about the internal ATA disk drive.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

driver gestalt

The `driver gestalt` status function provides the application information about the ATA disk driver and the attached device. Several calls are supported under this function. A Gestalt selector is used to specify a particular call.

The `DriverGestaltParam` data type defines the ATA Gestalt structure. Refer to *Designing PCI Cards and Drivers for the Macintosh Family* for information related to the ATA gestalt structure.

The fields `driverGestaltSelector` and `driverGestaltResponse` are 32-bit fields that contain the gestalt selector and possible responses. The selectors and responses are defined in the parameter block definition.

Parameter block

→	`csCode`	A value of 43.
→	`ioVRefNum`	The logical drive number.
→	`driverGestaltSelector`	Gestalt function selector. This is a 32-bit ASCII field containing one of the following selectors:
		`sync` Indicate synchronous or asynchronous driver.
		`devt` Specify type of device the driver is controlling.
		`intf` Specify the device interface.
		`boot` Specify PRAM value to designate this driver or device.
		`vers` Specify the version number of the driver.
		`lpwr` Indicates low power mode support.
		`purg` Request if the driver can be closed and or purged.
		`wide` Indicates large volume support.
		`ejec` Eject control function requirements.
←	`driverGestaltResponse`	Returned result based on the driver gestalt selector. The possible four-character return values are:
		`TRUE` If the `sync` driver selector is specified, this Boolean value indicates that the driver is synchronous; a value of `FALSE` indicates asynchronous.
		`'disk'` If the `devt` driver selector is specified, this value indicates a hard disk driver.
		`'ide '` If the `intf` driver selector is specified, this value indicates the interface is ATA.
		nnnn If the `vers` selector is specified, the current version number of the driver is returned.

		TRUE	If the `lpwr` selector is specified, this value indicates the power mode control and status function are supported.
		TRUE	If the `wide` selector is specified, this value indicates the driver supports large volumes.
		value	If the `ejec` selector is specified, this value indicates when the `ejec` call should be made. Bit 0, if set, means don't issue eject call on Restart. Bit 1, if set, means don't issue eject call on Shutdown. If the `purg` selector is specified, this value indicates whether the driver can close and or purged from memory. A value of 0 indicates that the driver cannot be closed. A value of 3 indicates that the driver can be closed, but not purged. A value of 7 indicates that the driver can be both closed and purged from memory.
←	`ioResult`		See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`statusErr`	–18	Unknown selector was specified
`nsDrvErr`	–56	The specified logical drive number does not exist

get startup partition

The `get startup partition` status function returns 1 if the specified partition is the startup partition, 0 if it is not. The partition is specified either by its logical drive number or its block address on the media.

Parameter block

→	`csCode`	A value of 44.
→	`ioVRefNum`	The logical drive number or 0 if using partition block address
→	`csParam[]`	The partition block address (long) if `ioVRefNum` param = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

get partition mount status

The `get partition mount status` status function returns 1 if the specified partition has mounting enabled, 0 if not enabled or the partition does not have a partition map entry on the media. The partition is specified either by its logical drive number or its block address on the media.

Parameter block

→	`csCode`	A value of 46.
→	`ioVRefNum`	The logical drive number or 0 if using partition block address
→	`csParam[]`	The partition block address (long) if `ioVRefNum` param = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

get partition write protect status

The `get partition write protect status` status function returns 1 if the specified partition is software write protected, 0 if it is not. The partition is specified either by its logical drive number or its block address on the media.

Parameter block

→	`csCode`	A value of 45.
→	`ioVRefNum`	The logical drive number or 0 if using partition block address
→	`csParam[]`	The partition block address (long) if `ioVRefNum` param = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

get power mode

The `get power mode` status function returns the current power mode state of the internal hard disk.

Parameter block

→	`csCode`	A value of 70.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`csParam[]`	The most significant byte of this field contains one of the following values: 1 = drive is in standby mode. 2 = drive is in idle mode. 3 = drive is in sleep mode.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred.
`statusErr`	–18	The power management information couldn't be returned due to a manager error.
`nsDrvErr`	–56	The specified logical drive number does not exist.

ATA Manager Reference

This section defines the data structures and functions that are specific to the version 3.0 of the ATA Manager. The section "The ATA Parameter Block" shows the data structure of the ATA parameter block. Version 3.0 of the ATA Manager supports DMA data transfers. The section "Setting Data Transfer Timing," discusses how the ATA Manager interacts with ATA devices to setup DMA transfers. The "Functions" section describes the functions for managing and performing data transfers through the ATA Manager.

The ATA Parameter Block

This section defines the fields that are common to all ATA Manager functions that use the ATA parameter block. The fields used for specific functions are defined in the description of the functions to which they apply. You use the ATA parameter block for all calls to the ATA Manager. The `ataPBHdr` data type defines the ATA parameter block.

ATA Manager 3.0 defines ATA parameter block version 3, which is required for the specification of ANSI ATA-2 compliant transfer timings, and DMA timing in particular. Parameter block versions 1 and 2 are still supported, but full use of version 3 is recommended when the best data transfer performance of the device is required.

The parameter block includes a field, `MgrFCode`, in which you specify the function selector for the particular function to be executed; you must specify a value for this field. Each ATA function may use different fields of the ATA parameter block for parameters specific to that function.

An arrow preceding the comment indicates whether the parameter is an input parameter, an output parameter, or both.

Arrow	Meaning
→	Input
←	Output
↔	Both

The following unique typedef identifiers are used in the ATA Manager parameter block and function definitions:

`SInt8`	A signed 8-bit field
`SInt16`	A signed 16-bit field
`SInt32`	A signed 32-bit field
`UInt8`	An unsigned 8-bit field
`UInt16`	An unsigned 16-bit field
`UInt32`	An unsigned 32-bit field

The ATA parameter block header structure is defined as follows:

```
typedef    struct    ataPBHdr       /* ATA Manager parameter */
                                    /* block header structure */
{
           Ptr       ataLink;       /* Reserved, initiaize to 0 */
           SInt16    ataQType;      /* Type byte */
           UInt8     ataPBVers;     /* → Parameter block */
                                    /* version number */
           UInt8     hdrReserved;   /* Reserved */
           Ptr       hdrReserved2;  /* Reserved */
           ProcPtr   ataCompletion; /* Universal completion */
                                    /* routine pointer */
           OSErr     ataResult;     /* ← Returned result */
           UInt8     MgrFCode;      /* → Manager function code */
           UInt8     ataIOSpeed;    /* → I/O timing class */
           UInt16    ataFlags;      /* → Control options */
           SInt16    hdrReserved3;  /* Reserved */
           long      deviceID;      /* → Device ID */
           UInt32    TimeOut;       /* → Transaction timeout */
                                    /* value */
           Ptr       ataPtr1;       /* Client storage Ptr 1 */
           Ptr       ataPtr2;       /* Client storage Ptr 2 */
           UInt16    ataState;      /* Reserved, init to 0 */
           SInt16    intSemaphores; /* Reserved */
           SInt32    hdrReserved5;  /* Reserved */
} ataPBHdr;
```

Field descriptions

`ataLink` This field is reserved for use by the ATA Manager. It is used internally for queuing I/O requests. It must be initialized to 0 before calling the ATA Manager and should be ignored upon return. This field should not be changed until the requested operation has completed.

`ataQType` This field is the queue type byte for safety check. It should be initialized to 0.

`ataPBVers` This field contains the parameter block structure version number. Values 1 through 3 are currently supported. Any higher values or a value of 0 result in a `paramErr`.

`hdrReserved` This field is reserved for future use. To ensure future compatibility, all reserved fields should be set to 0.

`hdrReserved2` This field is reserved for future use. To ensure future compatibility, all reserved fields should be set to 0.

`ataCompletion` This field contains the completion routine pointer to be called on completion of the request. When this field is set to 0, it indicates a synchronous I/O request; a nonzero value indicates an asynchronous I/O request. The routine this field points to is called either when the request has finished without error or when the request has terminated due to an error. This field is valid for any manager request. The completion routine is called as follows:

```
pascal void (*RoutinePtr) (ataPB *)
```

The completion routine is called with the associated manager parameter block in the stack.

`ataResult` Completion status. This field is returned by the ATA Manager after the request is completed. The value in this field is invalid until the operation is complete. Refer to Table 7-7 on page 144 for a list of the possible error codes returned in this field.

`MgrFCode` This field is the function selector for the ATA Manager. The functions are defined in Table 7-4 on page 118. An invalid code in this field results in an `ATAFuncNotSupported` error.

`ataPIOSpeed` This field specifies the I/O speed requirement for the ATA device. It is ignored in version 3.0 of the ATA Manager. The method for determining the I/O speed for version 3 of the ATA Manager is provided in the `ATA_SetDevConfig` function description.

For parameter block versions 1 and 2, this field specifies the I/O cycle timing requirement of the specified device. This field should contain the equivalent of word 51 of the identify drive data, as defined in the ATA-2 specification. Values 0 through 3 are supported by version 2 of the ATA Manager. See the ATA-2 specification for the definitions of the timing values. If a timing value higher than one supported is specified, the manager operates in the fastest timing mode supported by the manager. Until the timing value is determined by examining the identify drive data returned by the `ATA_Identify` function, the client should request operations using the slowest mode (PIO mode 0).

	In ATA Manager version 1, the value in this field is always valid. That is, this timing value is used to complete the requested operation. With version 2, the value in this field is only valid if the `CurrentSpeed` bit is set to 0 in the `ataFlags` field. If the `CurrentSpeed` bit is set to 1, the manager uses the timing mode set previously by the `ATA_SetDevConfig` command for the device, or the default value, which is mode 0.
`ataFlags`	This 16-bit field contains control settings that indicate special handling of the requested function. The control bits are defined in Table 7-3 on page 114.
`hdrReserved3`	This field is reserved for future use. To ensure future compatibility, all reserved fields should be set to 0.

`deviceID`	A number that uniquely identifies an ATA device. This field consists of the following structure:
	`typedef struct /* Device ID structure */`
	`{`
	`ushort Reserved; /* The upper word is reserved */`
	`ushort devNum; /* Consists of device ID */`
	`ushort busNum /* bus ID */`
	`} deviceIdentification;`
	Version 3. 0 of the ATA Manager supports two ATA devices per bus. The devices are physically numbered 0 and 1 respectively. Earlier versions of the ATA Manager used an unsigned 16-bit integer to specify the device number. In version 3.0 of the ATA Manager the `devNum` is used to distinguish between two devices on the bus specified by `busNum`. In systems with only one ATA device this value is always 0.

`TimeOut`	This field specifies the transaction timeout value in milliseconds. A value of zero disables the transaction timeout detection.
`ataPtr1`	This pointer field is available for application use. It is not modified or used by the ATA Manager.
`ataPtr2`	This pointer field is available for application use. It is not modified or used by the ATA Manager.
`ataState`	This field is used by the ATA Manager to keep track of the current bus state. This field must contain zero when calling the ATA Manager.
`intSemaphores`	This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.
`hdrReserved5`	This field is reserved for future use. To ensure future compatibility, all reserved fields should be set to 0.

Table 7-3 describes the functions of the control bits in the `ataFlags` field.

Table 7-3 Control bits in the `ataFlags` field

Name	Bit	Definition
—	0–2	Reserved.
`RegUpdate`	3	When set to 1 this bit indicates that a set of device registers should be reported back on completion of the request. This bit is valid for the `ATA_ExecI/O` function only. Refer to the description on page 118 for details. The following device registers are reported back: Sector count register Sector number register Cylinder register(s) SDH register
`ProtocolType`	5-4	These two bits specify the type of command: 00 = Standard ATA 11 = ATAPI These bits indicate how the protocol should be handled for the command type. Setting the bits to ATAPI and providing a non-zero packet command pointer indicates that a packet command should be sent prior to any data transfers. For ATA command values of A0 and A1 hexadecimal, this field should contain the ATAPI setting. For all other ATA commands, the field must contain the ATA setting.
—	6	Reserved
`UseDMA`	7	When set to 1, this bit indicates the data transfer is to be via DMA. DMA transfers are only valid with version 3.0 or higher of the ATA Manager and on system hardware that supports DMA. DMA transfers to and from ATA devices use different command codes from the PIO transfers. The state of this bit must correspond to the command code.

continued

Table 7-3 Control bits in the `ataFlags` field (continued)

Name	Bit	Definition

`SGType`	9-8	This 2-bit field specifies the type of scatter gather list passed in. This field is only valid for read/write operations.
		
		The following types are defined:
		
		00 = Scatter gather disabled
		
		01 = Scatter gather type I enabled
		
		10 = Reserved
		
		11 = Reserved
		
		When set to 0, this field indicates that the `ioBuffer` field contains the host buffer address for this transfer, and the `ioReqCount` field contains the byte transfer count.
		
		When set to 1, this field indicates that the `ioBuffer` and the `ioReqCount` fields of the parameter block for this request point to a host scatter gather list and the number of scatter gather entries in the list, respectively.
		
		The format of the scatter gather list is a series of the following structure definition:
		
		`typedef struct /* SG entry structure */`
		`{`
		`  uchar* ioBuffer; /* → Data buffer pointer */`
		`  ulong ioReqCount; /* → Byte count */`
		`} IOBlock;`

`QLockOnError`	10	When set to 0, this bit indicates that an error during the transaction should not freeze the I/O queue for the device. When an error occurs on an I/O request with this bit set to 0, the next queued request is processed following this request. When an error occurs on an I/O request with this bit set to 1, the I/O queue is halted and the user must issue an `ATA_QRelease` command to continue. A status code of hexadecimal $717 is returned for subsequent asynchronous I/O requests until the `I/O Queue Release` command is issued.
`Immediate`	11	When this bit is set to 1, it indicates that the request must be executed as soon as possible and the status of the request must be returned. It forces the request to the head of the I/O queue for immediate execution. When this bit is set to 0, the request is queued in the order received and is executed according to that order.

continued

Table 7-3 Control bits in the `ataFlags` field (continued)

Name	Bit	Definition
`ATAioDirection`	13-12	This bit field specifies the direction of data transfer. Bit values are binary and defined as follows: 00 = No data transfer 10 = Data direction in (read) 01 = Data direction out (write) 11 = Reserved These bits do not need to specify the direction of the ATAPI command packet bytes.
`ByteSwap`	14	When set to 1, this bit indicates that every byte of data prior to transmission on write operations and upon reception on read operations is to be swapped. When this bit is set to 0, it forces bytes to go out in the LSB-MSB format compatible with IBM clones. Typically, this bit should be set to 0. Setting this bit has performance implications because the byte swap is performed by the software. Use this bit with caution. ATAPI command packet bytes are swapped when this bit is set to 1.
`UseConfigSpeed`	15	When set to 1, this bit indicates that the current I/O speed setting specified in the most recent call to the `ATA_SetDevConfig` command should be used to transfer data across the ATA interface. If a `ATA_SetDevConfig` command has not been issued since power on, then the default setting of PIO mode 0 and singleword DMA mode 0 are used.

Setting Data Transfer Timing

This section defines the mechanism used by version 3.0 of the ATA Manager to setup and adjust the system hardware and software for optimized data transfers from and to the ATA devices.

Beginning with version 3.0 of the ATA Manager, all cycle timing for data transfer is accomplished through the `ATA_SetDevConfig` function, defined on page 139. The timing values in the `ataPIOSpeed` field (used by version 2.0 of the ATA Manager) in the parameter block header are ignored, and PIO, singleword DMA, and multiword DMA data transfer times are specified separately in the `ATA_SetDevConfig` function parameter block. In addition, minimum cycle times are determined for PIO and multiword DMA transfers with the `ATA_SetDevConfig` function.

The ATA-2 specification requires that ATA devices report cycle timing requirements and transfer mode information through the ATA Identify Device command. In order to synchronize the system ATA controller speed to the device speed, the Identify Device information must be interpreted by the ATA Manager. The ATA Manager receives the necessary information from the client in the `ATA_SetDevConfig` function. Five fields in

the ATA_SetDevConfig parameter block are used in various combinations to specify the timing and transfer mode values for PIO, multiword DMA, and singleword DMA data transfers.

Setting Up PIO Data Transfers

To set up PIO data transfers, the ATA Manager takes the values specified in the ataPIOSpeedMode and ataPIOCycleTime fields of the ATA_SetDevConfig parameter block to create a cycle time that approximates the specified cycle time and maintains the appropriate device signal timing requirements for the specified PIO transfer mode.

Setting Up Multiword and Singleword DMA Data Transfers

To set up multiword DMA data transfers, the ATA Manager takes the values in ataMultiSpeed and ataMultiCycle fields of the ATA_SetDevConfig parameter block to create a multiword DMA cycle time in system hardware that maintains the timing required by the multiword DMA mode while not exceeding the indicated cycle time.

To set up singleword DMA data transfers, the ATA Manager takes the value specified in the ataSingleDMASpeed field of the ATA_SetDevConfig parameter block to create the appropriate cycle timing for the device. The ATA-2 specification has no recommended timing values for singleword DMA data transfer modes, only minimum cycle times.

When both the ataSingleDMASpeed and ataMultiDMASpeed fields in the ATA_SetDevConfig function parameter block are set to zero and the UseDMA flag in the ataFlags field is set true, the ATA Manager uses singleword DMA mode 0 timing for data transfers.

The UseConfigSpeed flag of the ataFlags field in the ataPBHdr parameter block header must be set for both the ataExecIO and ATA_SetDevConfig functions to utilize new timing configuration information. When the UseConfigSpeed flag is not set, new timing values are not calculated and saved during a ATA_SetDevConfig function call. When the UseConfigSpeed flag is not set and the UseDMA flag is specified, timing is set to singleword DMA mode 0. If the UseConfigSpeed flag is not set for an ataExecIO function, PIO mode 0 timing is use for commands and PIO data transfers.

Additional reference documentation related to Identify Device data transfer timing information for ATA devices can be found in the ANSI ATA-2 specification.

Functions

This section describes the ATA Manager functions that are used to manage and perform data transfers. Each function is requested through a parameter block specific to that service. A request for an ATA function is specified by a function code within the parameter block. The entry point for all the functions is the same.

ATA Manager function names and codes are shown in Table 7-4.

Table 7-4 ATA Manager functions

Function name	Code	Description
`ATA_ExecIO`	$01	Execute ATA I/O
`ATA_MgrModifyEventMask`	$88	Modify driver event mask
`ATA_MgrDriveEject`	$89	Eject the drive
`ATA_MgrInquiry`	$90	ATA Manager inquiry
`ATA_BusInquiry`	$03	Bus inquiry
`ATA_QRelease`	$04	I/O queue release
`ATA_NOP`	$00	No operation
`ATA_Abort`	$10	Terminate command
`ATA_RegAccess`	$12	ATA device register access
`ATA_Identify`	$13	Get the drive identification data
`ATA_ResetBus`	$11	Reset ATA bus
`ATA_DrvrRegister`	$85	Register the driver reference number
`ATA_DrvrDeregister`	$87	Deregister the driver reference number
`ATA_FindRefNum`	$86	Look up the driver reference number
`ATA_GetDevConfig`	$8A	Get the device configuration
`ATA_SetDevConfig`	$8B	Set the device configuration
`ATA_GetLocationIcon`	$8C	Get device location icon and string

ATA_ExecIO

You can use the `ATA_ExecIO` function to perform all data I/O transfers to or from an ATA device. Your application must provide all of the parameters needed to complete the transaction prior to calling the ATA Manager. On return, the parameter block contains the result of the request.

A prior call to the `ATA_SetDevConfig` function is recommended to obtain the optimal performance from the device. See page 139 for information about the `ATA_SetDevConfig` function.

The manager function code for the `ATA_ExecIO` function is $01.

The parameter block associated with the ATA_ExecIO function is defined below:

```
typedef     struct            /* ATA_ExecIO structure */

{
   ataPBHdr                   /* See definition on page 111 */
   SInt8    ataStatusReg;     /* ← Last device status register */
                              /*   image */
   SInt8    ataErrorReg;      /* ← Last device error register */
                              /*  image (valid if bit 0 of */
                              /*  Status field is set) */
   SInt16   ataReserved;      /* Reserved */
   UInt32   BlindTxSize;      /* → Data transfer size */
   UInt8    *ioBuffer;        /* → Data buffer pointer */
   UInt32   ataActualTxCnt;   /* ← Actual number of bytes */
                              /*  transferred */
   UInt32   ataReserved2;     /* Reserved */
   devicePB RegBlock;         /* → Device register images */
   ATAPICmdPacket *packetCDBPtr; /* ATAPI packet command block
                                    pointer */
   UInt16   ataReserved3[6];  /* Reserved */

} ataExecIO;
```

Field descriptions

`ataPBHdr` See the definition of the ataPBHdr parameter block on page 111.

`ataStatusReg` This field contains the last device status register image. See the ATA-2 specification for status register bit definitions.

`ataErrorReg` This field contains the last device error register image. This field is valid only if the error bit (bit 0) of the `Status` register is set. See the ATA-2 specification for error register bit definitions.

`ataReserved` Reserved. All reserved fields are set to 0 for future compatibility.

`BlindTxSize` This field specifies the maximum number of bytes that can be transferred for each interrupt or detection of a data request. Bytes are transferred in blind mode (no byte level handshake). Once an interrupt or a data request condition is detected, the ATA Manager transfers up to the number of bytes specified in the field from or to the selected device. The typical number is 512 bytes.

The `BlindTxSize` field is used only for PIO transfers. It is ignored for DMA data transfers.

`ioBuffer` This field contains either the host buffer address and the requested transfer length, or the pointer to a scatter gather list and the number of scatter gather entries. If the `SGType` bits of the `ataFlags` field are set, an `IOBlk` contains the scatter gather information. The `IOBlk` is defined as follows:

```
typedef        struct
{
      uchar*   ioBuffer;   /* ↔ Data buffer ptr */
      ulong    ioReqCount; /* ↔ Transfer length */
} IOBlk;
```

`ioBuffer` This field contains the host buffer address for the number of bytes specified in the `ioReqCount` field. On returning, the `ioBuffer` field is updated to reflect data transfers. When the `SGType` bits of the `ataFlags` field are set, the `ioBuffer` field points to a scatter gather list. The scatter gather list consists of a series of `IOBlk` entries.

`ioReqCount` This field contains the number of bytes to transfer either from or to the buffer specified in `ioBuffer`. On returning, the `ioReqCount` field is updated to reflect data transfers (0 if successful; otherwise, the number of bytes that remained to be transferred prior to the error condition). When the `SGType` bits of the `ataFlags` field are set, the `ioReqCount` field contains the number of scatter gather entries in the list pointed to by the `ioBuffer` field.

`ataActualTxCnt` This field contains the total number of bytes transferred for this request. This field is currently not supported.

`ataReserved2` This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

`RegBlock` This field contains the ATA device register image structure. Values contained in this structure are written out to the device during the command delivery state. The caller must provide the image prior to calling the ATA Manager. The ATA device register image structure is defined as follows:

```
typedef      struct     /* Device register images */
{   UInt8    Features;  /* → Features register */
                        /*   image */
    UInt8    Count;     /* ↔ Sector count */
    UInt8    Sector;    /* ↔ Sector start/finish */
    UInt8    Reserved;  /* Reserved */
    UInt16   Cylinder;  /* ↔ Cylinder 68000 format */
    UInt8    SDH;       /* ↔ SDH register image */
    UInt8    Command;   /* → Command register image */
} devicePB;
```

`packetCDBPtr` This field contains the packet pointer for ATAPI. The ATAPI bit of the `ProtocolType` field must be set for this field to be valid. Setting the ATAPI protocol bit also signals the Manager to initiate the transaction without the DRDY bit set in the status register of the

device. For ATA commands, this field should contain 0 in order to insure compatibility in the future.

`ataReserved3[6]` These fields are reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_MgrInquiry

The `ATA_MgrInquiry` function gets information, such as the version number, about the ATA Manager.

The manager function code for the `ATA_MgrInquiry` function is $90.

The parameter block associated with this function is defined below:

```
typedef         struct          /* ATA inquiry structure */
{
   ataPBHdr                     /* See definition on page 111 */
   NumVersion   MgrVersion
   UInt8        MGRPBVers;      /* ← Manager PB version */
                                /*  number supported */
   UInt8        Reserved1;      /* Reserved */
   UInt16       ataBusCnt;      /* ← Number of ATA buses in
                                    system */
   UInt16       ataDevCnt;      /* ← Number of ATA devices
                                    detected */
   UInt8        ataPIOMaxMode;  /* ← Maximum PIO speed mode */
   UInt8        Reserved2;      /* Reserved */
   UInt16       Reserved3;      /* Reserved */
   UInt8        ataSingleDMAModes; /* ← Singleword DMA modes */
                                /*  supported */
   UInt8        ataMultiDMAModes;/* ← Multiword DMA modes */
                                /*  supported */
   UInt16       Reserved[16];   /* Reserved */
} ataMgrInquiry;
```

Field descriptions

`ataPBHdr` See the definition of the `ataPBHdr` parameter block on page 111.

`MgrVersion` On return, this field contains the version number of the ATA Manager.

`MGRPBVers`	This field contains the number corresponding to the latest version of the parameter block supported. A client may use any parameter block definition up to this version.
`Reserved`	Reserved. All reserved fields are set to 0 for future compatibility.
`ataBusCnt`	On return, this field contains the total number of ATA buses in the system. This field contains a zero if the ATA Manager has not been initialized.
`ataDevCnt`	On return, this field contains the total number of ATA devices detected on all ATA buses. The current architecture allows only one device per bus. This field will contain a zero if the ATA Manager has not been initialized.
`ataPIOMaxMode`	This field specifies the maximum PIO speed mode that the ATA Manager supports. Refer to the ATA-2 specification for information on mode timing.
`ataSingleDMAModes`	This bit-significant field specifies the maximum DMA mode that the manager can support. Refer to ATA-2 specification for information on DMA mode timing.
`ataMultiDMAModes`	This bit-significant field specifies the Multiword DMA transfer modes that the manager can support. The least-significant bit indicates support for Multiword DMA transfer mode 0. Refer to ATA-2 specification for information on DMA mode timing.
`Reserved[16]`	This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_BusInquiry

The `ATA_BusInquiry` function gets information about a specific ATA bus. This function is provided for possible future expansion of the Macintosh ATA architecture.

The manager function code for the `ATA_BusInquiry` function is $03.

The parameter block associated with this function is defined below:

```
typedef struct                  /* ATA bus inquiry structure */
{
  ataPBHdr                      /* See definition on page 111 */
  UInt16     ataEngineCount;    /* ← TBD; zero for now */
  UInt16     ataReserved;       /* Reserved */
  UInt32     ataDataTypes;      /* ← TBD; zero for now */
  UInt16     ataIOpbSize;       /* ← Size of ATA I/O PB */
```

```
  UInt16     ataMaxIOpbSize;     /* ← TBD; zero for now */
  UInt32     ataFeatureFlags;    /* ← TBD */
  UInt8      ataVersionNum;      /* ← HBA Version number */
  UInt8      ataHBAInquiry;      /* ← TBD; zero for now */
  UInt16     ataReserved2;       /* Reserved */
  UInt32     ataHBAPrivPtr;      /* ← Ptr to HBA private data */
  UInt32     ataHBAPrivSize;     /* ← Size of HBA private data */
  UInt32     ataAsyncFlags;      /* ← Capability for callback */
  UInt32     ataReserved3[4];    /* Reserved */
  UInt32     ataReserved4;       /* Reserved */
  SInt8      ataReserved5[16];   /* TBD */
  SInt8      ataHBAVendor[16];   /* ← HBA Vendor ID */
  SInt8      ataContrlFamily[16];/* ← Family of ATA controller */
  SInt8      ataContrlType[16];  /* ← Controller model number */
  SInt8      ataXPTversion[4];   /* ← Version number of XPT */
  SInt8      ataReserved6[4];    /* Reserved */
  SInt8      ataHBAversion[4];   /* ← Version number of HBA */
  UInt8      ataHBAslotType;     /* ← Type of slot */
  UInt8      ataHBAslotNum;      /* ← Slot number of the HBA */
  UInt16     ataReserved7;       /* Reserved */
  UInt32     ataReserved8;       /* Reserved */
} ataBusInquiry;
```

Field descriptions

`ataPBHdr` See the definition of the `ataPBHdr` structure on page 111.

`ataEngineCount` This field is currently set to 0.

`ataReserved` Reserved. All reserved fields are set to 0.

`ataDataTypes` Not supported by current ATA architecture. Returns a bit map of data types supported by this HBA. The data types are numbered from 0 to 30; 0 through 15 are reserved for Apple definition and 16 through 30 are available for vendor use. Returns 0.

`ataIOpbSize` This field contains the size of the I/O parameter block supported.

`ataMaxIOpbSize` This field specifies the maximum I/O size for the HBA. This field is currently not supported and returns 0.

`ataFeatureFlags` This field specifies supported features. This field is not supported; it returns a value of 0.

`ataVersionNum` The version number of the HBA is returned. The current version returns a value of 1.

`ataHBAInquiry` Reserved.

`ataHBAPrivPtr` This field contains a pointer to the HBA's private data area. This field is not supported; it returns a value of 0.

`ataHBAPrivSize` This field contains the byte size of the HBA's private data area. This field is not supported; it returns a value of 0.

`ataAsyncFlags`	These flags indicate which types of asynchronous events the HBA is capable of generating. This field is not supported; it returns a value of 0.
`ataHBAVendor`	This field contains the vendor ID of the HBA. This is an ASCII text field. It is not supported.
`ataContrlFamily`	Reserved.
`ataContrlType`	This field identifies the specific type of ATA controller. This field is not supported; it returns a value of 0.
`ataXPTversion`	Reserved.
`ataHBAversion`	This field specifies the version of the HBA. This field is not supported; it returns a value of 0.
`ataHBAslotType`	This field specifies the type of slot. This field is not supported; it returns a value of 0.
`ataHBAslotNum`	This field specifies the slot number of the HBA. This field is not supported; it returns a value of 0.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_QRelease

The `ATA_QRelease` function releases the frozen I/O queue of the selected device.

When the ATA Manager detects an I/O error and the `QLockOnError` bit of the parameter block is set for the request, the ATA Manager freezes the queue for the selected device. No pending or new requests are processed or receive status until the queue is released through the `ATA_QRelease` function. Only those requests with the `Immediate` bit set in the `ATAFlags` field of the `ataPBHdr` parameter block are processed. Consequently, for the ATA I/O queue release command to be processed, it must be issued with the `Immediate` bit set in the parameter block. An ATA I/O queue release command issued while the queue isn't frozen returns the `noErr` status.

The manager function code for the `ATA_QRelease` function is $04.

The parameter block associated with this function is defined below:

```
struct         ataQRelease       /* ATA QRelease structure */
{
   ataPBHdr                      /* See definition on page 111 */
   UInt16      Reserved[24];
}  ataQRelease;
```

Field descriptions

`ataPBHdr`	See the definition of the `ataPBHdr` structure on page 111.
`Reserved[24]`	Reserved. All reserved fields should be set to 0.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_NOP

The `ATA_NOP` function performs no operation across the interface and does not change the state of either the manager or the device. It returns `noErr` if the drive number is valid.

The manager function code for the `ATA_NOP` function is $00.

The parameter block associated with this function is defined below:

```
struct         ataNOP            /* ATA NOP structure */
{
   ataPBHdr                      /* See definition on page 111 */
   UInt16      Reserved[24];
}  ataQRelease;
```

Field descriptions

`ataPBHdr` See the definition of the `ataPBHdr` structure on page 111.

`Reserved[24]` Reserved. All reserved fields should be set to 0.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_Abort

The `ATA_Abort` function terminates a specified queued I/O request. This function applies to asynchronous I/O requests only. The `ATA_Abort` function searches through the I/O queue associated with the selected device and aborts the matching I/O request. The current implementation does not abort if the found request is in progress. If the specified I/O request is not found or has started processing, an `ATAUnableToAbort` status is returned. If aborted, the `ATAReqAborted` status is returned.

It is up to the application that called the `ATA_Abort` function to clean up the aborted request. Clean up includes parameter block deallocation and operating system reporting.

The manager function code for the `ATA_Abort` function is $10.

The parameter block associated with this function is defined as follows:

```
typedef      struct         /* ATA abort structure */
{
  ataPBHdr                  /* See definition on page 111 */
  ataPB*     AbortPB        /* Address of the parameter block */
                            /* of the function to be aborted */
  UInt16     Reserved[22]   /* Reserved */
} ataAbort;
```

Field descriptions

`ataPBHdr`	See the definition of the `ataPBHdr` parameter block on page 110.
`AbortPB`	This field contains the address of the I/O parameter block to be aborted.
`Reserved`	This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_RegAccess

The `ATA_RegAccess` function enables access to a particular device register of a selected device. This function is used for diagnostic and error recovery processes.

The manager function code for the `ATA_RegAccess` function is $12.

The parameter block associated with this function is defined below:

```
typedef        struct            /* Register access structure */
{
   struct      ataPBHdr          /* See definition on page 111 */
   UInt16      ataRegSelect      /* → Device register selector */
   union {
         UInt8 ataByteRegValue;  /* ↔ Byte register value to */
                                 /*  read or to be written */
         UInt16 ataWordRegValue; /* ↔ Word register value to */
                                 /*  read or to be written */
   } ataRegValue;
   UInt16      ataRegMask;       /* → Mask for registers(s) to */
                                 /*  update */
   ataTaskFile ataRegisterImage; /* ↔ Register images */
   UInt8       ataAltSDevCReg;   /* ↔ Alternate status(R) or */
                                 /*  Device Control(W) register */
```

```
                                    /*  image */
   UInt8       Reserved[3];         /* Reserved */
   UInt16      Reserved[16];        /* Reserved */
} ataRegAccess;
```

Field descriptions

`ataPBHdr` See the definition of the `ataPBHdr` parameter block on page 111.

`ataRegSelect` This field specifies which one of the device registers to access. The selectors for the registers supported by the `ATA_RegAccess` function are listed in Table 7-5. If `ataRegSelect` is "FFFF", then `ataRegMask` describes which register(s) are to be accessed as part of a multiregister access.

`ataRegValue` This field is either the source or destination of values for individual register accesses. For byte accesses, the upper half of the word is used. Word accesses (such as the data register) use the entire word. This field is the source or destination for the data register component of multiregister accesses.

`ataRegMask` This field is valid only if the `ataRegSelect` field contains 0xFFFF. It indicates which of the associated registers is to be read into or written from the `ataRegValue`, `ataRegisterImage` and `ataAltSDevCReg` fields in a single ATA Manager function call. The mask bits corresponding to the selected registers are listed in Table 7-6. Bit 0 is the least significant bit of the field.

`ataRegisterImage` An image of the task file registers. For a multiregister read, this field is the destination; it is the source for multiregister writes. Note that the data and alternate status and device control registers have individual fields separate from this field.

`ataAltSDevCReg` For multiregister writes, this field is the source for device control writes and the destination for alternate status reads.

Table 7-5 ATA register selectors

Selector name	Selector	Register description
`DataReg`	0	Data register (16-bit access only)
`ErrorReg`	1	Error register (R) or features register (W)
`SecCntReg`	2	Sector count register
`SecNumReg`	3	Sector number register
`CylLoReg`	4	Cylinder low register
`CylHiReg`	5	Cylinder high register

Table 7-5 ATA register selectors (continued)

Selector name	Selector	Register description
`SDHReg`	6	SDH register
`StatusReg` `CmdReg`	7	Status register (R) or command register (W)
`AltStatus` `DevCntr`	14	Alternate status (R) or device control (W)

The register mask selectors are defined in Table 7-6.

Table 7-6 Register mask selectors

Mask bit	Register description
0	Data register
1	Error register
2	Sector count register
3	Sector number register
4	Cylinder low register
5	Cylinder high register
6	ataTFSDH register
7	Status/command register
8-13	Reserved (set to 0)
14	Alternate status/device control register
15	Reserved (set to 0)

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_Identify

The `ATA_Identify` function returns the device identification data from the selected device. The identification data contains information necessary to perform I/O to the device. Refer to the ATA-2 Specification for the format and the information description provided by the data.

The manager function code for the `ATA_Identify` function is $13.

The parameter block associated with this function is defined below:

```
typedef     struct
{
  ataPBHdr
  SInt8     ataStatusReg;        /* ← Last ATA status image */
  sInt8     ataErrorReg;         /* ← Last ATA error image */
  SInt16    Reserved;            /* Reserved */
  UInt32    BlindTxSize;         /* ← Set to 512 on return */
  UInt8     *DataBuf;            /* ↔ Buffer for the data */
  UInt32    ataRequestCount;     /* ← Indicates remaining
                                  byte count */
  UInt32    ataActualTxCnt;      /* ← Actual transfer count */
  UInt32    ataReserved2;        /* Reserved */
  devicePB  RegBlock;            /* ← taskfile image sent */
                                 /* for the command */
  UInt16    Reserved3[8];        /* Reserved */
} ataIdentify;
```

Field descriptions

`ataPBHdr`	See the definition of the `ataPBHdr` parameter block on page 111.
`ataStatusReg`	Last ATA taskfile status register image.
`ataErrorReg`	Last ATA taskfile error register image. This field is only valid if the LSB (error bit) of the `ataStatusReg` field is set.
`BlindTxSize`	Last ATA taskfile error register image. This field is only valid if the LSB (error bit) of the `ataStatusReg` field is set. Set to 512 upon return.
`DataBuf`	A pointer to the data buffer for the device identify data. The length of the buffer must be at least 512 bytes.
`ataRequestCount`	Number of remaining bytes to transfer.
`ataActualTxCnt`	Number of bytes transferred.
`RegBlock`	Taskfile image sent to the device.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_ResetBus

The `ATA_ResetBus` function performs a soft reset operation to the selected ATA bus. The ATA interface doesn't provide a way to reset individual units on the bus. Consequently, all devices on the bus will be reset.

IMPORTANT

This function should be used with caution since it may terminate any active requests to devices on the bus. ▲

The manager function code for the `ATA_ResetBus` function is $11.

The parameter block associated with this function is defined below:

```
typedef     struct         /* ATA reset structure */
{
  ataPBHdr                 /* See definition on page 111 */
  SInt8     Status;        /* ← Last ATA status register image */
  SInt8     Reserved2;     /* Reserved */
  UInt16    Reserved[23];  /* Reserved */
} ataResetBus;
```

Field descriptions

`ataPBHdr`	See the definition of the `ataPBHdr` parameter block on page 111.
`Status`	This field contains the last device status register image following the bus reset. See the ATA-2 specification for definitions of the status register bits.
`Reserved[23]`	This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_DrvrRegister

The `ATA_DrvrRegister` function registers the driver reference number passed in for the selected drive. The function doesn't check for the existence of another driver.

The manager function code for the `ATA_DrvrRegister` function is $85.

The parameter block associated with `ataPBVers` of 1 is defined below:

```
typedef       struct             /* Driver registration */
                                 /* structure for ataPBVers 1 */
{
   ataPBHdr                      /* See definition on page 111 */
   SInt16     drvrRefNum;        /* → Driver reference number */
   UInt16     FlagReserved;      /* Reserved (should be zero)*/
   UInt16     deviceNextID;      /* Not used */
   SInt16     Reserved[21];      /* Reserved */
} ataDrvrRegister;
```

Field descriptions

`ataPBHdr`	See the `ataPBHdr` parameter block definition on page 111.
`drvrRefNum`	This field specifies the driver reference number to be registered. This value must be less than 0 to be valid.
`FlagReserved`	Reserved.
`deviceNextID`	Not used by this function.
`Reserved[21]`	This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

The parameter block associated with `ataPBVers` of 2 or greater is defined below:

```
typedef       struct             /* Driver registration */
                                 /* structure for ataPBVers 2 */
                                 /* or greater */
{
   ataPBHdr                      /* See definition on page 111 */
   SInt16     drvrRefNum;        /* → Driver reference number */
   UInt16     drvrFlags;         /* → Driver flags, set to 0 */
   UInt16     deviceNextID;      /* Not used */
   SInt16     Reserved;          /* Reserved (should be 0) */
   ProcPtr    ataEHandlerPtr;    /* → Event handler routine */
                                 /* pointer */
   SInt32     drvrContext;       /* → Value to pass in with */
                                 /* event handler */
   UInt32     ataEventMask;      /* → Masks of various events */
                                 /* for event handlers */
   SInt16     Reserved[14];      /* Reserved */
} ataDrvrRegister;
```

The version 2 parameter block also allows another type of registration; “notify-all” driver registration. The “notify-all” driver registration is identified by a value of -1 in the `deviceID` field of the header and bit 0 of `drvrFlags` set to 0. The “notify-all” driver registration is used if notification of all device insertions is desired. Registered default drivers are called if no media driver is found on the media. Typically, an INIT driver will register as a “notify-all” driver. The single driver may register as a “notify-all” driver, and then later, register for one or more devices on the bus.

Note

All PCMCIA/ATA and notify-all device drivers must register using the parameter block version 2 and utilize the event handling capability in order to insure proper operation. See the description of the `ataEHandlerPtr`, `driverContent`, and `ataEventMask` fields for additional information related to event handling.

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 111.

`drvrRefNum` This field specifies the driver reference number to be registered. This value must be less than 0 to be valid.

`drvrFlags` No bit definition has been defined for the field. This field shall be set to 0 in order to insure compatibility in the future.

`deviceNextID` Not used by this function.

`Reserved` Reserved.

`ataEHandlerPtr` A pointer to event handler routine for the driver. This routine is called whenever an event happens, and the mask bit for the particular event is set in the `ataEventMask` field.

The calling convention for the event handler is as follows:

```
pascal SInt16 (ataEHandlerPtr) (ATAEventRec*);
```

where `ATAEventRec` is defined as follows:

```
typedef     struct
{   UInt16  evenCode;   /* ATA event code */
    UInt16  phyDrvRef;  /* ID associated with */
                        /*  the event */
    SInt32  drvrContext;/* Context passed in by */
                        /*  driver */
} ATAEventRec;
```

`drvrContext` A value to be passed in when the event handler is called. This value is loaded in the `ATAEventRec` before calling the event handler.

`ataEventMask` The mask defined in this field is used to indicate whether the event handler should be called or not, based on the event. The event handler is only called if the mask for the event has been set (1). If the mask is not set (0) for an event, the ATA Manager takes no action. The following masks have been defined:

Bits	Event mask
0x00	Null event
0x01	Online event - a device has come online
0x02	Offline event - a device has gone offline
0x03	Device removed event - a device has been removed
0x04	Reset event - a device has been reset
0x05	Offline request event - a request to put the device offline has been detected
0x06	Eject request event - a request to eject a device has been detected
0x07	Configuration update event - the system configuration has changed (more devices)
0x08 - 0x1F	Reserved

`Reserved[21]` This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_DrvrDeregister

The `ATA_DrvrDeRegister` function deregisters the driver reference number passed in for the selected drive. After successful completion of this function, the driver reference number for the drive is set to 0, which indicates that there is no driver in control of this device.

The manager function code for the `ATA_DrvrDeRegister` function is $87.

For notify-all driver deregistration, the `ataEHandlerPtr` field is used to match the entry (the `deviceID` field is invalid for the notify-all driver registration/deregistration). If a driver is registered as both a "notify-all" and for a specific device, the driver must deregister for each separately.

All "notify-all" device drivers must deregister using the parameter block version 2. The version 1 and version 2 or greater parameter blocks for this function are as follows:

```
typedef       struct             /* Driver registration */
                                 /* structure for ataPBVers 1 */
{
   ataPBHdr                      /* See definition on page 111 */
   SInt16     drvrRefNum;        /* Not used*/
   UInt16     FlagReserved;      /* Reserved */
   UInt16     deviceNextID;      /* Not used */
   SInt16     Reserved[21];      /* Reserved */
} ataDrvrRegister;

typedef       struct             /* Driver registration */
                                 /* structure for ataPBVers 2 */
                                 /* or greater */
{
   ataPBHdr                      /* See definition on page 111 */
   SInt16     drvrRefNum;        /* → Driver reference number */
   UInt16     drvrFlags;         /* → Driver flags, set to 0 */
   UInt16     deviceNextID;      /* Not used */
   SInt16     Reserved;          /* Reserved (should be 0) */
   ProcPtr    ataEHandlerPtr;    /* → Event handler routine */
                                 /* pointer */
   SInt32     drvrContext;       /* → Value to pass in with */
                                 /* event handler */
   UInt32     ataEventMask;      /* → Masks of various events */
                                 /* for event handlers */
   SInt16     Reserved[14];      /* Reserved */
} ataDrvrRegister;
```

Field descriptions

`ataPBHdr`	
`drvrRefNum`	Not used for this function.
`drvrFlags`	No bits have been defined for this field. This field shall be set to 0 in order to insure compatibility in the future.
`deviceNextID`	Not used for this function.
`Reserved`	Reserved
`ataEHandlerPtr`	A pointer to the driver event handler routine. This field is only used for notify-all driver deregistration. This field is not used for other driver deregistration. Since this field is used to identify the correct "notify-all" driver entry, this field must be valid for "notify-all" driver deregistration.
`drvrContext`	Not used for this function.
`ataEventMask`	Not used for this function.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_FindRefNum

The `ATA_FindRefNum` function allows an application to determine whether a driver has been installed for a given device. You pass in a device ID and the function returns the current driver reference number registered for the given device. A value of 0 indicates that no driver has been registered. The `deviceNextID` field contains a device ID of the next device in the list. The end of the list is indicated with a value of `0xFF`.

To create a list of all drivers for the attached devices, pass in `0xFF` for `deviceID`. This causes `deviceNextID` to be filled with the first device in the list. Each successive driver can be found by moving the value returned in `deviceNextID` into `deviceID` until the function returns `0xFF` in `deviceNextID`, which indicates the end of the list.

The manager function code for the `ATA_FindRefNum` function is $86.

The parameter block associated with this function for `ataPBVers` version 1 is defined as follows:

```
typedef       struct           /* Driver registration for */
                               /*   ataPBVers version 1 */
{
   ataPBHdr                    /* See definition on page 111 */
   SInt16     drvrRefNum;      /* ← Contains the driver refNum */
   UInt16     FlagReserved;    /* Reserved */
```

```
   UInt16     deviceNextID;     /* ← Contains the next drive ID */
   SInt16     Reserved[21];     /* Reserved */
} ataDrvrRegister;
```

The parameter block associated with this function for `ataPBVer` version 2 is defined as follows:

```
typedef       struct             /* Driver registration for */
                                 /* ataPBVers version 2 */
                                 /* or greater */
{
   ataPBHdr                      /* See definition on page 111 */
   SInt16     drvrRefNum;        /* ← Driver reference number */
   UInt16     drvrFlags;         /* → Reserved, set to 0 */
   UInt16     deviceNextID;      /* Used to specify the next */
                                 /* drive ID */
   SInt16     Reserved;          /* Reserved (should be 0) */
   ProcPtr    ataEHandlerPtr;    /* ← Event handler routine */
                                 /* pointer */
   SInt32     drvrContext;       /* ← Value to pass in with */
                                 /* event handler */
   UInt32     ataEventMask;      /* ← Current setting of the */
                                 /* Mask of various events */
                                 /* for event handler */
   SInt16     Reserved[14];      /* Reserved */
} ataDrvrRegister;
```

Field descriptions

`ataPBHdr`	See the `ataPBHdr` parameter block definition on page 110.
`drvrRefNum`	On return, this field contains the reference number for the device specified in the `deviceID` field of the `ataPBHdr` data.
`FlagReserved`	This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.
`deviceNextID`	On return, this field contains the `deviceID` of the next device on the list.
`ataEHandlerPtr`	Currently registered event handler routine pointer for the selected device. This field is only valid for `ataPBVers` of 2 or greater.
`drvrContext`	Currently registered value to be passed along when the event handler is called. This field is only valid for `ataPBVers` of 2 or greater.
`ataEventMask`	Current event mask value for the selected device. This field is only valid for `ataPBVers` of 2 or greater.
`Reserved[nn]`	Reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_GetDevConfig

The `ATA_GetDevConfig` function allows an application to determine what the configuration is for a specified socket.

The manager function code for the `ATA_GetDevConfig` function is $8A.

The parameter block associated with this function is defined as follows:

```
typedef      struct
{
   ataPBHdr                     /* See definition on page 111 *
   SInt32    ConfigSetting;     /* → 32 bits of configuration */
                                /* information */
   UInt8     ataPIOSpeedMode;   /* → Default PIO mode setting*/
   UInt8     Reserved3;         /* Reserved for word alignment*/
   UInt16    pcValid;           /* → PCMCIA unique */
   UInt16    RWMultipleCount;   /* Reserved */
   UInt16    SectorsPerCyl;     /* Reserved */
   UInt16    Heads;             /* Reserved */
   UInt16    SectorsPerTrack;   /* Reserved */
   UInt16    socketNum;         /* ← Socket number */
   UInt8     socketType;        /* ← Type of socket */
   UInt8     deviceType;        /* ← Type of active device */
   UInt8     pcAccessMode;      /* → Access mode of socket */
   UInt8     pcVcc;             /* → device voltage */
   UInt8     pcVpp1;            /* → Vpp 1 voltage */
   UInt8     pcVpp2;            /* → Vpp 2 voltage */
   UInt8     pcStatus;          /* → Status register setting */
   UInt8     pcPin;             /* → Pin register setting */
   UInt8     pcCopy;            /* → Copy register setting */
   UInt8     pcConfigIndex;     /* → Option register setting */
   UInt8     ataSingleDMASpeed; /* → Single word DMA */
                                /* timing class */
   UInt8     ataMultiDMASpeed;  /* → Default Multiword DMA */
                                /* timing class */
   UInt16    ataPIOCycleTime;   /* → Default cycle time for */
                                /* PIO mode */
   UInt16    ataMultiCycleTime; /* → Default cycle time for */
```

```
                                    /* multiword DMA mode */
   UInt16     Reserved[7];          /* Reserved*/
} ataGetDevConfig;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 110.

`ConfigSetting` This 32-bit field contains various configuration information. The bits have the following definitions:

Bits 0-5: reserved
Bit 6: ATAPI packet DRQ handling setting
0 = Wait for an interrupt before sending the ATAPI command packet
1 = Wait for the assertion of DRQ in the status register before sending the ATAPI command packet. This is the default setting.
Bits 7-31: Reserved, set to 0

`ataPIOSpeedMode` This field indicates the value for the PIO mode currently used for commands and PIO data transfers. This value can be modified with the ATA_SetDevConfig function. In parameter block versions 1 and 2, this field is an integer. In parameter block versions 3 and higher, this field is bit-significant, where the low-order bit indicates that PIO mode 0 is the current mode.

`pcValid` This 16-bit field applies to systems that support PCMCIA card services. The following values are defined:
bit 0 = when set, the value in the `pcAccessMode` field is valid
bit 1 = when set, the value in the `pcVcc` field is valid
bit 2 = when set, the value in the `pcVpp1` field is valid
bit 3 = when set, the value in the `pcVpp2` field is valid
bit 4 = when set, the value in the `pcStatus` field is valid
bit 5 = when set, the value in the `pcPin` field is valid
bit 6 = when set, the value in the `pcCopy` field is valid
bit 7 = when set, the value in the `pcConfigIndex` field is valid
bits 14-8 = reserved (set to 0)
bit 15 = reserved

`PWMultipleCount` This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`SectorsPerCylinder`
This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`Heads` This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`SectorsPerTrack` This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`socketNum` This field contains the socket number for the device used by the card services. A value of 0xFF indicates the device is not a card services client.

`socketType` This field specifies the type of socket. The values are defined as:
00 = unknown socket type

	01 = internal ATA bus 02 = media bay socket 03 = PCMCIA socket
`deviceType`	This field specifies the type of device. The possible values are defined as: 00 = unknown or no device present 01 = standard ATA device detected 02 = ATAPI device detected 03 = PCMCIA ATA device detected
`pcAccessMode`	This field specifies the current mode of the socket. This field is valid only when bit 0 of the `pcValid` filed is set. The mode values are: 0 = I/O mode 1 = memory mode
`pcVcc`	This field specifies the voltage on Vcc in tenths of a volt. The value in this field is only valid when bit 1 of the `pcValid` field is set.
`pcVpp1`	This field specifies the voltage of Vpp1 in tenths of a volt. The value in this field is valid only when bit 2 of the `pcValid` field is set.
`pcVpp2`	This field specifies the voltage of Vpp2 in tenths of a volt. The value in this field is valid only when bit 3 of the `pccValid` field is set.
`pcStatus`	This field specifies the current card register setting of a PCMCIA device. The value in this field is valid only when bit 4 of the `pccValid` field is set.
`pcPin`	This field specifies the current card pin register setting of a PCMCIA device. The value in this field is valid only when bit 5 of the `pccValid` field is set.
`pcCopy`	This field specifies the current setting of the card socket/copy register of a PCMCIA device. The value in this field is valid only when bit 6 of the `pccValid` field is set.
`pcConfigIndex`	This field specifies the current setting of the card option register of a PCMCIA device. The value in this field is valid only when bit 7 of the `pccValid` field is set.
`ataSingleDMASpeed`	This bit-significant field indicates which single word DMA mode, if any, is currently configured for use with DMA transfers. The DMA transfer mode may be modified with the ATA_SetDevConfig function.
`ataMultiDMASpeed`	This bit-significant field indicates which multiword DMA mode, if any, is currently configured for use with DMA transfers. The DMA transfer mode may be modified by the ATA_SetDevConfig function.
`ataPIOCycleTime`	This word field specifies the minimum cycle time in microseconds of mode 3 or higher PIO transfers. For more information about the information contained in this field, see the `ataPIOCycleTime` field in the `ATA_SetDevConfig` description beginning on page 139.

The actual cycle time may be higher than this value if the system hardware is unable to create the requested cycle time while maintaining signal timing for the PIO mode in use.

`ataMultiCycleTime` This word field specifies the minimum cycle time in microseconds of mode 1 or higher multiword DMA data transfers. For more information about the information contained in this field, see the `ataMultiCycleTime` field in the `ATA_SetDevConfig` function description beginning on page 139.
The actual cycle time may be higher than this value if the system hardware is unable to create the requested cycle time while maintaining signal timing for the multiword DMA mode in use.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_SetDevConfig

The `ATA_SetDevConfig` function allows an application to set the configuration parameters of a specified socket. Part of the device configuration includes setting up the parameters for I/O transfer speed. The section "Setting Data Transfer Timing" beginning on page 116 includes a discussion of how to use the ATA Manager to setup the software for data transfers, including DMA data transfers.

The manager function code for the `ATA_SetDevConfig` function is $8B.

The parameter block associated with this function is defined as follows:

```
typedef      struct
{
   ataPBHdr
   char      ConfigSetting;     /* → 32 bits of configuration */
                                /* information */
   ushort    ataPIOSpeedMode;   /* → Default PIO mode setting*/
   ushort    Reserved3;         /* Reserved for word alignment*/
   ulong     pcValid;           /* → PCMCIA unique */
   ulong     RWMultipleCount;   /* Reserved */
   ulong     SectorsPerCyl;     /* Reserved */
   ulong     Heads;             /* Reserved */
   ulong     SectorsPerTrack;   /* Reserved */
   ulong     Reserved4[2];      /* Reserved */
   ushort    pcAccessMode;      /* → Access mode of socket */
   ushort    pcVcc;             /* → device voltage */
   ushort    pcVpp1;            /* → Vpp 1 voltage */
```

```
   ushort     pcVpp2;            /* → Vpp 2 voltage */
   ushort     pcStatus;          /* → Status register setting */
   ushort     pcPin;             /* → Pin register setting */
   ushort     pcCopy;            /* → Copy register setting */
   ushort     pcConfigIndex;     /* → Option register setting */
   ushort     ataSingleDMASpeed; /* → Single word DMA */
                                 /* timing class */
   ushort     ataMultiDMASpeed;  /* → Multiple word DMA */
                                 /* timing class */
   ulong      ataPIOCycleTime;   /* → Cycle time for PIO mode */
   ulong      ataMultiCycleTime; /* → Cycle time for multiword */
                                 /* DMA mode */
   ulong      Reserved[7];       /* Reserved*/
} ATA_SetDevConfig;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 110.

`ConfigSetting` This 32-bit field contains various configuration information. The bits have the following definitions:

Bits 0-5: Reserved, set to 0

Bit 6: ATAPI packet DRQ handling setting
0 = Wait for an interrupt before sending the ATAPI command packet
1 = Wait for the assertion of DRQ in the status register before sending the ATAPI command packet. This is the default setting.

Bits 7-31: Reserved, set to 0

`ataPIOSpeedMode` This field contains the PIO mode to be used for commands and PIO data transfers. For parameter block version 3 or higher, the value is bit-significant, with the low-order bit signifying PIO Mode 0. Be sure to carefully note the difference in bit positions between this field and the corresponding "Advanced PIO Modes" field of the ATA-2 Identify Device information.

`pcValid` This 16-bit field applies to systems that support PCMCIA card services. The following values are defined:
bit 0 = when set, the value in the `pcAccessMode` field is valid
bit 1 = when set, the value in the `pcVcc` field is valid
bit 2 = when set, the value in the `pcVpp1` field is valid
bit 3 = when set, the value in the `pcVpp2` field is valid
bit 4 = when set, the value in the `pcStatus` field is valid
bit 5 = when set, the value in the `pcPin` field is valid
bit 6 = when set, the value in the `pcCopy` field is valid
bit 7 = when set, the value in the `pcConfigIndex` field is valid
bits 14-8 = reserved (set to 0)
bit 15 = reserved

`PWMultipleCount` This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`SectorsPerCylinder`
: This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`Heads`
: This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`SectorsPerTrack`
: This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`Reserved4[2]`
: This field is reserved.

`pcAccessMode`
: This field specifies the mode of the socket. This field is valid only when bit 0 of the `pcValid` filed is set. The mode values are:
0 = I/O mode
1 = memory mode

`pcVcc`
: This field specifies the new voltage setting for Vcc in tenths of a volt. The value in this field is only valid when bit 1 of the `pcValid` field is set.

`pcVpp1`
: This field specifies the new voltage setting for Vpp1 in tenths of a volt. The value in this field is valid only when bit 2 of the `pcValid` field is set.

`pcVpp2`
: This field specifies the new voltage setting for Vpp2 in tenths of a volt. The value in this field is valid only when bit 3 of the `pccValid` field is set.

`pcStatus`
: This field specifies the new card register setting for a PCMCIA device. The value in this field is valid only when bit 4 of the `pccValid` field is set.

`pcPin`
: This field specifies the new card pin register setting for a PCMCIA device. The value in this field is valid only when bit 5 of the `pccValid` field is set.

`pcCopy`
: This field specifies the new card socket/copy register setting for a PCMCIA device. The value in this field is valid only when bit 6 of the `pccValid` field is set.

`pcConfigIndex`
: This field specifies the new card option register setting for a PCMCIA device. The value in this field is valid only when bit 7 of the `pccValid` field is set.

`ataSingleDMASpeed`
: This bit-significant field specifies the singleword DMA mode for DMA data transfers. It corresponds to the high-order byte of word 62 of the Identify Device data described in the ATA-2 specification. If word 62 is not supported by the device, then it reflects word 52 converted to bit significance. The ATA software supports word 62 modes 0 through 2, as defined in the ATA-2 specification. If the specified timing mode is higher than the values supported by the software, then the highest possible mode is selected for transfers.
The ATA Manager selects the transfer rate that satisfies the requirements of the mode and the system DMA hardware. The rate and mode are used on subsequent DMA transfers until changed by another `ATA_SetDevConfig` function call. The default singleword DMA mode is mode 0.
For additional information related to setting the I/O data

transfer speed, see "Setting Data Transfer Timing" beginning on page 116.

`ataMultiDMASpeed` This bit-significant field specifies the multiword DMA cycle mode for DMA data transfers. It corresponds to the high-order byte of word 63 of the Identify Device data described in the ATA-2 specification. If word 63 is not supported by the device, then this value should be zero which indicates that multiword DMA should not be attempted. The ATA software supports word 63 modes 0 through 2, as defined in the ATA-2 specification. If the specified timing mode is higher than the values supported by the software, then the highest supported mode is selected for DMA transfers. This field is used in conjunction with the value set in the `ataMultiCycleTime` field. The ATA Manager selects the transfer rate that satisfies the requirements of the mode specified in `ataMultiCycleTime` field and the system DMA hardware. The rate and mode are used on subsequent DMA transfers until changed by another `ATA_SetDevConfig` function call. The default setting for DMA mode is singleword DMA mode 0.
For additional information related to setting the I/O data transfer speed, see "Setting Data Transfer Timing" beginning on page 116.

`ataPIOCycleTime` This word field is used in conjunction with the `ataIOSpeed` field of the `ataPBHdr` structure to specify the cycle time for command and PIO data transfers. The value in this field represents word 68 of the Identify Device information, as defined in the ATA-2 specification. If this value is not zero, the ATA Manager selects the closest approximation of the cycle time supported by the system hardware which does not exceed the value and still meets the timing requirements of the selected mode.

If this value is zero, the ATA Manager uses the minimum cycle times from the ATA-2 specification for the mode. The resulting cycle timing represents the maximum timing for PIO mode 2, because that is the highest mode supported without reporting word 68 of the Identify Device information.

`ataMultiCycleTime` This word field is used in conjunction with the `ataMultiDMASpeed` field to specify the cycle time for multiword DMA data transfers. The value represents the same value reported in word 65 or word 66 of the Identify Device information, as specified in the ATA-2 specification. If the value specified in this field is not zero, the ATA Manager selects the closest cycle time supported by the system hardware that does not exceed the value and meets the other timing requirements of the mode specified in the `ataMultiDMASpeed` field.

If the value is zero, the ATA Manager uses the minimum cycle times specified in the ATA-2 specification for the selected mode. The resulting cycle timing represents the minimum timing for

multiword DMA mode 0, because that is the highest mode supported without reporting word 65 or word 66 of the Identify Device information.

RESULT CODES

See Table 7-7 on page 144 for possible result codes returned by the ATA Manager.

ATA_GetLocationIcon

The `ATA_GetLocationIcon` function returns a pointer to the structure defining the location icon data for the selected device. The structure contains the icon data and an icon string for the device.

The manager function code for the `ATA_SetDevConfig` function is $8C.

The parameter block associated with this function is defined as follows:

```
typedef      struct
{
   ataPBHdr
   ulong     iconData;   /* Pointer to icon data and */
                         /* the size of the data */
} ATA_GetLocationIcon;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 110.

`iconData` This field contains two fields, a pointer to a structure that contains the icon data, and the size in bytes of the icon data. The structure that contains the actual icon data is defined as:

```
struct DriverLocationIcon
{
ushort  locationIcon[256];
char    locationString;
} DriverLocationIcon;
```

The `locationIcon` field is the device icon data. The `locationString` field is string in C string format.

RESULT CODES

See Table 7-7 for possible result codes returned by the ATA Manager.

Result Code Summary

A summary of the ATA result codes is provided in Table 7-7. ATA Parameter block versions 2 and greater have a different numbering scheme from that of version 1. The error code number values for parameter block version 1 are contained in parenthesis.

Table 7-7 ATA Manager result codes

Error code		Name	Description
0	0	`noErr`	Successful completion, no error detected
-50		`paramErr`	Invalid parameter specified
-56		`nsDrvErr`	No such drive installed
-9396	(-1780)	`AT_AbortErr`	Command aborted bit set in error register
-9397	(-1781)	`AT_RecalErr`	Recalibrate failure detected by device
-9398	(-1782)	`AT_WrFltErr`	Write fault bit set in status register
-9399	(-1783)	`AT_SeekErr`	Seek complete bit not set on completion
-9400	(-1784)	`AT_UncDataErr`	Uncorrected data bit set in error register
-9401	(-1785)	`AT_CorDataErr`	Data corrected bit set in status register
-9402	(-1786)	`AT_BadBlkErr`	Bad block bit set in error register
-9403	(-1787)	`AT_DMarkErr`	Data mark not found bit set in error register
-9404	(-1788)	`AT_IDNFErr`	ID not found bit set in error register
-9405	(-1791)	`AT_NRdyErr`	Drive ready condition not detected
-9345	(-1817)	`AT_BusyErr`	Selected device bust (BUSY bit set)
-9376		`DRVRCantAllocate`	Global memory allocation error
-9375		`NoATAMgr`	No ATA Manager installed in the system (`MgrInquiry` failure)
-9374		`ATAInitFail`	ATA Manager initialization failure
-9373		`ATABufFail`	Device buffer test failed
-9372		`ATADevUnSupported`	Device type not supported

Table 7-7 ATA Manager result codes (continued)

Error code		Name	Description
-9371		ATAEjectDrvErr	Could not eject the drive
-9360	(-1802)	ATAMgrNotInitialized	ATA Manager not initialized
-9359	(-1803)	ATAPBInvalid	Invalid device base address detected (=0)
-9358	(-1804)	ATAFuncNotSupported	An unknown manager function code specified
-9357	(-1805)	ATABusy	Selected device is busy; device isn't ready to go to next phase yet
-9356	(-1806)	ATATransTimeOut	Timeout: Transaction timeout detected
-9355	(-1807)	ATAReqInProg	I/O channel in use—cannot proceed
-9354	(-1808)	ATAUnknownState	Device in unknown state
-9353	(-1809)	ATAQLocked	I/O queue locked—cannot proceed
-9352	(-1810)	ATAReqAborted	The request was aborted
-9351	(-1811)	ATAUnableToAbort	Request to abort couldn't be honored
-9350	(-1812)	ATAAbortedDueToRst	The I/O queue entry aborted due to a bus reset
-9349	(-1813)	ATAPIPhaseErr	Unexpected phase detected
-9348	(-1814)	ATAPIExCntErr	Warning: OVerrun/underrun condition detected (data valid)
-9347	(-1815)	ATANoClientErr	No client present to handle event
-9346	(-1816)	ATAInternalErr	Card services returned an error
-9345	(-1817)	ATABusErr	Bus error detected on I/O
-9344	(-1818)	AT_NoAddrErr	Invalid taskfile base address
-9343	(-1799)	DriverLocked	Current driver must be removed before adding another
-9342	(-1800)	CantHandleEvent	Particular event could not be handled
-9341		ATAMgrMemoryErr	Manager memory allocation error
-9340		ATASDFailErr	Shutdown failure
-9339		ATAXferParamErr	I/O transfer parameters inconsistent
-9338		ATAXferModeErr	I/O transfer mode not supported
-9337		ATAMgrConsistencyErr	Manager detected internal inconsistency
-9328		ATAInvalidDrvNum	Invalid driver number from event

Table 7-7 ATA Manager result codes (continued)

Error code	Name	Description
-9327	`ATAMemoryErr`	Memory allocation error
-9226	`ATANoDDMErr`	No DDM found on media
-9325	`ATANoDriverErr`	No driver found on the media

Index

E

F

G

H

I, J

K

L

M

N

O

P

Q

R

S

T, 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 were created on an Apple LaserWriter IINTX printer. Final pages were created on the Varityper VT600 imagesetter. Line art was created using Adobe™ Illustrator. 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
Steve Schwander

COPY EDITOR
John Hammett

ILLUSTRATOR
Sandee Karr

PRODUCTION EDITOR
Alex Solinski

Special thanks to Paul Freeburn, Steve Parsons, Paul Thompson, Stan Robbins, and Rich Schnell