Developer Note

PowerBook 520 and 520c and PowerBook 540 and 540c Computers

Developer Note

May 1994
Developer Press

Apple Computer, Inc.
20525 Mariani Avenue
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 Developer Note

This developer note describes the Macintosh PowerBook 520 and 540 computers, emphasizing the features that are new or different from those of other Macintosh PowerBook computers. 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 "Supplementary Documents."

Contents of This Note

The information in this note is arranged in six chapters:

- Chapter 1, "Introduction," introduces the PowerBook 520 and 540 computers and describes their new features.
- Chapter 2, "Architecture," describes the internal logic of the PowerBook 520 and 540 computers, including the main ICs that appear in the block diagram.
- Chapter 3, "I/O Features," describes the input/output features, including both the internal I/O devices and the external I/O ports.
- Chapter 4, "Expansion Modules," describes the expansion features of interest to developers. It includes development guides for the RAM expansion card, the PDS card, and the communications cards.
- Chapter 5, "Software," describes the new features of the ROM and system software, with the emphasis on software that is specific to this computer.
- Chapter 6, "Power Manager Interface," describes the application program interface for the Power Manager software.

Supplementary Documents

To supplement the information in this developer note, developers should have copies of the appropriate Motorola reference books for the MC68040 microprocessor. Software developers should have a copy of Motorola's *MC68040 Programmer's Reference Manual.* Hardware developers should have copies of Motorola's *MC68030 User's Manual, MC68040 User's Manual,* and *MC68040 Designer's Handbook.*

Developers should also have copies of the appropriate Apple reference books, including *Inside Macintosh; Guide to the Macintosh Family Hardware,* second edition; and *Designing Cards and Drivers for the Macintosh Family,* third edition. These Apple books are available in technical bookstores and through APDA.

For information about earlier PowerBook models, developers should also have copies of the *Macintosh Classic II, Macintosh PowerBook Family, and Macintosh Quadra Family Developer Notes;* and *Macintosh Developer Notes,* numbers 1 through 5, available on the Developer CD Series, and through APDA.

APDA is Apple's worldwide source for over three hundred development tools, technical resources, training products, and information for anyone interested in developing applications on Apple platforms. Customers receive the quarterly *APDA Tools Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. Ordering is easy; there are no membership fees, and application forms are not required for most of the products. APDA offers convenient payment and shipping options, including site licensing.

To order products or to request a complimentary copy of the *APDA Tools Catalog*, contact

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

Telephone	800-282-2732 (United States) 800-637-0029 (Canada) 716-871-6555 (International)
Fax	716-871-6511
AppleLink	APDA
America Online	APDAorder
CompuServe	76666,2405
Internet	APDA@applelink.apple.com

Conventions and Abbreviations

This developer note uses the following typographical conventions and abbreviations.

Sidebar

A sidebar is used for information that is not part of the main discussion. A sidebar may contain information about a related subject or technical details that are not required reading.

Typographical Conventions

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 of interest but is not essential for an understanding of the text. ◆

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

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

Standard Abbreviations

Standard units of measure used in this note include

A	amperes	MHz	megahertz
dB	decibels	mm	millimeters
GB	gigabytes	ms	milliseconds
Hz	hertz	mV	millivolts
K	1024	µF	microfarads
KB	kilobytes	ns	nanoseconds
kbps	kilobits per second	Ω	ohms
kHz	kilohertz	pF	picofarads
kΩ	kilohms	V	volts
M	1,048,576	VAC	volts alternating current
mA	milliamperes	VDC	volts direct current
MB	megabytes	W	watts

Other abbreviations used in this note include

$*n*	hexadecimal value *n*
AC	alternating current
ADB	Apple Desktop Bus
API	application program interface
ASIC	application-specific integrated circuit
AUI	auxiliary unit interface
BCD	binary coded decimal
CAS	column address strobe (a memory control signal)
CCFL	cold cathode fluorescent lamp
CD	compact disc
CLUT	color look-up table
CMOS	complementary metal oxide semiconductor
CPU	central processing unit
CSC	color screen controller
DAA	data access adapter (a telephone line interface)
DAC	digital-to-analog converter
DC	direct current
DOS	disk operating system
DRAM	dynamic RAM
DSP	digital signal processor
FIFO	first in, first out
FPU	floating-point unit
FSTN	film supertwist nematic (a type of LCD)
IC	integrated circuit
I/O	input/output
LCD	liquid crystal display
LS TTL	low-power Schottky TTL (a standard type of device)
MMU	memory management unit
NiCad	nickel cadmium
NiMH	nickel metal hydride
PCMCIA	Personal Computer Memory Card International Association
PDS	processor direct slot
PROM	programmable read-only memory
PWM	pulse width modulation
RAM	random-access memory
RAMDAC	random-access memory, digital to analog converter
RAS	row address strobe

RGB	red-green-blue (a type of color video system)
RISC	reduced instruction set computer
RMS	root-mean-square
ROM	read-only memory
SCC	Serial Communications Controller
SCSI	Small Computer System Interface
SNR	signal-to-noise ratio
SVGA	super video graphics adapter
TFT	thin-film transistor (a type of LCD)
TTL	transistor-transistor logic (a standard type of device)
VCC	positive supply voltage (voltage for collectors)
VGA	video graphics adapter
VRAM	video RAM

CHAPTER 1

Introduction

The Macintosh PowerBook 520 and PowerBook 540 computers are the first of a new generation of all-in-one notebook computers featuring the powerful Motorola MC68040 microprocessor, color displays, integrated communications architecture, and other advanced features. Inside the computer's contoured case are spaces for two rechargeable batteries, a new high-speed modem, and a processor-direct slot for expansion cards.

The modular design of these new computers offers new opportunities for developers to expand beyond traditional add-on products by integrating new devices within the computer's case. This modular design also makes it easier to upgrade the system with newer, faster, low-power microprocessors as they become available.

Features

Here is a summary of the major features of the PowerBook 520 and PowerBook 540 computers. Each feature is described more fully later in this note.

- Microprocessor: Motorola MC68LC040 running at 50/25 MHz or 66/33 MHz.

Note
The MC68LC040 uses two processor clocks: one for the system bus and another, at twice the speed, for the internal circuits. Thus, an MC68LC040 with a system bus clock of 33 MHz runs its internal processor at 66 MHz. ◆

- Random-access memory (RAM): built-in memory consists of 4 MB of low-power, self-refreshing dynamic RAM (DRAM).
- RAM expansion: support for RAM expansion cards of up to 32 MB, for a total of 36 MB of RAM.
- Read-only memory (ROM): 2 MB.
- Flat panel display: built-in 640-by-480-pixel LCD with 8-bit capability for 256 colors or shades of gray. The PowerBook 540 and 540c have active matrix LCDs; the PowerBook 520 and 520c have advanced passive matrix LCDs. Both types of displays are backlit by a cold cathode fluorescent lamp (CCFL).
- Video output: 8-bit color video output circuitry supports 256-color display on all Apple monitors up to 16 inches in size, 16 shades of gray on the Apple Macintosh Portrait Display, and 16-color display on Apple multiscan monitors; with appropriate adapter cables, also supports VGA displays and an 800-by-600-pixel SVGA mode.
- Floppy disk: one internal 1.4 MB Apple SuperDrive.
- Hard disk: one internal 2.5-inch SCSI hard disk drive with disk capacity of 240 or 320 MB. See "Configurations" on page 5.

- SCSI disk mode: operates in conjunction with an HDI-30 SCSI Disk Adapter cable to allow users to read and store data on the PowerBook computer's internal hard disk from another Macintosh computer.
- Networking: built-in Ethernet and LocalTalk network interfaces.
- Expansion: the PowerBook Expansion Bay, a 90-pin processor-direct slot (PDS) inside the left battery compartment.
- Sound: built-in microphone and stereo speakers; microphone jack and headphone jack.
- Modem: internal connector for an optional modem card. The modem card available from Apple is a 14.4kbps fax/data modem with a worldwide DAA interface.
- I/O (input/output): standard Macintosh inputs and outputs, including Ethernet and external video out. The I/O ports are an HDI-30 connector for external SCSI devices, a 14-pin Apple Ethernet AUI port, a 4-pin mini-DIN Apple Desktop Bus (ADB) port, an 8-pin mini-DIN serial port, stereo audio input and output jacks, and a video output connector with adapter for attaching standard Apple video cables.
- Keyboard: full-size keyboard with function keys and power on/off control.
- Trackpad, an integrated flat pad that replaces the trackball used in previous PowerBook computers.
- Batteries: one or two PowerBook Intelligent Batteries. The batteries are 9.6-V, 1.9-ampere-hour NiMH rechargeable batteries with built-in processors that communicate with the computer's Power Manager to measure battery charge and maximize battery life.
- Power supply: an external wall-mounted recharger/power adapter, included with the computer. The power adapter accepts any worldwide standard voltage from 100–240 VAC at 50–60 Hz. The power adapter can charge two batteries in two hours while the computer is shut down or in sleep mode, or in four hours while it is operating.
- Security connector: a connector on the back panel allows users to attach a security device.
- Weight: 6.3 pounds with one battery installed, 7 pounds with two batteries.
- Size: 11.3 inches wide, 9.5 inches deep, and 2.3 inches high.

Appearance

The PowerBook 520 and 540 models have a new, streamlined case. Figure 1-1 shows a front view of the PowerBook 540 and Figure 1-2 shows a back view.

Figure 1-1 Front view of the PowerBook 540 computer

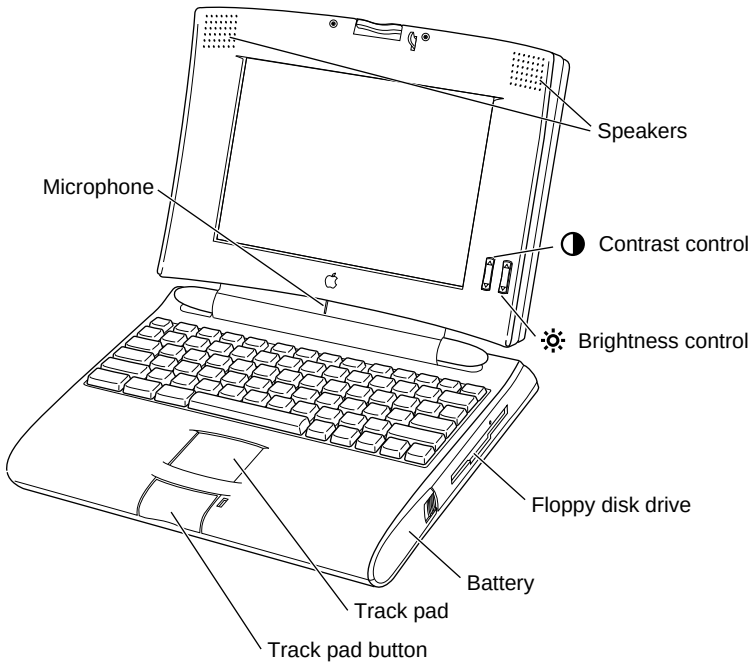
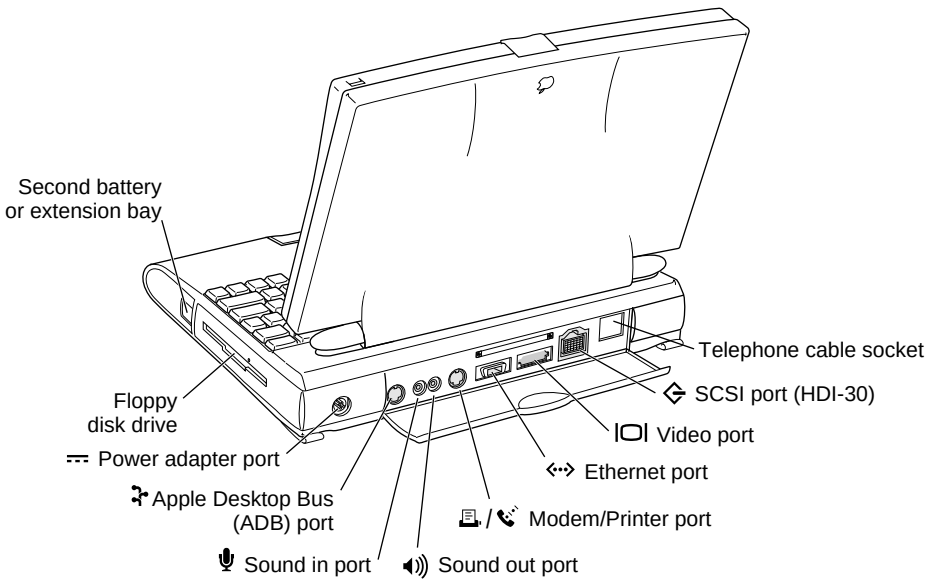


Figure 1-2 Back view of the PowerBook 540 computer



Configurations

The Macintosh PowerBook 520 and 540 computers are available in a total of six configurations, as shown in Table 1-1.

Table 1-1 Models and configurations

Model	Processor speed	Display type	Sizes of RAM and hard disk	Modem included	Number of batteries
PowerBook 540c	66/33 MHz	Active matrix color	12 MB/320 MB	Yes	2
PowerBook 540c	66/33 MHz	Active matrix color	4 MB/320 MB	No	2
PowerBook 540	66/33 MHz	Active matrix grayscale	12 MB/240 MB	Yes	2
PowerBook 540	66/33 MHz	Active matrix grayscale	4 MB/240 MB	No	2
PowerBook 520c	50/25 MHz	DualScan color	4 MB/160 MB	No	1
PowerBook 520	50/25 MHz	Supertwist grayscale	4 MB/160 MB	No	1

Peripheral Devices

In addition to the devices that are included with the computer, several peripheral devices are available separately:

- The PowerBook 8 MB Memory Expansion Kit expands the RAM in the PowerBook 520 and 540 computers to 12 MB.
- The PowerBook Intelligent Battery is available separately as an additional or replacement battery.
- The PowerBook AC Adapter, which comes with the computer, is also available separately. This AC adapter is different from the adapter for the PowerBook 140, 160, and 180 computers. Unlike that earlier model, the new adapter can recharge two internal batteries in just four hours while the computer is running or two hours while the computer is shut down or in sleep mode.
- The PowerBook PCMCIA Expansion Module fits into the PDS connector in the second battery bay and accepts either one Type III PCMCIA card or two stacked Type II cards.

- The PowerBook Express Modem II, available in the U.S. and Canada, is a 14.4kbps fax/data modem; it supports the CCITT V.32bis and V.42bis standards as well as industry standard MNP5.

  The modem comes with an internal DAA module that is software reconfigurable for use anywhere in the world. In the country where it was purchased, the DAA accepts the standard telephone connector. For travel outside the country of purchase, the user purchases a short cable adapter called the country key for the new country. When in that country, the user plugs the adapter into the DAA module and attaches the local telephone connector.

Compatibility Issues

The PowerBook 520 and 540 computers incorporate many significant changes from earlier PowerBook designs. This section highlights key areas you should investigate in order to ensure that your hardware and software work properly with the new PowerBook models. These topics are covered in more detail in subsequent sections.

RAM Expansion Cards

The RAM expansion card used in the PowerBook 520 and 540 computers has a new design. RAM expansion cards designed for earlier PowerBook models will not work in the new PowerBook models. See the section “RAM Expansion Card” beginning on page 34 for more information.

Number of Colors

The controller circuitry for the flat panel display includes a 256-entry color look-up table (CLUT) and is compatible with software that uses QuickDraw and the Palette Manager. The controller supports a palette of 32,768 colors. However, due to the nature of color LCD technology, some colors are dithered or exhibit noticeable flicker. Apple has developed a new gamma table for the color displays that minimizes flicker and optimizes the available colors. For the active matrix color display, the effective range of the CLUT is about 24,000 colors. For the DualScan color display, the effective range of the CLUT is about 4000 colors.

See the section “Flat Panel Displays” beginning on page 23 for more information about the internal display hardware and LCD screen.

Video Mirror Mode

The user can select video mirror mode, in which the external monitor mirrors (duplicates) the flat panel display. Applications that write directly to the display buffer may not be compatible with video mirror mode unless they take precautions to ensure that they do not write outside the active portion of the display. That is not a problem for applications that use QuickDraw and never write directly to the display buffer.

See the section “Video Mirroring” on page 25 for more information about video modes.

MacsBug Version 6.2.2

MacsBug version 6.2.2 does not work with the PowerBook 520 and 540 computers because it assumes that all MC68040 microprocessors have FPUs. This problem is corrected in newer versions of MacsBug. ◆

Sound Sample Rates

The PowerBook 520 and 540 computers provide sound sample rates of 22.05 kHz, 44.1 kHz, and 48 kHz. The 22.05 kHz sample rate is slower than the sample rate used in some older Macintosh models: 22.254 kHz, which was derived from the 16 MHz system clock. The 22.05 kHz rate was chosen for compatibility with the 44.1 kHz audio CD sample rate.

For sound samples made at the 22.254kHz rate, playback at the 22.05 kHz rate is about 1 percent low in pitch. Furthermore, programs that bypass the Sound Manager and write to the sound FIFOs at the older rate now write too many samples to the FIFOs, causing some samples to be dropped. The result is a degradation in sound quality for those programs. Programs that use the Sound Manager to generate sounds are not affected by the change.

Power Manager Control Panels

Developers have written software that provides expanded Power Manager control for other PowerBook models. That software will not work in the PowerBook 520 and 540 computers.

Until now, third-party software for the Power Manager has worked by reading and writing directly to the Power Manager's data structures, so it has had to be updated whenever Apple brings out a new model with changes in its Power Manager software. Starting with the PowerBook 520 and 540 computers, the system software includes interface routines for program access to the Power Manager functions, so it is no longer necessary for applications to deal directly with the Power Manager's data structures. For more information, see Chapter 6, "Power Manager Interface.".

Developers should not assume that the Power Manager's data structures are the same on all PowerBook models. In particular, developers should beware of the following assumptions regarding different PowerBook models:

- assuming that timeout values such as the hard disk spindown time reside at the same locations in parameter RAM
- assuming that the power cycling process works the same way or uses the same parameters
- assuming that direct commands to the Power Manager microcontroller are supported on all models

CHAPTER 2

Architecture

The architecture of the PowerBook 520 and 540 computers is partitioned into two subsystems: the processor/memory subsystem and the input/output subsystem. The processor/memory subsystem operates at 25 MHz or 33 MHz on the MC68040 bus. The input/output subsystem operates at 16 MHz on the I/O bus, an MC68030-compatible bus. An Apple custom IC called the Pratt IC acts as the bridge between the two buses, translating MC68040 bus cycles into single or multiple MC68030 bus cycles, as needed.

The block diagram shown in Figure 2-1 shows the two subsystems along with other modules that are attached to them.

Processor/Memory Subsystem

The processor/memory subsystem includes the MC68040 microprocessor, main RAM, and ROM. As Figure 2-1 shows, the processor/memory subsystem is a separate module; it occupies a secondary logic board mounted above the main logic board. An optional RAM expansion card attaches to the secondary logic board and becomes part of the processor/memory subsystem.

The modular design of the PowerBook 520 and 540 computers makes it possible to upgrade the system by plugging in a whole new processor/memory subsystem. For example, it will be possible to upgrade to a PowerPC RISC microprocessor when a low-power version becomes available.

Microprocessor

The microprocessor used in the PowerBook 520 and 540 computers is the MC68LC040. The MC68LC040 does not contain an FPU (floating-point unit). The MC68LC040 does include a built-in MMU (memory management unit).

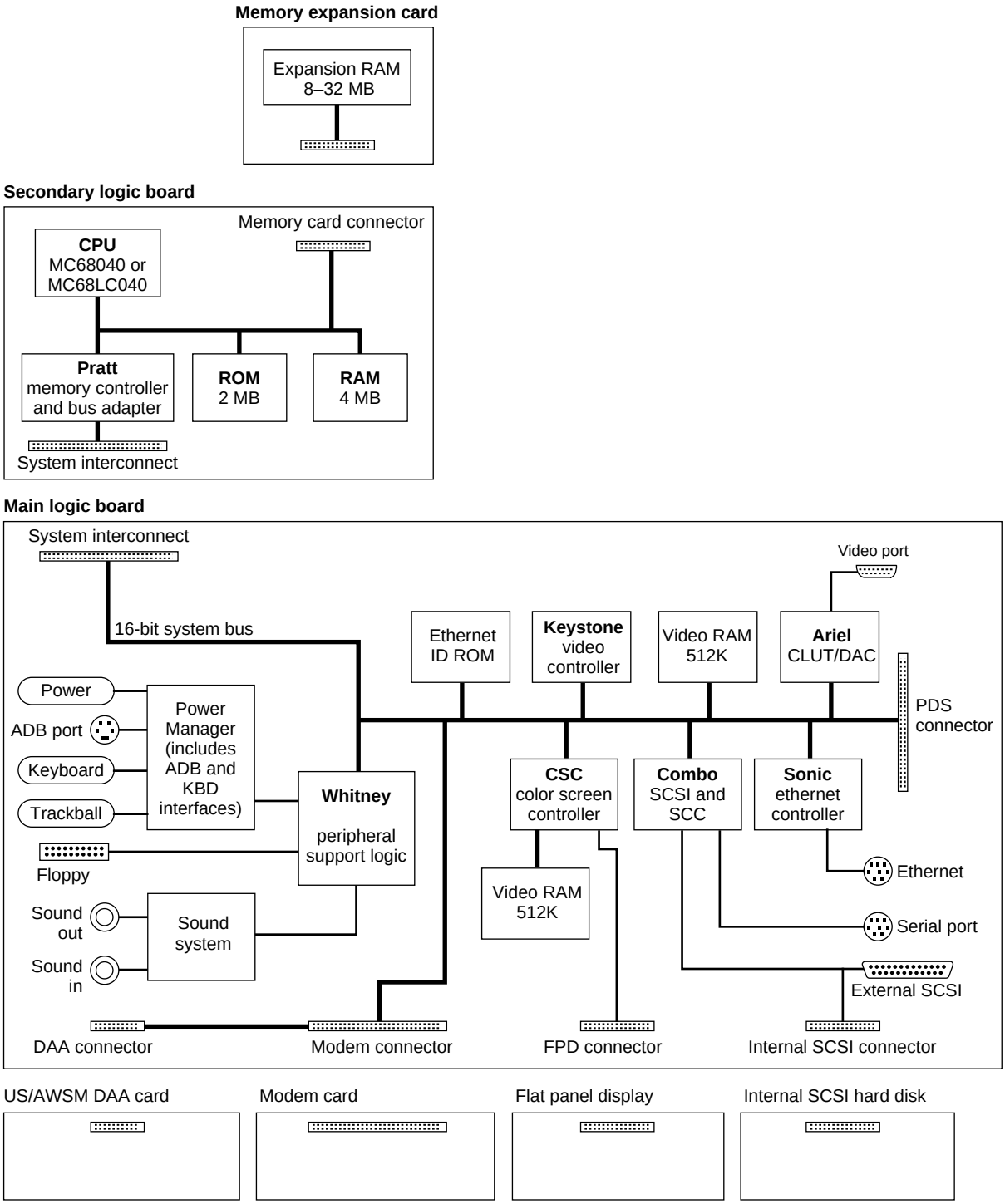
The MC68LC040 microprocessor runs at an internal clock rate that is double its external clock rate. With an external clock rate of 25 MHz, the processor's internal clock runs at 50 MHz; with an external rate of 33 MHz, the internal clock rate is 66 MHz. Table 1-1 on page 5 shows the microprocessor type and speed for each of the new models.

RAM

The built-in RAM consists of 4 MB of dynamic RAM (DRAM), supplied by eight 1 M by 4-bit ICs on the secondary logic board. The RAM ICs are low-power, self-refreshing type with an access time of 80 ns.

The RAM array is located in the system memory map between addresses $0000 0000 and $02FF FFFF, except following a system reset or sleep cycle, when it is overlaid by system ROM. As in other Macintosh models, the overlay is removed following the first program access to normal ROM space, making the RAM space accessible.

Figure 2-1 Block diagram of the PowerBook 520 and 540 computers



An optional RAM expansion card plugs into a 70-pin connector on the secondary logic board. With the RAM expansion card installed, the processor/memory subsystem supports up to 36 MB of RAM. The RAM card for the PowerBook 520 and 540 computers is not compatible with the RAM card used in earlier PowerBook models. See the section "RAM Expansion Card" beginning on page 34 for details.

ROM

The ROM in the PowerBook 520 and 540 computers is implemented as a 1M by 32-bit array consisting physically of two 1 M by 16-bit ROM ICs with an access time of 150 ns. Although these ICs provide 4 MB of storage, the actual ROM code occupies only the first 2 MB, located in the system memory map between addresses $4000 0000 and $401F FFFF. The ROM data path is 32 bits wide and addressable only as longwords. See Chapter 5, "Software," for a description of the features of this new ROM.

Like all Macintosh computers, the PowerBook 520 and 540 computers implement an overlay function at startup that maps the system's ROM address space ($4000 0000–$401F FFFF) to the address space starting at $0000 0000. This overlay allows the MC68040 processor to address a standard default set of exception vectors and trap addresses as well as a starting address at which to begin executing code. As soon as the ROM code makes an access to an address in the normal ROM address space, the hardware turns off the ROM overlay, making RAM accessible starting at address $0000 0000.

Pratt Memory Controller IC

The Pratt IC is a new Apple custom IC that provides RAM and ROM memory control and also acts as the bridge between the MC68040 bus on the secondary logic board and the MC68030 bus on the main logic board. The Pratt IC resides on the secondary logic board and transparently translates MC68040 bus cycles into single or multiple MC68030 dynamically sized bus cycles. Because the Pratt IC seamlessly integrates the two buses, the microprocessor and other bus masters operate as though they were on the same bus.

The Pratt IC supports read, write, and page mode cycles to the RAM. Pratt generates a 2048-byte CAS-before-RAS refresh cycle every 128 ms.

Input/Output Subsystem

The input/output subsystem includes the Whitney IC, the I/O controllers ICs, and the Power Manager IC.

Whitney Peripheral Support IC

The Whitney IC is a new Apple custom IC that provides the interface between the system bus and the I/O bus that supports peripheral devices in the Macintosh PowerBook 520 and 540 computers. The Whitney IC incorporates the following circuitry:

- VIA1 cell
- SWIM II cell
- CPU ID register

The Whitney IC also performs the following functions:

- bus error timing for I/O bus
- bus arbitration for I/O bus
- interrupt prioritization
- VIA2 functions
- sound data buffering
- clock generation
- power control signals

The Whitney IC contains the interface circuitry for the following peripheral ICs:

- Combo, which is a combination of SCC and SCSI ICs
- Singer, the sound codec IC
- the Ethernet ID PROM

The Whitney IC provides the device select signals for the following ICs:

- the Ethernet interface
- the flat panel display controller
- the external video controller

The Whitney IC also provides the power off and reset signals to the peripheral device ICs.

Power Manager IC

The Power Manager IC is a 68HC05 microprocessor that operates with its own RAM and ROM. The Power Manager IC has the following functions:

- controlling sleep, shutdown, and on/off modes
- controlling power to the other ICs
- controlling clock signals to the other ICs
- supporting the ADB

- scanning the keyboard
- controlling display brightness
- monitoring battery charge level
- controlling battery charging

Display Controller IC

The flat panel display is controlled by a type 65220 CSC (color support chip) IC manufactured by Chips and Technologies, Inc. The display controller provides the data and control interface to the LCD panel. The CSC IC contains a 256-entry CLUT, RAMDAC, display buffer controller, and flat panel control circuitry. For more information, see "Flat Panel Display Circuitry" on page 24.

Combo IC

The Combo custom IC combines the functions of the SCC (85C30 Serial Communications Controller) and the SCSI controller (53C80) ICs. The SCC portion of the Combo IC supports the serial I/O port. The SCSI controller portion of the Combo IC supports the internal and external SCSI devices.

Singer IC

The Singer custom IC is a 16-bit digital sound codec. It conforms to the IT&T *ASCO 2300 Audio-Stereo Code Specification.* The Whitney IC maintains sound I/O buffers in main memory for sound samples being send in or out through the Singer IC. For information about the operation of the Singer IC, see the section "Sound System" on page 31.

Keystone Video Controller IC

The Keystone custom IC contains the timing and control circuits for the external video. The Keystone IC has internal registers that the video driver uses to set the horizontal and vertical timing parameters. The Keystone IC also generates the video refresh addresses that are sent to the VRAM.

Ariel Video Output IC

The Ariel custom IC contains the video CLUT (color look-up table) and DAC. The Ariel is pin and software compatible with the AC843 but does not support 24 bits per pixel.

CHAPTER 3

I/O Features

This chapter describes both the built-in I/O devices and the interfaces for external I/O devices. Like the earlier chapters, it emphasizes the similarities and differences between the PowerBook 520 and 540 computers and other PowerBook models.

This chapter describes the following built-in devices and I/O ports:

- internal hard disk drive
- internal floppy disk drive
- built-in trackpad
- built-in keyboard
- built-in flat panel display
- external video port
- serial port
- SCSI port
- Ethernet port
- Apple Desktop Bus (ADB) port
- sound system

Internal Hard Disk Drive

The PowerBook 520 and 540 computers have an internal 2.5-inch hard disk drive connected to an internal SCSI connector. The following information is provided as a general guideline and is based on the specifications for Apple's 2.5-inch, 240 MB hard disk drive.

Hard Disk Power Budget

The hard disk drive operates on 5 VDC ± 5 percent. The maximum ripple voltage the drive must tolerate is 100 mV peak to peak at any frequency from DC to 10 MHz.

Table 3-1 shows the maximum and average current drain and power consumption for various operating modes of the hard disk drive. These limits include 1 kΩ pull-up terminator resistors on all signal lines. All mean specifications are RMS (root-mean-square) values.

IMPORTANT
The hard disk drive must not exceed the maximum power specifications given in Table 3-1. ▲

Table 3-1 Hard disk power budget

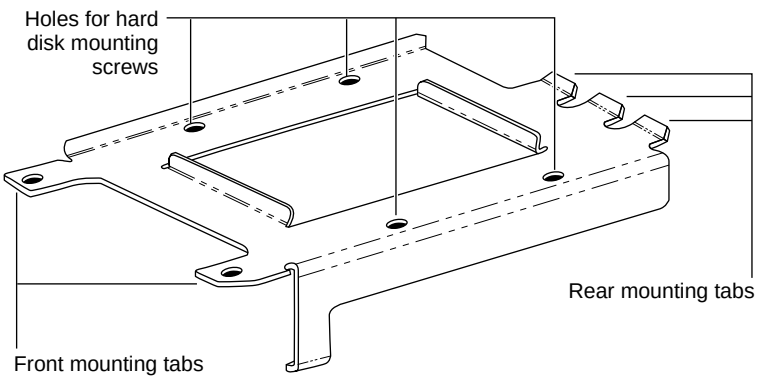
	Current (amps)		Power (watts)	
Mode	Average	Maximum	Average	Maximum
Startup*	n.a.	1.300	n.a.	6.500
Operation	0.500	0.600	2.500	3.000
Idle	0.300	0.380	1.500	1.750
Standby	0.200	0.250	1.000	1.250
Shutdown	0.050	0.075	0.250	0.380

* Maximum values between power on and drive ready (not more than 5 seconds).

Hard Disk Carrier Bracket

The hard disk drive is mounted upside down with a carrier bracket that holds the drive in a well in the computer's case. Figure 3-1 shows the carrier bracket; the drive is attached beneath the bracket by screws in the four holes in the bracket. When the drive is installed, the three tabs at the rear of the bracket engage cutouts in the case and the two tabs at the front of the bracket are held in place by screws.

Figure 3-1 Hard disk carrier bracket

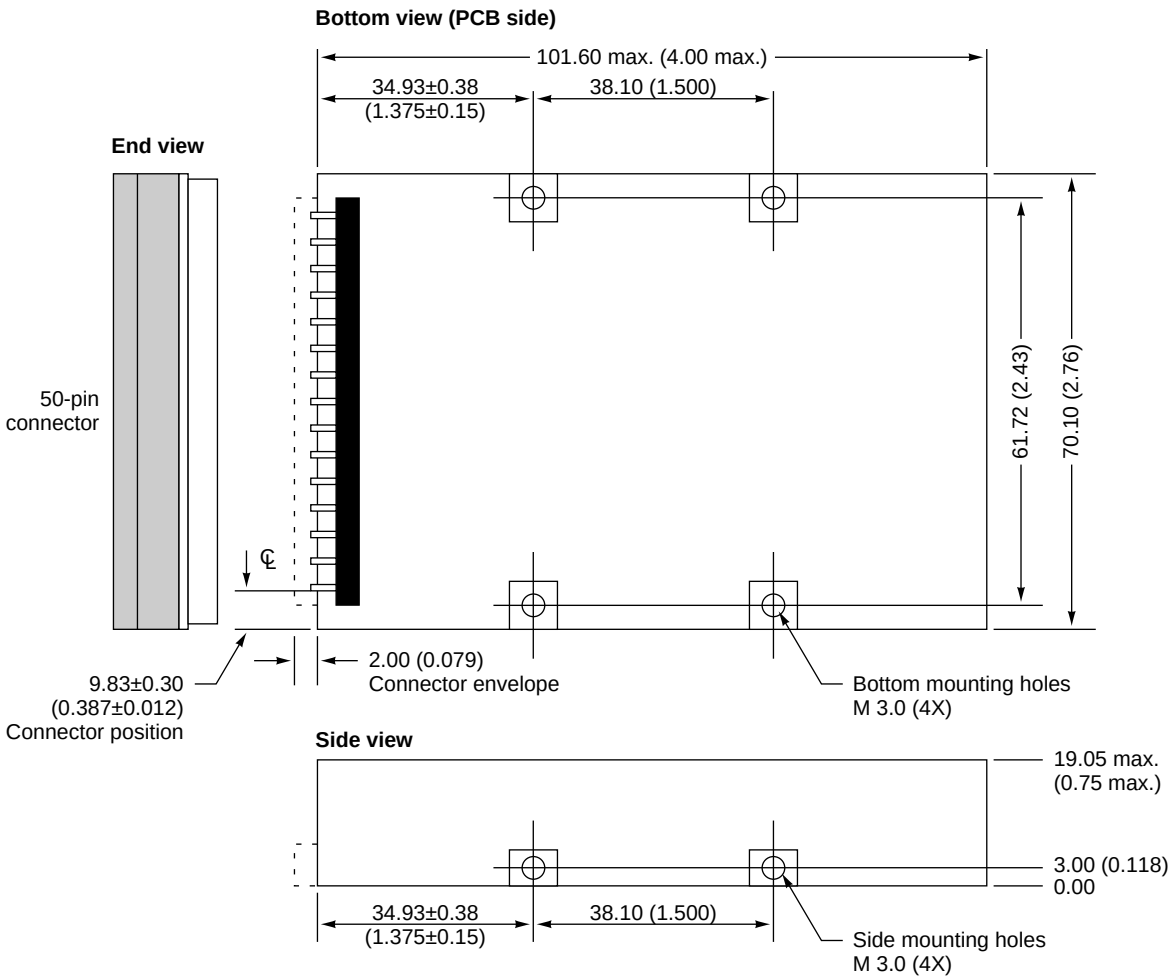


Apple's part number for the hard disk carrier bracket is 805-0608. Developers who wish to include a carrier bracket with their hard disk can obtain the bracket from Apple's vendor, subject to the usual restrictions. For information and authorization for obtaining the bracket, contact Apple's Developer Support Center.

Hard Disk Mounting Envelope

Figure 3-2 shows the drive and connector envelope requirements for the hard disk drive. The drive and its mating connectors must not extend outside the dimensions shown in the figure.

Figure 3-2 Hard disk drive dimensions



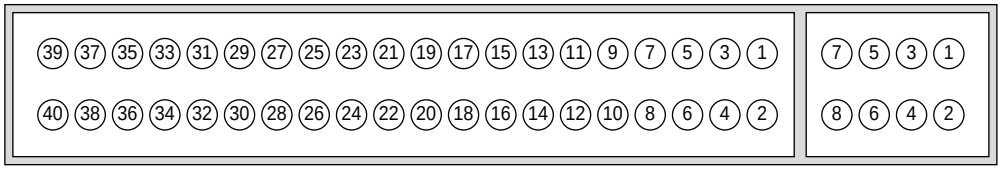
Note: Dimensions are in millimeters and (inches).

Hard Disk Connector

The connector for the internal hard disk is a combination of an 8-pin connector and a 40-pin connector. It is essentially a 50-pin connector with pins 9 and 10 removed. Pins 1–8 constitute the 8-pin connector (J1), and pins 11–50, renumbered as 1–40, constitute the 40-pin connector (J2). Figure 3-3 shows the pin numbering on the combination connector.

The pins on the hard disk connector are 0.5 mm square; pin spacing is 2 mm on centers, horizontal and vertical. Figure 3-2 shows the location of the connector by defining the distance from the edge of the drive to the center line of the nearest pin, which is pin 39.

Figure 3-3 Pins on the hard disk connector



Signals on Connector J1

As shown in Table 3-5, pins 5, 6, and 7 on the 8-pin connector (J1) are used as identification signals to encode the SCSI ID for the internal hard disk. Table 3-5 shows the values of the ID signals corresponding to possible SCSI ID values from 1 to 7.

Table 3-2 Signals on connector J1

Pin number	Signal name	Signal description
1	n.c.	No connection
2	n.c.	No connection
3	n.c.	No connection
4	n.c.	No connection
5	/ID1	Bit 0 of SCSI ID
6	/ID2	Bit 1 of SCSI ID
7	/ID4	Bit 2 of SCSI ID
8	n.c.	No connection

Table 3-3 SCSI ID encoding

SCSI ID value	/ID1	/ID2	/ID4	SCSI ID value	/ID1	/ID2	/ID4
0	High	High	High	4	High	High	Low
1	Low	High	High	5	Low	High	Low
2	High	Low	High	6	High	Low	Low
3	Low	Low	High	7	Low	Low	Low

Signals on Connector J2

Table 3-5 shows the signal assignments for the internal hard disk connector (J2). Entries in the table are arranged the same way as the pins on the connector: pin 1 across from pin 2, and so on.

Table 3-4 Signals on the internal hard disk connector (J2)

Pin number	Signal name	Pin number	Signal name
1	+5 V logic supply	2	+5 V logic supply
3	Logic return	4	Logic return
5	GND	6	/DB0
7	GND	8	/DB1
9	GND	10	/DB2
11	GND	12	/DB3
13	GND	14	/DB4
15	GND	16	/DB5
17	Key (n.c.)	18	/DB6
19	GND	20	/DB7
21	GND	22	/PARITY
23	GND	24	TERM_PWR
25	/ATN	26	/BSY
27	GND	28	/ACK
29	/RST	30	/MSG
31	GND	32	/SEL
33	/I/O	34	/C/D
35	GND	36	/REQ
37	Motor return	38	Motor return
39	+5 V motor supply	40	+5 V motor supply

Internal Floppy Disk Drive

The floppy disk interface is identical to those in previous PowerBook computers. The internal floppy disk drive is a 1.44 MB Apple SuperDrive. The drive is 15 mm high; it accepts either standard Macintosh disks or DOS floppy disks.

Table 3-5 shows the signal assignments for the internal floppy disk connector.

Table 3-5 Signals on the internal 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	/SYS.PWR	System power
10	/WREQI	Write data request
11	FD1.+5V/0	+5 V/0 V (awake/sleep)
12	HDSELI	Head select
13	FD1.+5V/0	+5 V/0 V (awake/sleep)
14	/DISK1EN	Drive enable
15	FD1.+5V/0	+5 V/0 V (awake/sleep)
16	RD	Read data
17	FD1.+5V/0	+5 V/0 V (awake/sleep)
18	WRDATA	Write data
19	n.c.	Not connected
20	n.c.	Not connected

Trackpad

For their pointing device, the PowerBook 520 and 540 computers have a trackpad, an integrated flat pad that replaces the trackball used in previous PowerBook computers. The trackpad provides precise cursor positioning in response to motions of the user's fingertip over the surface of the pad. A single button below the trackpad is used to make selections.

The trackpad is a solid-state device that emulates a mouse by sensing the motions of the user's finger over its surface and translating those motions into ADB commands. The trackpad is lighter and more durable than the trackball used in earlier PowerBook computers, and it consumes less power.

Also see the section "Trackpad Software" on page 74.

Keyboard

A new keyboard design provides 76 (U.S.) or 77 (ISO) keys, including 12 function keys. Figure 3-4 shows the version of the keyboard used on machines sold in the United States. Figure 3-5 shows the version of the keyboard used on machines sold in countries that require the ISO standard.

Figure 3-4 PowerBook 520 and 540 keyboard, U.S. layout

Figure 3-5 PowerBook 520 and 540 keyboard, ISO layout

By removing two screws, the user can lift out the keyboard to obtain access to the internal components and expansion connectors inside the PowerBook 520 and 540 computers.

Flat Panel Displays

The PowerBook 520 and 540 computers have built-in flat panel displays showing 640 by 480 pixels. Four types of flat panel display are used in the different PowerBook models, as shown in Table 3-6. All four types of displays are the same size, 9.5 inches (diagonal); all have a dot pitch of 0.30 mm and are backlit by a cold cathode fluorescent lamp (CCFL). The displays can show up to 8 bits per pixel, which provides 256 colors on color displays or 256 levels of gray on grayscale displays.

Table 3-6 Types of flat panel displays

Model	Display type
PowerBook 540c	Active matrix color
PowerBook 540	Active matrix grayscale
PowerBook 520c	DualScan color
PowerBook 520	Supertwist grayscale

Types of Displays

Flat-panel displays come in two types: active matrix and passive matrix.

Active matrix displays, also called thin-film transistor (TFT) displays, have a driving transistor for each individual pixel. The driving transistors give active matrix displays high contrast and fast response time.

Passive matrix refers to a display technology that does not have individual transistors. That technology is also called FSTN, for film supertwist nematic, sometimes shortened to just supertwist..

DualScan is Apple's new type of FSTN color, an improved version of the color display used in the PowerBook 165c.

Flat Panel Display Circuitry

The flat panel display circuitry in the PowerBook 520 and 540 computers emulate a NuBus™ video card installed in slot $0. There is no declaration ROM as such; its functions have been incorporated into the system ROM.

The display circuitry includes the CSC controller IC and a display buffer consisting of 512 KB of VRAM. The LCD display is compatible with software that uses QuickDraw and the Palette Manager. The display supports color table animation.

Note
The display circuitry in the PowerBook 520 and 540 computers is similar to that in the PowerBook 270c. ◆

Number of Colors

The display controller IC contains a 256-entry CLUT. Although the CLUT supports a palette of 32,768 colors, many of the possible colors do not look acceptable on the display. Due to the nature of color LCD technology, some colors are dithered or exhibit noticeable flicker. Apple has developed new gamma tables for these displays that minimizes flicker and optimizes available colors. With these gamma tables, the effective range of the CLUT for the active matrix color display is about 24,000 colors; for the DualScan color display, the effective range is about 4000 colors.

Note
Unlike the LCD controller on the PowerBook 160 and 180 models, the LCD controller on the PowerBook 520 and 540 computers is compatible with QuickDraw at all bit depths. ◆

External Video Port

The external video port provides video output for all Apple 12-inch, 13-inch, and 16-inch RGB monitors, the Apple Macintosh Portrait Display, and Apple's new 17-inch multiscan

display. With appropriate adapter cables, the external video port can also support a VGA display or a 800-by-600-pixel SVGA display.

The external video interface is enabled by attaching a monitor and restarting the computer. During the boot process, ROM software tests the monitor sense lines and activates the video output system if a recognized monitor is attached. If no monitor is found, the video output system is deactivated to conserve power.

Video Mirroring

There are two video output modes: dual mode and mirror mode. In dual mode, which is the normal Macintosh mode of operation, the external video monitor is independent of the flat panel display and displays additional information. Alternatively, the user can select mirror mode, in which the external monitor mirrors (duplicates) the flat panel display. In mirror mode, the display on the larger screen uses only the central portion of that screen and matches the horizontal and vertical dimensions of the smaller screen.

▲ **WARNING**
Applications that write directly to the display buffer may not be compatible with mirror mode unless they ensure that they do not write outside the active display area. That is not a problem for applications that use QuickDraw and never write directly to the display buffer. ▲

Because the video output circuitry consumes additional power, Apple recommends that customers use the AC adapter when using an external monitor.

Video Monitors

The PowerBook 520 and 540 computers support the external monitors and video modes listed in Table 3-7.

Table 3-7 Video monitors and modes

Monitor type	Width (pixels)	Height (pixels)	Pixel depth (bits)	Frame rate (Hz)
12-inch RGB	512	384	8	60.15
13-inch RGB*	640	480	8	66.67
Portrait	640	870	4	75.0
16-inch RGB	832	624	8	66.67
17-inch multiscan	640	480	8	66.67
17-inch multiscan	832	624	8	75.0
VGA or SVGA	640	480	8	59.95
SVGA	800	600	8	55.98

* Includes Macintosh Color Display and Apple High Resolution Monochrome Monitor.

Note
The largest image the PowerBook 520 and 540 computers can display on a multiscan monitor is 832 by 624 pixels. ◆

With a multiscan monitor, the Monitors control panel allows the user to select from the different modes available. Similarly, with an SVGA monitor, the user can select either the 640-by-480-pixel display or the 800-by-600-pixel display.

External Video Connector

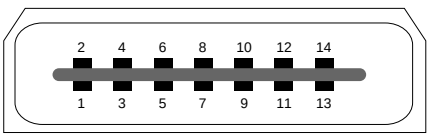
The PowerBook 520 and 540 computers have the same type VID-14 video output connector as earlier models. An adapter cable included with the computer allows the user to attach a standard Apple video cable. Figure 3-6 shows the pin configuration of the VID-14 connector on the back of the computer and the DB-15 connector on the adapter cable. Table 3-8 lists the signal pin assignments.

One source for the VID-14 adapter cable is

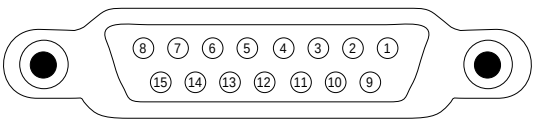
Hosiden America Corp.
10090 Pasadena Ave., Suite B2
Cupertino, CA 95014
408-252-0541

Refer to Hosiden part number CMP1220-010100.

Figure 3-6 Pin assignments on the video connectors



VID-14 connector socket



DB-15 connector socket

Monitor Sense Codes

To identify the type of monitor connected, the external video interface uses the standard Apple monitor sense codes as well as the newer extended sense codes. Table 3-9 shows the sense codes for each of the supported monitors. Refer to the Macintosh Technical Note 30 *SenseLines* for a description of the sense code system.

Table 3-8 Signals on the video connector

Pin			
VID-14	DB-15	**Signal name**	**Description**
1	2	RED.VID	Red video signal
2	1	RED.GND	Red video ground
3	4	SENSE0	Monitor sense signal 0
4	12	/VSYNC	Vertical synchronization signal
5	3	/CSYNC	Composite synchronization signal
6	11	GND	CSYNC and VSYNC ground
7	6	GRN.GND	Green video ground
8	5	GRN.VID	Green video signal
9	7	SENSE1	Monitor sense signal 1
10	14	HSYNC.GND	HSYNC ground
11	10	SENSE2	Monitor sense signal 2
12	15	/HSYNC	Horizontal synchronization signal
13	9	BLU.VID	Blue video signal
14	13	BLU.GND	Blue video ground
—	8	n.c.	Not connected
Shell	Shell	SGND	Shield ground

Table 3-9 Monitor sense codes

	Standard sense codes	**Extended sense codes**		
Monitor type	(2–0)	(1, 2)	(0, 2)	(0, 1)
12-inch RGB	0 1 0	n.a.	n.a.	n.a.
13-inch RGB	1 1 0	n.a.	n.a.	n.a.
Portrait	0 0 1	n.a.	n.a.	n.a.
16-inch RGB	1 1 1	1 0	1 1	0 1
17-inch multiscan	1 1 0	1 1	0 1	0 0
VGA/SVGA	1 1 1	0 1	0 1	1 1
No monitor	1 1 1	1 1	1 1	1 1

Serial Port

The PowerBook 520 and 540 computers have a standard Macintosh serial port for synchronous, asynchronous, or AppleTalk serial communication. The 8-pin mini-DIN connector on the back panel is the same as those on other Macintosh computers. Table 3-10 shows the signal assignments for the serial port.

Table 3-10 Serial port signals

Pin number	Signal name	Signal description
1	HSKo	Handshake output
2	HSKi	Handshake input
3	TxD–	Transmit data –
4	SG	Signal ground
5	RxD–	Receive data –
6	TxD+	Transmit data +
7	GPi	General-purpose input
8	RxD+	Receive data +

SCSI Port

The SCSI port on the PowerBook 520 and 540 computers supports the SCSI interface as defined by the American National Standards Institute (ANSI) X3T9.2 committee.

SCSI Disk Mode

The PowerBook 520 and 540 computers support SCSI disk mode, which allows the internal hard disk to be mounted and used as an external drive by another Macintosh. To operate a PowerBook computer in SCSI disk mode, the user connects an HDI-30 SCSI Disk Adapter cable between the external SCSI connectors on the PowerBook computer and the desktop computer and then starts both computers. The adapter cable grounds pin 1 of the HDI-30 connector, causing the PowerBook computer's ROM code to bypass the normal startup procedure and enter SCSI disk mode.

SCSI Connectors

The internal and external HDI-30 connectors are identical to those used in other PowerBook models. The SCSI portion of the Combo IC connects directly to the internal and external SCSI connectors and can sink up to 48 mA through each of the pins connected to the SCSI bus. The data and control signals on the SCSI bus are active low signals that are driven by open drain outputs.

Table 3-11 shows the signal assignments for the internal and external SCSI connectors. Note that pin 1 of the external HDI-30 connector is the /SCSI.DISK.MODE signal.

Table 3-11 Internal and external HDI-30 SCSI connector signals

Pin number	HDI-30 (internal)	HDI-30 (external)
1	DISK.+5	/SCSI.DISK.MODE
2	DISK.+5	/DB0
3	GND	GND
4	GND	/DB1
5	GND	TERMPWR (not used; reserved for future use)
6	/DB0	/DB2
7	/DB1	/DB3
8	/DB2	GND
9	/DB3	/ACK
10	/DB4	GND
11	/DB5	/DB4
12	/DB6	GND
13	/DB7	GND
14	/DBP	/DB5
15	DISK.+5	GND
16	/BSY	/DB6
17	/ATN	GND
18	/ACK	/DB7
19	GND	/DBP
20	/MSG	GND
21	/RST	/REQ
22	/SEL	GND

continued

Table 3-11 Internal and external HDI-30 SCSI connector signals (continued)

Pin number	HDI-30 (internal)	HDI-30 (external)
23	/C/D	/BSY
24	/I/O	GND
25	/REQ	/ATN
26	GND	/C/D
27	GND	/RST
28	GND	/MSG
29	DISK.+5	/SEL
30	DISK.+5	/I/O

Ethernet Port

The Ethernet connector is an Apple AUI connector. It accepts a Friendlynet adapter for either AUI (thick) cable, thin cable, or 10baseT (twisted pair) cable. The Ethernet port pin assignments are shown in Table 3-12.

Table 3-12 Pin assignments on the Ethernet port

Pin	Description	Pin	Description
1	+5 V	8	+5 V
2	DI+	9	DO+
3	DI–	10	DO–
4	Ground	11	Ground
5	CI+	12	NC
6	CI–	13	NC
7	+5 V	14	+5 V

ADB Port

The Apple Desktop Bus (ADB) port on the PowerBook 520 and 540 computers 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 PowerBook 520 and 540 computers. The custom ADB microcontroller drives the bus and reads status from the selected external device. A 4-pin mini-DIN connector connects the ADB controller to the outside world. Table 3-13 lists the ADB connector pin assignments. For more information about the ADB, see *Guide to the Macintosh Family Hardware,* second edition.

Table 3-13 ADB connector pin assignments

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

IMPORTANT
The total current available for all devices connected to the +5V pins on the ADB is 100 mA. ▲

Sound System

The 16-bit stereo audio circuitry provides high-quality sound input and output through the built-in microphone and speakers. The user can also connect external input and output devices by way of the sound input and output jacks.

The sound system is based on the Singer codec IC along with input and output amplifiers. In the PowerBook 520 and 540 computers, the Singer codec supports two channels with sample sizes up to 16 bits and sample rates of 11 kHz, 22.05 kHz, and 44.1 kHz.

The frequency response of the sound circuits, not including the microphone and speakers, is within plus or minus 2 dB from 20 Hz to 20 kHz. Total harmonic distortion and noise is less than 0.05 percent from 20 Hz to 20 kHz with a 1 V RMS sine wave input. The input signal-to-noise ratio (SNR) is 82 dB and the output SNR is 85 dB, with no audible discrete tones.

The sound system includes a built-in microphone and two speakers as well as external audio input and output jacks. The user can also plug in an external microphone and select either the built-in microphone or the external microphone by using the Sound control panel.

The speakers are located in the upper-left and upper-right corners of the display bezel. The speakers are modified versions of the ones used in the PowerBook 140 and 170 computers, with improved low-frequency response.

CHAPTER 4

Expansion Modules

The PowerBook 520 and 540 computers accept the following expansion modules, implemented as small circuit cards:

- RAM expansion card
- PDS card
- modem card

This chapter includes descriptions and design guides for each of these expansion cards.

RAM Expansion Card

A connector on the secondary logic board accepts a RAM expansion card containing from 8 MB to 32 MB of self-refreshing dynamic RAM. The card can contain from one to four identical banks, with 2 MB, 4 MB, or 8 MB in each bank. Table 4-1 shows how the banks can be implemented with standard RAM devices.

Table 4-1 Configurations of RAM banks

Size of bank	Number of devices per bank	Device size (bits)
2 MB	4	512K × 8
4 MB	8	1 M × 4
4 MB	2	1 M × 16
8 MB	4	2 M × 8

IMPORTANT

The RAM expansion card for the PowerBook 520 and 540 computers is a new design; cards designed for other PowerBook computers cannot be used in these new PowerBook models. ▲

▲ **WARNING**

Installation of a RAM expansion card in a PowerBook 520 or 540 computer must be done by an experienced technician. It requires care to avoid damage to the pins on the RAM expansion connector. ▲

Electrical Design Guidelines for the RAM Expansion Card

This section provides the electrical information you need to design a RAM expansion card for the PowerBook 520 and 540 computers.

Connector Pin Assignments

Table 4-2 lists the names of the signals on the RAM expansion connector. Entries in the table are arranged the same way as the pins on the connector: pin 1 across from pin 2, and so on. Signal names that end with _L are active low.

Table 4-2 Signal assignments on the RAM expansion connector

Pin	Signal name	Direction	Pin	Signal name	Direction
1	GND	—	2	DRAM_ADDR[0]	O
3	DRAM_ADDR[1]	O	4	DRAM_ADDR[2]	O
5	DRAM_ADDR[3]	O	6	GND	—
7	DRAM_ADDR[5]	O	8	DRAM_ADDR[4]	O
9	DRAM_ADDR[7]	O	10	DRAM_ADDR[6]	O
11	V_5P_MAIN	—	12	DRAM_ADDR[8]	O
13	DRAM_ADDR[9]	O	14	DRAM_ADDR[10]	O
15	DRAM_ADDR[11]	O	16	GND	—
17	DRAM_RAS_L[3]	O	18	DRAM_RAS_L[2]	O
19	DRAM_RAS_L[5]	O	20	DRAM_RAS_L[4]	O
21	GND	—	22	DRAM_CAS_L[0]	O
23	DRAM_CAS_L[1]	O	24	DRAM_CAS_L[2]	O
25	DRAM_CAS_L[3]	O	26	V_5P_MAIN	—
27	DRAM_WE_L	O	28	RESERVED[1]	X
29	RESERVED[2]	X	30	RESERVED[3]	X
31	GND	—	32	CPU_DATA[0]	B
33	CPU_DATA[1]	B	34	CPU_DATA[2]	B
35	CPU_DATA[3]	B	36	GND	—
37	CPU_DATA[5]	B	38	CPU_DATA[4]	B
39	CPU_DATA[7]	B	40	CPU_DATA[6]	B
41	V_5P_MAIN	—	42	CPU_DATA[8]	B
43	CPU_DATA[9]	B	44	CPU_DATA[10]	B
45	CPU_DATA[11]	B	46	GND	—
47	CPU_DATA[13]	B	48	CPU_DATA[12]	B
49	CPU_DATA[15]	B	50	CPU_DATA[14]	B
51	GND	—	52	CPU_DATA[16]	B

continued

Table 4-2 Signal assignments on the RAM expansion connector (continued)

Pin	Signal name	Direction	Pin	Signal name	Direction
53	CPU_DATA[17]	B	54	CPU_DATA[18]	B
55	CPU_DATA[19]	B	56	V_5P_MAIN	—
57	CPU_DATA[21]	B	58	CPU_DATA[20]	B
59	CPU_DATA[23]	B	60	CPU_DATA[22]	B
61	GND	—	62	CPU_DATA[24]	B
63	CPU_DATA[25]	B	64	CPU_DATA[26]	B
65	CPU_DATA[27]	B	66	GND	—
67	CPU_DATA[29]	B	68	CPU_DATA[28]	B
69	CPU_DATA[31]	B	70	CPU_DATA[30]	B

NOTE Signal directions are defined from the system side of the connector; O = output and B = bidirectional. Signals marked X are reserved.

Signal Descriptions

Table 4-3 describes the signals on the RAM expansion connector. Signal names that end with _L are active low.

Note

In the table, signals are specified as inputs or outputs with respect to the secondary logic board that contains the CPU and memory module; for example, an input is driven by the expansion card into the secondary logic board. ◆

Address Multiplexing

DRAM_ADDR[0–11] is a 12-bit multiplexed address bus and can support several different types of DRAM devices.

Depending on their internal design and size, different types of DRAM devices require different row and column address multiplexing. The operation of the multiplexing is determined by the way the address pins on the devices are connected to individual signals on the DRAM_ADDR[0–11] bus and depends on the exact type of DRAM used.

Table 4-4 on page 38 shows how the signals on the address bus are connected for several types of DRAM devices. The device types are specified by their size and by the number of row and column address bits they require.

Table 4-3 Descriptions of signals on the RAM expansion connector

Signal name	Description
CPU_DATA[0–31]	Bidirectional 32-bit DRAM data bus.
DRAM_ADDR[0–11]	Multiplexed row and column address to the DRAM devices. (See the section "Address Multiplexing" on page 36 to determine which bits to use for a given type of DRAM device.)
DRAM_CAS_L[0–3]	Column address select signals for the individual bytes in a longword. The signals are assigned to the bytes as follows:
	DRAM_CAS_L[3] selects CPU_DATA[24–31]
	DRAM_CAS_L[2] selects CPU_DATA[16–23]
	DRAM_CAS_L[1] selects CPU_DATA[8–15]
	DRAM_CAS_L[0] selects CPU_DATA[0–7]
DRAM_RAS_L[2–5]	Row address select signals for up to four banks of DRAM. (The first two banks, selected by DRAM_RAS_L[1:0], reside on the CPU and memory module.)
DRAM_WE_L	Write enable for the banks of DRAM.
GND	Chassis and logic ground.
RESERVED[1–3]	Reserved pins; must be left open.
V_5P_MAIN	5.0 ± 5%; 300 mA maximum

Table 4-4 shows how the signals are multiplexed during the row and column address phases. For each type of DRAM device, the first and second rows show the actual address bits that drive each address pin during row addressing and column addressing, respectively. The third row shows how the device's address pins are connected to the signals on the DRAM_ADDR bus.

IMPORTANT

Some types of DRAM devices don't use all 12 bits in the row or column address. Address-bit numbers for those unused bits are shown in plain style; bit numbers for the bits that are used are shown in bold. ▲

Table 4-4 Address multiplexing for some typical DRAM devices

	Individual signals on DRAM_ADDR bus											
Type of DRAM device	[11]	[10]	[9]	[8]	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
2 M × 8, 12 row bits, 9 column bits												
Row address bits	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	19	21	18	**22**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1 M × 16, 12 row bits, 8 column bits												
Row address bits	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	19	21	18	22	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1 M × 4, 12 row bits, 8 column bits												
Row address bits	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	19	21	18	22	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
2 M × 8, 11 row bits, 10 column bits												
Row address bits	**21**	**20**	19	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	**19**	21	18	**22**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	**9**	**10**	—	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1 M × 16, 10 row bits, 10 column bits												
Row address bits	21	**20**	**19**	18	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	19	**21**	**18**	22	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	—	**9**	**8**	—	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
512K × 8, 10 row bits, 9 column bits												
Row address bits	21	**20**	**19**	18	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	19	21	**18**	22	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	—	**9**	**8**	—	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

Banks of DRAM

The DRAM expansion card can have up to four banks of RAM. Banks can be 2 MB, 4 MB, or 8 MB in size; on a card with more than one bank, all banks must be the same size.

Because only one bank is active at a time, and because different-sized DRAM devices consume about the same amount of power when active, a card having fewer devices per bank consumes less power than a card having more devices per bank.

DRAM Device Requirements

The DRAM devices used in a DRAM expansion card must meet the following minimum specifications:

- fast page mode
- self-refreshing
- low-power grade
- row access time (t_{RAC}) of 70 ns or less
- column access time (t_{CAC}) of 20 ns or less
- page-mode cycle time (t_{PC}) of 50 ns or less

Note
The DRAM refresh operation depends on the state of the computer. When the computer is operating, the Pratt memory controller IC (described on page 12) provides refresh signals consisting of 2048 CAS before RAS cycles every 128 ms. When the computer goes into sleep mode, it switches the DRAM devices to their self-refresh feature to save power. ◆

Expansion Card Electrical Limits

The DRAM expansion card must not exceed the following maximum current limits on the +5 V supply:

active	300 mA
standby	24 mA
self-refresh	6 mA

The capacitative loading on the signal lines must not exceed the following limits:

DRAM_ADDR[0–11]	85 pF
DRAM_RAS_L[2–5]	25 pF
DRAM_CAS_L[0–3]	25 pF
DRAM_WE_L	85 pF
CPU_DATA[0–31]	25 pF

Mechanical Design Guidelines for the RAM Expansion Card

This section contains mechanical drawings showing the recommended design guidelines for RAM expansion cards.

RAM Expansion Connector

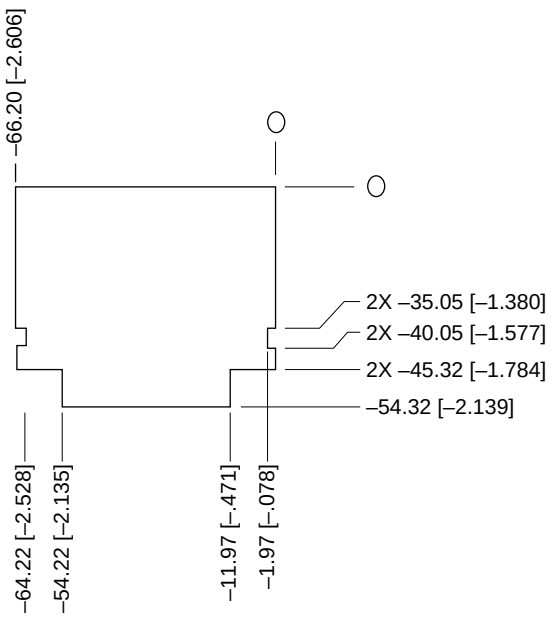
The RAM expansion connector is a 70-pin Metristack connector. The connector is available from AMP, Inc. (part number 535671-6). For a specification sheet or information about obtaining this connector, contact

AMP, Inc.
19200 Stevens Creek Blvd.
Cupertino, CA 95014-2578
408-725-4914
AppleLink: AMPCUPERTINO

RAM Expansion Card Design

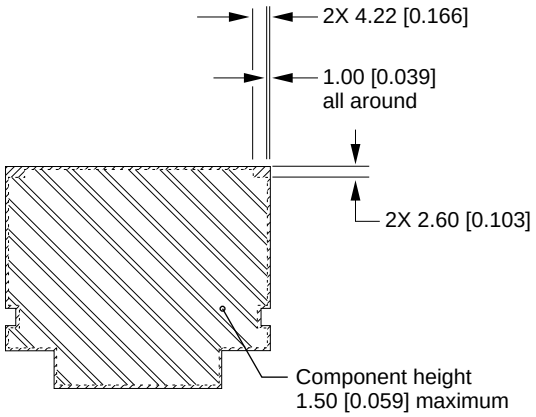
This section shows the mechanical outline and parts-placement guidelines for the RAM expansion card. Figure 4-1 is a top view of the card showing dimensions. Figure 4-2 and Figure 4-3 are top and bottom views showing the restrictions on placement of components and traces. Figure 4-4 is an edge view of the card; the dotted lines show the maximum component height on either side.

Figure 4-1 Dimensions of the RAM expansion card



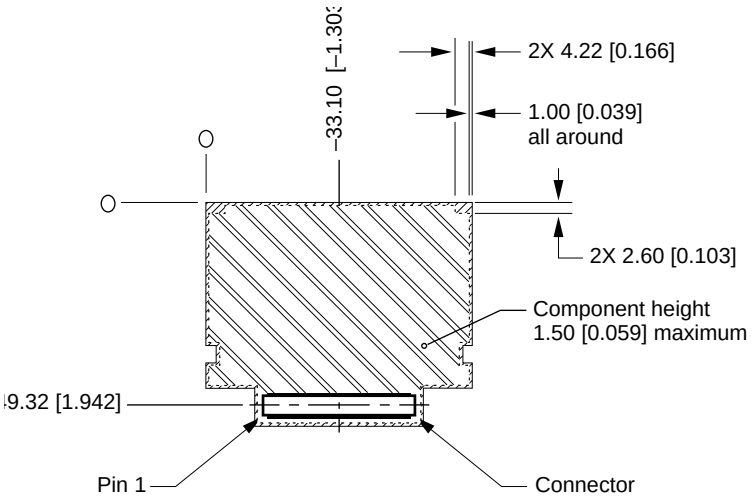
Note: Dimensions are in millimeters [inches].

Figure 4-2 Restricted areas on the top of the RAM expansion card



Note: Dimensions are in millimeters [inches].

Figure 4-3 Restricted areas on the bottom of the RAM expansion card

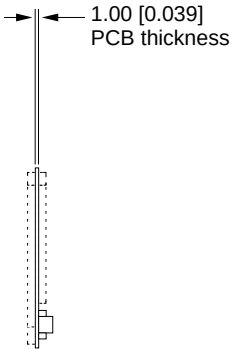


Note: Dimensions are in millimeters [inches].

Note

The drawings in Figure 4-1, Figure 4-2, Figure 4-3, and Figure 4-4 are taken from Apple engineering drawing number 056-0124, revision 02. Developers who are members of Apple's Developer Program may contact Apple's Developer Support Center for information about possible later revisions of the drawings. ◆

Figure 4-4 Edge view of the RAM expansion card



Note: Dimensions are in millimeters [inches].

▲ **WARNING**
Do not exceed the dimensions shown in the mechanical drawings. Cards that exceed these specifications may damage the computer and may be incompatible with future PowerBook models. ▲

PDS Expansion Card

The processor-direct slot (PDS) in the PowerBook 520 and 540 computers is a 90-pin connector located inside the left battery compartment. Users must remove the left battery to install a PDS card. A battery compartment cover is included with the computer for use when the battery is removed.

The PDS connector provides direct access to the MC68030-compatible bus on the main logic board. The PDS connector also accepts an optional adapter that can accommodate one or two industry-standard PCMCIA cards: two type II cards or one type III card.

Electrically, the PDS connector is not a true PDS because it is connected to the input/output subsystem's MC68030 bus and not directly to the MC68040 microprocessor bus. Because the PDS is independent of the processor bus, users can upgrade their microprocessor without having to purchase new PDS cards. The fact that the PDS is connected to the MC68030 bus also simplifies card design for developers with existing MC68030 PDS products.

User Installation of the PDS Card

The PDS card can be installed and removed by the user. Even though the instructions warn against inserting or removing the card while the computer is running, the user may not follow instructions. The card must be designed to prevent damage to the card and the computer in that eventuality.

Here are the requirements that minimize the possibility of damage if the user inserts or removes the PDS card with the computer running.

- The PDS connector is designed so that the power and ground pins make contact before the signal lines when the card is inserted.
- The PDS card must buffer all signals so as not to load down the computer's internal bus. The recommended way to do that is to install CMOS logic buffers on all the logic signals used on the PDS card. The buffers must draw their power directly from the +5 V supply line and must always be powered.
- The PDS card must indicate to the computer that the card is inserted by grounding the PDS insert signal. The line should be tied directly to ground so that it goes low as soon as the PDS card is inserted.
- The PDS card must not pull down the +5 V supply line even when the card is inserted in a discharged state. To prevent a glitch (momentary disturbance) in the +5 V supply line, the card must include a surge suppression circuit that limits the inrush current for the circuitry on the card. See the section "Power Control Circuits" on page 49.
- To prevent the card's bus drivers from presenting a load to the bus when the card is inserted, the card should have a pull-up resistor (100K) on the address strobe line. This will ensure that the card will have an invalid address and the drivers will be in the off state.

By following the guidelines outlined here, you can design a PDS card that minimizes the possibility of damage even when it is inserted hot, that is, while the computer is running or in sleep mode. Even so, your instructions to the user should include warnings against doing so.

IMPORTANT

Even with the design features described here, eventually a circuit may fail; if the user repeatedly inserts and removes the card with power on, damage to the computer or the card may eventually occur. The only time it is completely safe to insert or remove the card is while the computer is shut down. ▲

Electrical Description of the PDS

This section describes the electrical characteristics of the PDS. For mechanical specifications, see the section "Mechanical Design Guidelines for the PDS Card" beginning on page 52.

Signals on the PDS Connector

Table 4-5 shows the signal assignments on the PDS connector. Entries in the table are arranged the same way as the pins on the connector: pin 1 across from pin 46, pin 2 across from pin 47, and so on. Signal names that end with _L are active low.

Table 4-5 Signal assignments on the PDS connector

Pin number				Pin number			
New	Old	**Signal name**	**Direction**	New	Old	**Signal name**	**Direction**
1	90	GND	—	31	30	GND	—
2	88	ADDR[1]	B	32	28	DSACK_L[1]	B
3	86	ADDR[3]	B	33	26	DS_L	B
4	84	ADDR[5]	B	34	24	BERR_L	OC
5	82	ADDR[7]	B	35	22	PDS_BG_L	O
6	80	ADDR[9]	B	36	20	PDS_BUSCLK	O
7	78	V_5P_MAIN	—	37	18	GND	—
8	76	ADDR[11]	B	38	16	IPL_L[1]	O
9	74	ADDR[13]	B	39	14	SLEEP	O
10	72	ADDR[15]	B	40	12	IO_RESET_L	O
11	70	ADDR[17]	B	41	10	PDS_IRQ_L	I
12	68	ADDR[19]	B	42	8	PDS_CPU	O
13	66	GND	—	43	6	MDM_TDM[1]	X
14	64	ADDR[21]	B	44	4	MDM_TDM[3]	X
15	62	ADDR[23]	B	45	2	PDS_INSERT_L	I
16	60	ADDR[25]	B	46	89	ADDR[0]	B
17	58	ADDR[27]	B	47	87	ADDR[2]	B
18	56	ADDR[29]	B	48	85	ADDR[4]	B
19	54	GND	—	49	83	GND	—
20	52	ADDR[31]	B	50	81	ADDR[6]	B
21	50	SIZ[1]	B	51	79	ADDR[8]	B
22	48	DATA[17]	B	52	77	ADDR[10]	B
23	46	DATA[19]	B	53	75	ADDR[12]	B
24	44	DATA[21]	B	54	73	ADDR[14]	B
25	42	V_5P_MAIN	—	55	71	GND	—
26	40	DATA[23]	B	56	69	ADDR[16]	B
27	38	DATA[25]	B	57	67	ADDR[18]	B
28	36	DATA[27]	B	58	65	ADDR[20]	B
29	34	DATA[29]	B	59	63	ADDR[22]	B
30	32	DATA[31]	B	60	61	ADDR[24]	B

continued

Table 4-5 Signal assignments on the PDS connector (continued)

Pin number				Pin number			
New	Old	Signal name	Direction	New	Old	Signal name	Direction
61	59	V_5P_MAIN	—	76	29	DATA[30]	B
62	57	ADDR[26]	B	77	27	DSACK_L[0]	B
63	55	ADDR[28]	B	78	25	AS_L	B
64	53	ADDR[30]	B	79	23	V_5P_MAIN	—
65	51	RW_L	B	80	21	PDS_BR_L	I
66	49	SIZ[0]	B	81	19	BGACK_L	B
67	47	GND	—	82	17	IPL_L[0]	O
68	45	DATA[16]	B	83	15	IPL_L[2]	O
69	43	DATA[18]	B	84	13	PDS_RESET_L	O
70	41	DATA[20]	B	85	11	GND	—
71	39	DATA[22]	B	86	9	SND_SCLK	O
72	37	DATA[24]	B	87	7	PDS_POWER_SW_L	OC
73	35	GND	—	88	5	MDM_TDM[0]	X
74	33	DATA[26]	B	89	3	MDM_TDM[2]	X
75	31	DATA[28]	B	90	1	PDS_DIGITAL_SND	I

The PDS connector on the main logic board is a 90-pin hybrid: half the leads are attached to the board as through-hole pins and half as surface-mount leads. The through-hole pins are numbered from 1 through 45 and the surface-mount pins are numbered from 46 through 90. The pins are divided into two groups by a gap. Pins 1 and 46 are at the end of the connector nearest the gap; pins 45 and 90 are at the end farthest from the gap.

Note
On early schematic diagrams from Apple, the pins on the PDS connector were numbered starting from the other end. That numbering scheme is shown in Table 4-5 in the column labeled *Old*. ◆

The signals on the PDS connector are divided by function into three groups: MC68030-compatible signals, system support signals, and power leads. The next two sections describe the MC68030 signals and the system signals. The power leads are described in the section “PDS Power Consumption” beginning on page 49.

MC68030-Compatible Signals on the PDS Connector

Table 4-6 lists the MC68030-compatible signals and describes their functions. Signal names that end with _L are active low. Refer to the Motorola *MC68030 User’s Manual* for a complete description of these signals.

Table 4-6 MC68030-compatible signals on the PDS connector

Signal name	Description
ADDR[31–0]	Bidirectional 32-bit system address bus.
AS_L	Indicates the occurrence of an active bus transaction: a valid address is on the address bus and the size and read/write signals are valid.
BERR_L	Open collector signal connected to the system's bus error watchdog timer and the processor's /BERR line. This signal is used for two operations: (1) terminating an active bus transaction and (2) causing a bus cycle to be retried when asserted with /HALT.
BGACK_L	Bidirectional signal connected to the processor's /BGACK line; indicates that an alternate bus master has control of the system bus. The alternate master on the PDS card drives this line after it has been granted the system bus and has started a bus transaction. A bus master must never assert BR_L and BGACK_L simultaneously; instead, it should deassert PDS_BR_L and assert BGACK_L at the same time (internal PDS clock edge).
DATA[31–16]	Bidirectional 16-bit system data bus.
DS_L	Three-state output connected to the processor's /DS line. This signal is used to manage the transmission of data to and from the processor. During a read cycle, the PDS card places the requested data on the data bus when this signal is asserted. When this signal is asserted during a write, the processor drives the data bus with data to be written to the PDS card. When this signal is released, the data bus is not driven.
DSACK_L[1–0]	Data strobe acknowledge signals. These signals perform two functions: (1) they indicate the completion of the data transfer portion of a bus transaction by the addressed slave, and (2) they inform the bus master of the size of the data port being accessed during the current transaction.
PDS_BG_L	PDS bus granted; output signal is driven by the system's bus arbitrator. When asserted low, the alternate bus master on the PDS card may take ownership of the system bus after any pending bus traffic has completed—that is, /BGACK, /PDS/CPU_AS signals, and PDS/CPU_DSACK[1–0] (as appropriate to the card) have become inactive. /PDS_BG will be asserted only after all pending bus traffic has been completed.
PDS_BR_L	PDS bus request; logical-ORed with the bus request signals from other alternate bus masters to generate the CPU bus request. This signal is used to request the system's bus from the processor. A PDS alternate bus master must never assert PDS_BR_L and BGACK_L simultaneously. The alternate bus master should be designed to deassert /BR and assert /BGACK at the same time (internal PDS clock edge). Refer to the Motorola *MC68030 User's Manual* for a complete description of this signal. This line is active low.
RW_L	Indicates the type of bus cycle that is occurring: high indicates a read cycle and low indicates a write cycle.
SIZ[1–0]	Indicate the number of bytes remaining to be transferred in the current bus cycle.

System Support Signals on the PDS Connector

Table 4-7 lists the system support signals and describes their functions. Signal names that end with _L are active low.

Table 4-7 System support signals on the PDS connector

Signal name	Description
IO_RESET_L	PDS card reset signal from the main processor.
IPL_L[2–0]	Interrupt level; driven by the interrupt prioritizer in the system to encode the interrupt level. These signals are needed only by an alternate CPU on the card that must respond to interrupts.
MDM_TDM[3–0]	These lines are reserved for communication between the PDS slot and the modem slot. Contact Apple Developer Support for information about their use.
PDS_BUSCLK	PDS bus clock, 15.6672 MHz; supports PDS cards based on the MC68030 synchronous timing specifications.
PDS_CPU	PDS is CPU. This output, when asserted high by the Power Manager, disables the normal system CPU and enables the PDS card as an alternate system CPU. (This CPU function is not currently supported.)
PDS_DIGITAL_SND	Digital 1-bit sound signal (PWM or squarewave); gets mixed into both channels of the analog sound output.
PDS_INSERT_L	PDS card inserted; should be grounded on the PDS card. This signal is used by the system to detect the presence or absence of a PDS card.
PDS_IRQ_L	PDS interrupt request; when asserted low, causes the PDS card to interrupt the system at IPL level 2.
PDS_POWER_SW_L	PDS power switch; similar in operation to the ADB power signal. When driven low when the computer is off or in sleep mode, this signal causes the Power Manager to bring up the system.
PDS_RESET_L	PDS reset signal from the Power Manager IC; when asserted low, resets the PDS card.
SLEEP	Indicates that the system is in sleep mode (asserted high by the Power Manager).
SND_SCLK	Sound clock, 22.5792 kHz.

Descriptions of the Power Leads

Table 4-8 lists the electrical characteristics of the power leads on the PDS connector.

Table 4-8 Power leads on the PDS connector

Name	Description
V_5P_MAIN	+5V power to the PDS card
GND	PDS card ground

Characteristics of the PDS Signals

Table 4-9 lists the electrical characteristics of the signals on the PDS and indicates the signal directions with respect to the computer. The output drive capabilities are given in terms of standard TTL or LS TTL loads and include a safety factor.

Table 4-9 PDS signal load and drive limits

Signal name	Direction	Characteristics
ADDR[31–0]	Bidirectional	Output can drive one LS TTL load.
AS_L	Bidirectional	Output can drive one LS TTL load.
BERR_L	Open collector	Has internal 5K pull-up resistor.
BGACK_L	Bidirectional	Has internal 5K pull-up resistor.
DATA[31–16]	Bidirectional	Output can drive one LS TTL load.
DS_L	Bidirectional	Output can drive one LS TTL load.
DSACK_L[1–0]	Bidirectional	Output can drive one LS TTL load.
IO_RESET_L	Output	Has standard TTL output levels.
IPL_L[2–0]	Output	Can drive two standard LS TTL loads.
PDS_BG_L	Output	Can drive two standard LS TTL loads.
PDS_BR_L	Input	Has internal 100K pull-up resistor.
PDS_BUSCLK	Output	TTL voltage levels; source impedance 33 Ω.
PDS_CPU	Output	Can drive two standard LS TTL loads.
PDS_DIGITAL_SND	Input	Has internal 10K pull-up resistor.
PDS_INSERT_L	Input	Has internal 470K pull-up resistor.
PDS_IRQ_L	Input	Has internal 100K pull-up resistor.
PDS_POWER_SW_L	Open collector	Has internal 10K pull-up resistor.
PDS_RESET_L	Output	Can drive two standard LS TTL loads.
RW_L	Bidirectional	Output can drive one LS TTL load.
SIZ[1–0]	Bidirectional	Output can drive one LS TTL load.
SLEEP	Output	Can drive one standard TTL load.
SND_CLK	Output	Can drive two standard LS TTL loads.

PDS Power Consumption

This section specifies the maximum power available and provides guidelines for designing the power control circuitry on a PDS card.

Power Budget

The DC voltages supplied to the PDS connector and the maximum allowable current load for each voltage are listed in Table 4-10. The signal named V_BATT1 is not on the PDS connector but is on a separate battery connector accessible from the PDS card.

Table 4-10 PDS power budget

Signal name	Voltage	Maximum current
V_5P_MAIN	5.0 V ± 5 %	600 mA in operation 4 mA in sleep 0.00 mA in shutdown
V_BATT1	8–16 V	300 mA in operation 0.00 mA in shutdown

Note
The V_5P_MAIN signal is on the PDS connector itself. The V_BATT1 signal is only available on the battery terminals, which are accessible from a card installed in the battery well. ◆

Power and thermal restrictions limit the amount of power that can be dissipated by a PDS card to a maximum of 3 watts (average). The entire 3 watts can be from the 5-volt supply or from a combination of the two supply voltages, but the total cannot exceed 3 watts.

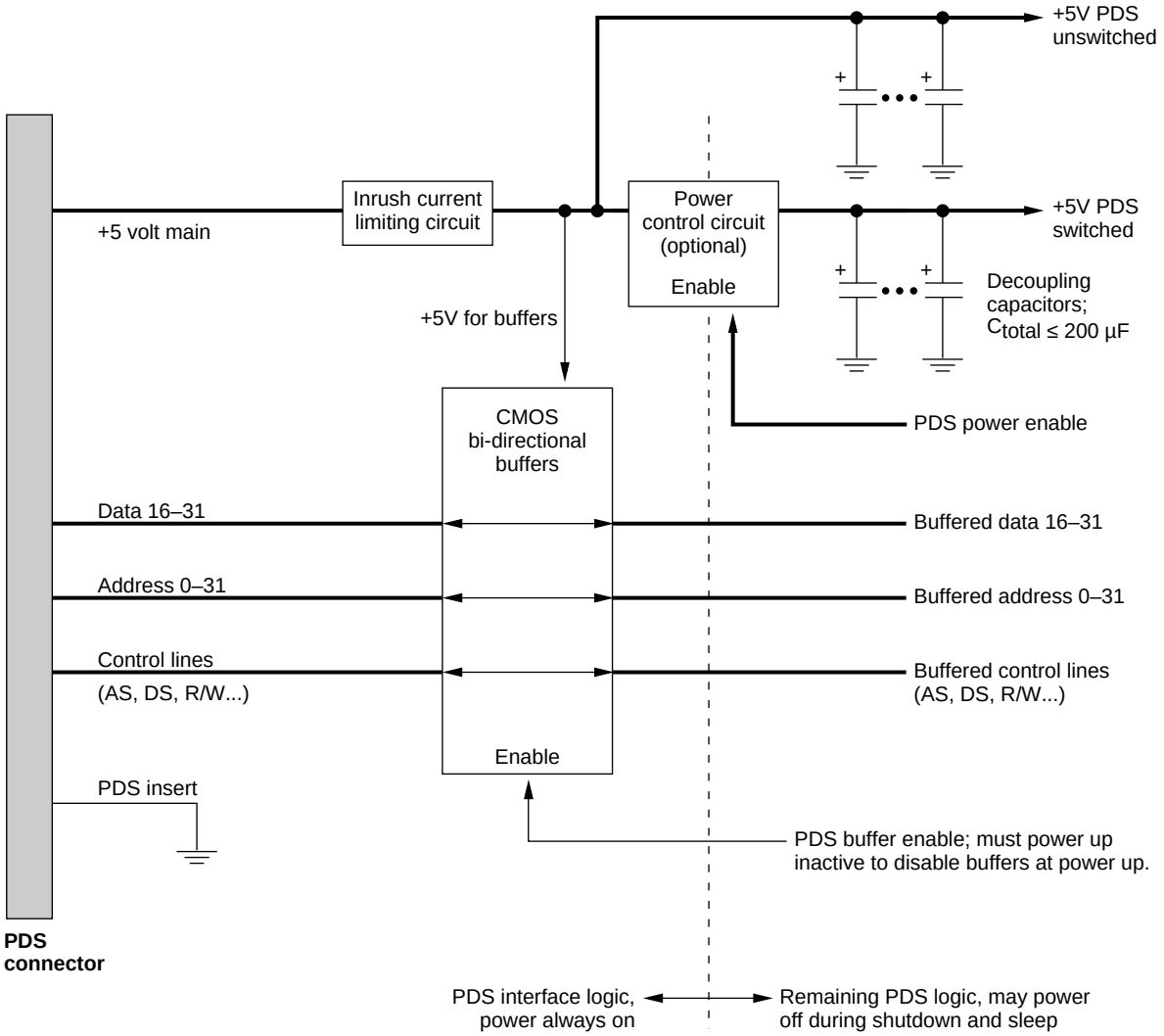
▲ **WARNING**
A card dissipating an average power of more than 3 watts may overheat and damage the computer's circuitry or cause it to become inoperable. ▲

Power Control Circuits

To prevent excessive battery drain, the PDS card must include circuitry to put the card into a low-power mode when the computer is in sleep mode. In addition, because the PDS card is not used all the time even when the computer is in run mode, you may wish to include some form of power cycling to reduce current consumption when the PDS card is not being used.

Figure 4-5 shows recommended interface circuitry to isolate the PDS card so that its power can be controlled independently of the computer. Figure 4-6 shows a recommended circuit to limit the inrush current to the PDS. Figure 4-7 is a diagram of the recommended circuit to control the power. The section "User Installation of the PDS Card" beginning on page 42 discusses the reasons for an inrush current limiting circuit.

Figure 4-5 Recommended connection logic on the PDS card



The bus interface controller on the PDS card should provide a register-controlled reset to the rest of the logic on the card. That arrangement makes it possible for software to reset the devices on the PDS card whenever it restores power to the card.

Figure 4-6 Recommended inrush current limiting circuit

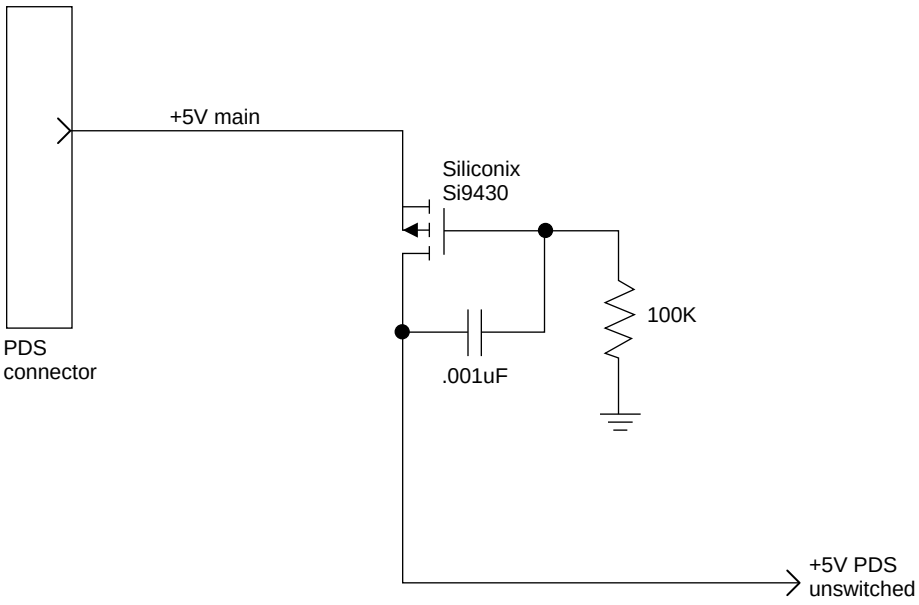
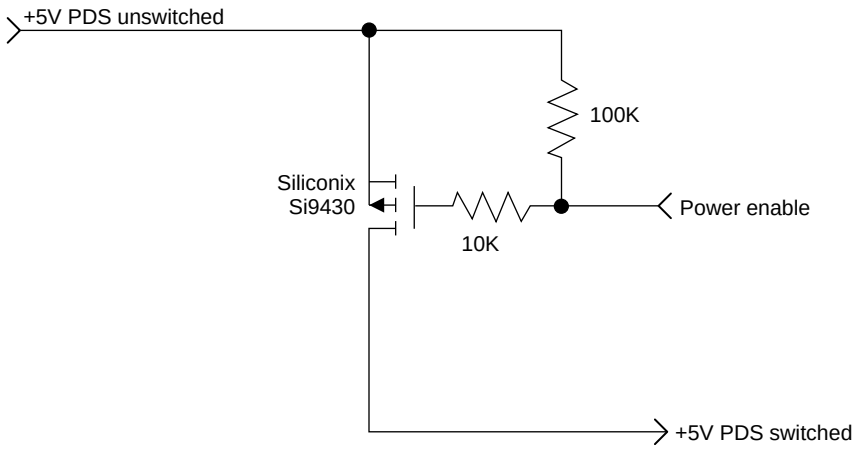


Figure 4-7 Recommended power control circuit



Logic Design Guidelines for the PDS Card

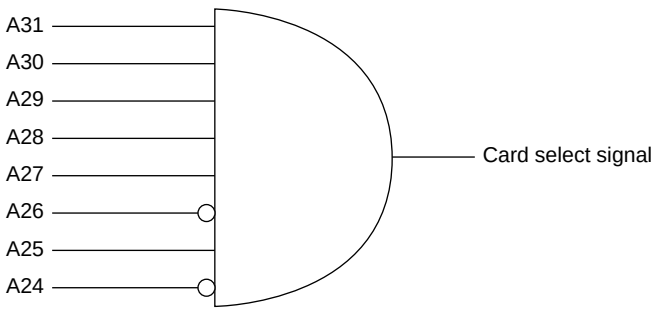
This section provides information about the operation of the logic circuits on a PDS expansion card for the PowerBook 520 and 540 computers.

Addressing Guidelines

The PDS card can only be addressed by 32-bit addressing; the card appears in the address space $8000 0000–$FFFF FFFF. To match the conventions used by the Slot Manager, software should address the card as if it were in slot space $A: either the 16 MB slot space $FA00 0000–$FAFF FFFF or the super slot space $A000 0000–$AFFF FFFF.

The expansion card must generate its own select signal from the address code signals on the connector. Figure 4-8 shows a typical logic circuit for generating the card select signal.

Figure 4-8 Generating the card select signal



Accessing Memory From the PDS Card

The PDS card in the PowerBook 520 and 540 computers reads and writes to memory in the same way as older PDS cards for the MC68030, as described in *Designing Cards and Drivers for the Macintosh Family*, third edition.

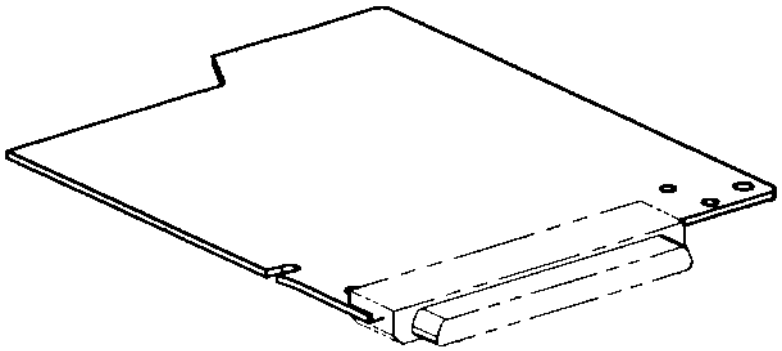
Timing Considerations

The PDS timing characteristics are identical to those of an MC68030 operating at a clock speed of 15.6677 MHz. See *Designing Cards and Drivers for the Macintosh Family,* third edition, for more information.

Mechanical Design Guidelines for the PDS Card

This section contains mechanical drawings showing the recommended design guidelines for PDS cards. Figure 4-12 shows the general appearance of the card with its connector.

Figure 4-9 The PDS card



PDS Card Shell

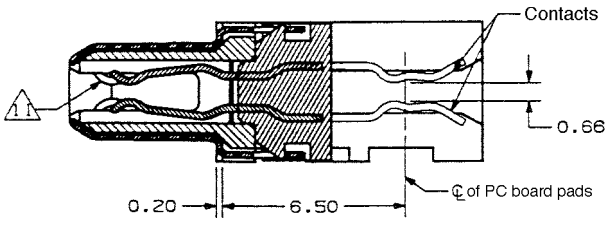
The PDS card is installed in a plastic shell that resembles a battery and fits into the battery well on the left side of the case. The plastic shell is made up of several components, including two clamshell halves, a bezel or cover, and an internal latch and spring. The shell fits into the computer's case with close tolerances; the cover fits into the opening of the battery well. Developers who wish to provide a PDS card can obtain shell components from Apple's vendor, subject to the usual restrictions. For information and authorization for obtaining the shell components, contact Apple's Developer Support Center.

PDS Card Connector

The PDS card connector is a 90-pin shielded connector. The connector does not sit on one face of the card, but straddles the edge and requires contact pads on both sides of the card. Figure 4-12 shows a section through the connector.

The PDS card connector is available as part number C-93-1817-54 from AMP, Inc. For a specification sheet or information about obtaining this connector, contact AMP at the address shown on page 40.

Figure 4-10 Section through the PDS card connector

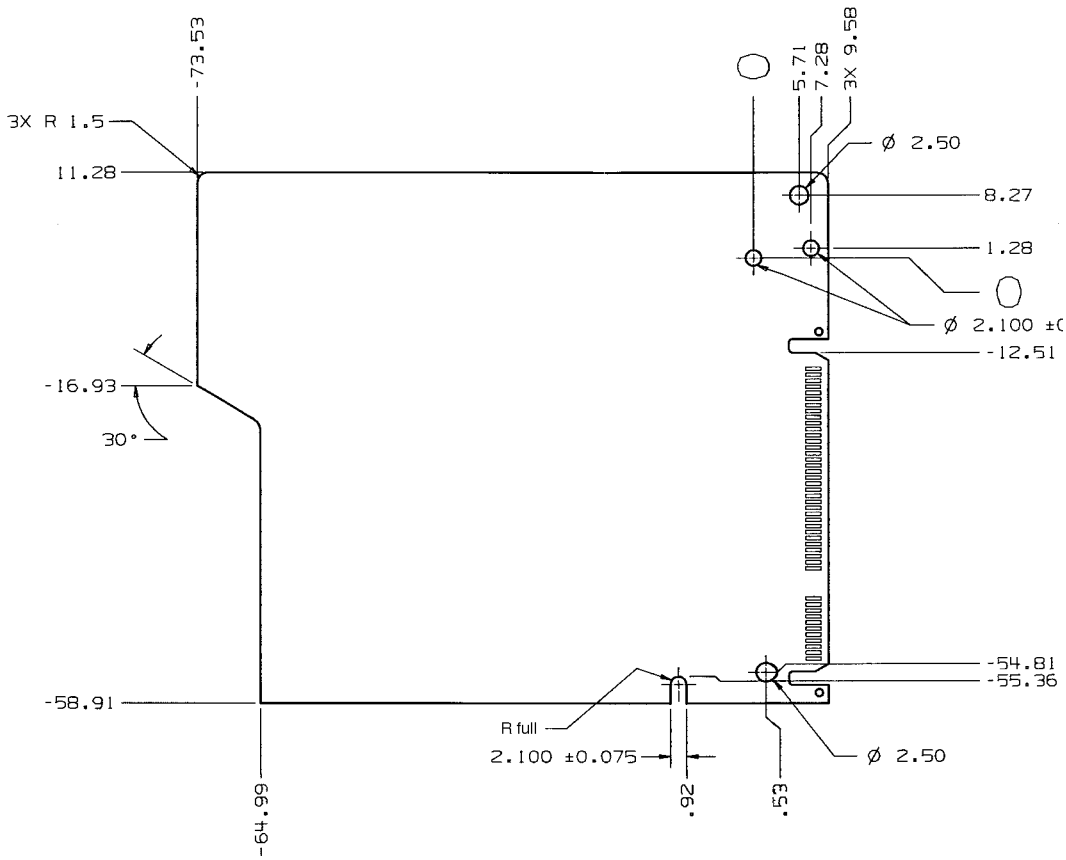


Note
The PDS connector is designed so that when the PDS card is inserted, the first connection is the ground by way of the connector shells, then the power pins make contact, and last of all the signal lines. The callout numbered 11 in Figure 4-12 indicates the long pins for the power connections. ◆

PDS Card Design

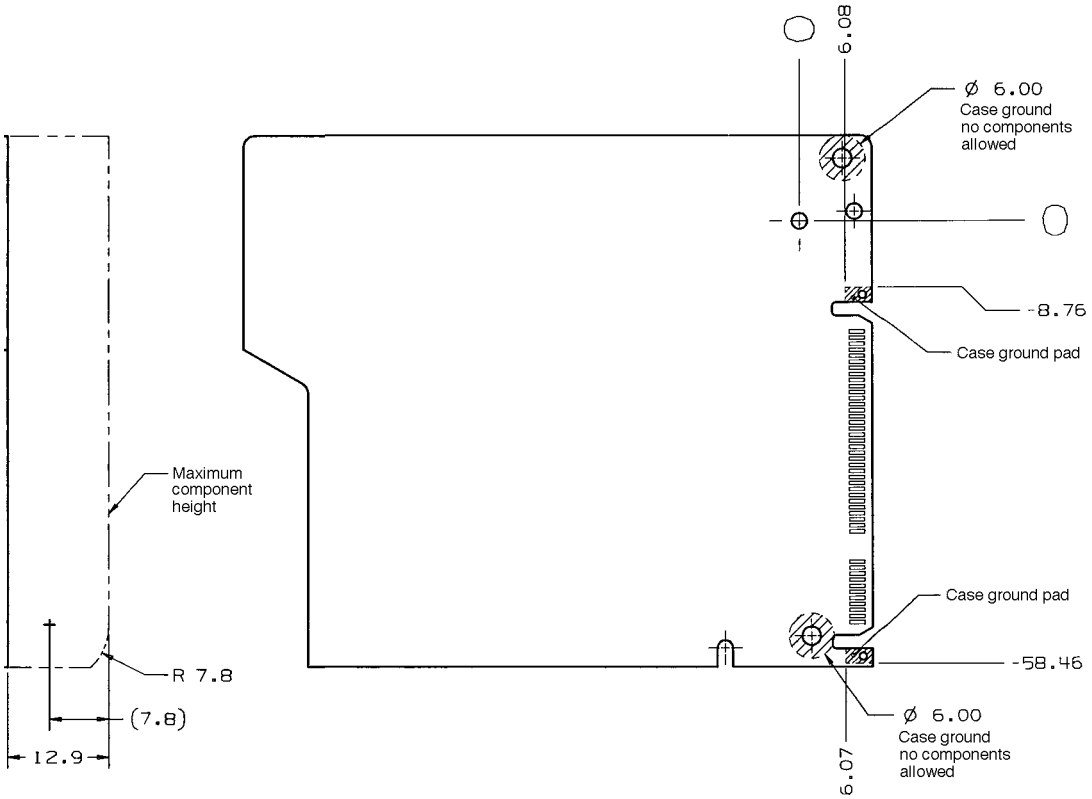
This section shows the mechanical outline and parts-placement guidelines for the PDS expansion card. Figure 4-12 shows the dimensions of the PDS expansion card. Figure 4-12 and Figure 4-13 show the restrictions on placement of components on the top and bottom of the card.

Figure 4-11 Dimensions of the PDS card



Note: Dimensions are in millimeters.

Figure 4-12 Restricted areas on the top of the PDS card



Note: Dimensions are in millimeters.

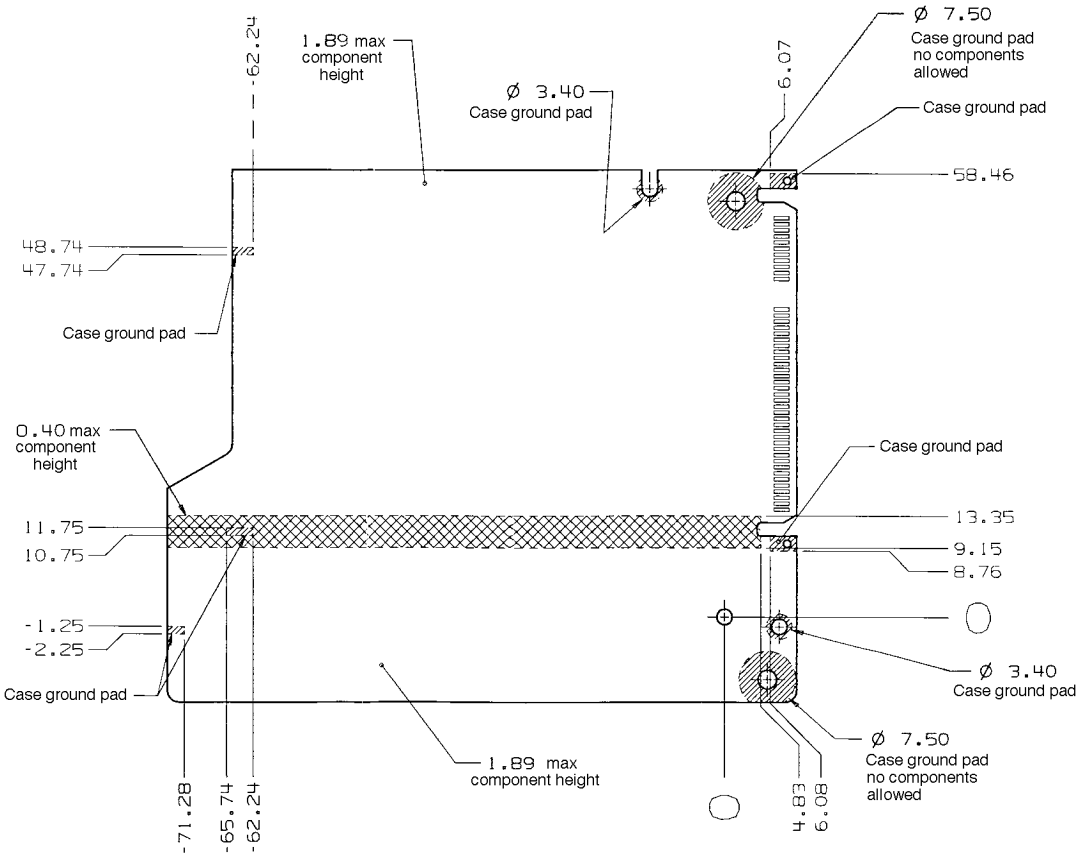
Note

The drawings in Figure 4-12, Figure 4-12, and Figure 4-13 are taken from Apple engineering drawing number 613-1706, revision 00. Developers who are members of Apple's Developer Program may contact Apple's Developer Support Center for information about possible later revisions of the drawings. ◆

▲ **WARNING**

Do not exceed the dimensions shown in the figures. Cards that exceed these specifications may damage the computer and may be incompatible with future PowerBook models. ▲

Figure 4-13 Restricted areas on bottom of the PDS card



Note: Dimensions are in millimeters.

Modem Expansion Card

The modem expansion slot in the PowerBook 520 and 540 computers is designed for a coprocessor card. The modem slot provides a superset of the PDS connector signals. Like the PDS card, a card installed in the modem slot can have a configuration ROM and emulate a NuBus card through the Slot Manager interface. The modem slot interrupt maps to NuBus slot $B.

To monitor sound through the computer's speaker, a modem card sends digitized sound data to the sound interface by way of the data bus and the Whitney IC. The Sound Manager has been modified to support such transfers of sound data.

The PowerBook Express Modem II, available in the U.S. and Canada, is a 14.4kbps fax/data modem; it supports the CCITT V.32bis and V.42bis standards as well as industry standard MNP5.

The PowerBook Express Modem II is designed around an Apple custom IC that includes the data pump, A-to-D and D-to-A converters, a microcontroller, and an interface to the computer's I/O bus. That custom IC is also available to other modem developers; contact Apple Developer Services for information about licensing.

The design of the PowerBook 520 and 540 computers incorporates a separate DAA (telephone line interface). The DAA provided with the PowerBook Express Modem II works with many different telephone services throughout the world. See the section "DAA Interface Card" on page 64.

Signals on the Modem Connector

The modem connector is a 100-pin connector and is described in the section "Modem Card Connector" on page 62. The signals on the modem connector are divided by function into four groups: MC68030-compatible signals, system support signals, DAA signals, and power. The next section shows the signal assignments on the connector. The following sections describe the first three groups of signals . For information about the power pins, see the section "Modem Power" on page 62.

Signal Assignments on the Modem Connector

Table 4-11 shows the signal assignments on the modem connector. Entries in the table are arranged the same way as the pins on the connector: pin 1 across from pin 2, and so on. Signal names that end with _L are active low.

Table 4-11 Signal assignments on the modem connector

Pin	Signal	Direction	Pin	Signal	Direction
1	ADDR[0]	B	2	GND	—
3	ADDR[2]	B	4	ADDR[1]	B
5	ADDR[4]	B	6	ADDR[3]	B
7	GND	—	8	ADDR[5]	B
9	ADDR[6]	B	10	ADDR[7]	B
11	ADDR[8]	B	12	ADDR[9]	B
13	ADDR[10]	B	14	V_5P_MAIN	—
15	ADDR[12]	B	16	ADDR[11]	B
17	ADDR[14]	B	18	ADDR[13]	B
19	GND	—	20	ADDR[15]	B
21	ADDR[16]	B	22	ADDR[17]	B
23	ADDR[18]	B	24	ADDR[19]	B
25	ADDR[20]	B	26	GND	—

continued

Table 4-11 Signal assignments on the modem connector (continued)

Pin	Signal	Direction	Pin	Signal	Direction
27	ADDR[22]	B	28	ADDR[21]	B
29	ADDR[24]	B	30	ADDR[23]	B
31	V_5P_MAIN	—	32	ADDR[25]	B
33	ADDR[26]	B	34	ADDR[27]	B
35	ADDR[28]	B	36	ADDR[29]	B
37	ADDR[30]	B	38	GND	—
39	RW_L	B	40	ADDR[31]	B
41	SIZ[0]	B	42	SIZ[1]	B
43	GND	—	44	DATA[17]	B
45	DATA[16]	B	46	DATA[19]	B
47	DATA[18]	B	48	DATA[21]	B
49	DATA[20]	—	50	V_5P_MAIN	B
51	DATA[22]	B	52	DATA[23]	B
53	DATA[24]	B	54	DATA[25]	B
55	GND	—	56	DATA[27]	B
57	DATA[26]	B	58	DATA[29]	B
59	DATA[28]	B	60	DATA[31]	B
61	DATA[30]	B	62	GND	—
63	DSACK_L[0]	B	64	DSACK_L[1]	B
65	AS_L	B	66	DS_L	B
67	V_5P_MAIN	—	68	BERR_L	OC
69	MDM_BR_L	I	70	MDM_BG_L	O
71	BGACK_L	B	72	MDM_BUSCLK	O
73	IPL_L[0]	O	74	GND	—
75	IPL_L[2]	O	76	IPL_L[1]	O
77	MDM_RESET_L	O	78	SLEEP	O
79	GND	—	80	IO_RESET_L	O
81	SND_SCLK	O	82	MDM_IRQ_L	I
83	PDS_POWER_SW	OC	84	n.c.	—
85	MDM_TDM[0]	B	86	MDM_TDM[1]	B
87	MDM_TDM[2]	B	88	MDM_TDM[3]	B

continued

Table 4-11 Signal assignments on the modem connector (continued)

Pin	Signal	Direction	Pin	Signal	Direction
89	GND	—	90	GND	—
91	MDM_DAA[0]	B	92	MDM_DAA[1]	B
93	MDM_DAA[2]	B	94	MDM_DAA[3]	B
95	MDM_DAA[4]	B	96	MDM_DAA[5]	B
97	MDM_RX_P	O	98	MDM_TX_P	I
99	MDM_RX_N	O	100	MDM_TX_N	I

NOTE Signal directions are defined from the main logic board side of the connector; I = input, O = output, B = bidirectional, and OC = open collector.

MC68030-Compatible Signals on the Modem Connector

Table 4-12 gives signal descriptions for the MC68030-compatible signals on the modem connector. Signal names that end with _L are active low.

Table 4-12 MC68030-compatible signals on the modem connector

Signal name	Description
ADDR[31–0]	System address bus; 32 bits, bidirectional.
AS_L	Indicates the occurrence of an active bus transaction; bidirectional.
BERR_L	System's bus error watchdog timer and the processor's /BERR line; open collector. This line is used for two operations: (1) to terminate an active bus transaction and (2) to cause a bus cycle to be retried when asserted with /HALT.
BGACK_L	Processor's /BGACK line; bidirectional. Indicates that an alternate bus master has control of the system bus. The alternate master drives this line after it has been granted the system bus and has started a bus transaction. A bus master must never assert BR_L and BGACK_L simultaneously, but rather should be designed to deassert MDM_BR_L and assert BGACK_L at the same time (an internal modem slot clock edge).
DATA[31–16]	System data bus; 16 bits, bidirectional.
DS_L	Processor's /DS signal; three state, output. This signal is used to manage the transmission of data to and from the processor. During a read cycle, the addressed bus slave places the requested data on the data bus when this signal is asserted. When this signal is asserted during a write, the processor drives the data bus with data to be written to the addressed bus slave. When this signal is released, the data bus is not driven.
DSACK_L[1–0]	Data strobe acknowledge. These signals perform two functions: (1) to indicate the completion of the data transfer portion of a bus transaction by the addressed slave and (2) to inform the bus master of the size of the data port being accessed during the current transaction.

continued

Table 4-12 MC68030-compatible signals on the modem connector (continued)

Signal name	Description
MDM_BG_L	Modem slot bus grant; output. This signal is driven by the system's bus arbitrator. When asserted low, the alternate bus master on the modem slot card may take ownership of the system bus after any pending bus traffic has completed (/BGACK, modem slot /CPU_AS signals, and modem slot /CPU_DSACK[1–0]—as appropriate to the card—have become inactive). MDM_BG_L will only be asserted after all pending bus traffic has been completed.
MDM_BR_L	Modem slot bus request; input. This signal is logical-ORed with the other alternate bus master's bus request signals to generate the CPU bus request. This signal is used to request the system's bus from the processor. An alternate bus master in the modem slot must never assert MDM_BR_L and BGACK_L simultaneously. The alternate bus master should be designed to deassert /BR and assert /BGACK at the same time (an internal modem slot clock edge). Refer to the Motorola MC68030 User's Manual for a complete description of this signal. This line is active low.
RW_L	Indicates the data direction for bus transfers; bidirectional.
SIZ[1–0]	Indicate the transfer size for a given bus transaction; bidirectional.

System Support Signals on the Modem Connector

Table 4-16 gives signal descriptions for the system support signals on the modem connector. Signal names that end with _L are active low.

Table 4-13 System support signals on the modem connector

Signal name	Description
IPL_L[2–0]	These outputs are driven by the interrupt prioritizer in the system and encode the interrupt level. These signals are not used by a modem; they are needed only when a card in the modem slot has an alternate CPU that must respond to interrupts.
MDM_BUSCLK	This output pin is connected to the system 15.6672 MHz bus clock and allows for designs based on the 68030 synchronous timing specifications.
MDM_INSERT_L	This input is grounded by the modem slot card. This allows the system to detect the insertion or removal of a modem slot card and take the appropriate action.
MDM_IRQ_L	This input, when asserted low, causes the modem slot card to interrupt the system. The modem driver can set the IPL level to either 2 or 3 by a call to the Internal Modem Manager.
MDM_POWER_SW_L	This input is similar in operation to the ADB power signal. When driven low, this signal causes the Power Manager to bring up the system. It can also be used as a ring detect; it is masked on the main logic board with the system is running.

continued

Table 4-13 System support signals on the modem connector (continued)

Signal name	Description
MDM_RESET_L	This output, when asserted low by the Power Manager, resets the modem slot card. (In addition to this signal, the modem card should have its own reset signal, controlled by a register in the bus interface logic, to reset the other devices on the card.)
MDM_TDM[3–0]	These lines are reserved for communication between the PDS slot and the modem slot. Contact Apple's Developer Support Center for information about their use.
SLEEP	This output, when asserted high by the Power Manager, indicates that the system is in sleep mode.
SND_SCLK	This output is a 22.5792 kHz sound clock generated by the Singer IC and used for transferring digital sound signals to the codec on the modem card.

Note

Developers who write their own modem driver software should make calls to the modem hardware by way of the Internal Modem Manager, which is part of the system software. For information about the Internal Modem Manager, contact Apple's Developer Support Center. ◆

DAA Signals on the Modem Connector

Table 4-16 lists the signals connecting the modem slot with the DAA slot. These signals are used only by the modem card and the DAA card. For more information, see the section "Descriptions of the DAA Signals" on page 65.

Table 4-14 DAA signals on the modem connector

Signal name	Description
MDM_DAA[5–0]	These lines are multipurpose I/O digital signal lines operating at CMOS logic levels. They are the control and status signals between the modem card and the DAA card.
MDM_TX_P, MDM_TX_N	These lines are a differential pair of analog signals for transmitting analog signals from the modem card to the DAA card.
MDM_RX_P, MDM_RX_N	These lines are a differential pair of analog signals for receving analog signals sent from the DAA card to the modem card.

Modem Power

The power pins on the modem connector are V_5P_MAIN, the 5-volt supply line, and GND, the ground. The voltage on V_5P_MAIN is 5.0V ± 5%. The maximum current available for the modem card is

- 200 mA in normal operation
- 2 mA in sleep mode
- 0.00 mA in shutdown mode

Physical Design Guidelines for Modem Card

This section contains mechanical drawings showing the recommended design guidelines for the modem card.

Modem Card Connector

The modem card is connected to the main logic board by means of a 100-pin connector available from MOLEX, Inc. (part number 53395-1000). For a specification sheet and information about obtaining the connector, contact

MOLEX, Inc.
194 S. Hillview Dr.
Milpitas, CA 95035
408-946-4700
AppleLink: MOLEX

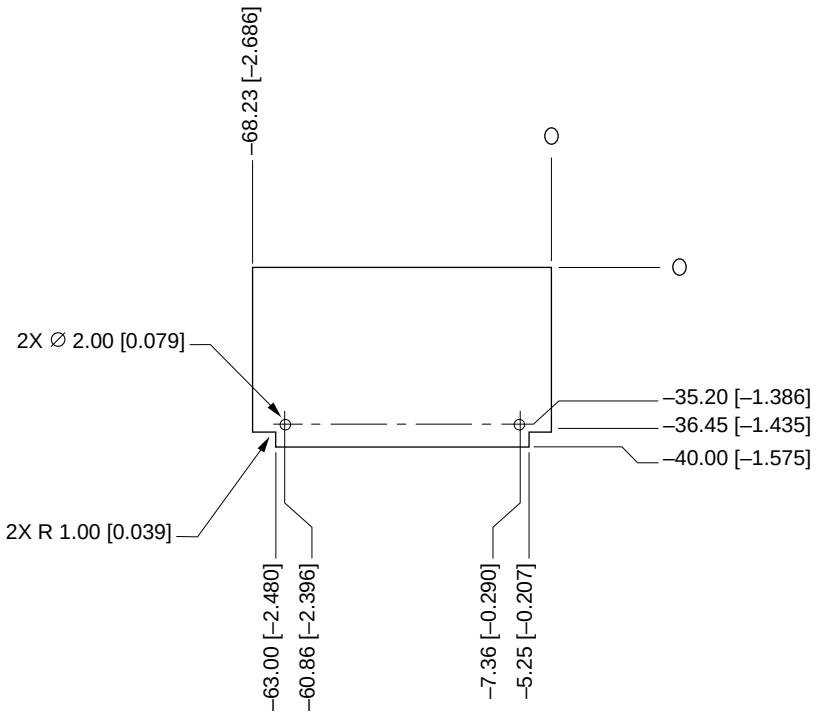
Design of the Modem Card

Figure 4-14 is a top view of the modem expansion card showing its dimensions. Figure 4-15 and Figure 4-16 are top and bottom views showing the restrictions on the placement of components and traces.

Note
The drawings in Figure 4-14, Figure 4-15, and Figure 4-16 are taken from Apple engineering drawing number 056-0104, revision 05. Developers who are members of Apple's Developer Program may contact Apple's Developer Support Center for information about possible later revisions of the drawings. ◆

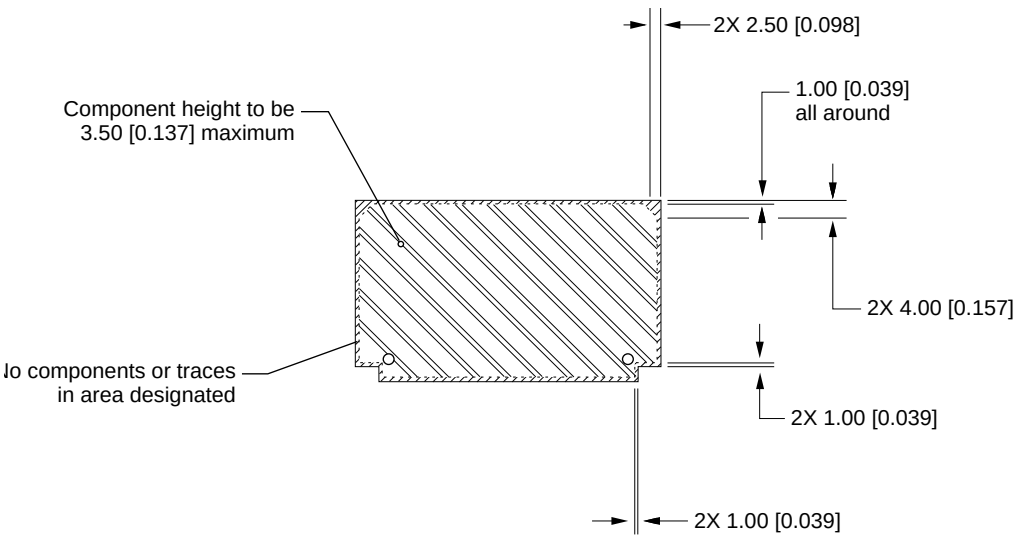
▲ **WARNING**
Do not exceed the dimensions shown in the mechanical drawings. Cards that exceed these specifications may damage the computer and may be incompatible with future PowerBook models. ▲

Figure 4-14 Top view of the modem card



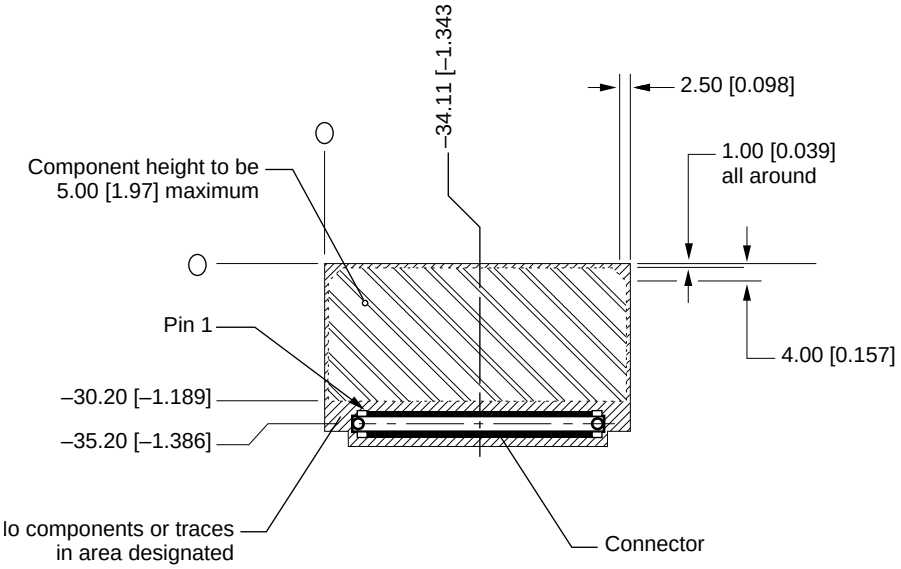
Note: Dimensions are in millimeters [inches].

Figure 4-15 Restricted areas on the top of the modem card



Note: Dimensions are in millimeters [inches].

Figure 4-16 Restricted areas on the bottom of the modem card



Note: Dimensions are in millimeters [inches].

DAA Interface Card

The modem architecture in the PowerBook 520 and 540 computers is unlike most others in that the telephone line interface unit, also called the DAA (data access arrangement), is separate from the rest of the modem's controller and data pump. This architecture makes it possible to reduce the length of the high-speed buses and still locate the telephone line socket at the rear of the computer's case.

In addition to describing the signals to the DAA card, this section describes a typical DAA card and then describes the multi-country DAA card that Apple includes with the PowerBook Express Modem II.

Signals on the DAA Connector

The DAA card plugs into a connector on the main logic board and the signal lines from that connector are routed to the modem-card connector. Except for its power leads, the DAA card is not connected to any other part of the computer.

Signal Assignments on the DAA Connector

Table 4-15 lists the signals on the DAA connector. Entries in the table are arranged the same way as the pins on the connector: pin 1 across from pin 2 and pin19 across from pin 20. These signals are used only by the modem card and the DAA card.

Table 4-15 Signals on the DAA connector

Pin	Signal name	Direction	Pin	Signal name	Direction
1	GND	—	2	GND	—
3	MDM_TX_P	O	4	MDM_RX_P	I
5	MDM_TX_N	O	6	MDM_RX_N	I
7	GND	—	8	GND	—
9	MDM_DAA[1]/ H_UKNUM	O	10	MDM_DAA[0]/ H_AWSMDAT	O
11	MDM_DAA[3]/ H_OHRC	O	12	MDM_DAA[2]/ L_RINGREADY	I
13	MDM_DAA[5]/ H_AWSMCLK	O	14	MDM_DAA[4]/ L_PHONE	I
15	GND	—	16	GND	—
17	n.c.	—	18	n.c.	—
19	MAIN_5V	—	20	MAIN_5V	—

Signal directions are defined from the main logic board side of the connector; I = input and O = output.

Descriptions of the DAA Signals

Table 4-16 gives descriptions of the signals on the DAA connector. All these signals are connected to the modem connector.

Table 4-16 Signals on the DAA connector

Signal name	Description
MDM_DAA[5–0]	These lines are multipurpose I/O digital signal lines operating at CMOS logic levels. They are the control and status signals between the modem card and the DAA card.
MDM_RX_P, MDM_RX_N	These lines are a differential pair of analog signals for receving analog signals sent from the DAA card to the modem card. Impedance is 47K in parallel with 1.3 Henries.
MDM_TX_P, MDM_TX_N	These lines are a differential pair of analog signals for transmitting analog signals from the modem card to the DAA card. Impedance is 47K in parallel with 1.3 Henries.

The signals MDM_DAA[5-0] are multipurpose I/O digital control and status lines that operate at CMOS logic levels. The signals MDM_TX_P, MDM_TX_N, MDM_RX_P, and MDM_RX_N are two differential pairs for receiving and transmitting analog signals between the modem and the DAA. The nominal impedance of each differential pair is 47 K in parallel with a transformer winding that has an inductance of 1.3 Henries.

Physical Design of the DAA Card

This section contains information about two versions of the DAA card: a simple DAA such as might be used in a single country, and Apple's multi-country DAA card.

DAA Connector

The DAA card connector is a 20-pin SMT-type dual-row receptable header manufactured by AMP Incorporated, Harrisburg, PA 17105. The connector on the main logic board is AMP part number 5-174904-0; the mating connector on the DAA board is AMP part number 5-174645-0. For specification sheets or information about obtaining these connectors, contact AMP at the address on page 40.

Design of a Simple DAA Card

Figure 4-17 shows the appearance of a simple DAA expansion card. Figure 4-18 and Figure 4-19 are top and bottom views showing the restrictions on the placement of components and traces.

Figure 4-17 A simple DAA card

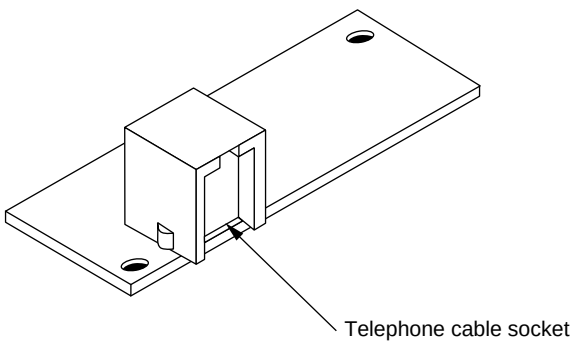
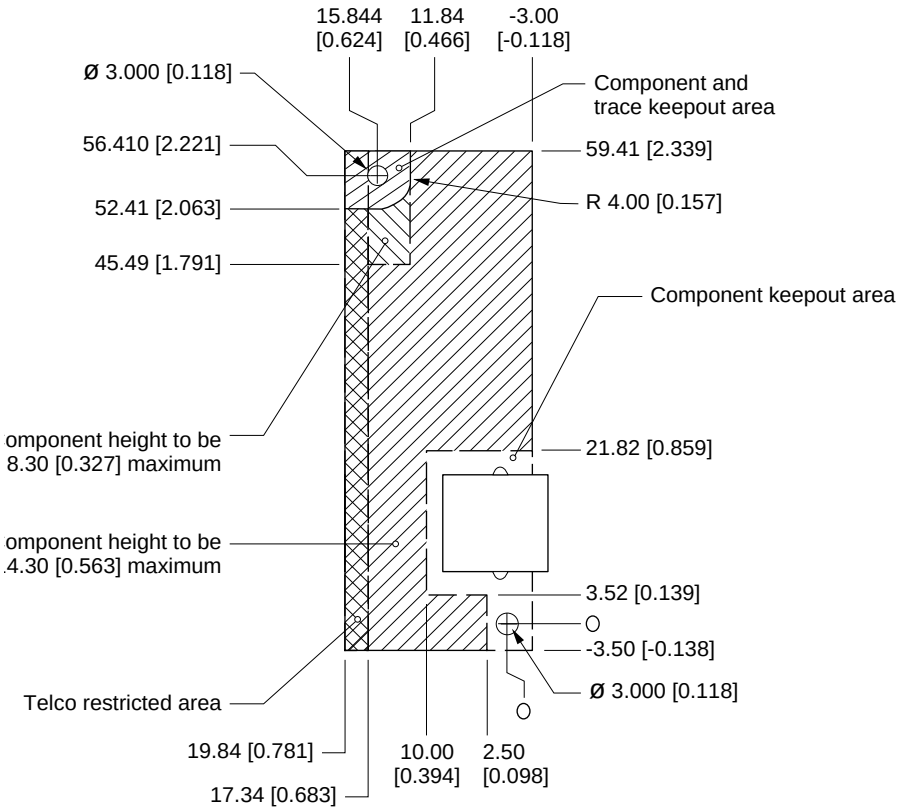


Figure 4-18 Top view of a simple DAA card



Note: Dimensions are in millimeters [inches].

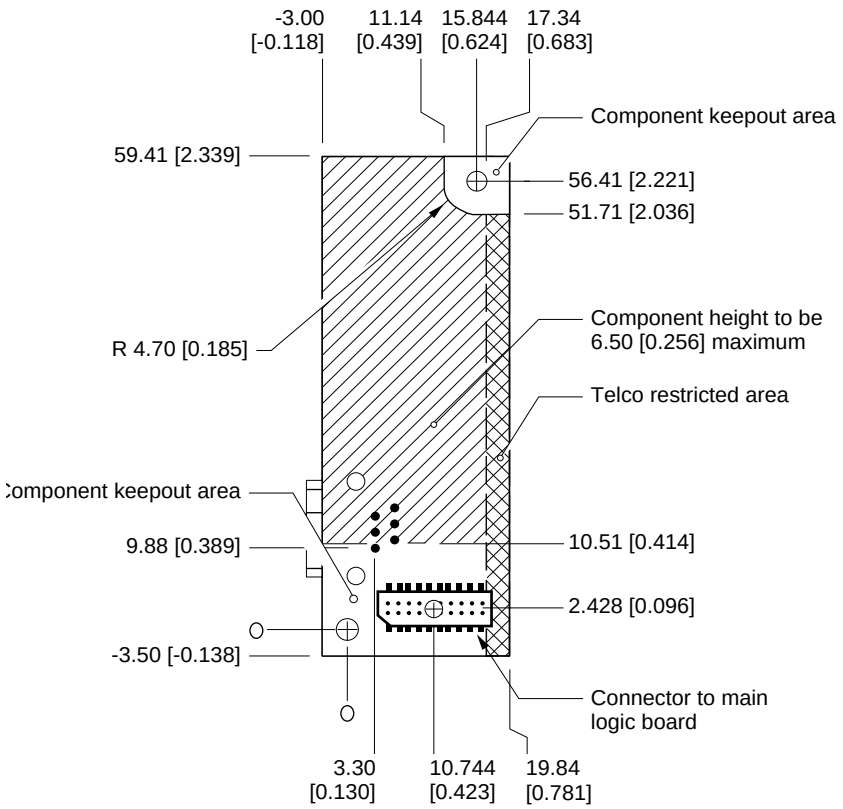
Note

The drawings in Figure 4-18 and Figure 4-19 are taken from Apple engineering drawing number 062-1227, revision 00. Developers who are members of Apple's Developer Program may contact Apple's Developer Support Center for information about possible later revisions of the drawings. ◆

▲ **WARNING**

Do not exceed the dimensions shown in the mechanical drawings. Cards that exceed these specifications may damage the computer and may be incompatible with future PowerBook models. ▲

Figure 4-19 Bottom view of a simple DAA card



Note: Dimensions are in millimeters [inches].

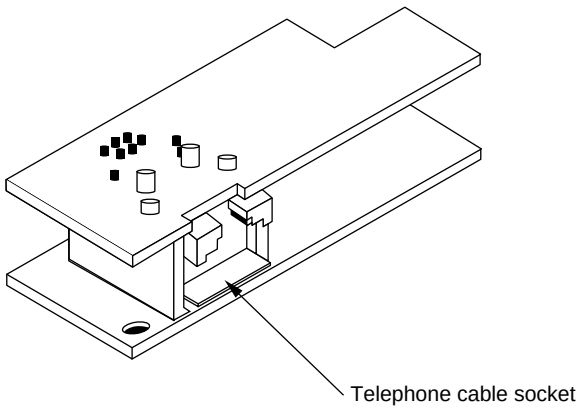
The Multi-Country DAA Card

The multi-country DAA is included with the PowerBook Express Modem II. The multi-country DAA has a special telephone line socket that includes additional contacts. Those contacts are used with different adapter cables, called country keys, that allow the computer to identify the country where the it is being used and to reconfigure the DAA interface accordingly.

The multi-country DAA, called Submarine, uses a custom chip set that make it possible to create a single system that works with many different telephone systems. The multi-country DAA is also available for use with modems from other developers; for information about licensing it, contact Apple Developer Services.

Figure 4-20 shows the appearance of the multi-country DAA card. It fits into the same space as the simple DAA card.

Figure 4-20 View of the multi-country DAA card



CHAPTER 5

Software

This chapter describes the new ROM and system software features of the PowerBook 520 and 540 computers.

ROM Software

The ROM software in the PowerBook 520 and 540 computers is based on the ROM used in previous PowerBook computers, with enhancements to support the many new features of these computers. Some of the features this ROM was designed to support include the following:

- MC68040 and MC68LC040 microprocessors
- Pratt memory controller
- new Power Manager software
- new display controller
- new sound features
- Ethernet
- PDS support
- function keys
- smart batteries
- trackpad

The following sections describe each of these features.

MC68040 and MC68LC040 Microprocessors

The MC68LC040 microprocessors used in the PowerBook 520 and 540 computers differ from the MC68030 used in earlier PowerBook models in important ways such as the sizes of the caches and the configurations of the control registers. The differences make it necessary to use a different power cycling scheme on the PowerBook 520 and 540 computers.

The PowerBook 520 and 540 computers do not provide the economode reduced speed feature found on the PowerBook 160 and 180 models.

Memory Controller Software

The memory control routines have been rewritten to operate with the Pratt IC, which has a control register configuration different from that of the memory controller used in earlier PowerBook models. The memory initialization and size code has been rewritten to deal with

- larger ROM size
- new type of DRAM devices
- new memory configurations

Power Manager Software

Changes to the Power Manager software include

- power cycling and sleep mode for the MC68040 microprocessor
- support for the new smart batteries
- support for turning on and off power to the Ethernet interface

In addition to those changes, the PowerBook 520 and 540 computers include a new public API for power management; it is described in Chapter 6, “Power Manager Interface.”

Display Controller Software

The PowerBook 520 and 540 computers have a new custom IC, the CSC (color support chip), that provides the data and control interface to the flat panel display. The ROM software includes new video drivers for that IC.

The new drivers also support a wider range of external video monitors. See the section “Video Monitors” beginning on page 25 for information about external video.

Sound Features

The ROM software includes new sound driver software to support the new Sound Manager, which is part of the system software. The new driver software also supports the following new features:

- improved sound performance by way of a new interface to the Singer sound IC
- support for 16-bit stereo sound input
- support for automatic gain control in software
- mixing of sound output from the modem

The new ROM software also includes routines to arbitrate the control of the sound hardware between the modem and the Sound Manager.

Ethernet Driver

The driver for the Ethernet interface can now put a sleep task for Ethernet into the Power Manager’s sleep table. This sleep task first makes a control call to the Ethernet driver to prepare the Ethernet interface IC for sleep mode. The sleep task then makes a Power Manager call to turn off power to the IC. The sleep task installs a corresponding wake task that turns the interface power back on and reinitializes the interface IC.

PDS Support Software

As in the desktop Macintosh models, the PDS expansion card is treated like a NuBus expansion slot. The Slot Manager software looks for a configuration ROM on the PDS card and then treats the card like a NuBus expansion card. PDS cards with configuration ROM that conforms to the requirements described in *Designing Cards and Drivers for the Macintosh Family,* third edition, will work with the existing Slot Manager. The Slot Manager used with the PowerBook 520 and 540 computers is the same as that used in previous PowerBook models.

The interrrupt from the PDS slot maps to an interrupt from expansion slot $A.

Support for Function Keys

The keyboards on the PowerBook 520 and 540 computers have a row of 12 function keys across the top. Except for the function keys, the keyboards are similar to those on previous PowerBook models. The function keys are added to the key matrix in the same way as the function keys on the Apple Extended Keyboard and return the same key codes.

Intelligent Battery Support

The Power Manager IC communicates with the processors in the PowerBook Intelligent Batteries by means of a serial interface. The Power Manager's command set has been expanded to provide system access to the data from the batteries.

Trackpad Software

The trackpad hardware, the Power Manager IC, and the system software work together to translate the movements of a finger across the surface of the trackpad into cursor movements.

The control registers for the trackpad hardware are part of the Power Manager IC. The Power Manager's software takes the raw data from the trackpad hardware and converts it to the same format as ADB mouse data before sending it on to the system software.

The ADB software that supports the trackpad includes the Cursor Device Manager, which provides a standard interface for a variety of devices. The ADB software checks to see whether a device connected to the ADB port is able to use the Cursor Device Manager. For more information, see the January 1994 revision of Technical Note HW 01, *ADB—The Untold Story: Space Aliens Ate My Mouse.*

System Software

The PowerBook 520 and 540 computers are shipped with system software version 7.1. A new system enabler file is required in order to run System 7.1 on the PowerBook 520 and 540 computers.

Identifying the PowerBook 520 and 540 Computers

The correct method for software to use to identify the Macintosh model it is running on is by using the Gestalt Manager routines. The `gestaltMachineType` value returned by the PowerBook 520 and 540 computers is 72 (hexadecimal $48). *Inside Macintosh: Overview* describes the Gestalt Manager and tells how to use the `gestaltMachineType` value to obtain the machine name string.

New System Enabler

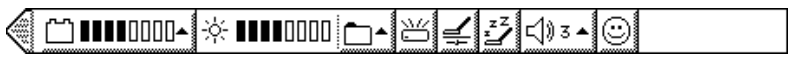
The PowerBook 520 and 540 computers use a new system enabler. Many of the patches that were in the system enabler for earlier PowerBook models have been incorporated into the ROM. The new system enabler includes

- the new Sound Manager
- the sound input driver
- the sound output sifter
- the new Display Manager

Control Strip

The desktop on the PowerBook 520 and 540 computers has a new status and control element called the control strip. It is a strip of graphics with small button controls and indicators in the form of various icons. Figure 5-1 shows the control strip.

Figure 5-1 Control strip



The control strip is a control panel that provides the operating environment for control strip modules. It runs on any Macintosh computer with System 7.0 or later.

The control strip is implemented in a private layer that appears in front of the windows in all the application layers so that the windows will not obscure it. The user can move the window for the control strip to any location on the display as long as the right or left edge of the strip is attached to the right or left edge of the display.

The control strip has a tab on its unattached end. The user can drag the tab to adjust the length of the strip or hold down the Option key and drag the tab to move the strip to a new position. The user can hide the control strip, except for the tab, by clicking the tab. Clicking the tab when the control strip is hidden makes the control strip visible again. To make the control strip disappear completely, the user can click the Hide button in the Control Strip control panel, described on page 80.

The different parts of the control strip either display status information or act as buttons. When the user clicks a button, it is highlighted; some buttons also display additional elements such as pop-up menus.

By holding down the Option key and clicking a display area, the user can drag the display area to another position in the control strip. After the user rearranges the parts of the control strip, the new arrangement is saved when the computer is shut down and restarted.

The control strip software provides a standard screen location for a collection of individual modules that provide status and control functions. The control strip functions include

- AppleTalk Switch: shows whether AppleTalk is on or off and lets the user turn AppleTalk on or off without having to open the Chooser.
- Battery Monitor: displays the status of the battery or batteries.
- File Sharing: displays the state of file sharing (on, off, or users connected), lets the user turn file sharing on or off, and lets the user open the Sharing Setup control panel.
- HD Spin Down: shows whether the internal hard disk is on or off; lets the user turn off the hard disk.
- Power Settings: lets the user select between maximum conservation or maximum performance without opening the PowerBook control panel; also lets the user open the PowerBook control panel.
- Sleep Now: puts the computer into sleep mode.
- Sound Volume: lets the user select the sound volume.
- Video Mirroring: lets the user turn video mirroring on or off if an external monitor is connected.

Note
Several of the functions of the control strip were implemented in separate control panels on earlier PowerBook models. ◆

Developers can add modules to the control strip. For information, see the section "Adding Control Strip Modules" beginning on page 80.

Selecting SVGA

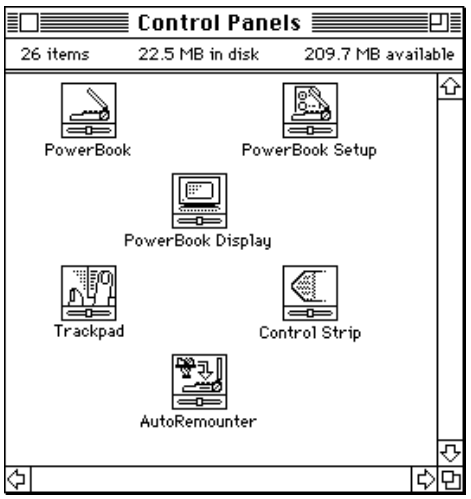
An SVGA monitor is considered to be a member of the VGA monitor family. To work with the PowerBook 520 and 540 computers, an SVGA monitor must be capable of operating as a standard VGA monitor. With the monitor operating in VGA mode, the user selects the Monitors control panel and clicks the Options button. The user then selects SVGA in the Monitor types list shown in the Options dialog box.

Similarly, the PowerBook 520 and 540 computers treat the different scanning modes available on multiscan monitors as a family of settings from which the user can select by means of the Options dialog box in the Monitors control panel.

New Control Panels

Several control panels are new or revised for the PowerBook 520 and 540 computers. Figure 5-2 shows the appearance of the new control panel icons.

Figure 5-2 New control panels



The new control panels are

- PowerBook control panel
- PowerBook Setup control panel
- PowerBook Display control panel
- Trackpad control panel
- Control Strip control panel
- AutoRemounter control panel

The following sections describe the new control panels.

PowerBook Control Panel

The PowerBook control panel includes several controls that allow the user to balance performance against battery conservation. Figure 5-3 and Figure 5-4 show the PowerBook control panel in two modes: easy, which allows the user to balance performance against battery life with a single slider, and custom, which gives the user control of individual power conservation features.

Figure 5-3 PowerBook control panel in easy mode

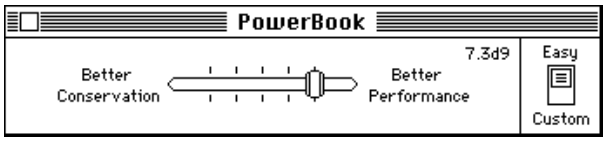
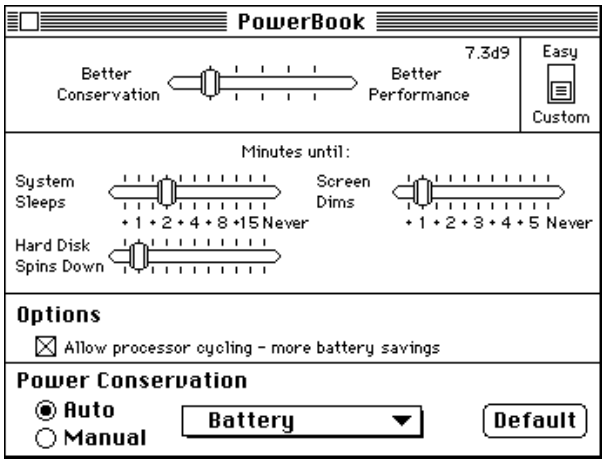


Figure 5-4 PowerBook control panel in custom mode

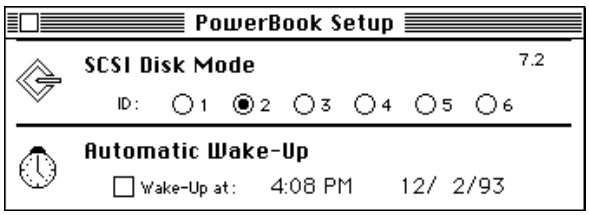


In custom mode, the PowerBook control panel lets the user set the time delays in minutes until the computer dims the display, goes into sleep mode, and goes into shutdown mode. The user can also select processor cycling, which slows down the processor and saves power.

PowerBook Setup Control Panel

Figure 5-5 shows the PowerBook Setup control panel.

Figure 5-5 PowerBook Setup control panel

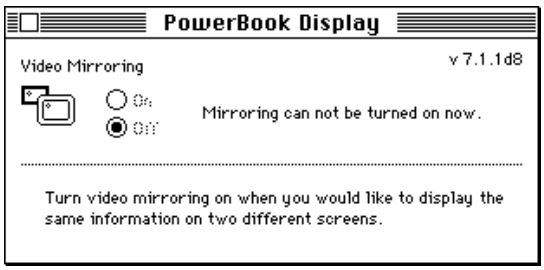


The PowerBook Setup control panel allows the user to set the SCSI ID value for the internal hard disk so that the PowerBook 520 and 540 computers can be used as a hard disk connected to another computer. The control panel also allows the user to program an automatic wake-up time.

PowerBook Display Control Panel

Figure 5-6 shows the PowerBook Display control panel.

Figure 5-6 PowerBook Display control panel

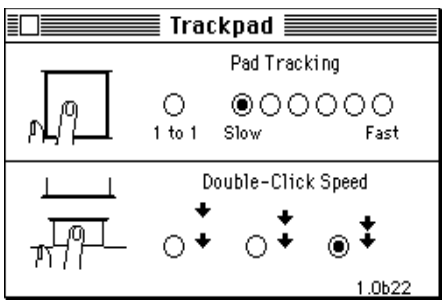


The PowerBook Display control panel allows the user with an external video monitor to select video mirroring mode. In normal operation, an external monitor provides additional desktop space, as on any Macintosh computer with multiple displays. When the user selects video mirroring mode, the built-in display is duplicated or mirrored on the external video monitor. For example, mirroring mode is useful for presentations when the user wishes to see the same display that the audience sees on the video monitor.

Trackpad Control Panel

Figure 5-7 shows the Trackpad control panel.

Figure 5-7 Trackpad control panel



The Trackpad control panel allows the user to set the characteristics of the trackpad to match the way the user operates it. The user can set the scale and sensitivity of the trackpad and the time interval for a double click.

Control Strip Control Panel

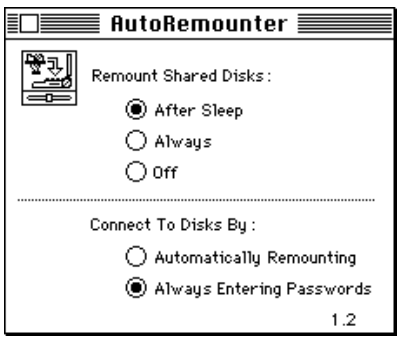
Figure 5-8 shows the Control Strip control panel. The user can hide or show the control strip by clicking the corresponding button in the control panel.

Figure 5-8 Control Strip control panel

AutoRemounter Control Panel

Figure 5-9 shows the AutoRemounter control panel.

Figure 5-9 AutoRemounter control panel



When the user has connected a PowerBook 520 or 540 computer to another computer to use its internal hard disk as a shared disk, the AutoRemounter control panel allows the user to determine how the PowerBook computer operates when it resumes operation after a period in sleep mode.

Adding Control Strip Modules

The control strip is implemented in software as a shell with individual control and status modules added. The control strip software draws the strip that acts as the background for the individual modules. Each module is responsible for drawing the icons and other objects that make up its user interface.

Contents of Module Files

The only required resource in a module file is a resource containing the code necessary for the module to interact with the control strip. A module file may contain more than one code resource if it is to provide multifunctional support. In that case, each module in the file is loaded and initialized separately and treated as an independent entity.

If a file contains only a single code resource, the resource may be unnamed, and the module will be referenced by its filename. If more than one module is contained within a module file, each module is required to have a unique name describing its functionality.

All other resources in a module file are optional, but there are several that are recommended in order to support a custom icon and version information. The recommended resources are

- `'BNDL'`
- `'FREF'`
- `'ICN#'`, `'icl4'`, `'icl8'`, `'ics#'`, `'ics4'`, `'ics8'`
- signature resource (same type as file's creator)
- `'vers'`, ID=1

Developers should confine their resources to the range 256–32767.

Module Interface

The module's interface to the control strip consists of a code resource of type `'sdev'`. This code is responsible for performing all of the functions required by the control strip (see below) as well as any functions that are custom to the module itself. The module's entry point is at the beginning of the resource and is defined as

```
pascal long ControlStripModule(long message,
                               long params,
                               Rect *statusRect,
                               GrafPtr statusPort);
```

Interactions between a module and the control strip are managed by passing messages to the module to tell it what to do or to obtain information about the module and its capabilities. Each module is required to observe Pascal register saving conventions. A module may trash registers D0, D1, D2, A0, and A1, but must preserve all other registers across its call.

Field descriptions

`message` A message number, from the list in the section "Control Strip Module Messages", that tells the module what action to perform.

`params` This is the result returned by the initialize call to the module. This would typically be the handle to the module's private variables since modules can't have global variables. It will be passed to the module on all subsequent calls.

`statusRect`	A pointer to a rectangle defining the area that a module may draw within.
`statusPort`	A pointer to the control strip's graphics port. This will be either a color or black-and-white graphics port depending on which PowerBook Model the control strip is running on.

The result value returned by the module will vary depending on the message sent to it. Results for each message are described in the sections on the individual messages.

Module Reentrancy

Any module that makes calls to routines such as `GetNextEvent`, `ModalDialog` or `pop-upMenuSelect` should assume that it could be called reentrantly; that is, the module could be called again while the initial call is still in progress. Situations to avoid are such things as reusing a single parameter block for multiple calls, or indiscriminately locking and unlocking your global variables around the module's invocation.

Instead of using a single parameter block, it's better, if possible, to allocate the parameter block on the stack. In the case of asynchronous calls, using the stack could cause problems; in that case, preventing the block's reuse should be sufficient.

If you need to lock and unlock your global variables, it's better to use `HGetState` and `HLock` at the beginning of the call, and `HSetState` at the end, so that the state is restored to what it was on entry.

Control Strip Module Reference

Control strip modules interact with the control strip software in three ways: by accepting messages, by calling utility routines, and by calling `Gestalt` seletors. The next three sections describe each of those interactions.

Control Strip Module Messages

All control strip modules must respond to messages from the control strip. The following messages have been defined:

Message name	Message number	Description
`sdevInitModule`	0	Initialize the module
`sdevCloseModule`	1	Clean up before being closed
`sdevFeatures`	2	Return the feature bits
`sdevGetDisplayWidth`	3	Return the width of the module's display
`sdevPeriodicTickle`	4	Periodic tickle when nothing else is happening
`sdevDrawStatus`	5	Update the interface in the control strip

Message name	Message number	Description
sdevMouseClick	6	User has clicked on the module's display area
sdevSaveSettings	7	Save any changed settings in thee module's preferences file
sdevShowBalloonHelp	8	Display a help balloon, if the module has one

sdevInitModule

The sdevInitModule message is the first message sent to a module after the module has been loaded from its file. Initialization allows the module to initialize its variables and to determine whether it can run on a particular machine: for example, if the module's function is to display battery information it can run only on a PowerBook.

The module needs to load and detach any resources in the module's resource file that will be used, because the resource file will not be kept permanently open. What that means is that your code can't use GetResource() or the like to retrieve the handle to one of the module's resources on a subsequent call. Typically you would allocate space in your global variables for handles to those detached resources.

The sdevInitModule message returns a result depending on its success at installing itself. A positive result (≥0) indicates successful installation. This result value will be passed to the module on all subsequent calls. A negative result indicates an error condition, and installation of the module is aborted by the control strip software. The module will not receive a close message when installation has been aborted.

sdevCloseModule

The sDevCloseModule message is sent to a module when it should be closed. Typically the module itself will decide when this ought to happen. When the module receives this message, it should dispose of all the detached resources it loaded as well as its global storage. No result is expected.

sdevFeatures

The sdevFeatures message queries the module for the features it supports. It returns as its result a bitmap consisting of 1 bits for supported features and 0 bits for unsupported features. All undefined bits are reserved by Apple for future features, and must be set to 0. The bits are defined as

sdevWantMouseClicks 0 If this bit is set, the control strip will notify the module of mouse down events. If this bit is not set, the control

		strip assumes that the module only displays status information with no user interaction.
`sdevDontAutoTrack`	1	If this bit is set, the control strip highlights the module's display and then calls the module to perform mouse tracking; this bit is usually set when, for example, a module has a pop-up menu associated with it. If this bit is cleared, the control strip tracks the cursor until the mouse button is released, then sends an `sdevMouseClick` message to the module to notify it that there was a mouse-down event.
`sdevHasCustomHelp`	2	If this bit is set, the module is responsible for displaying its own help messages, which can be customized depending on its current state. If the bit is cleared, the control strip will display a generic help message when the cursor passes over the module's display area and Balloon Help is on.
`sdevKeepModuleLocked`	3	If this bit is set, the module's code will be kept locked in the heap. This bit should be set only if the module is passing the address of one of its routines to the outside world (for example, installing itself in a queue).

sdevGetDisplayWidth

The `sdevGetDisplayWidth` message is sent to a module to determine how much horizontal space (in pixels) its display currently requires on the control strip. The module should return the number of pixels as its result. The returned width should not be the maximum width it requires for any configuration, but should reflect how much space it currently requires, because it's possible for a module to request that its display be resized.

IMPORTANT
You should be conservative in your use of control strip display space, which is limited. Because several modules could be requesting space, it's possible that your module could be shoved off the end. ▲

sdevPeriodicTickle

The `sdevPeriodicTickle` message is passed to the module periodically to allow the module to update its display due to changes in its state. You should not assume any minimum or maximum interval between tickles. The module should return, as its result, some bits that signal requests for actions from the control strip software. All undefined

bits in the result are reserved for future use by Apple and must be set to 0. The bits are defined as

`sdevResizeDisplay`	0	If this bit is set, the module needs to resize its display. The control strip sends a `sdevGetDisplayWidth` message to the module and then update the control strip on the display.
`sdevNeedToSave`	1	If this bit is set, the module needs to save changed settings to disk. The control strip software will mark the request but may defer the actual save operation to a better time (for example, when the hard disk is spinning).
`sdevHelpStateChange`	2	If this bit is set, the module's state has changed so it needs to update its help message. If a help balloon is being displayed for this module, the control strip software will remove it and put up a new help balloon for the current state.
`sdevCloseNow`	3	If this bit is set, the module is requesting to be closed. The control strip software will call the module to save its settings, then call it again to close itself.

sdevDrawStatus

The `sdevDrawStatus` message indicates that the module has to redraw its display to reflect the most recent state. This message is typically sent when the user clicks on the module's display area, when any of the module's displays is resized, or when the control strip itself needs to be updated, perhaps in response to a screen saver deactivation.

The `statusRect` parameter points to a rectangle bounding the module's display area, in local coordinates. All drawing done by a module within the bounds of the control strip must be limited to the module's display rectangle. The graphics port's `clipRgn` will be set to the visible portion of this rectangle so you can draw all the elements in the display. If you need to change the `clipRgn`, you should observe the initial `clipRgn` to avoid drawing over other items in the control strip.

sdevMouseClick

When the user clicks in a module's display area, the control strip software calls the module with the `sdevMouseClick` message if the `sdevWantMouseClicks` bit is set in the module's features.

If the `sdevDontAutoTrack` bit is also set, the control strip draws the module's display in its highlighted state and then sends the `sdevMouseClick` message to the module. If the `sdevDontAutoTrack` bit is not set, the control strip software tracks the cursor until the mouse button is released. If the cursor is still within the module's display area, the control strip software sends the `sdevMouseClick` message to notify the module that a click occurred. In either case, the module can then perform the appropriate function in response to a mouse-down event.

This message returns the same result as the `sdevPeriodicTickle` message.

sdevSaveSettings

The `sdevSaveSettings` message is passed to the module when the control strip software has determined that it's a good time to save configuration information to the disk. This message will be sent only if the module had previously set the `sdevNeedToSave` bit in the result of a `sdevPeriodicTickle` or `sdevMouseClick` message. The call returns an error code (File Manager, Resource Manager, or the like) indicating the success of the save operation. The control strip software will continue to send this message to the module until the module returns a result of 0, indicating a successful save.

sdevShowBalloonHelp

The control strip software calls the module with the `sdevShowBalloonHelp` message if Balloon Help is turned on, the module has previously set the `sdevHasCustomHelp` bit in its features, and the cursor is over the module's display area. The module should then call the Help Manager to display a help balloon describing the current state of the module. The module should return a value of 0 if it's successful or an appropriate error result if not.

Utility Routines

The control strip software provides a set of utility routines that are available to control strip modules. They are provided to promote a consistent user interface within the control strip and to reduce the amount of duplicated code that each module would have to include to support common functions.

The utility routines are called through a selector-based trap, `_ControlStripDispatch` ($AAF2). If an unimplemented routine is called, it will return `paramErr` as the result.

IMPORTANT
These routines should not be called at interrupt time because they all move memory. ▲

SBIsControlStripVisible

You can use the `SBIsControlStripVisible` routine to find out whether the control strip is visible.

```
pascal Boolean SBIsControlStripVisible();
```

The `SBIsControlStripVisible` routine returns a Boolean value indicating whether or not the control strip is currently visible. It returns a value of `true` if the control strip is visible, or a value of `false` if it's hidden.

It is possible for this call to return a value of `true` even when the control strip is not visible. That happens whenever the control strip is not accessible in the current environment. As soon as that condition changes, the control strip becomes visible again and the returned value correctly reflects the actual state.

SBShowHideControlStrip

You can use the `SBShowHideControlStrip` routine to show or hide the control strip.

```
pascal void SBShowHideControlStrip(Boolean showIt);
```

The `SBShowHideControlStrip` routine determines the visibility state for the control strip based on the value of the `showIt` parameter. Passing a value of `true` makes the control strip visible, and passing a value of `false` hides it. Modules shouldn't typically need to call this routine, but it's provided as a means for other software to hide the control strip when it might get in the way.

Calling `SBShowHideControlStrip` with a `showIt` value of `true` may or may not show the control strip, depending on the current environment: if the control strip is not accessible, it does not become visible. If a `showIt` value of `true` is passed to this routine, then when the environment changes, the control strip will become visible.

SBSafeToAccessStartupDisk

You can use the `SBSafeToAccessStartupDisk` routine to find out whether the internal hard disk is spinning so that your software can determine whether to make a disk access or postpone it until a time when the disk is already spinnings.

```
pascal Boolean SBSafeToAccessStartupDisk();
```

The `SBSafeToAccessStartupDisk` routine returns a Boolean value of `true` if the disk is spinning and `false` if it is not.

SBOpenModuleResourceFile

You can use the `SBOpenModuleResourceFile` routine to open a module resource file.

```
pascal short SBOpenModuleResourceFile(OSType fileCreator);
```

The `SBOpenModuleResourceFile` routine opens the resource fork of the module file whose creator is `fileCreator`, and return the file's reference number as its result. If the file cannot be found or opened, `SBOpenModuleResourceFile` returns a result of –1.

`SBOpenModuleResourceFile` provides a means for a module to load in large or infrequently used resources that it doesn't usually need, but that it requires for a particular operation.

SBLoadPreferences

You can use the `SBLoadPreferences` routine to load a resource from a preferences file.

```
pascal OSErr SBLoadPreferences(ConstStr255Param prefsResourceName,
                               Handle *preferences);
```

The `SBLoadPreferences` routine loads a resource containing a module's configuration information from the control strip's preferences file. The `PrefsResourceName` parameter points to a Pascal string containing the name of the resource. The `Preferences` parameter points to a variable that will hold a handle to the resource read from the file. The handle does not need to be preallocated.

If either `prefsResourceName` or `preferences` contains a nil pointer, `SBLoadPreferences` does nothing and returns a result of `paramErr`. If the resource is successfully loaded, it returns a result of 0. `SBLoadPreferences` can also return other Memory Manager and Resource Manager errors if it fails during some part of the process.

SBSavePreferences

You can use the `SBSavePreferences` routine to save a resource to a preferences file.

```
pascal OSErr SBSavePreferences(ConstStr255Param prefsResourceName,
                               Handle preferences);
```

The `SBSavePreferences` routine saves a resource containing a module's configuration information to the control strip's preferences file. The `PrefsResourceName` parameter

points to a Pascal string containing the name of the resource. The `preferences` parameter contains a handle to a block of data which will be written to the file.

If either `prefsResourceName` or `preferences` has a nil value, `SBSavePreferences` does nothing and returns a result of `paramErr`. If the resource is successfully saved, `SBSavePreferences` returns a result of 0. `SBSavePreferences` can also return other Memory Manager and Resource Manager errors if it fails during some part of the process.

SBGetDetachedIndString

You can use the `SBGetDetachedIndString` routine to get a string from a detached resource.

```
pascal void SBGetDetachedIndString(StringPtr theString,
                                   Handle stringList,
                                   short whichString);
```

The `SBGetDetachedIndString` routine is the detached resource version of `GetIndString`. The parameter `theString` points to a Pascal string; `stringList` is a handle to a detached `'STR#'` resource; and `whichString` is the index (1–n) into the array of Pascal strings contained in the detached resource. `SBGetDetachedIndString` will copy the string whose index is `whichString` into the space pointed to by `theString`. If `whichString` is out of range, `SBGetDetachedIndString` will return a zero-length string.

SBGetDetachIconSuite

You can use the `SBGetDetachIconSuite` routine to set up a detached icon suite.

```
pascal OSErr SBGetDetachIconSuite(Handle *theIconSuite,
                                  short theResID,
                                  unsigned long selector);
```

The `SBGetDetachIconSuite` routine creates a new icon suite, loads all of the requested icons, and then detaches the icons. The parameter `theIconSuite` points to the location where the handle to the icon suite will be stored; the parameter `theResID` is the resource ID of the icons that make up the icon suite; and the parameter `selector` tells which icons should be loaded into the suite. The `selector` parameter should typically contain one (or a combination of) the following values:

`svAllLargeData` `0x000000FF` load large 32-by-32-pixel icons (`'ICN#'`, `'icl4'`, `'icl8'`)

`svAllSmallData`	`0x0000FF00`	load small 16-by-16-pixel icons (`'ics#'`, `'ics4'`, `'ics8'`)
`svAllMiniData`	`0x00FF0000`	load mini 12-by-12-pixel icons (`'icm#'`, `'icm4'`, `'icm8'`)

These values may be OR-ed together to load combinations of icon sizes. `SBGetDetachIconSuite` returns an appropriate error code if it's unsuccessful, or 0 if if was able to load the icon suite. Note that if none of the icons comprising the icon suite could be found, the call returns the error `resNotFound`.

IMPORTANT

You should call `SBGetDetachIconSuite` only when the module's resource file is open, which is typically the case during a module's initialization call. ▲

SBTrackpopupMenu

You can use the `SBTrackpopupMenu` routine to manage a pop-up menu.

```
pascal short SBTrackpopupMenu(const Rect *moduleRect,
                              MenuHandle theMenu);
```

The `SBTrackpopupMenu` routine handles setting up and displaying a pop-up menu associated with a module. The module should pass a pointer to its display rectangle and a handle to the menu to use. The menu will be displayed just above the module's display rectangle, allowing the user to view the current configuration or to change the settings. `SBTrackpopupMenu` returns which menu item was selected, or 0 if no item was selected because the user moved the cursor outside the menu's bounds.

IMPORTANT

Menus are displayed in the control strip's font, so don't use `the CheckItem()` routine to mark menu items, because a checkmark is supported only in the system font. Use the `SetItemMark()` routine instead and pass it a bullet (•). ▲

SBTrackSlider

You can use the `SBTrackSlider` routine to display and set an arbitrary parameter.

```
pascal short SBTrackSlider(const Rect *moduleRect,
                           short ticksOnSlider,
                           short initialValue);
```

The SBTrackSlider routine displays an unlabeled slider above the module's display rectangle. You can use the slider for displaying and setting the state of an arbitrary parameter. The parameter ModuleRect contains a pointer to the module's display rectangle; ticksOnSlider is the upper bounds of the value returned by the slider; and initialValue is the starting position (0 to ticksOnSlider–1). When the user releases the mouse button, SBTrackSlider returns the final position.

SBShowHelpString

You can use the SBShowHelpString routine to display a help balloon.

```
pascal OSErr SBShowHelpString(const Rect *moduleRect,
                              StringPtr helpString);
```

The SBShowHelpString routine displays a module's help balloon. The module passes a pointer to its display rectangle and a pointer to a Pascal string, and the routine displays the balloon if possible. If the help string has a length of 0 or the Help Manager is unable to display a balloon, an error result is returned. If SBShowHelpString successfully displays the help balloon, it returns a result of 0.

SBGetBarGraphWidth

You can use the SBGetBarGraphWidth routine to find out the how wide a bar graph drawn by SBDrawBarGraph (described next) will be so that a module can calculate its display width.

```
pascal short SBGetBarGraphWidth(short barCount);
```

The SBGetBarGraphWidth routine returns the width of a bar graph containing barCount segments. If barCount has a value less than 0, the SBGetBarGraphWidth routine returns a width of 0.

SBDrawBarGraph

You can use the SBDrawBarGraph routine to draw a bar graph.

```
pascal void SBDrawBarGraph(short level, short barCount,
                           short direction,
                           Point barGraphTopLeft);
```

The `SBDrawBarGraph` routine draws a bar graph containing the number of segments specified by the `barCount` parameter in a module's display area. If the value of barCount is less than or equal to 0, `SBDrawBarGraph` does nothing.

The bar graph is drawn relative to the location specified by `barGraphTopLeft`. Figure 5-10 shows the way the point barGraphTopLeft determines the position of the bar graph.

Figure 5-10 Positioning a bar graph

The `level` parameter determines how many segments are highlighted. The value of `level` should be in the range of 0 to `barCount`–1. If the value of `level` is less than 0, no segments in the bar graph are highlighted; if `level` is greater than or equal to `barCount`, all segments in the bar graph are highlighted.

The direction parameter specifies which way the bar graph will be drawn to show a larger level. It should be one of the following values:

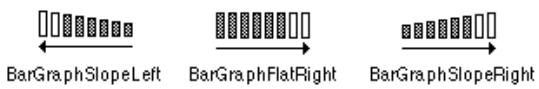
```
#define BarGraphSlopeLeft   -1   // max end of sloping graph is on the left

#define BarGraphFlatRight   0    // max end of flat graph is on the right

#define BarGraphSlopeRight  1    // max end of sloping graph is on the right
```

Figure 5-11 shows the resulting bar graph for each direction value. The arrows indicate which way an increasing level value is displayed. For sloped versions of the bar graph, the number of segments specified by the `barCount` value may not be larger than 8. If a larger `barCount` value is passed, `SBDrawBarGraph` draws nothing.

Figure 5-11 Directions of a bar graph



SBModalDialogInContext

You can use the `SBModalDialogInContext` in place of `ModalDialog` routine to keep background applications from getting run while your modal dialog window is visible.

```
pascal void SBModalDialogInContext(ModalFilterProcPtr filterProc,
                                   short *itemHit);
```

The `SBModalDialogInContext` routine is a special version of `ModalDialog` that doesn't allow background applications to get time while a modal dialog window is visible. You should use `SBModalDialogInContext` when you don't want any context switching to occur.

Gestalt Selectors

The control strip software installs two `Gestalt` selectors to return information to the outside world. One selector returns software attributes, and the other returns the software version.

gestaltControlStripAttr

The selector `gestaltControlStripAttr` (`'sdev'`) returns 32 bits describing the software attributes of this version of the control strip. Currently only the following bit is defined:

`gestaltControlStripExists` 0 1=control strip is installed

gestaltControlStripVersion

The selector `gestaltControlStripVersion` (`'sdvr'`) returns the version of control strip software that is installed. The format of the returned version is the same as that of the numeric part of a `'vers'` resource, that is:

Bits 31-24	Major part of the version, in BCD
Bits 23-20	Minor part of the version, in BCD
Bits 19-16	Bug release version, in BCD
Bits 15- 8	Release stage: $80=final $60=beta $40=alpha $20=development
Bits 7- 0	Revision level of nonreleased version, in binary

Thus, if the software version were 1.5.3b25, the `gestaltControlStripVersion` selector would return $01536019.

CHAPTER 6

Power Manager Interface

This chapter describes the new application programming interface (API) to the Power Manager control software. Developers who provide expanded control panel software for the PowerBook Duo 280 and 280c computers will no longer need access to the Power Manager's internal data structures.

About the Power Manager Interface

Developers have written control panel software for previous PowerBook models that gives the user more control over the power management settings than is provided in the PowerBook control panel. Because that software reads and writes directly to the Power Manager's private data structure and parameter RAM, the software needs to be updated any time Apple Computer makes a change to the internal operation of the Power Manager.

System software for the PowerBook 520 and 540 computers and for future PowerBook models includes interface routines for program access to the Power Manager functions, so it is no longer necessary for applications to deal directly with the Power Manager's data structures. The new routines provide access to most of the Power Manager's parameters. Some functions will be reserved because of their overall effect on the system. The interface is extensible; it will probably grow over time to acccommodate new kinds of functions.

Things That May Change

By using the Power Manager interface, developers can isolate themselves from future changes to the internal operation of the Power Manager software.

IMPORTANT

Apple Computer reserves the right to change the internal operation of the Power Manager software. Developers should not make their applications depend on the Power Manager's internal data structures or parameter RAM. ▲

Starting with the PowerBook 520 and 540 models, developers should not depend on the Power Manager's internal data structures staying the same. In particular, developers should beware of the following assumptions regarding different PowerBook models:

- assuming that timeout values such as the hard disk spindown time reside at the same locations in parameter RAM
- assuming that the power cycling process works the same way or uses the same parameters
- assuming that direct commands to the Power Manager microcontroller are supported on all models

Checking for Routines

Before calling any of the Power Manager interface routines, it's always a good idea to call the Gestalt Manager to see if it they're present on the computer. The Gestalt Manager is described in *Inside Macintosh: Overview.*

A new bit has been added to the `gestaltPowerMgrAttr` selector:

```
#define gestaltPMgrDispatchExists 4
```

If that bit is set to 1, then the routines are present.

Because more routines may be added in the future, one of the new routines simply returns the number of routines that are implemented. The following code fragment determines both that the routines in general exist and that at least the hard disk spindown routine exists.

```
long   pmgrAttributes;
Boolean routinesExist;

routinesExist = false;
if (! Gestalt(gestaltPowerMgrAttr, &pmgrAttributes))
 if (pmgrAttributes & (1<<gestaltPMgrDispatchExists))
  if (PMSelectorCount() >= 7)
   routinesExist = true;
```

▲ **WARNING**
If you call a routine that does not exist, the call to the public Power Manager trap (if the trap exists) will return an error code, which your program could misinterpret as data. ▲

Power Manager Interface Routines

This section tells you how to call the interface routines for the Power Manager software. The interface routines are listed here in the order of their routine selector values, as shown in Table 6-1.

Assembly-language note:
All the routines share a single trap, `_PowerMgrDispatch` ($A09E). The trap is register based: parameters are passed in register D0 and sometimes also in A0. A routine selector value passed in the low word of register D0 determines which routine is executed. ◆

Table 6-1 Interface routines and their selector values

Routine name	Selector value	
	decimal	hexadecimal
`PMSelectorCount`	0	$00
`PMFeatures`	1	$01
`GetSleepTimeout`	2	$02
`SetSleepTimeout`	3	$03
`GetHardDiskTimeout`	4	$04
`SetHardDiskTimeout`	5	$05
`HardDiskPowered`	6	$06
`SpinDownHardDisk`	7	$07
`IsSpindownDisabled`	8	$08
`SetSpindownDisable`	9	$09
`HardDiskQInstall`	10	$0A
`HardDiskQRemove`	11	$0B
`GetScaledBatteryInfo`	12	$0C
`AutoSleepControl`	13	$0D
`GetIntModemInfo`	14	$0E
`SetIntModemState`	15	$0F
`MaximumProcessorSpeed`	16	$10
`CurrentProcessorSpeed`	17	$11
`FullProcessorSpeed`	18	$12
`SetProcessorSpeed`	19	$13
`GetSCSIDiskModeAddress`	20	$14
`SetSCSIDiskModeAddress`	21	$15
`GetWakeupTimer`	22	$16
`SetWakeupTimer`	23	$17
`IsProcessorCyclingEnabled`	24	$18
`EnableProcessorCycling`	25	$19
`BatteryCount`	26	$1A
`GetBatteryVoltage`	27	$1B
`GetBatteryTimes`	28	$1C

PMSelectorCount

You can use the `PMSelectorCount` routine to determine which routines are implemented.

```
short PMSelectorCount();
```

DESCRIPTION

The `PMSelectorCount` routine returns the number of routine selectors present. Any routine whose selector value is greater than the returned value is not implemented.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `PMSelectorCount` is 0 ($00) in the low word of register D0. The number of selectors is returned in the low word of register D0.

PMFeatures

You can use the `PMFeatures` routine to find out which features of the Power Manager are implemented.

```
unsigned long PMFeatures();
```

DESCRIPTION

The `PMFeatures` routine returns a 32-bit field describing hardware and software features associated with the Power Manager on a particular machine. If a bit value is 1, that feature is supported or available; if the bit value is 0, that feature is not available. Unused bits are reserved by Apple for future expansion.

Field descriptions

Bit name	Bit number	Description
`hasWakeupTimer`	0	The wakeup timer is supported.
`hasSharedModemPort`	1	The hardware forces exclusive access to either SCC port A or the internal modem. (If this bit is not set, then typically port A and the internal modem may be used simultaneously by means of the Communications Toolbox.)
`hasProcessorCycling`	2	Processor cycling is supported; that is, when the computer is idle, the processor power will be cycled to reduce the power usage.

`mustProcessorCycle`	3	The processor cycling feature must be left on (turn it off at your own risk).
`hasReducedSpeed`	4	Processor can be started up at a reduced speed in order to extend battery life.
`dynamicSpeedChange`	5	Processor speed can be switched dynamically between its full and reduced speed at any time, rather than only at startup time.
`hasSCSIDiskMode`	6	The SCSI disk mode is supported.
`canGetBatteryTime`	7	The computer can provide an estimate of the battery time remaining.
`canWakeupOnRing`	8	The computer supports waking up from the sleep state when an internal modem is installed and the modem detects a ring.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `PMFeatures` is 1 ($01) in the low word of register D0. The 32-bit field of supported features is returned in register D0.

GetSleepTimeout

You can use the `GetSleepTimeout` routine to find out how long the computer will wait before going to sleep.

```
unsigned char GetSleepTimeout();
```

DESCRIPTION

The `GetSleepTimeout` routine returns the amount of time that the computer will wait after the last user activity before going to sleep. The value of `GetSleepTimeout` is expressed as the number of 15-second intervals that the computer will wait before going to sleep.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetSleepTimeout` is 2 ($02) in the low word of register D0. The sleep timeout value is returned in the low word of register D0.

SetSleepTimeout

You can use the `SetSleepTimeout` routine to set how long the computer will wait before going to sleep.

```
void SetSleepTimeout(unsigned char timeout);
```

DESCRIPTION

The `SetSleepTimeout` routine sets the amount of time the computer will wait after the last user activity before going to sleep. The value of `SetSleepTimeout` is expressed as the number of 15-second intervals making up the desired time. If a value of 0 is passed in, the routine sets the `timeout` value to the default value (currently equivalent to 8 minutes).

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetSleepTimeout` is 3 ($03) in the low word of register D0. The sleep timeout value to set is passed in the high word of register D0.

GetHardDiskTimeout

You can use the `GetHardDiskTimeout` routine to find out how long the computer will wait before turning off power to the internal hard disk.

```
unsigned char GetHardDiskTimeout();
```

DESCRIPTION

The `GetHardDiskTimeout` routine returns the amount of time the computer will wait after the last use of a SCSI device before turning off power to the internal hard disk. The value of `GetHardDiskTimeout` is expressed as the number of 15-second intervals the computer will wait before turning off power to the internal hard disk.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetHardDiskTimeout` is 4 ($04) in the low word of register D0. The hard disk timeout value is returned in the low word of register D0.

SetHardDiskTimeout

You can use the `SetHardDiskTimeout` routine to set how long the computer will wait before turning off power to the internal hard disk.

```
void SetHardDiskTimeout(unsigned char timeout);
```

DESCRIPTION

The `SetHardDiskTimeout` routine sets how long the computer will wait after the last use of a SCSI device before turning off power to the internal hard disk. The value of `SetHardDiskTimeout` is expressed as the number of 15-second intervals the computer will wait before turning off power to the internal hard disk. If a value of 0 is passed in, the routine sets the `timeout` value to the default value (currently equivalent to 4 minutes).

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetHardDiskTimeout` is 5 ($05) in the low word of register D0. The hard disk timeout value to set is passed in the high word of register D0.

HardDiskPowered

You can use the `HardDiskPowered` routine to find out whether the internal hard disk is on.

```
Boolean HardDiskPowered();
```

DESCRIPTION

The `HardDiskPowered` routine returns a Boolean value indicating whether or not the internal hard disk is powered up. A value of `true` means that the hard disk is on, and a value of `false` means that the hard disk is off.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `HardDiskPowered` is 6 ($06) in the low word of register D0. The Boolean result is returned in the low word of register D0.

SpinDownHardDisk

You can use the `SpinDownHardDisk` routine to force the hard disk to spin down.

```
void SpinDownHardDisk();
```

DESCRIPTION

The `SpinDownHardDisk` routine immediately forces the hard disk to spin down and power off if it was previously spinning. Calling `SpinDownHardDisk` will not spin down the hard disk if spindown is disabled by calling `SetSpindownDisable` (defined later in this section).

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SpinDownHardDisk` is 7 ($07) in the low word of register D0.

IsSpindownDisabled

You can use the `IsSpindownDisabled` routine to find out whether hard disk spindown is enabled.

```
Boolean IsSpindownDisabled();
```

DESCRIPTION

The `IsSpindownDisabled` routine returns a Boolean `true` if hard disk spindown is disabled, or `false` if spindown is enabled.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `IsSpindownDisabled` is 8 ($08) in the low word of register D0. The Boolean result is passed in the low byte of register D0.

SetSpindownDisable

You can use the `SetSpindownDisable` routine to disable hard disk spindown.

```
void SetSpindownDisable(Boolean setDisable);
```

DESCRIPTION

The `SetSpindownDisable` routine enables or disables hard disk spindown, depending on the value of `setDisable`. If the value of `setDisable` is `true`, hard disk spindown will be disabled; if the value is `false`, spindown will be enabled.

Disabling hard disk spindown affects the `SpinDownHardDisk` routine, defined earlier, as well as the normal spindown that occurs after a period of hard disk inactivity.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetSpindownDisable` is 9 ($09) in the low word of register D0. The Boolean value to set is passed in the high word of register D0.

HardDiskQInstall

You can use the `HardDiskQInstall` routine to notify your software when power to the internal hard disk is about to be turned off.

```
OSErr HardDiskQInstall(HDQueueElement *theElement);
```

DESCRIPTION

The `HardDiskQInstall` routine installs an element into the hard disk power down queue to provide notification to your software when the internal hard disk is about to be powered off. For example, this feature might be used by the driver for an external battery-powered hard disk. When power to the internal hard disk is turned off, the external hard disk could be turned off as well.

The structure of `HDQueueElement` is as follows.

```
typedef pascal void (*HDSpindownProc)(HDQueueElement *theElement);
struct HDQueueElement {
  Ptr             hdQLink;       /* pointer to next queue element */
  short           hdQType;       /* queue element type (must be HDQType) */
  short           hdFlags;       /* miscellaneous flags (reserved) */
  HDSpindownProc hdProc;         /* pointer to routine to call */
  long      hdUser;              /* user-defined (variable storage, etc.) */
} HDQueueElement;
```

When power to the internal hard disk is about to be turned off, the software calls the routine pointed to by the `hdProc` field so that it can do any special processing. The routine will be passed a pointer to its queue element so that, for example, the routine can reference its variables.

Before calling `HardDiskQInstall`, the calling program must set the `hdQType` field to

```
#define HDPwrQType 'HD'      /* queue element type */
```

or the queue element won't be added to the queue and `HardDiskQInstall` will return an error.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `HardDiskQInstall` is 10 ($0A) in the low word of register D0. The pointer to the `HDQueue` element is passed in register A0. The result code is returned in the low word of register D0.

HardDiskQRemove

You can use the `HardDiskQRemove` routine to discontinue notification of your software when power to the internal hard disk is about to be turned off.

```
OSErr HardDiskQRemove(HDQueueElement *theElement);
```

DESCRIPTION

The `HardDiskQRemove` routine removes a queue element installed by `HardDiskQInstall`. If the `hdQType` field of the queue element is not set to `HDPwrQType`, `HardDiskQRemove` simply returns an error.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `HardDiskQRemove` is 11 ($0B) in the low word of register D0. The pointer to the `HDQueue` element is passed in register A0. The result code is returned in the low word of register D0.

GetScaledBatteryInfo

You can use the `GetScaledBatteryInfo` routine to find out the condition of the battery or batteries .

```
void GetScaledBatteryInfo(short whichBattery, BatteryInfo*theInfo);
```

DESCRIPTION

The `GetScaledBatteryInfo` routine provides a generic means of returning information about the battery or batteries in the system. Instead of returning a voltage value, the routine returns the battery level as a fraction of the total possible voltage.

Note
New battery technologies such as NiCad (nickel cadmium) and nickel metal hydride (NiMH) have replaced the sealed lead acid batteries of the original Macintosh Portable. The algorithm for determining the battery voltage that is documented in the Power Manager chapter of *Inside Macintosh,* Volume VI, is no longer correct for all PowerBook models. ◆

The value of `whichBattery` determines whether `GetScaledBatteryInfo` returns information about a particular battery or about the total battery level. The value of `GetScaledBatteryInfo` should be in the range of 0 to `BatteryCount()`. If the value of `whichBattery` is 0, `GetScaledBatteryInfo` returns a summation of all the batteries, that is, the effective battery level of the whole system. If the value of `whichBattery` is out of range, or the selected battery is not installed, `GetScaledBatteryInfo` will return a result of 0 in all fields. Here is a summary of the effects of the `whichBattery` parameter:

Value of `whichBattery`	Information returned
0	Total battery level for all batteries
From 1 to `BatteryCount`()	Battery level for the selected battery
Less than 0 or greater than `BatteryCount`	0 in all fields of `theInfo`

The `GetScaledBatteryInfo` routine returns information about the battery in the following data structure:

```
typedef struct BatteryInfo {
  unsigned char flags;              /* misc flags (see below) */
  unsigned char   warningLevel;     /* scaled warning level (0-255) */
  char            reserved;         /* reserved for internal use */
  unsigned char   batteryLevel;     /* scaled battery level (0-255) */
} BatteryInfo;
```

The `flags` character contains several bits that describe the battery and charger state. If a bit value is 1, that feature is available or is operating; if the bit value is 0, that feature is not operating. Unused bits are reserved by Apple for future expansion.

Field descriptions

Bit name	Bit number	Description
`batteryInstalled`	7	A battery is installed.
`batteryCharging`	6	The battery is charging.
`chargerConnected`	5	The charger is connected.

The value of `warningLevel` is the battery level at which the first low battery warning message will appear. The routine returns a value of 0 in some cases when it's not appropriate to return the warning level.

The value of `batteryLevel` is the current level of the battery. A value of 0 represents the voltage at which the Power Manager will force the computer into sleep mode; a value of 255 represents the highest possible voltage.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetScaledBatteryInfo` is 12 ($0C) in the low word of register D0. The `BatteryInfo` data are returned in the low word of register D0 as follows:

Bits 31–24	Flags
Bits 23–16	Warning level
Bits 15–8	Reserved
Bits 7–0	Battery level

AutoSleepControl

You can use the `AutoSleepControl` routine to turn the automatic sleep feature on and off.

```
void AutoSleepControl(Boolean enableSleep);
```

DESCRIPTION

The `AutoSleepControl` routine enables or disables the automatic sleep feature that causes the computer to go into sleep mode after a preset period of time. When `enableSleep` is set to `true`, the automatic sleep feature is enabled (this is the normal state). When `enableSleep` is set to `false`, the computer will not go into the sleep mode unless it is forced to either by some user action—for example, by the user's selecting Sleep from the Special menu of the Finder—or in a low battery situation.

IMPORTANT

Calling `AutoSleepControl` with `enableSleep` set to `false` multiple times increments the auto sleep disable level so that it requires the same number of calls to `AutoSleepControl` with `enableSleep` set to `true` to reenable the auto sleep feature. If more than one piece of software makes this call, auto sleep may not be reenabled when you think it should be. ▲

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `AutoSleepControl` is 13 ($0D) in the low word of register D0. The Boolean value is passed in the high word of register D0.

GetIntModemInfo

You can use the `GetIntModemInfo` routine to find out information about the internal modem.

```
unsigned long GetIntModemInfo();
```

DESCRIPTION

The `GetIntModemInfo` routine returns a 32-bit field containing information that describes the features and state of the internal modem. It can be called whether or not a modem is installed and will return the correct information.

If a bit is set, that feature or state is supported or selected; if the bit is cleared, that feature is not supported or selected. Undefined bits are reserved by Apple for future expansion.

Field descriptions

Bit name	Bit number	Description
`hasInternalModem`	0	An internal modem is installed.
`intModemRingDetect`	1	The modem has detected a ring on the telephone line.
`intModemOffHook`	2	The internal modem has taken the telephone line off hook (that is, you can hear the dial tone or modem carrier).
`intModemRingWakeEnb`	3	The computer will come out of sleep mode if the modem detects a ring on the telephone line and the computer supports this feature (see the `canWakeupOnRing` bit in `PMFeatures`).
`extModemSelected`	4	The external modem is selected (if this bit is set, then the modem port will be connected to port A of the SCC; if the modem port is not shared by the internal modem and the SCC, then this bit can be ignored).

Bits 15–31 contain the modem type, which will take on one of the following values:

–1 Modem is installed but type not recognized.

0 No modem is installed.

1 Modem is a serial modem.

2 Modem is a PowerBook Duo–style Express Modem.

3 Modem is a PowerBook 160/180–style Express Modem.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetIntModemInfo` is 14 ($0E) in the low word of register D0. The bit field to set is passed in the high word of register D0.

SetIntModemState

You can use the `SetIntModemState` routine to set some parts of the state of the internal modem.

```
void SetIntModemState(short theState);
```

DESCRIPTION

The `SetIntModemState` routine configures some of the internal modem's state information. Currently the only items that can be changed are the internal/external modem selection and the wakeup-on-ring feature.

To change an item of state information, the calling program sets the corresponding bit in `theState`. In other words, to change the internal/external modem setting, set bit 4 of `theState` to 1. To select the internal modem, bit 15 should be set to 0; to select the external modem, bit 15 should be set to 1.Using this method, the bits may be set or cleared independently, but they may not be set to different states at the same time.

Note
In some PowerBook computers, there is a hardware switch to connect either port A of the SCC or the internal modem to the modem port. The two are physically separated, but software emulates the serial port interface for those applications that don't use the Communications Toolbox. You can check the `hasSharedModemPort` bit returned by `PMFeatures` to determine which way the computer is set up. ◆

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetIntModemState` is 15 ($0F) in the low word of register D0. The bit field is returned in register D0.

MaximumProcessorSpeed

You can use the `MaximumProcessorSpeed` routine to find out the maximum speed of the computer's microprocessor.

```
short MaximumProcessorSpeed();
```

DESCRIPTION

The `MaximumProcessorSpeed` routine returns the maximum clock speed of the computer's microprocessor, in MHz.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `MaximumProcessorSpeed` is 16 ($10) in the low word of register D0. The processor speed value is returned in the low word of register D0.

CurrentProcessorSpeed

You can use the `CurrentProcessorSpeed` routine to find out the current clock speed of the microprocessor.

```
short CurrentProcessorSpeed();
```

DESCRIPTION

The `CurrentProcessorSpeed` routine returns the current clock speed of the computer's microprocessor, in MHz. The value returned will be different from the maximum processor speed if the computer has been configured to run with a reduced processor speed to conserve power.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `CurrentProcessorSpeed` is 17 ($11) in the low word of register D0. The processor speed value is returned in the low word of register D0.

FullProcessorSpeed

You can use the `FullProcessorSpeed` routine to find out whether the computer will run at full speed the next time it restarts.

```
Boolean FullProcessorSpeed();
```

DESCRIPTION

The `FullProcessorSpeed` routine returns a Boolean value of `true` if, on the next restart, the computer will start up at its maximum processor speed; it returns `false` if the computer will start up at its reduced processor speed.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `FullProcessorSpeed` is 18 ($12) in the low word of register D0. The Boolean result is returned in the low byte of register D0.

SetProcessorSpeed

You can use the `SetProcessorSpeed` routine to set the clock speed the microprocessor will use the next time it is restarted.

```
Boolean SetProcessorSpeed(Boolean fullSpeed);
```

DESCRIPTION

The `SetProcessorSpeed` routine sets the processor speed that the computer will use the next time it is restarted. If the value of `fullSpeed` is set to `true`, the processor will start up at its full speed (the speed returned by `MaximumProcessorSpeed`, described on page 109). If the value of `fullSpeed` is set to `false`, the processor will start up at its reduced speed.

For PowerBook models that support changing the processor speed dynamically, the processor speed will also be changed. If the speed is actually changed, `SetProcessorSpeed` will return `true`; if the speed isn't changed, it will return `false`.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetProcessorSpeed` is 19 ($13) in the low word of register D0. The Boolean value to set is passed in the high word of register D0. The Boolean result is returned in register D0.

GetSCSIDiskModeAddress

You can use the `GetSCSIDiskModeAddress` routine to find out the SCSI ID the computer uses in SCSI disk mode.

```
short GetSCSIDiskModeAddress();
```

DESCRIPTION

The `GetSCSIDiskModeAddress` routine returns the SCSI ID that the computer uses when it is started up in SCSI disk mode. The returned value is in the range 1 to 6.

Note
When the computer is in SCSI disk mode, the computer appears as a hard disk to another computer. ◆

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetSCSIDiskModeAddress` is 20 ($14) in the low word of register D0. The SCSI ID is returned in the low word of register D0.

SetSCSIDiskModeAddress

You can use the `SetSCSIDiskModeAddress` routine to set the SCSI ID for the computer to use in SCSI disk mode.

```
void SetSCSIDiskModeAddress(short scsiAddress);
```

DESCRIPTION

The `SetSCSIDiskModeAddress` routine sets the SCSI ID that the computer will use if it is started up in SCSI disk mode.

The value of `scsiAddress` must be in the range of 1 to 6. If any other value is given, the software sets the SCSI ID for SCSI disk mode to 2.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetSCSIDiskModeAddress` is 21 ($15) in the low word of register D0. The SCSI ID to set is passed in the high word of register D0.

GetWakeupTimer

You can use the `GetWakeupTimer` routine to find out when the computer will wake up from sleep mode.

```
void GetWakeupTimer(WakeupTime *theTime);
```

DESCRIPTION

The `GetWakeupTimer` routine returns the time when the computer will wake up from sleep mode.

If the PowerBook model doesn't support the wakeup timer, `GetWakeupTimer` returns a value of 0. The time and the enable flag are returned in the following structure:

```
typedef struct WakeupTime {
   unsigned long wakeTime;    /* wakeup time (same format as the time) */
   char      wakeEnabled;     /* 1=enable timer, 0=disable timer */
} WakeupTime;
```

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetWakeupTimer` is 22 ($16) in the low word of register D0. The pointer to `WakeupTime` is passed in register A0.

SetWakeupTimer

You can use the `SetWakeupTimer` routine to set the time when the computer will wake up from sleep mode.

```
void SetWakeupTimer(WakeupTime *theTime);
```

DESCRIPTION

The `SetWakeupTimer` routine sets the time when the computer will wake up from sleep mode and enables or disables the timer. On a PowerBook model that doesn't support the wakeup timer, `SetWakeupTimer` does nothing.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetWakeupTimer` is 23 ($17) in the low word of register D0. The pointer to `WakeupTime` is passed in register A0.

IsProcessorCyclingEnabled

You can use the `IsProcessorCyclingEnabled` routine to find out whether processor cycling is enabled.

```
Boolean IsProcessorCyclingEnabled();
```

DESCRIPTION

The `IsProcessorCyclingEnabled` routine returns a Boolean value of `true` if processor cycling is currently enabled, or `false` if it is disabled.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `IsProcessorCyclingEnabled` is 24 ($18) in the low word of register D0. The Boolean result is returned in register D0.

EnableProcessorCycling

You can use the `EnableProcessorCycling` routine to turn the processor cycling feature on and off.

```
void EnableProcessorCycling(Boolean enable);
```

DESCRIPTION

The `EnableProcessorCycling` routine enables processor cycling if a value of `true` is passed in, and disables it if `false` is passed.

▲ **WARNING**
You should follow the advice of the `mustProcessorCycle` bit in the feature flags when turning processor cycling off. Turning processor cycling off when it's not recommended can result in hardware failures due to overheating. ▲

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `EnableProcessorCycling` is 25 ($19) in the low word of register D0. The Boolean value to set is passed in the high word of register D0.

BatteryCount

You can use the `BatteryCount` routine to find out how many batteries the computer supports.

```
short BatteryCount();
```

DESCRIPTION

The `BatteryCount` routine returns the number of batteries that are supported internally by the computer. The value of `BatteryCount` returned may not be the same as the number of batteries currently installed.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `BatteryCount` is 26 ($1A) in the low word of register D0. The number of batteries supported is returned in the low word of register D0.

GetBatteryVoltage

You can use the `GetBatteryVoltage` routine to find out the battery voltage.

```
Fixed GetBatteryVoltage(short whichBattery);
```

DESCRIPTION

The `GetBatteryVoltage` routine returns the battery voltage as a fixed-point number.

The value of `whichBattery` should be in the range 0 to `BatteryCount()`–1. If the value of `whichBattery` is out of range, or the selected battery is not installed, `GetBatteryVoltage` will return a result of 0.0 volts.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetBatteryVoltage` is 27 ($1B) in the low word of register D0. The battery number is passed in the high word of register D0. The 32-bit value of the battery voltage is returned in register D0.

GetBatteryTimes

You can use the `GetBatteryTimes` routine to find out about how much battery time remains.

```
void GetBatteryTimes (short whichBattery, BatteryTimeRec *theTimes);
```

DESCRIPTION

The `GetBatteryTimes` routine returns information about the time remaining on the computer's battery or batteries. The information returned has the following data structure:

```
typedef struct BatteryTimeRec {
  unsigned long expectedBatteryTime;   /* estimated time remaining */
  unsigned long minimumBatteryTime;    /* minimum time remaining */
  unsigned long maximumBatteryTime;    /* maximum time remaining */
  unsigned long timeUntilCharged;      /* time until full charge */
} BatteryTimeRec;
```

The time values are in seconds. The value of `expectedBatteryTime` is the estimated time remaining based on current usage patterns. The values of `minimumBatteryTime` and `maximumBatteryTime` are worst-case and best-case estimates, respectively. The value of `timeUntilCharged` is the time that remains until the battery or batteries are fully charged.

The value of `whichBattery` determines whether `GetBatteryTimes` returns the time information about a particular battery or the total time for all batteries. The value of `GetScaledBatteryInfo` should be in the range of 0 to `BatteryCount()`. If the value of `whichBattery` is 0, `GetBatteryTimes` returns a total time for all the batteries, that is, the effective battery time for the whole system. If the value of `whichBattery` is out of range, or the selected battery is not installed, `GetBatteryTimes` will return a result of 0 in all fields. Here is a summary of the effects of the `whichBattery` parameter:

Value of `whichBattery`	Information returned
0	Total battery time for all batteries
From 1 to `BatteryCount( )`	Battery time for the selected battery
Less than 0 or greater than `BatteryCount`	0 in all fields of `theTimes`

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetBatteryTimes` is 28 ($1C) in the low word of register D0. The pointer to `BatteryTimeRec` is passed in register A0.

Header File for Power Manager Dispatch

Here is a sample header file for access to the Power Manager.

```
/*****************************************************************************

    file:  PowerMgrDispatch.h

    contains:  header file for access to the Power Manager

    Copyright © 1992-1993 by Apple Computer, Inc. All rights reserved.

*****************************************************************************/

#ifndef __PowerMgrDispatch__

#define __PowerMgrDispatch__

#ifndef __TYPES__

#include <Types.h>

#endif


#ifndef gestaltPMgrDispatchExists

#define gestaltPMgrDispatchExists       4   /* gestaltPowerMgrAttr bit:
                                               1=PowerMgrDispatch exists */

#endif


/* bits in bitfield returned by PMFeatures */

#define hasWakeupTimer          0   /* 1=wakeup timer is supported  */

#define hasSharedModemPort      1   /* 1=modem port shared by SCC and internal modem */

#define hasProcessorCycling     2   /* 1=processor cycling is supported */

#define mustProcessorCycle      3   /* 1=processor cycling should not be turned off */

#define hasReducedSpeed         4   /* 1=processor can be started up at reduced speed */

#define dynamicSpeedChange      5   /* 1=processor speed can be switched dynamically */

#define hasSCSIDiskMode         6   /* 1=SCSI disk mode is supported */

#define canGetBatteryTime       7   /* 1=battery time can be calculated */

#define canWakeupOnRing      8   /* 1=can wake up when the modem detects a ring  */
```

```
/* bits in bitfield returned by GetIntModemInfo and set by SetIntModemState */
#define hasInternalModem     0   /* 1=internal modem installed */
#define intModemRingDetect   1   /* 1=internal modem has detected a ring */
#define intModemOffHook      2   /* 1=internal modem is off hook */
#define intModemRingWakeEnb  3   /* 1=wake up on ring is enabled */
#define extModemSelected     4   /* 1=external modem selected */
#define modemSetBit          15  /* 1=set bit, 0=clear bit (SetIntModemState) */

/* information returned by GetScaledBatteryInfo */
struct BatteryInfo {
    unsigned charflags;             /* misc flags (see below) */
    unsigned charwarningLevel;      /* scaled warning level (0-255) */
    char   reserved;                /* reserved for internal use */
    unsigned charbatteryLevel;      /* scaled battery level (0-255) */
};

typedef struct BatteryInfo BatteryInfo;
/* bits in BatteryInfo.flags */
#define batteryInstalled     7      /* 1=battery is currently connected */
#define batteryCharging      6      /* 1=battery is being charged */
#define chargerConnected     5      /* 1=charger is connected to the PowerBook */
                                    /* (this doesn't mean the charger is plugged in) */

/* hard disk spindown notification queue element */
typedef struct HDQueueElement HDQueueElement;

typedef pascal void (*HDSpindownProc)(HDQueueElement *theElement);
```

```
struct HDQueueElement {
    Ptr            hdQLink;      /* pointer to next queue element */
    short          hdQType;      /* queue element type (must be HDQType) */
    short          hdFlags;      /* miscellaneous flags */
    HDSpindownProc  hdProc;      /* pointer to routine to call */
    long           hdUser;       /* user-defined (variable storage, etc.) */
};

#define HDPwrQType'HD'           /* queue element type */
/* wakeup time record */
typedef struct WakeupTime {
    unsigned long     wakeTime;     /* wakeup time (same format as current time) */
    char              wakeEnabled;  /* 1=enable wakeup timer, 0=disable wakeup timer */
} WakeupTime;

/* battery time information (in seconds) */
typedef struct BatteryTimeRec {
    unsigned long     expectedBatteryTime;     /* estimated battery time remaining */
    unsigned long     minimumBatteryTime;      /* minimum battery time remaining */
    unsigned long     maximumBatteryTime;      /* maximum battery time remaining */
    unsigned long     timeUntilCharged;        /* time until battery is fully charged */
} BatteryTimeRec;

#ifdef __cplusplus
extern "C" {
#endif
```

```
#pragma parameter __D0 PMSelectorCount(__D0)
short PMSelectorCount()
    = {0x7000, 0xA09E};

#pragma parameter __D0 PMFeatures
unsigned long PMFeatures()
    = {0x7001, 0xA09E};

#pragma parameter __D0 GetSleepTimeout
unsigned char GetSleepTimeout()
    = {0x7002, 0xA09E};

#pragma parameter __D0 SetSleepTimeout(__D0)
void SetSleepTimeout(unsigned char timeout)
    = {0x4840, 0x303C, 0x0003, 0xA09E};

#pragma parameter __D0 GetHardDiskTimeout
unsigned char GetHardDiskTimeout()
    = {0x7004, 0xA09E};

#pragma parameter __D0 SetHardDiskTimeout(__D0)
void SetHardDiskTimeout(unsigned char timeout)
    = {0x4840, 0x303C, 0x0005, 0xA09E};

#pragma parameter __D0 HardDiskPowered
Boolean HardDiskPowered()
    = {0x7006, 0xA09E};
```

```
#pragma parameter __D0 SpinDownHardDisk
void SpinDownHardDisk()
    = {0x7007, 0xA09E};

#pragma parameter __D0 IsSpindownDisabled
Boolean IsSpindownDisabled()
    = {0x7008, 0xA09E};

#pragma parameter __D0 SetSpindownDisable(__D0)
void SetSpindownDisable(Boolean setDisable)
    = {0x4840, 0x303C, 0x0009, 0xA09E};

#pragma parameter __D0 HardDiskQInstall(__A0)
OSErr HardDiskQInstall(HDQueueElement *theElement)
    = {0x700A, 0xA09E};

#pragma parameter __D0 HardDiskQRemove(__A0)
OSErr HardDiskQRemove(HDQueueElement *theElement)
    = {0x700B, 0xA09E};

#pragma parameter __D0 GetScaledBatteryInfo(__D0,__A0)
void GetScaledBatteryInfo(short whichBattery, BatteryInfo *theInfo)
    = {0x4840, 0x303C, 0x000C, 0xA09E, 0x2080};

#pragma parameter __D0 AutoSleepControl(__D0)
void AutoSleepControl(Boolean enableSleep)
    = {0x4840, 0x303C, 0x000D, 0xA09E};
```

```
#pragma parameter __D0 GetIntModemInfo(__D0)
unsigned long GetIntModemInfo()
    = {0x700E, 0xA09E};

#pragma parameter __D0 SetIntModemState(__D0)
void SetIntModemState(short theState)
    = {0x4840, 0x303C, 0x000F, 0xA09E};

#pragma parameter __D0 MaximumProcessorSpeed
short MaximumProcessorSpeed()
    = {0x7010, 0xA09E};

#pragma parameter __D0 CurrentProcessorSpeed
short CurrentProcessorSpeed()
    = {0x7011, 0xA09E};

#pragma parameter __D0 FullProcessorSpeed
Boolean FullProcessorSpeed()
    = {0x7012, 0xA09E};

#pragma parameter __D0 SetProcessorSpeed(__D0)
Boolean SetProcessorSpeed(Boolean fullSpeed)
    = {0x4840, 0x303C, 0x0013, 0xA09E};

#pragma parameter __D0 GetSCSIDiskModeAddress
short GetSCSIDiskModeAddress()
    = {0x7014, 0xA09E};
```

```
#pragma parameter __D0 SetSCSIDiskModeAddress(__D0)
void SetSCSIDiskModeAddress(short scsiAddress)
    = {0x4840, 0x303C, 0x0015, 0xA09E};

#pragma parameter __D0 GetWakeupTimer(__A0)
void GetWakeupTimer(WakeupTime *theTime)
    = {0x7016, 0xA09E};

#pragma parameter __D0 SetWakeupTimer(__A0)
void SetWakeupTimer(WakeupTime *theTime)
    = {0x7017, 0xA09E};

#pragma parameter __D0 IsProcessorCyclingEnabled
Boolean IsProcessorCyclingEnabled()
    = {0x7018, 0xA09E};

#pragma parameter __D0 EnableProcessorCycling(__D0)
void EnableProcessorCycling(Boolean enable)
    = {0x4840, 0x303C, 0x0019, 0xA09E};

#pragma parameter __D0 BatteryCount
short BatteryCount()
    = {0x701A, 0xA09E};

#pragma parameter __D0 GetBatteryVoltage(__D0)
Fixed GetBatteryVoltage(short whichBattery)
    = {0x4840, 0x303C, 0x001B, 0xA09E};
```

```
#pragma parameter __D0 GetBatteryTimes(__D0,__A0)
void GetBatteryTimes(BatteryTimeRec *theTimes)
    = {0x4840, 0x303C, 0x001C, 0xA09E};


#ifdef __cplusplus
}
#endif
#endif
```

Index

D

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 and final pages were created on an Apple LaserWriter Pro printer. Line art was created using Adobe Illustrator™ and Adobe Photoshop™. PostScript™, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

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

WRITER
Allen Watson

DEVELOPMENTAL EDITORS
Wendy Krafft, Beverly Zegarski

ILLUSTRATORS
Deb Dennis, Sandee Karr

Special thanks to Larry Agbulos, Mark Baumwell, Faith Bradford, Steve Christensen, Gary Embler, Eric Gradeler, Ted Hammer, and Rosemary Mylod