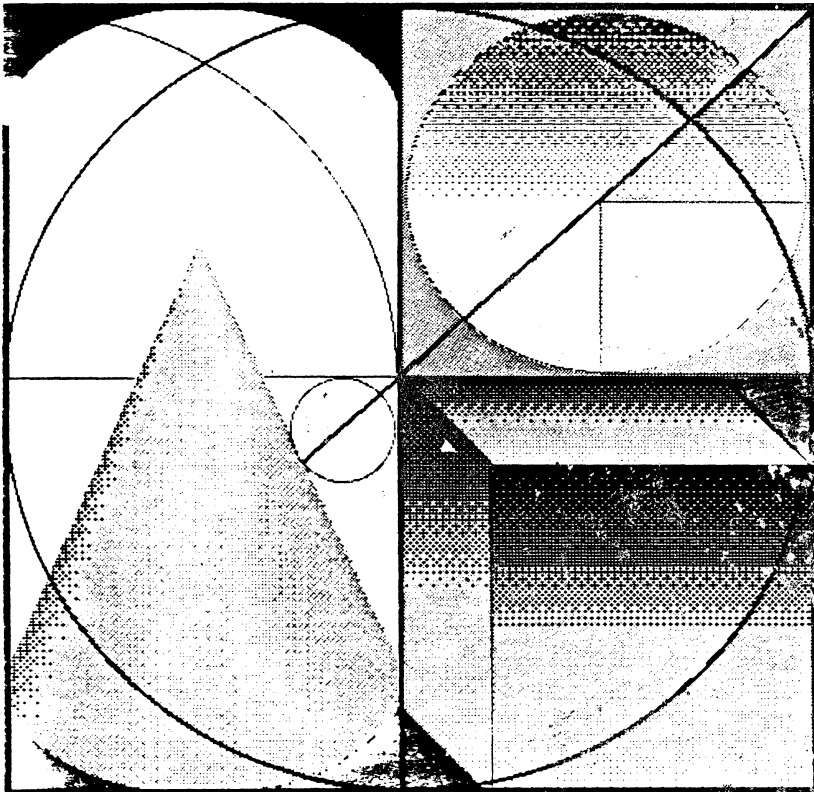


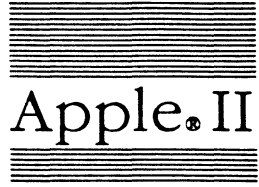


# Apple® II SCSI Card Technical Reference Manual

APDA =A2G0029







# Apple II SCSI Card Technical Reference

🍏 APPLE COMPUTER, INC.

This manual is copyrighted by Apple or by Apple's suppliers, with all rights reserved. Under the copyright laws, this manual may not be copied, in whole or in part, without the written consent of Apple Computer, Inc. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased may be sold, given, or lent to another person. Under the law, copying includes translating into another language.

© Apple Computer, Inc., 1988  
20525 Mariani Avenue  
Cupertino, CA 95014  
(408) 996-1010

Apple, Apple IIGS, the Apple logo, LaserWriter, Macintosh, and ProDOS are registered trademarks of Apple Computer, Inc.

ITC Avant Garde Gothic, ITC Garamond, and ITC Zapf Dingbats are registered trademarks of International Typeface Corporation.

Microsoft is a registered trademark of Microsoft Corporation.

Adobe Illustrator is a trademark and POSTSCRIPT is a registered trademark of Adobe Systems Incorporated.

Simultaneously published in the United States and Canada.



# Contents



**Figures and tables v**  
**Preface vii**

<b>Chapter 1</b>	<b>Introduction to the Apple II SCSI Card 1</b>
	The Apple II SCSI Card 1
	A quick look at the hardware 1
	A quick look at the firmware 2
	Using the SCSI card as a startup device 2
<b>Chapter 2</b>	<b>The Hardware 3</b>
	Functional description of the SCSI card 5
	Apple II microprocessor 5
	Apple II internal bus 6
	Bank select logic 6
	RAM 6
	ROM 7
	CPU-5380 interface logic 7
	NCR 5380 SCSI IC 7
	The Apple II SCSI bus 8
<b>Chapter 3</b>	<b>The Firmware 11</b>
	SCSI management 11
	Device tables 12
	SDATs 12
	SDAT example 13
	DIBTABs 13
	DIBTAB example 16
	Interpretation 16
	Making a ProDOS call 17
	Command parameters zero-page write 17
	ProDOS call 18
	Making a SmartPort call 18
	SmartPort location 18
	SmartPort call 19
	SmartPort command definitions 20
	Status (\$00) 20
	Read Block (\$01) 22
	Write Block (\$02) 23
	Format (\$03) 23
	Control (\$04) 24
	Init (\$05) 33

Open (\$06)	33
Close (\$07)	33
Read (\$08)	34
Write (\$09)	34
Summary of error codes	35
Sample program	36

**Appendix A Device Partitioning 45**

Creating device partitions	45
The Device Control Block (DCB)	45
The Device Information Block (DIB)	46
The Driver Descriptor Map (DDM)	46
The Device Partition Map (DPM)	47
The Partition Descriptor Map (PDM)	47

**Appendix B Using Unsupported SCSI Commands 53**

Select the SCSI card	53
Set up device tables (SDAT/DIBTAB)	54
Load the command block	54
Call the SCSI management routines	54
Wait for next bus phase	57
Check the command execution status	58

**Glossary 59**

**Index 63**

---

---

## Figures and tables

### Chapter 2 The Hardware 3

Figure 2-1	Unit number assignment by ProDOS partition	4
Figure 2-2	Apple II SCSI configuration	4
Figure 2-3	SCSI card block diagram	5
Figure 2-4	Bank select register data structure	6
Figure 2-5	RAM map	6
Figure 2-6	ROM map	7
Table 2-1	Device select space I/O address map	8
Table 2-2	System cable pin-out chart	9
Table 2-3	SCSI card cable pin-out chart	10

### Chapter 3 The Firmware 11

Figure 3-1	SDATAT data structure	12
Figure 3-2	DIBTAB data structure	14
Figure 3-3	SmartPort call command block data structure	19
Table 3-1	SDAT software ID codes	13
Table 3-2	Apple device type codes	15
Table 3-3	Apple device subtype codes	15
Table 3-4	Microprocessor register state	18
Table 3-5	General status byte contents	21
Table 3-6	SCSI bus status bytes contents	22
Table 3-7	Device-specific status bytes	22
Table 3-8	Control codes	25
Table 3-9	Error codes	35

### Appendix A Device Partitioning 45

Figure A-1	DDM data structure	46
Figure A-2	Driver Descriptor data structure	47
Figure A-3	DPM data structure	47
Figure A-4	PDM data structure	48
Figure A-5	Status byte data structure	49

### Appendix B Using Unsupported SCSI Commands 53

Table B-1	ROM entry points for SCSI management routines	55
Table B-2	Data transfer mode selection address	55
Table B-3	SCSI bus condition	57







# Preface

The Apple® II SCSI Card allows you to connect **SCSI**-compatible peripheral devices, like disk or tape drives, to your Apple II family computer. Depending on the operating system running on your Apple II, you can connect from four to seven devices to the SCSI card. Chapter 1 contains more information about the number of SCSI devices you can connect to your Apple II.

The SCSI card contains **firmware** that performs all the operations required to use and control SCSI devices, under the control of the Apple II microprocessor. The firmware is described in Chapter 3.

This manual is written for anyone with a desire to know more about the operation of their SCSI card. In an effort to make using this manual easier, it has been divided into five major sections. The following paragraphs briefly describe what information you can expect to find in each of these sections.

Chapter 1 introduces you to the Apple II SCSI card. In this chapter you'll find a brief explanation of what SCSI is, and how Apple has made use of this industry standard. You'll also get an overview of how the SCSI card hardware and firmware work with the Apple II microprocessor and with SCSI devices.

Chapter 2 describes the functional operation of the SCSI card and the general architecture of the Apple II SCSI bus. This chapter describes the electrical interface between the SCSI card and the Apple II CPU, as well as that between the SCSI card and SCSI devices on the external bus.


Chapter 3 describes the operation of the firmware. In this chapter you'll find the information you need to find and use the firmware, including descriptions of the SmartPort and ProDOS® command sets.

Appendix A explains how to set up SCSI block-type devices, like hard disk drives, for multiple-operating system use. This type of setup operation is called *device partitioning*.

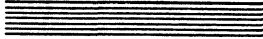
Appendix B provides information on using nonstandard device commands.

A glossary of terms related to SCSI and the SCSI card is included at the back of this manual. Terms set off in boldface type in this manual are defined in the glossary.





# Chapter 1



## Introduction to the Apple II SCSI Card

The abbreviation SCSI stands for *Small Computer System Interface*. The Small Computer System Interface is based on the ANSI **standard** for small computer systems. Because the SCSI standard defines all the aspects of a computer interface, a detailed description of the standard itself is beyond the scope of this manual. If you want to know more about the SCSI standard, see the *SCSI American National Standard for Information Systems (ANSI, X3T9.282-2)*.

The idea behind the SCSI standard is very simple: Rather than designing products that are compatible with only one or two small computers, a developer designs products that are compatible with *any* small computer. The developer accomplishes this by designing the device to work with the **interface**, rather than the actual computer. Not only does this free developers from worrying about various computer manufacturers' hardware designs, it gives the owners of small computers a tremendous amount of flexibility in selecting equipment for use in their systems.

---

---

### The Apple II SCSI Card

The SCSI standard is implemented in Apple® II family computers by the Apple II SCSI Card and the firmware it contains. The SCSI card will work with any SCSI device, *provided the device actually conforms to the SCSI standard*.

---

### A quick look at the hardware

The Apple II SCSI Card is a self-contained peripheral card for use in all of the Apple II family computers that have expansion slots. The SCSI card contains on-board RAM and ROM. Special logic circuits are used to access and control the RAM and ROM. The SCSI bus is controlled by a NCR 5380 SCSI IC. The SCSI card also contains circuitry that provides an interface between the 5380 IC and the Apple II CPU.

The SCSI card supports four external devices, two each in slots 2 and 5, when mounted in an Apple II computer running ProDOS® 16, version 1, or ProDOS 8, version 1.2. With ProDOS 8, version 1.1, the SCSI card supports two external devices. Future versions of ProDOS will support up to seven external devices.

---

**Important**

Although it is using two slots, the SCSI card is physically installed in only one slot; the card "takes over" the other slot's I/O space in order to support another two external SCSI devices. The use of two slots by a single peripheral card is called **shadowing**.

---

The SCSI card performs data I/O operations in pseudo-DMA (PDMA) mode.

The SCSI card hardware has two principal functions:

- It provides an electrical interface between external SCSI devices and the host computer.
- It provides the address and control lines required by the Apple II microprocessor to access and control the NCR 5380 and the firmware.

More information about the SCSI card hardware is provided in Chapter 2.

---

**A quick look at the firmware**

The firmware on the SCSI card is a program that handles all of the communications protocol and bus management for Apple II SCSI. The firmware also provides a means for the operating system, or an application program, to access and control devices on the SCSI bus without requiring a highly-specialized SCSI driver program. This feature converts **calls** from the operating system or an application program to SCSI commands and manages their execution. The firmware calls are described in Chapter 3.

---

**Using the SCSI card as a startup device**

The Apple II SCSI card is recognized as a bootable device by the Autostart ROM, so all you need to do to use it as a startup device is install it in the slot with the highest boot priority.

If the SCSI card is installed in the slot with the highest boot priority, the firmware initializes the SCSI bus and attempts to boot the operating system off a SCSI device.

---

**Important**

In the Apple II, the slot with the highest boot priority is the highest numbered slot. Boot priority thus descends from slot 7 through slot 1, with slot 1 the lowest priority slot. However, a device must still be recognized as a *valid boot device* by the operating system to be considered the highest priority boot device.

---

If there is more than one device on the SCSI bus, the device with the highest SCSI ID number (see Chapter 2) is tried first. If this device is not a valid boot device, the firmware returns control to the Apple II, and the boot search continues through lower priority slots.



## Chapter 2



# The Hardware

The Apple II SCSI bus is a peripheral-device bus capable of supporting a maximum of eight SCSI devices. The Apple II SCSI Card itself counts as one device, so you can connect up to seven *external* devices to the bus. A device that is connected to the SCSI bus is said to be *resident* on the bus.

The external SCSI devices are daisy-chained together with cables. These cables form the physical SCSI bus. There are three different cables used to connect devices:

- The system cable connects the first *external* device to the SCSI card's 25-pin back-panel connector.
- The peripheral interface cable connects all of the other SCSI devices in the chain together (one cable is required for each device).
- The cable extender is an inline bus extender that increases the length of a peripheral interface cable or the system cable by three feet.

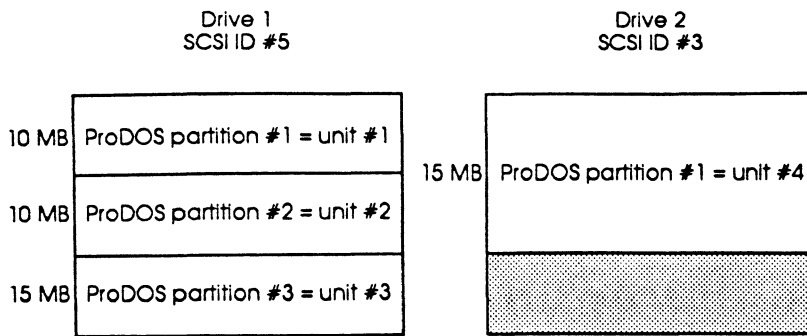
The devices resident on the SCSI bus are identified by two numbers. The first number, called the **SCSI ID number**, is set by the position of a strap or switch on the device, or by configuration software. Each device has a unique SCSI ID number. SCSI ID numbers range from 7 (the highest) to 0 (the lowest).

A device's SCSI ID number determines its priority with respect to the other devices on the bus. The device with the highest unit number (7) has the highest priority, while the device with the lowest unit number (0) has the lowest priority.

The Apple II SCSI Card's SCSI ID number is set by positioning a jumper on a strip of pins. With the jumper positioned across pins 1 and 16, the SCSI card's ID number is set to 7. With the jumper across pins 8 and 9, the SCSI card's ID number is set to 0. For most applications, the SCSI card should be set to ID number 7 (jumper across pins 1 and 16).

The second number, called the **unit number**, is set by the card firmware during bus initialization. The firmware sets the unit number based on the SCSI ID number, as follows:

The first ProDOS partition of the device with the highest SCSI ID is unit number 1. The second ProDOS partition on this same device is assigned unit number 2, and so on for all valid ProDOS partitions (up to the maximum of 7) on the same device. If there are no ProDOS partitions on the device, or if all of the available space on the device is a single partition, the entire device is assigned unit number 1. The device with the next-highest SCSI ID number is then assigned unit numbers in exactly the same manner, starting with unit number  $n+1$ , where  $n$  is the unit number of the last partition on the previous device.



**Figure 2-1**  
Unit number assignment by ProDOS partition

For example:

A 40 MB and a 20 MB hard disk are resident on the SCSI bus. The 40 MB drive is SCSI ID #5, and the 20 MB drive is SCSI ID #3. There are two 10 MB and one 15 MB ProDOS partitions on the 40 MB drive and one 15 MB ProDOS partition on the 20 MB drive. The firmware will assign unit numbers as follows:

40 MB drive (ID #5): unit numbers 1 and 2 for the two 10 MB partitions (assuming they are the first two ProDOS partitions), and unit number 3 for the 15 MB ProDOS partitions.

Thus,  $n=3$ , so  $n+1=4$ ; the first ProDOS partition on the 20 MB drive will be assigned unit number 4.

20 MB drive (ID# 3): unit number 4 for the 15 MB partition.

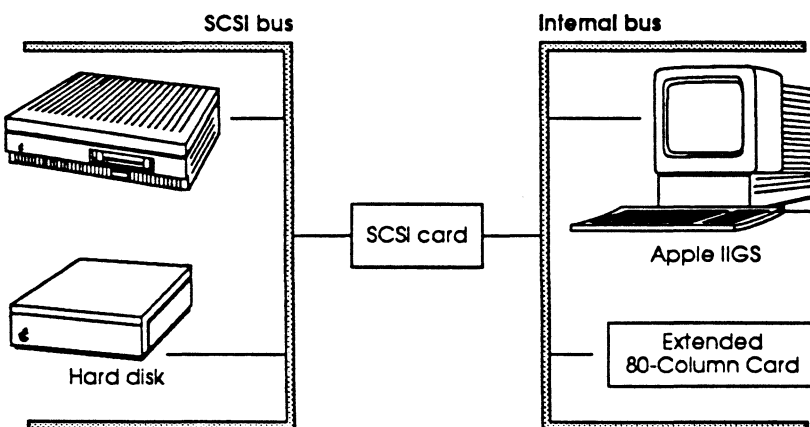
---

**Important**

A device's physical location in the chain does *not* determine its unit number.

---

The SCSI card is resident on both the Apple II internal bus and the SCSI bus. (See Figure 2-2.) However, the Apple II accesses and controls the SCSI card across the internal bus, as it would any other peripheral card.



**Figure 2-2**  
Apple II SCSI configuration

External devices reside on the SCSI bus only. The Apple II cannot access these devices directly because they are not on the internal bus. To perform any I/O or control operation on an external SCSI device, the Apple II must send a command to the SCSI card. The SCSI card firmware then executes the operation (I/O or control) requested by the Apple II. More information about the role of the firmware is given in Chapter 3.

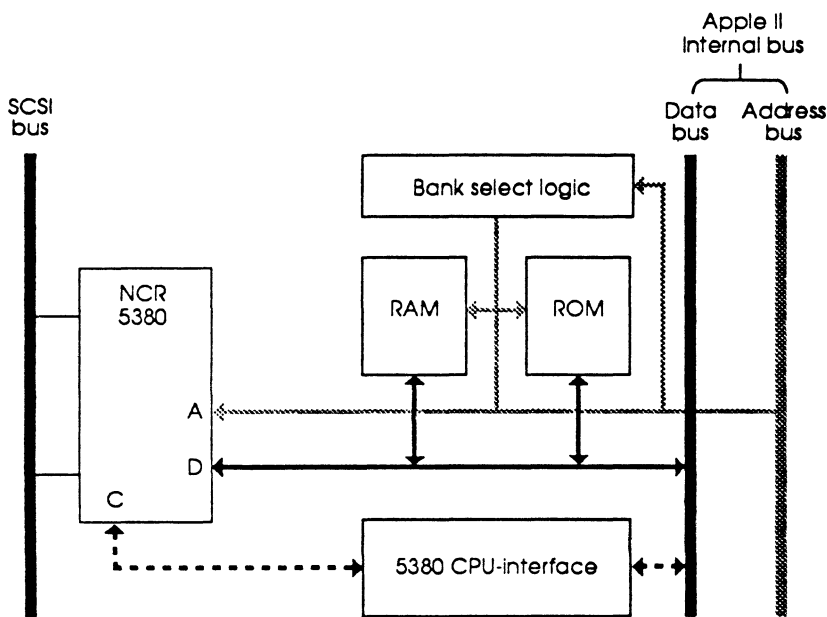
---



---

## Functional description of the SCSI card

The following sections describe the basic tasks performed by each of the functional blocks shown in Figure 2-3.



Note: A = address line, D = data line, C = control line

**Figure 2-3**  
SCSI card block diagram

---

## Apple II microprocessor

The Apple II microprocessor controls the access to, and operation of, the SCSI card and the external devices connected to it. It does so by reading and writing the 5380 IC's control registers. The microprocessor also sends all the addressing signals used by the SCSI card RAM and ROM.

---

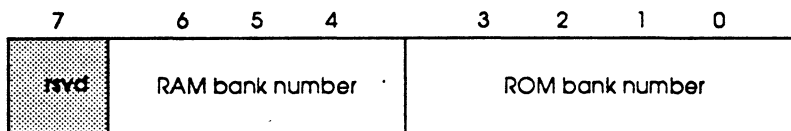
## Apple II internal bus

The Apple II internal bus is the standard I/O and control bus used by the Apple II microprocessor. The SCSI card is connected to the internal bus by the card edge connector that fits into the slot on the Apple II logic board. The Apple II internal bus is described in the technical reference manual for your Apple II.

---

## Bank select logic

The on-board ROM and RAM are each mapped into 1K banks in the Apple II's I/O address space at \$C800-\$CFFF. To select a specific bank, the microprocessor writes the bank address into the memory bank select register at \$C0nA, where *n* is the SCSI card slot number plus eight. Figure 2-4 defines the data structure of the bank select register.

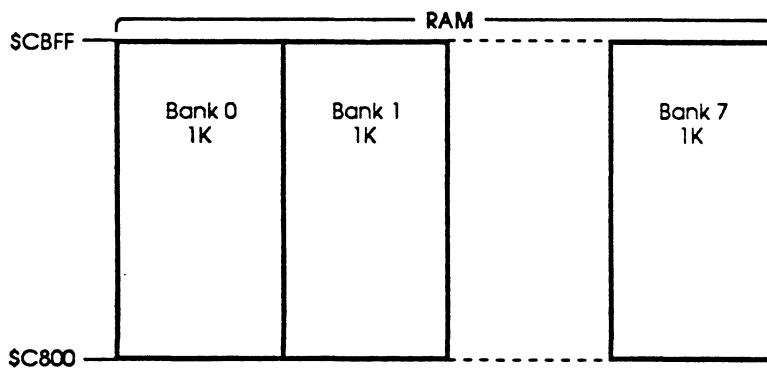


**Figure 2-4**  
Bank select register data structure

---

## RAM

The SCSI card contains 8K of static RAM in a single 8Kx8 IC. The RAM is divided by the bank select logic into eight 1K banks and mapped into the Apple II main memory I/O space at \$C800-\$CBFF. Card RAM is used to store information about the external SCSI devices. Figure 2-5 is a map of the RAM.



**Figure 2-5**  
RAM map

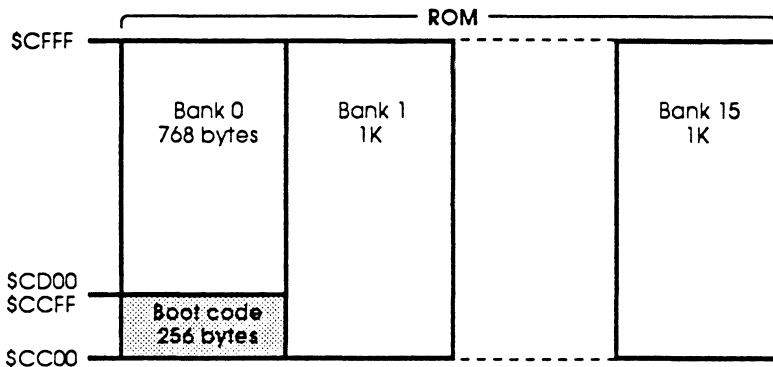


---

## ROM

The SCSI card contains 16K of ROM in a single 16Kx8 UVEPROM. Like the RAM, the ROM is divided into eight selectable 1K banks. Except for the lower 256 ROM locations, the ROM banks are mapped into the Apple II main memory I/O space at \$C000-\$CFFF. The lower 256 ROM locations are mapped into the boot space at \$Cn00-\$CnFF, where  $n$  is the slot number for the SCSI card. Bank selection is controlled by the microprocessor through the bank select logic. The SCSI card's ROM is used to store the firmware. Whenever a call is issued to the firmware, whether from the operating system or from a software program, the SCSI card ROM is read.

Figure 2-6 is a map of the ROM.



**Figure 2-6**  
ROM map

---

## CPU-5380 interface logic

The control signals issued to the NCR 5380 SCSI IC by the Apple II microprocessor are handled by the CPU-5380 interface logic. This logic decodes control and address signals sent across the Apple II internal bus and, in turn, sets the 5380 control inputs to the correct state. Responses from the 5380 are also passed back to the CPU through this logic.

---

## NCR 5380 SCSI IC

The NCR 5380 IC provides all the bus control and device protocol required for a SCSI bus. The 5380 IC is the physical I/O point for all external devices connected to the Apple II SCSI bus. All the data and control signals sent to external SCSI devices by the microprocessor or software are handled by the 5380 IC. The 5380 IC internal registers are addressed as memory-mapped I/O space at \$C0n0-\$C0n7 (the device select space), where  $n$  is the SCSI card slot number plus 8.

Table 2-1 describes the device select space I/O address map.

**Table 2-1**  
Device select space I/O address map

I/O address	R/W	Name
\$C0n0	R	Current SCSI data register
\$C0n0	W	Output data register
\$C0n1	R/W	Initiator command register
\$C0n2	R/W	Mode Select register
\$C0n3	R/W	Target command register
\$C0n4	R	SCSI bus status
\$C0n4	W	Select enable register
\$C0n5	R	Bus and Status register
\$C0n5	W	Not used
\$C0n6	R	Input data register
\$C0n6	W	Not used
\$C0n7	R	Reset parity/interrupts
\$C0n7	W	Not used
\$C0n8	R/W	PDMA/DACK
\$C0n9	R	SCSI device ID
\$C0nA	W	Memory Bank Select register
\$C0nB	W	Reset 5380 IC
\$C0nC	W	Not used
\$C0nD	W	PDMA mode enable
\$C0nE	R	Read DRQ status bit through D7 bit
\$C0nF	R/W	Not used

---

## The Apple II SCSI bus

The Apple II SCSI bus is a standard SCSI peripheral device bus, as defined in the ANSI specification referred to earlier in this manual. Bus control and device protocol are provided by the NCR 5380 SCSI IC. Physical support for the SCSI bus is provided by the system cable. The system cable is terminated with a DB-25 connector on the Apple II side and a 50-pin connector on the SCSI device side.

The SCSI card itself is connected to the DB-25 connector on the back panel of the Apple II by a 25-pin ribbon cable.

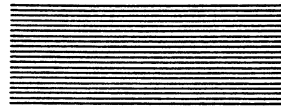
Table 2-2 is a pin-out chart for the system cable, and Table 2-3 is a pin-out chart for the 25-pin ribbon cable.

**Table 2-2**  
System cable pin-out chart

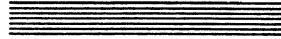
50-Pin connector	Signal	DB-25 connector
1	DB0-GND	14
2	DB1-GND	14
3	DB2-GND	14
4	DB3-GND	16
5	DB4-GND	16
6	DB5-GND	16
7	DB6-GND	18
8	DB7-GND	18
9	DBP-GND	18
11	DIFFSENS-GND	18
16	ATN-GND	7
18	BSY-GND	7
19	ACK-GND	7
20	RST-GND	9
21	MSG-GND	9
22	SEL-GND	9
23	C/D-GND	24
24	REQ-GND	24
25	I/O-GND	24
26	-DB0	8
27	-DB1	21
28	-DB2	22
29	-DB3	10
30	-DB4	23
31	-DB5	11
32	-DB6	12
33	-DB7	13
34	-DBP	20
38	TERMPWR	25
41	-ATN	17
43	-BSY	6
44	-ACK	5
45	-RST	4
46	-MSG	2
47	-SEL	19
48	-C/D	15
49	-REQ	1
50	-I/O	3

**Table 2-3**  
SCSI card cable pin-out chart

26-pin P.C.B. connector	Signal	DB-25 connector
1	-REQ	1
2	GND	14
3	-MSG	2
4	-C/D	15
5	-I/O	3
6	GND	16
7	-RST	4
8	-ATN	17
9	-ACK	5
10	GND	18
11	-BSY	6
12	-SEL	19
13	GND	7
14	-DBP	20
15	-DB0	8
16	-DB1	21
17	GND	9
18	-DB2	22
19	-DB3	10
20	-DB4	23
21	-DB5	11
22	GND	24
23	-DB6	12
24	TPWR	25
25	-DB7	13
26	No connection	



## Chapter 3



# The Firmware

The Apple II SCSI Card firmware provides the software interface between the Apple II (and any program running on it) and the devices resident on the SCSI bus.

This interface has two primary functions: management and interpretation. The management functions support SCSI command execution, SCSI message protocol, and SCSI bus arbitration. The interpretation feature provides a simplified command interface for use by application programs that do not include SCSI device drivers.

---

### Important

The firmware commands described in this chapter make use of special data variables called parameters. All parameters have names associated with them. Whenever a parameter is being described, its name is set off in italics, as shown here:

*enable* is set to 1.

---

---

---

## SCSI management

The SCSI card firmware performs all of the read/write and record-keeping operations required by the SCSI message protocol using the on-board RAM. In this sense, the firmware acts like a mail carrier, taking mail (a SCSI message) from one person (a device) and delivering it to another person (another device on the bus). Along the way, the mail carrier (the firmware) logs the mail in and out of the post office, cancels the postage, and sorts the mail out for easier distribution (record-keeping).

The firmware supports SCSI bus arbitration by maintaining the device tables (described later in this section).

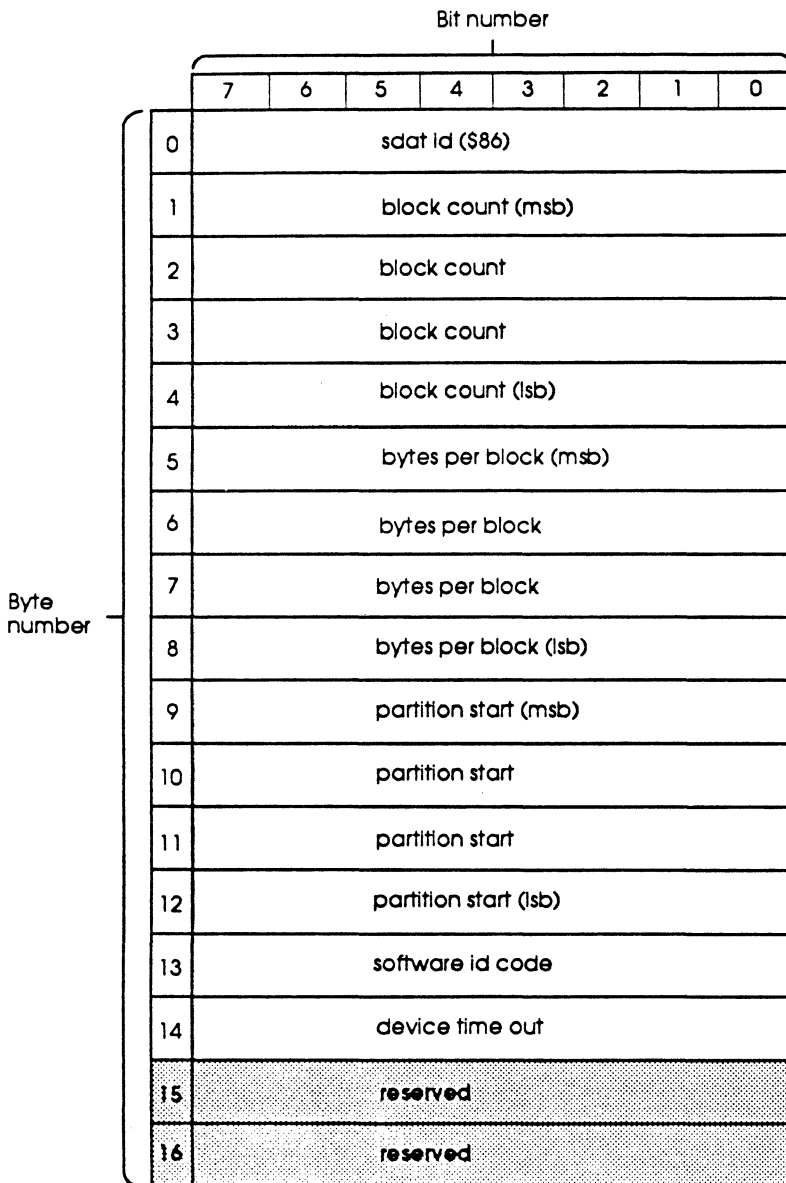
The SCSI message protocol and the SCSI bus arbitration scheme are described in the ANSI standard referred to in Chapter 1.

## Device tables

In order to manage SCSI devices, the firmware needs to store certain information where it can be quickly accessed, namely the SCSI card RAM. The firmware stores this critical information in two software tables, the SCSI Device Access Table (SDAT) and the Device Information Block Table (DIBTAB). Each SCSI device has an SDAT and a DIBTAB constructed for it by the firmware at SCSI bus initialization.

### SDATs

The SDATs are loaded into SCSI card RAM beginning at \$C831 and grow upwards in RAM to \$C897. Each SDAT is 17 bytes long. Figure 3-1 defines the SDAT data structure. All entries are in hex unless otherwise specified.



**Figure 3-1**  
SDAT data structure

**sdatt id:** This field contains a hexadecimal code identifying the bytes that follow as an SDAT. This field is always set to \$86.

**block count:** This field contains the number of blocks available on the device described in the SDAT.

**bytes per block:** This field contains the size, in bytes, of each block on the device described in the SDAT.

**partition start:** This field contains the starting block address of the partitions on the device described in the SDAT.

**software id code:** This field contains a hexadecimal code for the SCSI ID number of the device described in the SDAT. The software codes for SCSI unit numbers are shown in Table 3-1.

**Table 3-1**  
SDAT software ID codes

Code	ID number
\$01	0
\$02	1
\$04	2
\$08	3
\$10	4
\$20	5
\$40	6
\$80	7

**device timeout:** This field contains a hexadecimal code that defines the timeout parameter for the device described in the SDAT.

### SDAT example

Here is an example of an SDAT:

```
C831: 86
C832: 00 01 49 45 ;Total blocks in device, msb first
C836: 00 00 02 00 ;Bytes per block field
C83A: 00 00 00 40 ;Partition start offset, msb first
C83E: 20 ;SCSI device ID.
C83F: 08 ;Device specific timeout constant
```

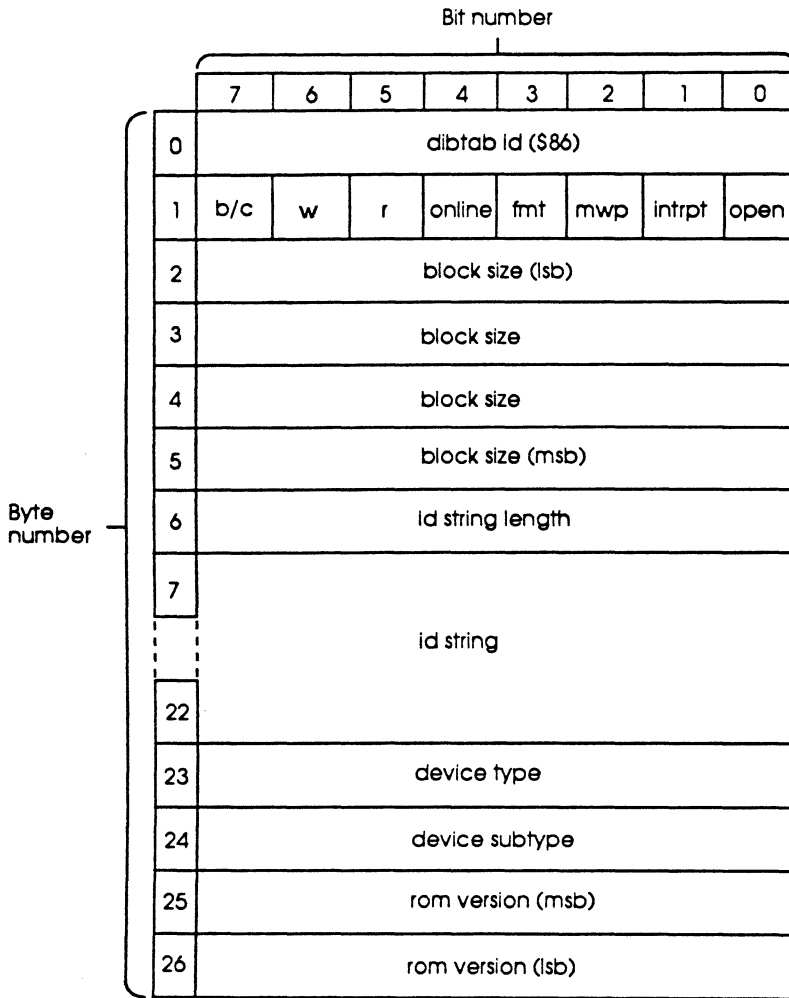
This SDAT describes a device with a size of 84,293 blocks. Each block on the device consists of 512 bytes. Device partitions begin at block 64. The device number is set to 5 on the device, because the software ID code is \$20.

### DIBTABs

The DIBTAB contains status information describing a specific device on the SCSI bus. DIBTAB information can be obtained from the device itself by reading out the Device Information Block (DIB) using the Status (\$00) SmartPort call with *status code* set to \$03.

If the target device does not have a DIB, you can build one by using the INQUIRY (\$0F), MODE SENSE (\$09), and REQUEST SENSE (\$06) SmartPort control calls to obtain the necessary information, and then writing the appropriate data back to the device in the block reserved for the DIB. See the SmartPort Control call description, later in this chapter, for more information.

DIBTABs are loaded into SCSI card RAM beginning at \$C931. Each DIBTAB is 27 bytes long. Figure 3-2 defines the DIBTAB data structure.



**Figure 3-2**  
DIBTAB data structure

**dibtab id:** This field contains a hexadecimal code identifying the bytes that follow as a DIBTAB. This field is always set to \$86.

**device status:** This field contains eight 1-bit flags that provide information related to the status of the target device, as described in the following paragraphs.

- b/c:** This field contains the block/character device flag. When set to 1, it indicates the device is a block-type device. When set to 0, it indicates the device is a character-type device.
- w:** This field contains the write allowed flag. When set to 1, it indicates that write operations are accepted by the device. When set to 0, it indicates that write operations are not accepted by the device. This flag applies *only to the device itself*. Certain blocks on the device may be reserved, or otherwise protected.
- r:** This field contains the read allowed flag. When set to 1, it indicates that read operations are accepted by the device. When set to 0, it indicates that read operations are not accepted by the device. This flag applies *only to the device itself*. Certain blocks on the device may be reserved, or otherwise protected.



- **online:** This field contains the on-line flag. When set to 1, it indicates that the device is on-line. When set to 0, it indicates the device is not on-line.
- **fmt:** This field contains the format allowed flag. When set to 1, it indicates that format operations are accepted by the device. When set to 0, it indicates that format operations are not accepted by the device. This flag applies *only to the device itself*. Certain blocks on the device may be reserved, or otherwise protected.
- **mwp:** This field contains the medium write-protected flag. When set to 1, it indicates the data medium mounted on the device is write-protected.
- **Intrpt:** This field contains a flag used by the Apple IIc (which *does not* support SCSI) for SmartPort block device I/O; it is not valid for the SCSI card or any SCSI device.
- **open:** This field contains the device open flag. For character devices, this flag set to 1 indicates that the device is logically open. When set to 0, it indicates that the device is logically closed. For block devices, this flag set to 1 indicates a disk switch took place. When set to 0, it indicates that no switch took place.

**block count:** This field contains the number of logical blocks available on the device.

**id string length:** This field contains the size, in bytes, of *id string*.

**id string:** This field contains an ASCII string identifying the device. This field *must* be 16 bytes long; pad any extra bits with space characters.

**device type:** This field contains a hexadecimal code identifying the general class of the device. The device type codes are shown in Table 3-2.

**Table 3-2**  
Apple device type codes

Code	Device type
\$03	Nonspecific SCSI
\$05	CD-ROM
\$06	Direct-access tape drive
\$07	Hard disk
\$08	Scanner
\$09	Printer

**device subtype:** This field contains a hexadecimal code providing additional information on the type of the device. The device subtype codes are shown in Table 3-3.

**Table 3-3**  
Apple device subtype codes

Bit number	Function
0-4	Reserved
5	If set to 0, device medium is removable If set to 1, medium is not removable
6	If set to 1, device supports disk-switched errors
7	If set to 1, device supports extended format SmartPort calls

**rom version:** This field contains a hexadecimal code that defines the firmware version installed on the SCSI card. The Rev. C SCSI card ROM version is \$00 02.

### DIBTAB example

Here is an example of a DIBTAB:

```
C931: 86
C932: F8                ;status byte
C933: 45 49 01 00      ;block count
C937: 10                ;ID string length
C938: 51 55 41 4E 54 55 4D 51 ;'QUANTUM Q'
C940: 32 35 30 37 36 30 34 35 ;'25576-45'
C948: 07                ;type byte
C949: 80                ;subtype byte
C94A: 00 02            ;SCSI Rom rev number
```

This DIBTAB describes an 40 MB Quantum Q hard disk, model number 25576-45, identified as a SCSI hard disk that supports extended-format SmartPort calls. The drive supports SCSI card Rev. C ROM. The status byte indicates that this device

- has not had a disk switch take place
- is not write-protected
- can be formatted
- is on-line
- can be read from
- can be written to
- is a block device

---

### Interpretation

In order to access and control a SCSI device, the host must be able to send and receive SCSI commands from either the operating system or the active application program. Rather than force each application or operating system to provide all of the functions necessary to send and receive SCSI commands, Apple has created a command interpreter in the SCSI card's firmware. This interpreter accepts calls from either the ProDOS operating system or the SmartPort I/O interface. Thus, an application program need only call the SmartPort, or ProDOS, and leave the rest of the details to the command interpreter.

Requests from ProDOS or the SmartPort are much like routines, in that the command interpreter takes a simple input, such as Status, and uses several lower-level SCSI commands, such as MODE SENSE (\$1A) and REQUEST SENSE (\$03), to accomplish the operation requested. The command interpreter receives the command request from ProDOS or the SmartPort and proceeds to select and execute the SCSI commands required to carry it out.

Even with the command interpreter, you must still provide all of the data necessary to accomplish the operation requested. For instance, the interpreter would select the correct commands to accomplish a Write Block (\$02) call, but would not be able to provide the data to write to the target device unless you had specified the location of the outgoing data in the original SmartPort call.

A ProDOS or SmartPort call includes a **command number**, a **pointer**, and a **parameter list**.

A command number is a hexadecimal number that indicates to the interpreter the operation being requested. Each operation has a unique command number. SCSI commands also have a command number, but this number is not directly related to the command number sent to the interpreter in a ProDOS or SmartPort call. *Do not confuse SCSI command number with ProDOS/SmartPort command numbers.*

A pointer is a data field that contains the address in RAM of another field, such as the parameter list.

A parameter list is a set of related data fields loaded into RAM at a specific address. Parameter lists can contain special codes, pointers to RAM locations loaded with outgoing data blocks, control variables, or any other information the command requires to be executed properly.

Basically, you build the command in Apple II RAM, load the RAM address of your command into the pointer field, make the appropriate call to the command interpreter via ProDOS or the SmartPort, and wait for the call to finish.

Once a call is issued, the Apple II turns control over to the interpreter until the requested operation is done. When the interpreter completes the operation, it returns control to the program issuing the original call. Program execution resumes at the address immediately following the parameter list pointer for the call.

---

---

## Making a ProDOS call

The operating system or program running on the Apple II executes a ProDOS call to the SCSI card firmware in two basic stages:

1. Command parameters zero-page write
2. ProDOS call

---

### Command parameters zero-page write

Prior to issuing a firmware call, the operating system passes a set of command parameters to the SCSI card. These parameters are passed in zero-page locations \$42-\$47, as follows:

\$42	Command number
\$43	Unit number
\$44-\$45	Buffer pointer
\$46-\$47	Block number

The command number is the firmware command number. The unit number indicates the slot and drive numbers of the target SCSI device, with the slot number set in bits 4 and 5, and the drive number set in bits 6 and 7. The buffer pointer indicates the start of a 512-byte data buffer. The block number is the address of the target block on the SCSI device. If the command is not a read/write operation, no block number is required.

---

## ProDOS call

The second stage is the actual ProDOS call. A ProDOS call is coded as a JSR to the entry point.

You can calculate the ProDOS entry point as follows:

$$\$Cn00 + (CnFF)$$

where  $n$  is the SCSI card slot number and  $CnFF$  is the value of the byte located at  $\$CnFF$ .

When the firmware has completed the operation requested by the call, it sets certain flags in the microprocessor's Status register and accumulator (A register). The state of these flags depends on whether or not the ProDOS call was successfully completed. Table 3-4 defines the state of the microprocessor flags for both successful and unsuccessful ProDOS calls.

**Table 3-4**  
Microprocessor register state

Bit	Successful call	Unsuccessful call
N	x	x
Z	x	x
C	0	1
D	0	0
V	x	x
1	Unchanged	Unchanged
B	x	x
Xreg	x	x
Yreg	x	x
Acc	0	Error code
PC	JSR+3	JSR+3
SP	Unchanged	Unchanged

---

---

## Making a SmartPort call

The operating system or program running on the Apple II executes a call to the SmartPort in two basic stages:

1. SmartPort location
2. SmartPort call

---

### SmartPort location

Before you issue a call to the SmartPort, it's a good idea to make sure that the SmartPort is present on the SCSI card. This step helps assure compatibility between software and future hardware developments.

To locate the SmartPort, search for the following bytes:

```
$Cn01 $20 (where n is the SCSI card slot number)
$Cn03 $00
$Cn05 $03
$Cn07 $00
```

An additional byte, at \$CnFB, should contain \$82, indicating that the device is the SCSI card (\$2) and that it supports extended calls (\$8).

## SmartPort call

The next stage is the actual SmartPort call. A SmartPort call is coded as a JSR to the SmartPort entry point (DISPATCH), followed by the 1-byte command number, followed by the 2-byte command parameter-list pointer.

You can calculate the DISPATCH address as follows:

$$\$Cn00 + (CnFF) + 3$$

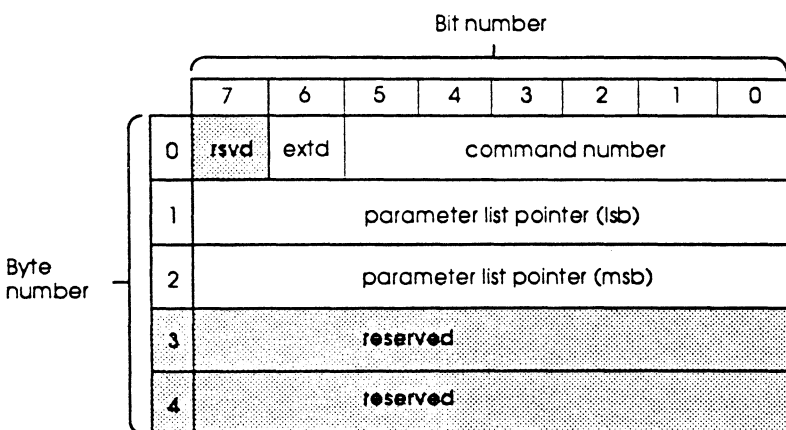
where *n* is the SCSI card slot number and (*CnFF*) is the value of the byte located at \$CnFF.

The following is an example of a nonextended SmartPort call:

```
PCCALL JSR DISPATCH ;Call SMP entry point and dispatcher
        DFB CMDNUM   ;SMP command number
        DW  CMDLIST  ;Command parameter list pointer
        BCS ERROR    ;Carry is set on an error
```

Extended calls include a 4-byte field for the parameter list pointer. Extended call command numbers are calculated by adding \$40 to the nonextended command number. For example, a nonextended Read Block call is coded as \$01, so the extended Read Block call is coded as \$41. Figure 3-3 is a data diagram of the SmartPort command block.

For extended calls, the two bytes shown as reserved in Figure 3-3 are used for the extended address field, so that byte 4 becomes the msb of the address field, while bytes 2 and 3 become the middle bytes of that field.



**Figure 3-3**  
SmartPort call command block data structure

**rsved:** All reserved fields must be set to \$0.

**extd:** This field contains the extended call flag. If set to 1, it indicates that an extended call is being made. If set to 0, it indicates that a nonextended call is being made.

**command number:** This field contains the SmartPort command number for the call, as described earlier in this chapter.

**parameter list pointer:** This field contains the parameter list pointer, as described earlier in this chapter. The parameter lists for each call are described in the section covering that call. This field includes the two reserved bytes for extended calls.

When the SmartPort has completed the operation requested by the call, it sets certain flags in the microprocessor's Status register and accumulator (A register). There are two differences between the states of the microprocessor register flags after a SmartPort call and those given in Table 3-4:

- The X and Y registers (Xreg and Yreg) are set to the number of bytes of data transferred in successful device-to-host transfers; unsuccessful and host-to-device transfers remain undefined.
- The PC register is set to JSR+5 for extended SmartPort calls.

---

---

## SmartPort command definitions

The following sections describe each SmartPort command in detail. Each definition contains the following information:

- command name: the name of the SmartPort command
- command number: the hexadecimal number of the SmartPort command
- description: the purpose of the command
- parameter list: the command parameter list required by or for the command, by byte number (given in parentheses)
- parameter description: describes the parameters in the parameter list

---

### Important

Extended SmartPort calls have address pointers that are twice as long (4 bytes) as nonextended calls. When using an extended call format, remember that the byte numbers given in the following descriptions are for nonextended calls; refer to the data diagrams given earlier for the correct byte-order.

---

---

## Status (\$00)

The Status command returns information about a specific device on the SCSI bus. The information returned by this command is determined by *status code*.

On return from a Status call, the microprocessor X and Y registers are set to indicate the number of bytes transferred to the Apple II by the command. The X register is set to the low byte of the count, and the Y register is set to the high byte.

The parameter list for this call is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	status list pointer (lsb-msb)
4	status code

### Parameter description

**unit number:** This parameter contains the SmartPort unit number of the target SCSI device. If the unit number is set to \$00, all devices resident on the SCSI bus are targeted. See Chapter 2 for details on selecting the correct SmartPort unit number for your target device.

**status list pointers:** This parameter contains the address of the data buffer used to store the device status information returned by this command. The first byte (byte 2) is the low byte, and the second byte (byte 3) is the high byte of the address.

**status code:** This parameter contains the hexadecimal code number indicating which status request is being issued, as follows:

\$00	Return device status
\$01	Not supported
\$02	Not supported
\$03	Return Device Information Block (DIB)
\$04	Return Device Information Block (DIB), extra
\$05	Return last error status
\$06	Return bytes/block parameter for device

**Code \$00:** The status list returned for this code is 4 bytes long (5 bytes for extended format). The first byte is the general status byte, and the remaining 3 (4 extended) bytes are the size of the device in blocks. The general status byte is returned for all device types, but the size bytes are only returned for block-type devices. Table 3-5 defines the information returned in the general status byte.

**Table 3-5**  
General status byte contents

Bit	Definition
7	1=Block device, 0=Character device
6	1=Write allowed
5	1=Read allowed
4	1=Device on-line
3	1=Format allowed
2	1=Media write-protected (block device)
1	1=Reserved
0	1=Device open (character device) 1=Disk switch took place (block device)

- ❖ **Unit number \$00:** A code \$00 Status command with the unit number set to \$00 returns the status of the SCSI bus. The status list returned is 8 bytes long. Table 3-6 defines the contents of the SCSI bus status list.

**Table 3-6**  
SCSI bus status bytes contents

Byte	Definition
0	Number of SCSI devices on SCSI bus
1-7	Reserved

**Code \$01-\$02:** These codes return error code \$21.

**Code \$03:** The status list returned for this code is 25 bytes long (26 bytes extended). The first 4 bytes are identical to the bytes returned for code \$00. The remaining bytes are device-specific status information. Table 3-7 defines the information returned in the device-specific status bytes.

**Table 3-7**  
Device-specific status bytes

Byte	Definition
1	General status byte
2	Block count (lsb)
3	Block count
4	Block count (msb)
5	Length of device ID string in bytes (16 max)
6-21	Device name, in ASCII (16 bytes)
22	Device type
23	Device subtype
24-25	ROM version number

❖ *Note:* For extended calls, byte 5 becomes the msb of the block count, and all subsequent fields “move” 1 byte down.

**Code \$04:** The status list returned for this code is the same as that for code \$03.

**Code \$05:** The status list returned for this code is the SCSI REQUEST SENSE (\$03) return block.

**Code \$06:** The status list returned for this code is 2 bytes long. This list returns the current bytes/block parameter being used on the target device.

---

## Read Block (\$01)

The Read Block command reads one 512-byte block from the target device specified in the unit number parameter. The block read by this command is written into RAM at the address specified in the input buffer pointer. The parameter list for this call is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	input buffer pointer (lsb-msb)
4	target block number



### Parameter description

**unit number:** This parameter contains the SmartPort unit number of the target SCSI device. See Chapter 2 for details on selecting the correct SmartPort unit number for your target device.

**input buffer pointer:** This parameter contains the starting address of the data buffer used to store the block read by this command.

**target block number:** This parameter contains the logical address, on the host device, of the target block.

---

### Write Block (\$02)

The Write Block command writes one 512-byte block to the target device specified in the unit number parameter. The block written by this command is read from RAM at the address specified in the output buffer pointer. The parameter list for this call is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	output buffer pointer (lsb-msb)
4	target block number

### Parameter description

**unit number:** This parameter contains the SmartPort unit number of the target SCSI device. See Chapter 2 for details on selecting the correct SmartPort unit number for your target device.

**output buffer pointer:** This parameter contains the beginning address of the data buffer from which the target block is written.

**target block number:** This parameter contains the logical address, on the target device, to which the target block is to be written. Byte 4 is the low byte of this address, byte 5 is the middle byte, and byte 6 is the high byte.

---

### Format (\$03)

The Format command prepares all the blocks on the device specified in the unit number parameter for read/write use. This command is for use on block-type devices only.

The Format command checks the target device for existing partitions prior to beginning execution. If valid partitions are found, the command terminates and control is turned over in the normal fashion. If no partitions are found, a full SCSI FORMAT (\$04) command is executed on the target device, and the routine lays down a DDM and DPM for the drive, treating the entire storage volume as a single partition.

The parameter list for this call is as follows:

Byte	Definition
0	parameter list length (\$01)
1	unit number

### Parameter description

**unit number:** This parameter contains the SmartPort unit number of the target SCSI device. See Chapter 2 for details on selecting the correct SmartPort unit number for your target device.

---

## Control (\$04)

The Control command provides two basic functions. The first is to execute device control routines designed by Apple. The second is to execute SCSI commands. The command interpreter handles all of the functions required to execute the routine or command, including the routines necessary to read your routine or command out of RAM and pass it to the device, and receive any data returned by the command and write it into the RAM address you have specified (where appropriate).

If you need to use a SCSI command not supported by a SmartPort control code, you must write some special code in order to pass your command to the interpreter. See Appendix B for details on using unsupported SCSI commands with the command interpreter.

If there is a particular unsupported call that you use frequently, it is possible to patch it into the SCSI card RAM. This allows the command to be treated by the command interpreter just as if it were a supported command. To do this, use Patch1Call (\$1E).

Although each control code has its own parameter list, the following generic list shows the basic format they all adhere to.

Byte	Definition
0	parameter list length (\$03 or \$04)
1	unit number
2-3	pointer (lsb-msb)
4	control code
5	transfer count

---

### Important

The Control command makes extensive use of SCSI commands, as well as some Apple-designed routines. Programming these commands is very complicated and requires thorough knowledge of both the commands you are using and the devices you are targeting. Do not attempt to use this command unless you are familiar with both the ANSI SCSI commands and the command set supported by your device or devices; the paragraphs describing the commands later in this section are intended only as a quick reference, not a programming guide.

---

## Parameter description

**unit number:** This parameter contains the SmartPort unit number of the target SCSI device. See Chapter 2 for details on selecting the correct SmartPort unit number for your target device.

**pointer:** This parameter contains a pointer to the starting address of a buffer used to store data sent or received by the host during the execution of the control call.

**control code:** This parameter contains the SmartPort code number indicating which control routine or SCSI command is being handled. Table 3-8 shows the control codes and their associated routine or SCSI command, with the SCSI operation code (if any) given in parentheses.

**transfer count:** This parameter, when used, contains a count of the number of bytes transferred to or from the host during the execution of the command requested by the control code.

**Table 3-8**  
Control codes

Code	Definition	Code	Definition
\$00	DeviceReset	\$17	ReceiveDiagnostic (\$1C)
\$01–\$03	reserved	\$18	StartUnit (\$1B)
\$04	Eject (\$C0)	\$19	StopUnit (\$1B)
\$05	TestUnitReady (\$00)	\$1A	PreventRemoval (\$1E)
\$06	RequestSense (\$03)	\$1B	AllowRemoval (\$1E)
\$07	ReassignBlock (\$07)	\$1C	Verify (\$2F)
\$08	ModeSelect (\$15)	\$1D	RezeroUnit (\$01)
\$09	ModeSense (\$1A)	\$1E	Patch1Call
\$0A	Reserve (\$18)	\$1F	SetNewSDAT
\$0B	Release (\$17)	\$20	AudioSearch (\$C8)
\$0C	ReadDefectData (\$nn)	\$21	AudioPlay (\$C9)
\$0D	ReadCapacity (\$25)	\$22	AudioPause (\$CA)
\$0E	SendDiagnostic (\$1D)	\$24	AudioStatus (\$CC)
\$10	not supported	\$25	AudioScan (\$CD)
\$11	not supported	\$26	Eject (\$C0)
\$12	HardReset	\$27	ReadTOC (\$C1)
\$13	SetBlockSize	\$28	ReadQSubcode (\$C2)
\$14	SetTimeout	\$29	ReadHeader (\$C3)
\$15	FormatUnit (\$04)	\$2A	SetInterleave (\$04)
\$16	ExtendedSeek (\$2B)		

**DeviceReset (\$00):** This code orders a soft reset of the target device. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2–3	reserved (\$00)
4	control code (\$00)

**Code \$01–\$03:** These codes are not supported by the command interpreter. Use of these code numbers will return an error code.

**Eject (\$04):** This code executes the SCSI EJECT (\$C0) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	reserved (\$00)
4	control code (\$04)

---

### Important

This command is only valid for CD-ROM devices. Do not use it for other devices.

---

**TestUnitReady (\$05):** This code executes SCSI TEST UNIT READY (\$00) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	reserved (\$00)
4	control code (\$05)

**RequestSense (\$06):** This code executes the SCSI REQUEST SENSE (\$03) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$06)

**ReassignBlock (\$07):** This code executes the SCSI REASSIGN BLOCK (\$07) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$07)

**ModeSelect (\$08):** This code executes the SCSI MODE SELECT (\$15) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$04)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$08)
5	transfer byte count

**ModeSense (\$09):** This code executes the SCSI MODE SENSE (\$1A) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$04)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$09)
5	transfer byte count

**Reserve (\$0A):** This code executes the SCSI RESERVE (\$16) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$0A)

**Release (\$0B):** This code executes the SCSI RELEASE (\$17) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$0B)

**ReadDefectData (\$0C):** This code reads out information about data medium defects on the target device's mounted data medium. Information returned by this command is written to the buffer identified by the buffer pointer parameter. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$0C)

**ReadCapacity (\$0D):** This code executes the SCSI READ CAPACITY (\$25) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-5	buffer pointer (lsb-msb)
6	control code (\$0D)

**SendDiagnostic (\$0E):** This code executes the SCSI SEND DIAGNOSTIC (\$1D) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	reserved
4	control code (\$0E)

**Inquiry (\$0F):** This code executes the SCSI INQUIRY (\$12) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$0F)

**ExtendedRead (\$10):** This code is not supported by the command interpreter. Use of this code number will return an error code.

**Code \$11:** This code is not supported by the command interpreter. Use of this code number will return an error code.

**HardReset (\$12):** This code executes a full SCSI bus reset and then passes a SCSI Test Unit Ready (\$00) command to the target device. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$12)

---

### Warning

Do not use this code on a CD-ROM device.

---

**SetBlockSize (\$13):** This code sets the number of bytes per block on the target device. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	block size (bytes) msb-lsb
4	control code (\$13)

**SetTimeout (\$14):** This code sets the amount of time the initiator will wait for a response from the target before terminating the command (timeout). The timeout constant ranges from \$00-\$FF, and is loaded into the Control call parameter list in byte 2. For this command, byte 3 of the parameter list must be set to 0. The default timeout constant is \$08, for a timeout duration of approximately 10 seconds. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2	timeout
3	reserved (\$00)
4	control code (\$14)

**FormatUnit (\$15):** This code executes the SCSI FORMAT (\$04) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	reserved (\$00)
4	control code (\$15)

**ExtendedSeek (\$16):** This code executes the SCSI EXTENDED SEEK (\$2B) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-5	seek address buffer pointer
6	control code (\$16)

**ReceiveDiagnostic (\$17):** This code executes the SCSI RECEIVE DIAGNOSTIC RESULTS (\$1C) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$17)

**StartUnit/StopUnit (\$18/\$19):** These codes execute the SCSI START/STOP UNIT (\$1B) command. Each must be used separately; StartUnit (\$18) only starts the unit, while StopUnit (\$19) only stops it. The parameter list for these control codes is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	reserved (\$00)
4	control code (\$18/\$19)

**PreventRemoval/AllowRemoval (\$1A/\$1B):** These codes execute the SCSI PREVENT/ALLOW REMOVAL (\$1E) command. Each must be used separately. The parameter list for these control codes is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	reserved (\$00)
4	control code (\$1A/\$1B)

**Verify (\$1C):** This code executes the SCSI VERIFY (\$2F) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-5	reserved (\$00)
6	control code (\$1C)

**RezeroUnit (\$1D):** This code executes the SCSI REZERO UNIT (\$01) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	reserved (\$00)
4	control code (\$1D)

**Patch1Call (\$1E):** This code is used to patch one unsupported command to the command interpreter. It executes program code loaded into RAM bank 2 at \$C803. To use this call, you must write the program code, exactly as you want it to execute, into bank 2 RAM starting at \$C803. Once you have loaded the program code, executing Patch1Call will automatically execute your command.

---

**Important**

Remember that your command is loaded in RAM; any reinitialization or loss of power will erase it.

---

The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	reserved (\$00)
2-3	reserved (\$00)
4	control code (\$1E)

**SetNewSDAT (\$1F):** This code forces the SCSI card to reinitialize the SDAT and DIBTAB device tables for all devices resident on the bus. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number (any valid unit number will work)
2-3	RAM address pointer
4	control code (\$1E)

**AudioSearch (\$20):** This code executes the SCSI AUDIO TRACK SEARCH (\$C8) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-5	buffer pointer
6	control code (\$20)

The buffer contains the control parameters required by the SCSI command for proper execution, as follows:

Byte	Definition
0	play flag (0=hold after search, 1=play after search)
1	play mode (\$00-\$FF)
2-5	search address (lsb-msb)
6	type (\$00-\$02)

**AudioPlay (\$21):** This code executes the SCSI AUDIO PLAY (\$C9) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-5	buffer pointer (lsb-msb)
6	control code (\$21)



The buffer contains the control parameters required by the SCSI command for proper execution, as follows:

Byte	Definition
0	stop flag (0=stop address in 2-5, 1=play address)
1	play mode (\$00-\$FF)
2-5	playback address (lsb-msb)
6	type (\$00-\$02)

**AudioPause (\$22):** This code executes the SCSI AUDIO PAUSE (\$CA) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-5	buffer pointer (lsb-msb)
6	control code (\$22)

The buffer contains the control parameters required by the SCSI command for proper execution, as follows:

Byte	Definition
0	pause flag (0=pause, 1=resume)

**AudioStop (\$23):** This code executes the SCSI AUDIO STOP (\$CB) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-5	buffer pointer (lsb-msb)
6	control code (\$23)

The buffer contains the control parameters required by the SCSI command for proper execution, as follows:

Byte	Definition
0-3	stop address (msb-lsb)
4	type (\$00-\$02)

**AudioStatus (\$24):** This code executes the SCSI AUDIO STATUS (\$CC) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-5	buffer pointer (lsb-msb)
6	control code (\$24)

**AudioScan (\$25):** This code executes the SCSI AUDIO SCAN (\$CD) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-5	buffer pointer (lsb-msb)
6	control code (\$25)

The buffer contains the control parameters required by the SCSI command for proper execution, as follows:

Byte	Definition
0	direction flag (0=forward, 1=reverse)
1	reserved (\$00)
2-5	scan start address (lsb-msb)
6	type (\$00-\$02)

**ReadTOC (\$27):** This code executes the SCSI READ TABLE OF CONTENTS (\$C1) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$27)

The buffer contains the control parameters required by the SCSI command for proper execution, as follows:

Byte	Definition
0	track number
1-2	allocation length (lsb-msb)
3	type (\$00-\$02)

**ReadQSubcode (\$28):** This code executes the SCSI READ Q SUBCODE (\$C2) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$28)

**ReadHeader (\$29):** This code executes the SCSI READ HEADER (\$C3) command. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2-3	buffer pointer (lsb-msb)
4	control code (\$29)

The buffer contains the control parameters required by the SCSI command for proper execution, as follows:

Byte	Definition
0-3	block address

**SetInterleave (\$2A):** This code executes the SCSI FORMAT (\$04) command, changing only the interleave. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number
2	interleave
3	reserved (\$00)
4	control code (\$2A)

**ResetBus (special):** This special code forces a reset of the SCSI bus. To use it, you must load *unit number* with \$00 and *control code* with \$00. The parameter list for this control code is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number (\$00)
2-3	reserved (\$00)
4	control code (\$00)

---

## Init (\$05)

The Init command forces the firmware to reinitialize the SCSI bus. All the devices on the SCSI bus are hard reset, the partition offsets for each of the devices online are reinitialized, and new unit numbers are assigned, where necessary. The parameter list for this call is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number

### Parameter description

**unit number:** This parameter contains the SmartPort unit number of the target SCSI device. See Chapter 2 for details on selecting the correct SmartPort unit number for your target device.

---

## Open (\$06)

The Open command opens a logical file on the target device for data I/O. This command is used for character devices only. The parameter list for this call is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number

### Parameter description

**unit number:** This parameter contains the SmartPort unit number of the target SCSI device. See Chapter 2 for details on selecting the correct SmartPort unit number for your target device.

---

## Close (\$07)

The Close command closes a logical file on the target device after a data I/O sequence is completed. This command is used for character devices only. The parameter list for this call is as follows:

Byte	Definition
0	parameter list length (\$03)
1	unit number

### Parameter description

**unit number:** This parameter contains the SmartPort unit number of the target SCSI device. See Chapter 2 for details on selecting the correct SmartPort unit number for your target device.

---

### Read (\$08)

The Read command reads a specified number of bytes from the target device specified in the unit number parameter. The bytes read by this command are written into RAM, beginning at the address specified in the data buffer pointer. The number of bytes to be read is specified in the byte count parameter. The parameter list for this call is as follows:

Byte	Definition
0	parameter list length (\$04)
1	unit number
2-3	buffer pointer
4-5	byte count
6-7	address pointer

### Parameter description

**unit number:** This parameter contains the SmartPort unit number of the target SCSI device. See Chapter 2 for details on selecting the correct SmartPort unit number for your target device.

**data buffer pointer:** This parameter contains the beginning address of the host data buffer to which the target bytes are written.

**byte count:** This parameter contains the number of bytes to read for this command.

**address pointer:** This parameter contains the block address of the target block.

---

### Write (\$09)

The Write command writes a specified number of bytes to the target device specified in the unit number parameter. The bytes written by this command are read from RAM, beginning at the address specified in the data buffer pointer. The number of bytes to be written is specified in the byte count parameter. The parameter list for this call is as follows:

Byte	Definition
0	parameter list length
1	unit number
2-3	buffer pointer
4-5	byte count
6-7	address pointer

## Parameter description

**unit number:** This parameter contains the SmartPort unit number of the target SCSI device. See Chapter 2 for details on selecting the correct SmartPort unit number for your target device.

**data buffer pointer:** This parameter contains the beginning address of the data buffer from which the target bytes are written.

**byte count:** This parameter contains the number of bytes to write for this command.

**address pointer:** This parameter contains the block address of the target block.

---

---

## Summary of error codes

Following is a summary of error codes returned by the command interpreter, including a brief description of the possible causes for each. If there is no error, the C flag (in the Status register of the microprocessor) is cleared (0), and the accumulator (the A register) contains zeros. If the call was unsuccessful, the C flag is set (1), and the A register contains the error code.

**Table 3-9**  
Error codes

Code	Error	Explanation
\$00		No error.
\$01	BadCmd	A nonexistent command was issued. Check the command number.
\$04	BadPCnt	Bad call parameter count. The call parameter list was not properly constructed. Make sure the parameter list has the correct number of parameters.
\$06	BusErr	A communications error between the device controller and the host. Make sure that RAM is both read-enabled and write-enabled. Check the hardware (cables and connectors) between the device and the host. Check for noise sources; make sure the cable is properly shielded.
\$11	BadUnit	Unit number \$00 was used in a call other than Status, Control, or INIT.
\$21	BadCtl	The Control or Status code is not supported by the device.
\$22	BadCtlParm	The Control parameter list contains invalid information. Make sure each value is within the range allowed for that parameter.
\$27	IOError	The device encountered an I/O error trying to read or write to the recording medium. Make sure that the medium device is formatted and not defective. Make sure the device is operating correctly.

**Table 3-9 (continued)**  
Error codes

Code	Error	Explanation
\$28	NoDrive	The device is not connected. This can occur if the device is not connected but its controller is, or if there is no device with the unit number specified.
\$2B	NoWrite	The medium in the device is write-protected.
\$2D	BadBlock	The block number is outside the range allowed for the medium in the device. Note that this range depends on the type of device and the type of medium in the device (single-sided versus double-sided disk, for example).
\$2E	DiskSw	Disk switch took place.
\$2F	OffLine	Device off-line or no disk in drive. Check the cables and connections; make sure the medium is present in the drive, and that the drive is functioning correctly.
\$30-\$3F	DevSpec	Errors that differ from device to device. See the technical manual for the device in question for details.
\$40-\$4F	Reserved	Reserved for future expansion.
\$50-\$7F	NonFatal	A device-specific soft error. The operation was successfully completed but some exception condition was detected. See the technical manual for the device in question for details.

---



---

## Sample Program

The following program is written as an example of coding SmartPort calls to a SCSI device. The device used for this program is a CD-ROM drive. The program is written to exercise the basic functions of the drive.

```

        verbose      on
        longa        off
        longl        off
        absaddr      on
        gen          on
        65c02        on
        mcopy        cons.macros
        Keep         CDTest

CDTester      START                ;Begin program segment
              using                ConsData
;
;
; CD Test Mainline - Initializes the screen and performs Status, Control,
;                   and Read calls to the CD-ROM.
;
;
CDTestMain    jsr Inits              ;Do program initialization
              jsr FindCDROM          ;Look for CD-ROM to test
              jsr DoStatus           ;Do all drive Status calls
              jsr DoControl          ;Do all drive Control calls
              jsr RandRead           ;Do a random read sequence test
              jmp ExitProgram

```

```

;
;
; FIND CD-ROM - This routine first locates the SCSI card and saves its
; slot value. Then the card is accessed to see if any
; CD-ROMs are connected. If a CD-ROM is located, the
; test sequence continues.
;
; On Entry:      No params.
;
; On Exit:       C=0 for CD-ROM found
;               Leaves program if no CD-ROM located
;

```

FindCD-ROM

```

GotoXY 5,5
WriteStr FindCDstr ;Print subtest identifier string
jsr IsSCSI         ;See if SCSI card is hooked up
bcs FindOut
jsr IsCD           ;See if a CD-ROM is attached
bcc FindOut

```

FindNot

```

GotoXY 38,5
WriteStr NotCDstr ;Print no CD-ROM message and exit
jmp ExitProgram

```

FindOut

```

GotoXY 38,5
WriteStr OKStr    ;Print OK CD-ROM found message
rts

```

```

;
;
; IS_SCASI - This routine determines if there is a SCSI card hooked up.
; Each slot is cycled through and the SCSI card ID bytes are
; searched for.
;
; On Entry:      No params.
;
; On Exit:       C=0 for SCSI card found.
;               C=1 for no SCSI card located.
;

```

IsSCSI

```

lda #$C0          ;Get initial slot pointer hi
sta Buffhi       ;Place in high buffer pointer
sta CNval
lda #$FB         ;Get SCSI ID location offset
sta Bufflo      ;Point to in buffer
ldx #7           ;Keep counter of slot spaces, start with 7

```

Findloop

```

lda $CFFF        ;Clear the I/O space
txa
ora CNval        ;Set the proper $CNFA buffer pointer
sta Buffhi
ldy #0           ;Use indirect addressing for reading ID byte
lda (Bufflo),y
cmp #SCSIID     ;Did we find SCSI card ID?
beq FoundCD
dex
bne Findloop    ;Check next lowest slot, stop at zero

```

NotFound

```

sec             ;No SCSI card found
rts

```

FoundCD

```

clc
rts

```

```

;
;
; IS_CD - This routine determines if there is a CD-ROM attached on
; the SCSI card bus.
;
; On Entry: No params.
;
; On Exit: C=0 for SCSI CD-ROM found.
; C=1 for no SCSI CD-ROM located.
;
;-----
IsCD
        lda #Status           ;Set Status call SmartPort opcode
        sta SPcall
        lda #0                ;Do unit number zero call for Status
        sta SPUnit           ;Set for all units, zero
        sta SPCode1          ;Set SmartPort status call
        jsr SmartPort        ;Do the SmartPort status call
        ldx Mybuffer          ;Look at result # of devices
        lda #DIB              ;Set DIB status mode
        sta SPCode1          ;Set in status code byte

IsCDLoop
        stx SPUnit           ;Get DIB status for each connected device
        jsr SmartPort        ;Do DIB status call
        lda Mybuffer          ;Look at first status byte
        cmp #CDCODE          ;Look for disk status now!!!
        beq YesCD            ;Did we find the CD?
        ldx SPUnit           ;Get last x value
        dex                  ;Can we check another device?
        bne IsCDLoop         ;Thank you sir, may I have another?
        sec
        rts

YesCD
        clc
        rts

;
;-----
; Routine SMARTPORT - The SmartPort call is executed from here
;
;-----
SmartPort   jsr $C660           ;SmartPort call segment
SPcall      dc H'00'           ;Place for command number
SPList      dc I2'SPparams'    ;Pointer to SmartPort parameter list
           rts

;
; Parameter list for SmartPort call
;
SPparams
Numparams   dc H'03'           ;Number of parameters in list
SPUnit      dc H'01'           ;SmartPort unit number
SPBuffer    dc I2'Mybuffer'    ;Pointer to buffer for call
Blocklo
SPCode1     dc H'03'           ;Block number or cmd code
Blockhi     dc H'00'
Blockvhi    dc H'00'
Blockxhi    dc H'00'
SPCode2     dc H'05'

```



```

;
;
; DO_STATUS - Issues a SmartPort status call on the device
;
; On Entry: No params.
;
; On Exit: C=0 for status call good.
;          C=1 for status call error.
;

```

```

DoStatus
    GotoXY 5,7
    WriteStr Statustr
    lda #Status          ;Set Status call
    sta SPcall
    lda #DIB
    sta SPCode1         ;Set Device Status Block (DIB) call opcode
    jsr SmartPort      ;Do the SmartPort status call
    bcc DoStatOK       ;No error on status call?
    WriteStr Staterr

```

```

DoStatOK
    GotoXY 38,7
    WriteStr OKStr      ;Print OK status message
    rts

```

```

;
;
; DO_CONTROL- Test a SmartPort control call on the device.
;
; On Entry: No params.
;
; On Exit: C=0 for SmartPort control call good.
;          C=1 for SmartPort control call error.
;

```

```

DoControl
    GotoXY 5,9
    WriteStr Contrstr   ;Write control subtest string
    lda #Control        ;Set Control call
    sta SPcall
    lda #TstUnit
    sta SPCode1         ;Set test until ready control opcode
    jsr SmartPort      ;Do the SmartPort control call
    bcc DoCntrOK       ;No error on control call?
    WriteStr Cntrerr

```

```

DoCntrOK
    GotoXY 38,9
    WriteStr OKStr      ;Print OK CD-ROM found message
    rts

```

```

;
;
; RAND_READ - Test the device with a series of SmartPort read calls.
;
; On Entry: No params.
;
; On Exit: C=0 for SmartPort read call good.
;          C=1 for SmartPort read call error.
;

```

```

RandRead
    bit $C010          ;Clear keyboard strobe like Sam says
    GotoXY 5,11
    WriteStr Readstr
    lda #0
    sta blocklo
    sta blockhi       ;Init block counters
    sta blockvhi
    lda #Read         ;Set read call SmartPort opcode
    sta SPcall
    lda $CFFF         ;Reset IO space
    ldy #0
    lda (Bufflo),y   ;Access $CN00 card space

RandRloop
    jsr SmartPort     ;Do the SmartPort status call
    bcs RandRdErr     ;Report read error

RandRl
    jsr random
    and #$04
    sta blockvhi
    jsr random
    sta blockhi       ;Set the new block number hi
    jsr random
    sta blocklo       ;Set the new block number lo
    lda #70
    sta $24           ;CH
    lda #21
    sta $25           ;CV
    jsr $FC22         ;Vtab
    lda blockvhi
    jsr $FDDA
    lda blockhi
    jsr $FDDA
    lda blocklo
    jsr $FDDA         ;Print block numbers
    lda $C000
    bmi Randdie
    bpl RandRloop

Randdie
    GotoXY 44,11
    WriteStr OKStr    ;Print OK reads message
    clc
    rts

```

```

;
; Do request sense, report error, log it, then continue
;
RandRdErr
        WriteStr Readerrstr ;Report the read error
        jmp      RandR1      ;Go do some more
        rts

;
; Random - Returns a new random number
;
Random
        inc Increment
        ldy Increment
        lda $D000,y
        jsr Strip
Randout
        rts

Strip
        iny
        eor $D000,y
        asl a
        rts

;
; -----
; DO_INITS - Initialize all program space, screen, and variables
;
; On Entry: No params.
; On Exit: No params.
;
; -----
Inits
        InitConsole
        SViewPort 0,0,64,64
        WriteChar ClrWind ;Clear the current screen port
        WriteChar Home
        ldx #5
        stx SelXPos ;Set the current x select position
        ldx #3
        stx SelYPos
        GotoXY 1,1
        WriteStr Title1 ;Write test program title to screen
        GotoXY 0,2
        ReptChar TitLine ;Write window top underline
        GotoXY 0,22
        ReptChar TitLine ;Write window bottom underline
        GotoXY 0,23
        WriteStr Title2 ;Write bottom title text
        rts

;
; -----
; EXIT_PROGRAM - Execute a ProDOS-8 QUIT command
;
; -----
ExitProgram
        GotoXY 0,23
        WriteStr Quitstr
        GetKey ;What is the input
        jsr mli ;Do the "Quit" mli call, code $65
        dc H'65'
        dc I2'Exitlist'
        dc H'00'

```

```

;
; Parameter list for the exit call
;
Exitlist          dc H'04'
                  dc H'00'
                  dc H'00'
                  dc H'00'
                  dc H'00'
                  dc H'00'
                  dc H'00'

;
; -----
; Global data storage and equates
; -----
;
; Zero page equates
;
Bufflo            equ    $18
Buffhi            equ    $19
CrHB              equ    $8d
TabHB             equ    $89
StrPtr            equ    $83
StrPtr2           equ    $85
AtCall           equ    $42
;
; Program equates

mli               equ    $BF00           ;Entry point for all mli calls
Status            equ    $00           ;SmartPort status command value
Read              equ    $01           ;SmartPort read command value
Control           equ    $04           ;SmartPort control command value
TstUnit           equ    $05           ;SmartPort Test Unit Ready command
;
; Constants
;
CDCODE            equ    $B4           ;Device status code for CD-ROM
DIB               equ    $03           ;SmartPort DIB status code value
SCSIID            equ    $82           ;ID value for SCSI ROM in $CNFA
;
; Strings
;
Title1            str    'Apple //e           CD-ROM Test Utility  V1.1A'
Title2            str    'Currently testing CD-ROM!'
Statustr          str    'Testing CD-ROM Status calls...'
Contrstr          str    'Testing CD-ROM Control calls...'
Readstr           str    'Testing CD-ROM Random Read calls...'
FindCDstr         str    'Locating CD-ROM drive...'
Quitstr           str    'Any key to exit program -->'
NotCDstr          str    'No CD-ROM drives have been found.'
Readerrstr        str    'Read error encountered'
StatErr           str    ' ...Error on Status call !!!'
CntrErr           str    ' ...Error on Control call !!!'
OKStr             str    'OK.'

```

```

;
; Variable storage
;
SelXPosds 1
SelYPosds 1
Increment    ds 1
readcnt      ds 1                ;Loop counter for reads
readx2       ds 1                ;Second loop counter for reads
CNval        ds 1                ;Keep the $CN slot pointer here
Initparm     dc  H'80'

TitLine      anop
             dc    a'ULChars'
             dc    i'5'

ULChars      anop
             dc    I1'MTON'
             dc    I1'ReptChar'

LinCnt       dc    I1'80'
             dc    c'L'           ;Top of the window underline character
             dc    I1'MTOFF'

Mybuffer     ds 1024             ;Data buffer for transfers

            END
CD-ROM Test Utility Program

```



---

---

## Appendix A

---

---

# Device Partitioning

Device partitioning is a means of dividing a SCSI block-type device into a number of sections for use by multiple operating systems. These sections are called **device partitions**.

A device partition is a set of blocks on a device. Although a single partition may be set up for use by more than one operating system, all the operating systems that share a partition must be compatible with each other.

Each of the partitions on a device is defined in a software table called the **Device Partition Map (DPM)**. The DPM is itself a partition and is stored on the device it describes. The DPM is described under "The Device Partition Map (DPM)."

---

---

### Creating device partitions

To create a partition on a SCSI block device, you must create the Partition Descriptor Map (PDM) in the Device Partition Map (DPM). These structures are described in this appendix. You create the PDM by writing data into a series of fields that define the partition's physical, logical, and operating characteristics. The utilities disk supplied with your Apple II SCSI Card contains a utility called HD SC Partition. Running the HD SC Partition partitioning routine does all of this work for you.

The *Apple II SCSI Card Owner's Guide* contains step-by-step instructions on running HD SC Partition.

---

---

### The Device Control Block (DCB)

The **Device Control Block (DCB)** is a special block of device-specific data, stored on the device itself. The data in the DCB usually consists of setup data for the device's SCSI controller.

---

---

## The Device Information Block (DIB)

The **Device Information Block (DIB)** is a special block of device-specific data. The data in the DIB is used to construct the device's DIBTAB and SDAT in SCSI Card RAM. The information contained in this block varies by device.

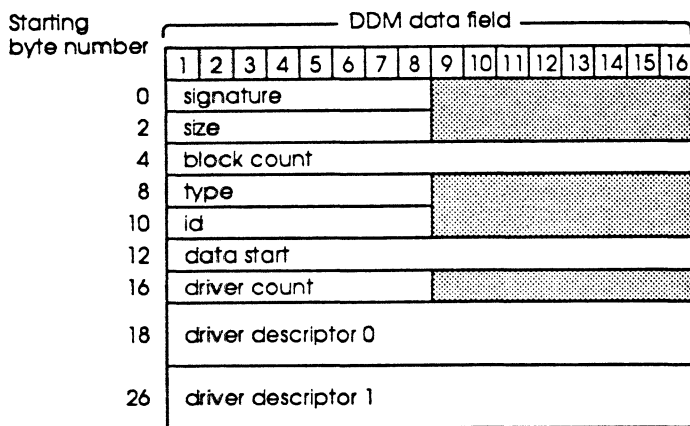
---

---

## The Driver Descriptor Map (DDM)

The **Driver Descriptor Map (DDM)** is a software table that contains the starting address, size, and operating system type of all device drivers resident on the device. This information is stored in an 8-byte entry in the DDM, one entry per device driver. In addition to the information on device drivers, the DDM contains some general information about the device itself.

Figure A-1 shows the structure of the DDM. The following sections describe the DDM entries.



**Figure A-1**  
DDM data structure

**signature:** This is a 2-byte field containing the hexadecimal number that identifies this block as the DDM to the operating system or active program. The value of this field is always \$45 52.

**size:** This is a 2-byte field containing the size of the DDM block in bytes.

**block count:** This is a 4-byte field containing the number of blocks on the device.

**type:** This is a 2-byte field containing an ASCII string describing the device type of the host device. This field usually contains a mnemonic for the device type.

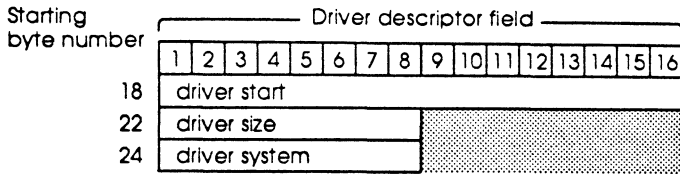
**id:** This is a 2-byte field containing the SCSI device ID of the host device.

**data start:** This is a 4-byte field containing the starting address of the first *driver descriptor* entries.

**driver count:** This is a 2-byte field containing the number of device driver descriptor entries in the DDM.



**driver descriptor:** This is an 8-byte field containing three data fields describing a device driver. One *driver descriptor* is made for each device driver resident on the device. The data structure of *driver descriptor* is shown in Figure A-2.



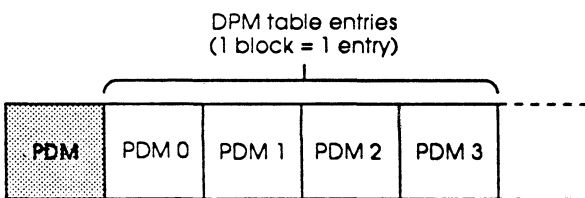
**Figure A-2**  
Driver Descriptor data structure

The *driver descriptor* fields are described in the following paragraphs.

- **driver start:** This 4-byte field contains the starting address of the device driver code.
- **driver size:** This 2-byte field contains the size of the device driver, in bytes.
- **driver system:** This 2-byte field contains an ASCII string describing the operating system type of the device driver. This field usually contains a mnemonic for the operating system type.

## The Device Partition Map (DPM)

The Device Partition Map is a software table made up of a variable number of 1-block entries. The entries in the DPM are called Partition Descriptor Maps (PDMs). One PDM is constructed for each partition on the device, including the DPM partition itself. The DPM always begins on *physical block 1* on the device, and is always defined as *logical block 0*. The DPM consists of as many blocks as there are PDMs; that is, as many blocks as there are partitions on the device. Figure A-3 shows the structure of the DPM.



**Figure A-3**  
DPM data structure

## The Partition Descriptor Map (PDM)

The Partition Descriptor Map (PDM) is a one-block-wide entry in the DPM table. The PDM consists of a series of data fields that define the partition. Figure A-4 shows the structure of the PDM. The following sections define the PDM data fields.

Starting byte number	PDM data field
0	signature
4	DPM block count
8	start
12	size
16	name
48	type
80	data start
84	data size
88	status
92	boot start
96	boot size
100	boot load
104	MP load
108	jump to
112	MP jump to
116	boot check
120	MP id
136	boot arg

**Figure A-4**  
PDM data structure

**signature:** This 2-byte field is set to the PDM ID string, indicating that this block is a PDM. The PDM ID string is \$50 4D.

**DPM block count:** This 32-bit field is set to the number of blocks in the Device Partition Map. Because the PDM for the Device Partition Map may not be the first PDM on the device, this field must be included in every PDM.

**start:** This 32-bit field is set to the number of the first physical block allocated to the partition.

**size:** This 32-bit field is set to the number of blocks allocated to the partition.

**name:** This field is filled with the *physical* name of the partition, which may differ from the *logical* name of the partition. This field is an ASCII string with a length of 32 bytes. Uppercase and lowercase characters are not distinguished.

---

**Important**

If you fill this field with a string of less than 32 bytes, you must terminate the entry with a NUL (binary zero) character.

You may leave this field empty, provided that you fill the first byte with the NUL character.

**type:** This field is filled with an ASCII string describing the purpose or use of the partition, such as the name of the operating system that uses it. This field is an ASCII string with a length of 32 bytes. Uppercase and lowercase characters are not distinguished.

---

**Important**

If you fill this field with a string of less than 32 bytes, you must terminate the entry with a NUL (binary zero) character.

---

You may leave this field empty, provided that you fill the first byte with the NUL character.

All *type* strings that begin with the characters *Apple* are reserved for use by Apple Computer.

**data start:** This 32-bit field is set to the number of the first block allocated to the partition that contains valid data. This number may differ from the number set in *start* due to the presence of bad blocks or blocks used for boot code or other special uses.

**data size:** This 32-bit field is set to the number of blocks allocated to the partition that contain valid data.

**status:** This 32-bit field contains nine 1-bit status flags, as shown in Figure A-5.

Bit number	Status flag
0	PDM valid?
1	allocated?
2	in use?
3	boot ok?
4	read ok?
5	write ok?
6	pin d boot?
7	os sf 1
8	os sf 2
9-1F	not used

**Figure A-5**  
Status byte data structure

The *status* fields are described in the following paragraphs.

- PDM valid:** This flag is set to 1 for a valid PDM.
- allocated?:** This flag is set to 1 if the partition has been allocated by an operating system or systems.
- in use?:** This flag is set to 1 if the partition is currently being accessed. This flag is intended for use in multiprocessor environments.
- boot ok?:** This flag is set to 1 if the partition contains valid boot code.
- read ok?:** This flag is set to 1 if the management operating system for this partition allows reading of the partition. This flag does not apply to operating systems other than the operating system to which the partition is allocated.
- write ok?:** This flag is set to 1 if the management operating system for this partition allows writing to the partition. This flag does not apply to operating systems other than the operating system to which the partition is allocated.
- pin d boot?:** This flag is set to 1 if the boot code contained in the partition is position-independent. For some microprocessors, this flag indicates whether the Boot Load and Jump To (defined later in this section) addresses must be adhered to.
- os sf 1/os sf 2:** These operating system-specific flags are not defined; they are available for use by the operating system to which the partition is allocated.

---

**Important**

Any of the following fields marked with the ● symbol are required only if the PDM Valid flag is set to 1.

---

**boot start:** This 32-bit field contains the starting block number of the boot code.

●**boot size:** This 32-bit field is set to the number of bytes of boot code contained in the partition.

●**boot load:** This 32-bit field is set to the address in main memory where the boot code must be loaded.

●**mp load:** This 32-bit field contains any load address data that is microprocessor specific.

●**jump to:** This 32-bit field contains the boot code JUMP address in main memory.

●**mp jump to:** This 32-bit field contains any additional boot code JUMP address data that is microprocessor specific.

●**boot check:** This 32-bit field contains the boot code checksum. The C code for this routine is as follows:

```
unsigned short      calculate_checksum(data, length)
    unsigned char    *data;
    unsigned int     length;
{
    unsigned short    checksum;
    checksum = 0;
    while (length--)
    {
        checksum += *data++;
        if (checksum & 0x8000)
        {
            checksum = (checksum << 1) | 1;
        }
        else
        {
            checksum <<= 1;
        }
    }
    if (checksum == 0)
    {
        checksum = 0xffff;
    }
    return(checksum);
}
```

●**mp id:** This field is filled with an ASCII string describing the microprocessor for which the boot code is valid. This field is an ASCII string with a length of 16 bytes. Uppercase and lowercase characters are not distinguished.

---

**Important**

If you fill this field with a string of less than 16 bytes, you must terminate it with a NUL (binary zero) character.

---

You may leave this field empty, provided that you fill the first byte with the NUL character.

●**boot arg:** These four 32-bit fields contain arguments that are boot code specific.



---

---

## Appendix B

---

---

# Using Unsupported SCSI Commands

The Apple II SCSI Card command interpreter supports 40 different commands. If, however, you should find that you need to use a SCSI command for which there is no control code, follow the six basic steps described in the following sections to execute your command on the SCSI card.

---

### Important

SCSI commands contain a 3-bit field called *logical unit number (lun)*. The command interpreter does not use *lun* to target devices. Instead, it uses the SCSI device IDs established by the card firmware during bus initialization and loaded into the SDAT.

Be sure to load *lun* with \$00.

---

---

---

### Select the SCSI card

Before you can begin the process of executing your command, you must enable SCSI card RAM and ROM (\$C000-\$CFFF). To do this, select the slot that the SCSI card is installed in by reading or writing any byte to \$CFFF (to clear any previous selection) and \$Cn00 (to select slot *n*, the SCSI card slot).

If, for example, your SCSI card is installed in slot 2, you might run the following code to select it:

```
lda #0
sta $CFFF ; clears slot already selected
sta $C200 ; selects slot 2
```

The SCSI card is now selected, enabling card RAM and ROM.

---

---

## Set up device tables (SDAT/DIBTAB)

Your device should already have a SDAT and a DIBTAB, set up for it by the SCSI management firmware during initialization. If, for some reason, your device does not have device tables already set up, or if you wish to change the parameters loaded into the SDAT or DIBTAB for your device, create the new tables according to the structure shown in Chapter 3 and load the data in by hand.

If you do set up your own tables, you must set *init*. To set *init*, write \$77 to address \$C809.

◆ *Note:* Data transfer (r/w) commands require some additional setup. Refer to the section on data transfer commands later in this appendix.

---

---

## Load the command block

The command block for an interpreted SCSI command consists of a 1-byte header and the standard SCSI command block for your command. The header contains a count of the number of bytes in the command block (exclusive of the header). Construct your command in RAM, count the number of bytes it contains, load the count into the header, and write the whole command block into the SCSI buffer. The buffer begins at \$C80D.

If, for example, your device is a Tape Backup 40SC and you want to execute a RECEIVE QIC-100 SYSTEM DATA (\$06) command, you would construct the command block as follows:

byte 0	\$06	;length is 6 bytes
byte 1	\$06	;command number is 6
byte 2	\$00	;lun is set to 0
byte 3	\$00	;reserved
byte 4	\$00	;reserved
byte 5	\$00	;reserved
byte 6	\$00	;reserved

---

---

## Call the SCSI management routines

To begin executing your command, you need to call the firmware routines responsible for managing the SCSI bus phases. DoPhases handles the arbitration, selection, and command phases. DoStatus handles the status and message-in phases. You must call these two routines for all of your SCSI commands. Data transfer commands (r/w) also require calls to DataXin, DataXout, and/or LongData. Other commands may require calls to other routines.

The management routines are loaded at \$CC00–CFFF in 16 ROM banks. In order to call a routine you must select its ROM bank and then pass its address. In most cases, you can call BankSwitch to perform your switching operation. BankSwitch is located at \$CFCC in all banks. To use BankSwitch, load the X register with \$00 and the Y register with \$n0 (where *n* equals the SCSI card slot number plus eight). Then load the accumulator as follows:



upper nibble = target routine number (\$0-\$F)

lower nibble = target bank number (\$0-\$F)

With the SCSI card in slot 2, you call DoStatus as follows:

```
ldx    #$00
ldy    $A0          only if card in slot 2, n=(2+8)=A
jsr    banksw      ;execute DoStatus
```

**Table B-1**  
ROM entry points for SCSI management routines

Routine	Entry point	Description
DoPhases	\$00	Arbitration/selection/command
DoStatus	\$03	Status/message-in
DataXin	\$10	Transfers data from target to initiator
DataXout	\$11	Transfers data from initiator to target
LongData	\$92	Special data transfer routine
BusFree	\$3A	Clears SCSI bus to Bus Free from any other phase

Commands involving data transfer, like the RECEIVE QIC-100 SYSTEM DATA (\$06) command, require some additional work to execute. First, you must select the transfer mode your command will use. There are four modes you can choose among, as follows:

- Iie Pseudo-DMA (PDMA): This mode transfers one 512-byte block of data. This mode is used for the Apple Iie when the necessary handshaking is supported by hardware.
- IIGS Pseudo-DMA (PDMA): This mode transfers one 512-byte block of data. This mode is used for the Apple IIGS when the necessary handshaking is supported by hardware.
- Programmed I/O (PIO): This mode is used when the hardware involved does not support handshaking. PIO is executed in software. This mode must be used for commands that involve less than a full block of data.
- LongData I/O (LDIO): This mode transfers any number of bytes, up to 64K. LongData I/O uses a special routine located in ROM bank 10, rather than the standard DataXin/DataXout routines located in ROM bank 2.

To select the mode you wish to use, you must execute a read to the correct address, as shown in Table B-2.

**Table B-2**  
Data transfer mode selection address

Mode	Address	Contents
Iie PDMA	\$C804	\$00
IIGS PDMA	\$C804	\$01
PIO	\$C804	\$02-\$FF
	\$C820	\$FF
LDIO	Special	

You must also store a pointer to the buffer you are using for your command. The firmware looks for the buffer address at \$FA-\$FB in page zero RAM for nonextended calls, and at \$FA-\$FD for extended calls.

For example, in a nonextended call, code to read into a buffer located at \$3300, bank 0, might look like this:

```
lda  #$00      ;store low byte in $FA
sta  $FA
lda  #$33      ;store high byte in $FB
sta  $FB
lda  #$00      ;store high byte in $FB
sta  $FC
      $FD
```

For an extended call, code to read into a buffer located at \$3300, bank 9, might look like this:

```
lda  #$00      ;store low byte in $FA
sta  $FA
lda  #$30      ;store next byte in $FB
sta  $FB
lda  #$09      ;store next byte (the bank number) in $FC
sta  $FC
lda  #$00      ;store high byte in $FD
sta  $FD
```

You must now set the extended flag at \$C81D. Write \$00 to this address to set the flag for a nonextended call. Write \$40 to set the flag for an extended call.

If you are using LongData I/O mode, there are a few more steps you must perform. Because LongData I/O mode allows you to read any number and write any block-multiple number of bytes (up to \$FFFF), you must load some extra parameters.

After you have loaded the buffer pointer, as directed earlier, you must load the number of bytes to be transferred. Load the lsb of the count at \$C9EE, and the msb at \$C9FF.

Calculate the number of blocks to be transferred using the following algorithm:

Reading:  $\text{count} = (\text{bytecount} / \text{size}) + 1$

Where *size* is equal to the bytes per block value for the target device.

Writing:  $\text{count} = \text{bytecount} / \text{size}$

❖ *Note:* To read an even block-multiple number of bytes, such as 4096 (2 blocks in a 2048 bytes/block device), you do not need to add 1 to  $(\text{bytecount} / \text{size})$ .

For example, if you wanted to read 7000 bytes from a device that had 2048-byte blocks, the algorithm would look like this:

$\text{count} = (7000 / 2048) + 1 = 4$

To write 7000 bytes to the same device, you should pad the last block to an even multiple of 2K, as follows:

$\text{count} = 8192 / 2048 = 4$  blocks

---

---

## Wait for next bus phase

Before you can call the next management routine, you must be sure the previous one has completed its execution. Also, you must ensure that the target device has sufficient time to properly execute the previous command. You control these timing considerations in one of two ways, depending on the type of management routine you are executing.

For the DoPhases and DoStatus routines, you control the execution timing by setting TimeConstant at \$C9F6. To do this, load a value between \$00 and \$FF \$C9F6. Each value is roughly equal to 1.25 seconds of time, so a value of \$02 sets the timeout parameter on the card to 2.5 seconds.

To set TimeConstant for your command, you could use the following code:

```
lda    #$04      ;(1.25 sec)x4 = 5 sec timeout
wait
sta    $C9F6     ;TimeConstant
```

For data transfer routines, you need to write a routine that checks the bus phase the target device is in (its phase condition), and waits for that phase to be completed. You should call this routine prior to calling DataXin, DataXout, LongData, or DoStatus.

The target device's phase condition is loaded at \$C0n4, bits 2, 3, and 4, where *n* is the slot number plus eight. Table B-3 shows the values loaded for the various bus conditions.

**Table B-3**  
SCSI bus condition

Phase	b4	b3	b2
Data-In	0	0	1
Data-Out	0	0	0

Code to wait for a status phase might be as follows:

```
        lda    #$FF
        sta    timer
loop    lda    $C0A4      ;card in slot 2, m= 2+8 =A
        and    #$1C      ;mask out for bits 2,3,4
        cmp    #$04      ;are we in data in phase yet?
        beq    datainyes ;if yes, quit waiting
        dec    timer     ;if no, keep waiting
        bne    loop
        sec    datainno  ;flag error if timeout occurred
        rts
        clc    datainyes ;data in phase is happening!
        rts
```

- ❖ *Note:* The time you must wait for a command to be executed varies from device to device, and sometimes from command to command. Making your timeout loops longer than seems necessary, just to be on the safe side, costs you very little time and is an excellent precaution against timeout variances.

---

---

## Check the command execution status

In order to determine whether or not your command was successfully executed at the target device, check the status byte loaded at \$C81E. A value of \$00 indicates the Good status. A value of \$02 indicates the Check Condition status, which means that something went wrong. To find out what went wrong, execute the SmartPort RequestSense (\$06) routine (by making a SmartPort Control call) or the SCSI REQUEST SENSE (\$03) command.

Code to check the status byte might look like this:

```
jsr  do_a_read          ;
lda  $C81E              ;check status byte
cmp  #02                ;is status $02?
bne  done               ;if not, then quit
jsr  do_request_sense   ;if so, do RequestSense
```



## Glossary

**accumulator:** The register in the microprocessor where most computations are performed.

**active program:** The program that is currently running on the Apple II.

**address:** (1) A number that specifies the location of a single **byte** of memory. Addresses can be given as decimal integers or as hexadecimal integers. A 64K system has addresses ranging from 0 to 65535 (in decimal) or from \$0000 to \$FFFF (in hexadecimal). (2) In data transmission, a code for a specific terminal. Multiple terminals on one communication line, for example, must have unique addresses.

**Apple II:** A family of computers, including the original Apple II, the Apple II Plus, the Apple IIe, the Apple IIc, and the Apple IIGS.

**ASCII:** Acronym for *American Standard Code for Information Interchange*, pronounced *ASK-ee*. A code in which the numbers from 0 to 127 stand for text characters. ASCII code is used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.

**assembly language:** A low-level programming language in which individual machine-language instructions are written in a symbolic form that's easier to understand than machine language itself. Each assembly language instruction produces one machine-language instruction.

**binary:** The representation of numbers in the base-2 system, using only the two digits 0 and 1. For example, the numbers 0, 1, 2, 3, and 4 become 0, 1, 10, 11, and 100 in binary notation. The binary system is commonly used in computers because the values 0 and 1 can easily be represented in a variety of ways, such as the presence or absence of current, positive or negative voltage, or a white or black dot on the

display screen. A single binary digit—a 0 or a 1—is called a **bit**. Compare **decimal**, **hexadecimal**.

**binary digit:** The smallest unit of information in the binary number system; a 1 or 0. Also called a **bit**.

**bit:** A contraction of *binary digit*. The smallest unit of information that a computer can hold. The value of a bit (1 or 0) represents a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing. See also **binary**.

**bit map:** A set of bits that represent the position and state of a corresponding set of items.

**block:** A unit of data. A standard block is 512 bytes in size. Nonstandard blocks must be specially defined.

**buffer:** A "holding area" of the computer's RAM where information can be stored on a temporary basis (buffered). Buffering is often done when data is being transferred between two devices operating at different communication rates.

**bus:** An electrical or electronic connection between devices. The devices connected by the bus are said to be resident on the bus, and may be as small as ICs or as large as mainframe computers. A bus provides a means to send the same data, signals, or voltages (for power supply buses) to more than one device across a single carrier (wire, fiber-optic cable, and so forth).

**byte:** A unit of data consisting of eight contiguous bits numbered from 0 to 7. Bit 7 of a byte is called the *most significant bit*, while bit 0 is called the *least significant bit*.

**call:** A request issued by the CPU or a program to the SCSI card firmware. A call consists of a **command number**, a **pointer**, and a **parameter list**.

**catalog:** A list of all files stored on a disk. Sometimes called a directory.

**central processing unit (CPU):** The Apple II computer; specifically, the microprocessor and its supporting hardware exclusive of any peripheral cards or devices, RAM, and ROM.

**character:** Any symbol that has a widely understood meaning and thus can convey information. Some characters—such as letters, numbers, and punctuation—can be displayed on the monitor screen and printed on a printer.

**character code:** A number used to represent a character for processing by a computer system.

**clear:** To erase data from memory or reset a control register. Clearing is usually done by loading the memory location or register to be cleared with zeros.

**code:** (1) A number or symbol that corresponds to a set of parameters or instructions. (2) The statements that make up a program.

**command:** An instruction that causes the target device to perform a specific operation. Commands are passed to the firmware in **calls**.

**command number:** A hexadecimal number that corresponds to a specific ProDOS, SmartPort, or SCSI command; each command has a unique command number within its group (SCSI, ProDOS, or SmartPort).

**command register:** A location in a device controller that stores control information. Compare **data register**.

**control code:** A hexadecimal number that corresponds to a particular control function for an external SCSI device.

**CPU:** See **central processing unit**.

**daisy-chain:** A group of devices connected to a host device, where the first device in the "chain" is connected to the host, the second device is connected to the first, the third device is connected to the second, and so on. In the Apple II, devices in a daisy-chain are connected in parallel.

**data register:** A location in a peripheral device controller that stores data. Compare **command register**.

**DCB:** See **Device Control Block**.

**decimal:** The common form of number representation used in everyday life, in which numbers are expressed in the base-10 system, using the digits 0 through 9. Compare **binary**, **hexadecimal**.

**default:** The value or condition to which a variable or parameter is set automatically. The default value or condition is used by a device or program unless the user specifies otherwise.

**device:** A hardware unit, such as a computer, a disk drive, or a peripheral card.

**Device Control Block (DCB):** A block of device-specific data stored on the device itself. Data in the DCB is usually setup information for use in initializing the device after power-up or reset.

**Device Information Block (DIB):** A block of device-specific data stored on the device itself. Data in the DIB is used to construct the SDAT and DIBTAB for the device.

**device partition:** A set of blocks on a device set up for use by one or more operating systems. All the operating systems using a partition must be compatible.

**Device Partition Map (DPM):** A table made up of a number of 1-block entries called **Partition Descriptor Maps**. The DPM always begins on physical block 1 of any device, and is defined as logical block 0.

**DIB:** See **Device Information Block**.

**DIBTAB:** A software table loaded into SCSI card RAM that contains information on the type and version of the device, as well as the logical size and accessibility of the device. Often this is the same information as that contained in the **DIB**.

**DPM:** See **Device Partition Map**.

**Driver Descriptor Map (DDM):** A table that contains the starting address, size, and operating system type of all device drivers resident on the device. Each descriptor is 8-bytes long, one descriptor for every driver on the device.

**error code:** A number or other symbol representing a type of error.

**field:** A specific set of data that is related. A field is always defined by its size, given in bits or bytes. A field usually has a name as well.

**firmware:** Programs stored permanently in read-only memory (ROM). Such programs (for example, the Applesoft Interpreter and the Monitor program) are built into the computer at the factory. They can be executed at any time but cannot be modified or erased from main memory.

**flag:** A variable whose value (1 or 0) indicates the state of a specific parameter.

**hexadecimal:** The representation of numbers in the base-16 system, using the ten digits 0 through 9 and the six letters A through F. For example, the decimal numbers 0, 1, 2, 3, 4, ... 8, 9, 10, 11, ... 15, 16, 17 would be shown in hexadecimal notation as 00, 01, 02, 03, 04, ... 08, 09, 0A, 0B, ... 0F, 10, 11. Hexadecimal numbers are easier for people to read and understand than binary numbers, and they can be converted easily and directly to binary form. Each hexadecimal digit corresponds to a sequence of four **binary digits**, or bits. Hexadecimal numbers are usually preceded by a dollar sign (\$).

**host:** The device that controls the operation of peripheral devices. The Apple II is a host for the SCSI card and the external devices connected to it.

**host device:** See **host**.

**IC:** See **integrated circuit**.

**initialize:** To set to a predetermined starting state or value.

**input:** Data transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem.

**input/output (I/O):** The process by which information is transferred to and from the computer and peripheral devices.

**instruction:** A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

**integrated circuit:** An electronic circuit—including components and interconnections—entirely contained in a single piece of semiconducting material, usually silicon. Often referred to as an IC or a chip.

**interface:** The rules of interaction between two devices or programs; the electrical, procedural, and functional conventions that govern the exchange of data and control signals between devices.

**interface card:** A circuit card that implements a particular interface (such as SCSI) by which the computer can communicate with peripheral devices such as a hard disk drive.

**interrupt:** An electronic attention-getter; a signal sent to the microprocessor that is intended to force the microprocessor to stop its current activity and accept input from the device that sent the interrupt.

**least significant bit:** The rightmost bit of a binary number; bit 0. The least significant bit contributes the smallest quantity to the value of the number. Compare **most significant bit**.

**logical block:** A block on a device that can be used by software to store data. Logical blocks on a device are numbered consecutively from 0, but may not directly correspond to their physical block number due to block sparing conducted during initialization of the device. Compare **physical block**.

**logical name:** The name of a block, file, table, device, or other entity in software, that may or may not correspond to its physical name. Compare **physical name**.

**main memory:** The part of a computer's volatile (nonpermanent) memory that is directly accessible to the microprocessor. Auxiliary memory, such as that contained on peripheral cards, is mapped into main memory to allow the microprocessor to access it.

**mnemonic:** A type of abbreviation consisting of a series of letters and/or numbers that represent a longer or more complicated name or title.

**most significant bit:** The leftmost bit of a **byte**; bit 7. The most significant bit contributes the largest quantity to the value of the number. Compare **least significant bit**.

**null:** Any character or character code that has no meaning to the operating system or program interpreting it.

**parameter:** Any of a set of characteristics whose value or condition determines the operation of a program or device.

**parameter list:** The list of characteristics whose value or condition determines the precise execution of a SCSI command.

**Partition Descriptor Map:** A 1-block entry in the **Device Partition Map** consisting of a series of data fields describing the state of a specific partition.

**PDM:** See **Partition Descriptor Map**.

**physical block:** The blocks on the disk itself, created during initialization, that represent a specific storage capacity on the device medium. A physical block may or may not be a logical block. Compare **logical block**.

**physical name:** The name of a block, file, table, device, or other entity, used for convenience. A physical name may or may not correspond to the logical name. Compare **logical name**.

**pointer:** A data item that provides the memory address of some other data item; a pointer contains the address of some other data.

**protocol:** A formal set of rules for sending and receiving data on a communication line.

**register:** A memory location of a specific size in which each bit (or byte, depending on the size and design of the register) has some meaning to a microprocessor, program, or "smart" IC. IC registers are sometimes internal to the IC.

**resident:** Present on; connected to.

**SCSI:** See **Small Computer System Interface**.

**SCSI device access table (SDAT):** A software table stored in the SCSI card RAM that contains a set of parameters for each SCSI device connected to the Apple II.

**SCSI ID number:** The number assigned to a SCSI device based on the position of a hardware strap, jumper or switch on the device itself. Compare **unit number**.

**shadowing:** A process through which the SCSI card takes over an additional slot in order to work with ProDOS in supporting four external device ports.

**Small Computer System Interface (SCSI):** A set of mechanical, electrical, and functional specifications covering the design of peripheral devices, software, firmware, and computers for use in small computer systems.

**stack:** A list in which entries are added (pushed) or removed (popped) at one end only (the top of the stack), causing them to be removed in last-in, first-out (LIFO) order.

**standard:** A set of rules, specifications, and procedures used to standardize the design of products.

**system:** A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose—for example, a computer system comprising a processor, keyboard, monitor, and disk drive.

**unit number:** A hexadecimal number assigned to a SCSI device by the firmware, based on the device's SCSI ID number. Compare **SCSI ID number**.

**valid partition:** A device partition that has a **PDM** created for it in the **DPM**.

**word:** In the Apple II, a word is 2 bytes (16 bits).

**X register:** One of the two index registers in the 6502 microprocessor.

**Y register:** One of the two index registers in the 6502 microprocessor.

**zero page:** The first page (256 bytes) of memory in the Apple II family of computers. Zero page is also called page zero or, in the Apple IIGS, the direct page. Because the high-order byte of any address in this page is zero, only the low-order byte is needed to specify a zero-page address; this makes zero-page locations more efficient to address, in both time and space, than locations in any other page of memory.





# Index

## A

Apple II internal bus 6  
Apple II microprocessor 5, 18  
Apple II SCSI bus 8–10  
AudioPause 31  
AudioPlay 30–31  
AudioScan 31–32  
AudioSearch 30  
AudioStatus 31  
AudioStop 31  
Autostart ROM 2

## B

bank select logic 6  
boot priority 2  
bus phase 57

## C

cable extender 3  
cables  
    peripheral interface 3  
    SCSI card 10  
    system 3, 9  
Close command 33–34  
command block, loading 54  
command execution status 58  
command interpreter 16–17  
    unsupported commands and  
    53–58  
command number, defined 17  
command parameters zero-page  
    write 17  
Control command 24–33  
CPU-5380 interface logic 7

## D

Device Control Block (DCB) 45  
Device Information Block (DIB) 46  
Device Information Block Table  
    (DIBTAB) 13–16, 54  
    example 16  
Device Partition Map (DPM) 47

device partitions 45–51  
    creating 45  
    defined 45  
DeviceReset 25  
device select space I/O address  
    map 8  
device tables. *See* Device  
    Information Block Table; SCSI  
    Device Access Table  
DIBTAB. *See* Device Information  
    Block Table  
Driver Descriptor Map (DDM) 46–47

## E

Eject 26  
enabling RAM/ROM 53  
error codes 35–36  
ExtendedRead 28  
ExtendedSeek 29  
extended SmartPort calls 19–20,  
    55–56

## F, G

5380 interface logic 7  
5380 SCSI IC 7–8  
firmware 11–43  
    overview 2  
Format command 23–24  
FormatUnit 28  
functional description  
    (of SCSI card) 5

## H

HardReset 28  
hardware 3–10  
    overview 1–2  
HD SC Partition 45

## I, J, K

Init command 33  
Inquiry 27  
internal bus (Apple II) 6  
interpreter. *See* command  
    interpreter

## L

loading command block 54  
locating SmartPort 18–19  
LongData I/O (LDIO) mode 55

## M

microprocessor (Apple II) 5  
    register state 18  
ModeSelect 26  
ModeSense 26

## N

NCR 5380 SCSI IC 7–8  
nonextended SmartPort calls  
    19–20, 55–56

## O

Open command 33

## P, Q

parameter list, defined 17  
parameters, defined 11  
Partition Descriptor Map (PDM)  
    47–51  
partitions. *See* device partitions  
Patch1Call 30  
peripheral interface cable 3  
pin-outs  
    SCSI card cable 10  
    system cable 9  
pointer, defined 17  
PreventRemoval 29  
ProDOS calls 17–18  
Programmed I/O (PIO) mode 55  
programs, sample 36–43

## R

RAM 6  
    enabling 53  
Read Block command 22–23  
ReadCapacity 27  
Read command 34

ReadDefectData 27  
ReadHeader 32  
ReadQSubcode 32  
ReadTOC 32  
ReassignBlock 26  
ReceiveDiagnostic 29  
register state (microprocessor) 18  
Release 27  
RequestSense 26  
Reserve 27  
ResetBus 33  
RezeroUnit 29  
ROM 7  
    enabling 53

## S

sample program 36-43  
SCSI, defined 1  
*SCSI American National Standard  
for Information Systems* 1  
SCSI bus (Apple II) 8-10  
SCSI card cable, pin-outs 10  
SCSI commands. *See* command  
    interpreter  
SCSI Device Access Table (SDAT)  
    12-13, 54  
    example 13

SCSI ID number 3-5  
    defined 3  
SCSI management 11-17, 54-56  
SDAT. *See* SCSI Device Access  
    Table  
selecting SCSI card 53  
SendDiagnostic 27  
SetBlockSize 28  
SetInterleave 32  
SetNewSDAT 30  
SetTimeout 28  
shadowing, defined 2  
Small Computer System Interface.  
    *See* SCSI  
SmartPort  
    calls 18-20, 36-43  
    command definitions 20-35  
    extended vs. nonextended calls  
        19-20, 55-56  
    locating 18-19  
StartUnit 29  
startup device, using SCSI card as  
    2  
Status command 20-22  
StopUnit 29  
system cable 3  
    pin-outs 9

## T

TestUnitReady 26  
Ile Pseudo-DMA (PDMA) mode 55  
IIGs Pseudo-DMA (PDMA)  
    mode 55

## U

unit number 3-5  
    defined 3  
unsupported commands (command  
    interpreter) 53-58

## V

Verify code 29

## W, X, Y

Write Block command 23  
Write command 34-35

## Z

zero-page write 17

## THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh® computers and Microsoft® Word. Proof and final pages were created on the Apple LaserWriter® Plus printer. POSTSCRIPT®, the LaserWriter page-description language, was developed by Adobe Systems Incorporated. Some of the illustrations were created using Adobe Illustrator™.

Text type is ITC Garamond® (a downloadable font distributed by Adobe Systems). Display type is ITC Avant Garde Gothic®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier, a fixed-width font.

