Apple II SCSI Card Reference Manual Alpha Draft

June, 1986
Writer: Steven Ford High
Apple User Education Department

Contents

Chapter 1

Introduction to the Apple II SCSI Card

What is SCSI?

The abbreviation SCSI stands for Small Computer System Interface. The Small Computer System Interface is a *standard*, a set of rules governing the creation of peripheral devices, software, and firmware for use with computer systems like the Apple II family of computers.

The idea behind the SCSI standard is very simple: rather then designing products which are compatible with only one or two small computers, a developer designs products which 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 manufacturer's hardware design, it gives the owners of small computers a tremendous amount of flexibility in selecting equipment for use in their systems.

The SCSI standard defines all of the apsects involved in a computer interface, so 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 (ANSC), X3T9.282-2, Rev. 15, 5-8-85".

The Apple II SCSI Card

The SCSI standard is implemented in Apple II family computers by the SCSI Card and the Smartport firmware it contains. The Apple II SCSI Card will work with any SCSI device, *provided that 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 CPUs which support expansion slots. The SCSI Card contains on-board RAM and ROM . Special logic circuits are used to to access and control the RAM and ROM. The SCSI bus is implemented by a NCR 5380 SCSI IC. The SCSI Card also contains circuitry to interface the 5380 IC with the Apple II CPU control signals.

The SCSI Card supports seven external devices when mounted in an Apple II computer running ProDos 8.1.2, or ProDos 16.0. In computers running ProDos 8.1.1 (or earlierversions), the SCSI Card supports four external devices.

The SCSI Card performs data I/O operations in either psuedo–DMA (PDMA) or Block Move mode. Block Move mode is not supported by all Apple II CPUs, so the SCSI Card defaults to PDMA mode. To activate the Block Move mode, you must set a flag in the SCSI Card's firmware. More information on mode selection is provided in Chapter 3.

The SCSI Card hardware has two principal functions, as follows:

- Provide an electrical interface between external SCSI devices and the CPU
- Provide the address and control lines required by the Apple II's microprocessor to access and control the SCSI chip and Smartport firmware

More information on the SCSI Card hardware is provided in Chapter 2.

A quick look at the firmware

The firmware on the SCSI Card is called the Smartport. The Smartport is a program that converts commands issued by the CPU to a format which is compatable with the external SCSI devices connected to the Apple II. Microprocessor commands issued to the Smartport are known as *calls.*

Smartport calls provide the microprocessor with information on the status of the external SCSI devices and allow it to control their operation. The Smartport 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 Smartport will initialize the SCSI bus and boot the operating system off of 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 0, with slot 0 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 the Smartport is unable to find a SCSI device capable of booting the system, it returns control to the Apple II and the boot search continues through lower-priority slots.

Chapter 2

The Apple II SCSI Card hardware

SCSI bus configuration

The Apple II SCSI bus is a peripheral device bus capable of supporting a maximum of eight SCSI devices connected in parallel. The Apple II SCSI Card itself counts as one device, so you can connect up to seven *external* devices to the bus.

The external SCSI devices connected to the Apple II are daisey-chained together, with the SCSI Card the first device in the chain.

Important: A device's position in the chain determines its Unit Number. Unit Numbers are set by the boot firmware during SCSI bus initialization. The Unit Number for the SCSI Card is determined by the position of a jumper on the circuit card.

The SCSI Card is resident on both the Apple II internal bus and the SCSI bus (see Figure 2-1). However, the Apple II accesses and controls the SCSI Card across the internal bus, like any other peripheral card.

External devices reside on the SCSI bus only. The Apple II cannot access these devices directly since 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 (the Smartport) then executes the operation (I/O or control) requested by the Apple II. More information on the role of the Smartport 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 2–2.

Figure'2—2
SCSI Card Block Diagram

CPU Microprocessor

The Apple II microprocessor controls the access to, and operation of, the SCSI Card and the external devices connected to it. The microprocessor controls the operation of the SCSI Card and

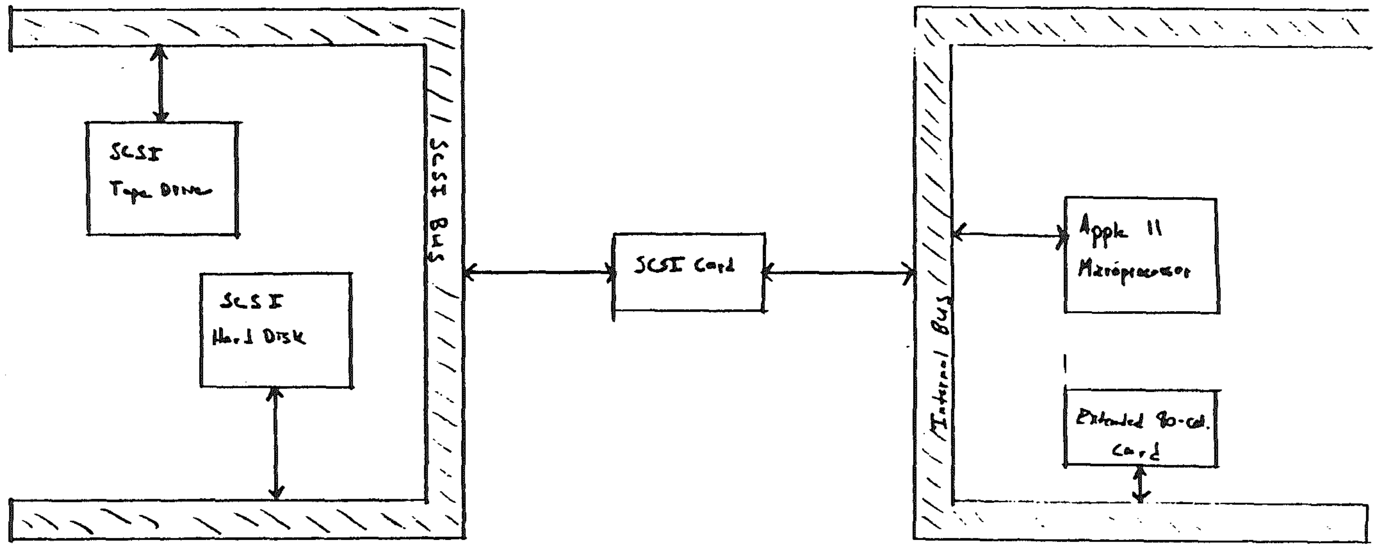

Figure 2-1, Apple II SCSI System Configuration

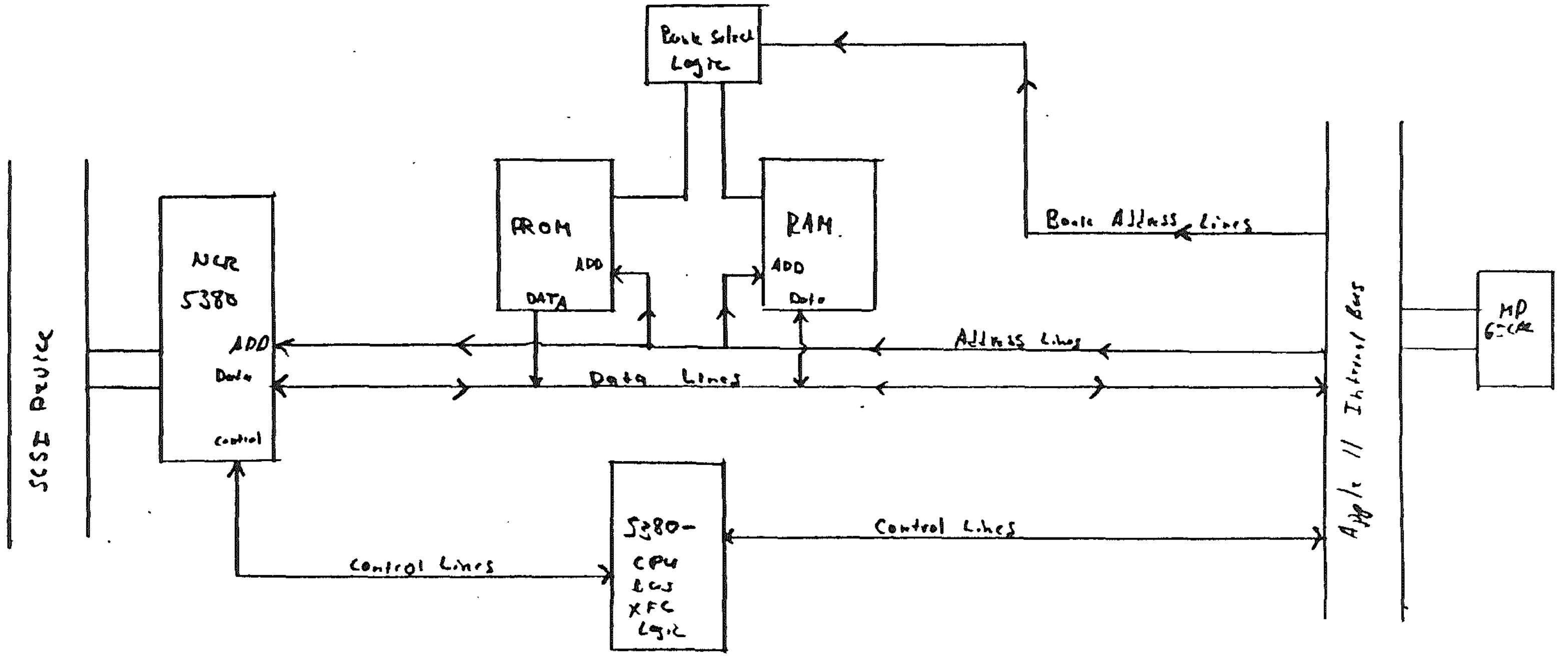

Figure 2-2, SCSI Card Block Diagram

external devices by sending executable instructions to the 5380 IC. The microprocessor also sends all of the addressing signals used by the SCSI Card's RAM and ROM. The addressing signals required by the external devices are sent to the 5380 IC by the microprocessor.

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 your Apple II's technical reference manual.

Bank select logic

In order to access (Read/Write) the on-board ROM and RAM, the Apple II CPU must first select which of the eight 1K banks it wants to use. To select a specific bank, the CPU sends an adressing signal to the bank select logic. This signal is decoded by the bank select logic, which then outputs its own address signal to the ROM or RAM IC (depending on which was addressed in the CPU signal). The signal from the bank select logic sets the ROM or RAM up to receive the data addressing signal sent by the CPU.

RAM

The SCSI Card contains 8K of static RAM in a single 8Kx8 IC. The RAM is divided into 1K banks. The microprocessor controls bank selection through the bank select logic, described above. The SCSI Card's RAM is used to store information about the external SCSI devices collected during bus intitialization. Whenever the microprocesor (or a software program) needs this information, it selects the appropriate bank and sends the Read address to the RAM IC. Figure 2-3 is a map of the RAM.

Figure 2—3
RAM Memory Map

ROM

The SCSI Card contains 8K of ROM in a single 16Kx8 UVEPROM. Like the RAM, the ROM is divided into selectable 1K banks. Bank selection is controlled by the microprocessor through the bank select logic, described above. The SCSI Card's ROM is used to store the Smartport firmware. Whenever a call is issued to the Smartport, whether from the operating system or from a software program, the SCSI Card ROM is Read.

The 8K UVEPROM (2764) used on the SCSI Card can be replaced by a 16K UVEPROM (27128).

Figure 2–4 is a map of the ROM.

Figure'2—4
ROM Memory 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.

The CPU-5380 interface circuitry also contains the the data I/O mode selection logic.

NCR 5380 SCSI IC

The SCSI bus on which the SCSI Card and the external devices reside is implemented by the NCR 5380 IC. This IC provides all of 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 of the data, addressing, and control signals sent to external SCSI devices by the microprocessor or software are handled by the 5380 IC. Appendix B contains the data sheets for the NCR 5380 IC.

The Apple II SCSI bus

The Apple II SCSI bus is implemented 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 CPU side, and a 50–pin connector on the SCSI device side. The SCSI Card is connected to the DB–25 connector on the back panel of the Apple II by a 25–pin ribbon cable. Table 2–1 is a pin–out chart for the System Cable, and Table 2–2 is a pin–out chart for the 25–pin ribbon cable.

Table 2—1, 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—1, System Cable Pin—Out Chart

26-pin P.C.B. connector	Signal Name	Female 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 WIRE	

Chapter 3

The Smartport

The Smartport is a program that converts commands issued by the Apple II microprocessor into SCSI commands. The Smartport provides the software interface bewteen the Apple II (and any program running on it) and the devices resident on the SCSI bus.

How the Smartport works

Whenever the Apple II wants to access a device on the SCSI bus, it must issue a *call* to the Smartport. A Smartport call instructs the Smartport to perform one of several operations on a SCSI device. The call includes a ***command number***, which indicates the precice operation being requested, and a ***pointer*** to a buffer memory in the Apple II that contains a ***parameter list***. The parameter list provides the Smartport with any special command parameters it may need to execute the operation requested by the Apple II microprocessor .

In order to access and control the external SCSI devices, the Smartport needs to store certain information in its on-board RAM. The Smartport stores this critical information in a software table called the SCSI Device Access Tabel (SDAT). each SCSI device, including the SCSI Card, has a SDAT contructed for it by the Smartport at SCSI bus intitialization. Figure 3–1 defines the SDAT data structure.

Figure 3—1
SDAT Data Structure

After it has contructed all of the SDATs, the Smartport constructs another software table in the SCSI Card RAM. This table, called the SDAT Pointer table (PSDAT), contains pointers to the location of each SDAT in SCSI Card RAM. Figure 3–2 defines the data structure for the PSDAT.

Figure 3—2
PSDAT Data Structure

There are other, device-specific, parameters the Smartport needs to know about each of the devices on the SCSI bus. These parameters are stored on the device itself. These parameters, and the way they are stored and used, are described in Appendix A of this manual.

Once a call is issued, the Apple II turns processing control over to the Smartport until the operation requested is finished. When the Smartport completes the operation, it turns control back over to the

Apple II. Program execution resumes at the statement immediately following the Smartport call.

A quick look atSmartport commands

The following sections describe the commands supported by the Apple II SCSI Card Smartport. The number in parenthesis beside each command name is the Smartport command number. Smartport commands may be issued in either standard or extended format. Extended format calls include a 24–bit (or greater) address field in the parameter list. The larger address field is used for data buffer and read/write addresses. Figure 3–3 diagrams the various command formats. The Smartport commands are described in detail later in this Chapter.

Figure'3—3
Parameter List Formats

STATUS ($00)

The STATUS command returns information about a specific device on the SCSI bus. The information returned by this command includes the following:

- *General* status information
  - Type of device (character or block)
  - Type of protection (Read/Write), if any
  - Format or formats allowed on the device
  - Device state (on or off-line)
- *Device-specific* status information
  - Number of logical blocks residing on the device
  - Device name (in ASCII)
  - Device type
  - Device subtype
  - Device revision number

The device type and device subtype fields describe the device's physical type, such as "tape drive", etc... The device type and subtype is described using a *mnemonic* .

Some of these calls are not necessary!

A **mnemonic** is a special type of abbreveation, series of letters and/or numbers which represent a longer or more complicated name.

READ BLOCK ($01)

The READ BLOCK command reads a 512–byte block from a device.

WRITE BLOCK ($02)

The WRITE BLOCK command writes a a 512–byte block to a device.

FORMAT ($03)

The FORMAT command prepares all of the blocks on a block device for READ/WRITE use. FORMAT **does not** prepare blocks for use by a specific operating system.

CONTROL ($04)

The CONTROL command sends control function information to an external device. With the exception of one code, control codes sent by the CONTROL command are device-specific. Code $00 is a soft resset order; it is not device-specific.

INIT ($05)

The INIT command resets the Smartport. During initialization, the Smartport executes a hard reset on all devices connected to the SCSI bus.

OPEN ($06)

The OPEN command prepares a character device for Read/Write operations.

CLOSE ($07)

The CLOSE command informs a character device that a Read/Write operation sequence is complete.

READ ($08)

The READ command reads a specified number of bytes from a device.

WRITE ($09)

The WRITE command writes a specified number of bytes to a device.

Using Block Move mode

Apple II CPUs with the Motorola 65C816 microprocessor can run data I/O operations in Block Move mode.

A—DRAFT NOTE: The specifics of using the Block Move mode have not yet been defined by engineering.

Making a Smartport call

The Operating System or program running on the Apple II executes a call to the Smartport in three basic stages, as follows:

- Smartport location
- Command parameters Zero-Page write
- 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 compatability 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 (this is the address used by the SCSI Card)

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 compatability between software and future hardware developments.

Command parameters Zero-Page write

Prior to issuing a Smartport 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 Smartport 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.

Smartport call

The final stage is the actual Smartport call. A Smartport call is coded as a JSR to the Smartport entry point (DISPATCH), followed by the one-byte command number, followed by the two-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 address CnFF.

An example of a Smartport call is as follows:

```
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
```

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). The state of these flags depends on whether or not the Smartport call was successfully completed. Table 3-1 defines the state of the microprocessor flags for both successful and unsuccessful Smartport calls.

Table 3—1, 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
SMP	JSR+3	JSR+3
S	unchanged	unchanged

*x is undefined unless index information is returned in X and Y.

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
- **Parameter Description:** describes the parameters in the parameter list
- **Error Codes:** a list of possible error codes returned by the command

STATUS ($00)

Description

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

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

Parameter list

- **Parameter Count (0):** 1 byte, set to 3 for this command
- **Unit Number (1):** 1 byte, in the range $00—$7E
- **Status List Pointer (2,3)*:** 1 word, data buffer start address
- **Status Code (4):** 1 byte, in the range $00—$FF

*These two bytes are processed as a single, one-word (16–bit) parameter.

Parameter description

Unit Number:"This parameter contains the unit number of the target SCSI device, as loaded into the Zero–Page address $43. If the Unit Number is set to$00, all devices resident on the SCSI bus are targeted.

Status List Pointers:"This parameter contains the beginning 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 Device Status
- $01 Device Control Block
- $02 Newline Status (character devices only)

- $03 Device Information Block

Code $00: the status list returned for this code is four bytes long. The first byte is the general status byte, and the remaining three bytes are the size of the device in 512–byte blocks. The general status byte is returned for all device types, but the size bytes are only returned for block–type devices. Table 3-2 defines the information returned in the general status byte.

Table 3—2, 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=Device issuing interrupt
0	1=Device Open (character device)

Unit Number $00: a Code $00 STATUS command with the Unit Number set to $00 returns the status of the Smartport itself. The status list returned is 8 bytes long. Table 3–3 defines the content of the Smartport status list.

Table 3—3, Smartport Status Bytes Contents

Byte	Definition
0	Number of SCSI devices on SCSI bus
1	Interrupt state (Bit 7=0 indicates inactive)
2—7	Reserved

Bit 7 of Byte 1 is used to indicate that a device on the SCSI bus has generated an interrupt. The status list does not indicate the specific device that generated the interrupt; the Apple II must poll each of the devices on the SCSI bus to determine which device requires interrupt handling.

Code$01: the status list returned for this code ranges from 1 to 256 bytes long, depending on the content of the Device Control Block (DCB). The first byte of the DCB contains the number of bytes in the block. The information returned in this status list is device-dependent.

Code$02: the status list returned for this code has not been defined.

Code$03: the status list returned for this code is 23 bytes long. The first four bytes are identical to the bytes returned for **Code** $00. The remaining bytes are device-specific status information. Table 3-4 defines the information returned in the device-specific status bytes.

Table 3—4, Device—Specific Status Bytes

Byte	**Definition**
1	General Status Byte
2	Number of blocks on device, low byte
3	Number of blocks on device, middle byte
4	Number of blocks on device, high byte
5	Length of device ID string in bytes (16 max)
6—20	Device Name, in ASCII (16 bytes)
21	Device type mnemonic
22	Device subtype mnemonic
23	Device firmware version number

Error Codes:"Table 3–5 defines the possible error codes for the STATUS command.

Table 3—5, Possible Error Codes

Code	**Mnemonic**	**Definition**
$06	BUSERR	SCSI bus error
$21	BADCTL	Illegal Status Code
$30—$3F	$50—$7F	Device-specific error

READ BLOCK($01)

Description

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 bank $00 RAM at the address specified in the data buffer pointer.

Parameter list

- **Parameter Count (0):** 1 byte, set to 3 for this command
- **Unit Number (1):** 1 byte, in the range $00—$7E
- **Data Buffer Pointer (2,3)*:** 1 word, data buffer address
- **Block Number (4,5,6)*:** 3 bytes, logical address of target block

*These bytes are processed as a single parameter.

Parameter description

Unit Number:"This parameter contains the unit number of the target SCSI device, as loaded into the Zero–Page address $43.

Data Buffer Pointer:"This parameter contains the beginning address of the data buffer used to store the block read 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.

Block Numbers:"This parameter contains the logical address, on the host device, of the target block. Byte 4 is the low byte of this address, byte 5 the middle byte, and byte 6 the high byte.

Important: Logical–to–physical address translation is not performed by the Smartport; translation must be performed by the target device.

Error Codes:"Table 3–6 defines the possible error codes for the READ BLOCK command.

Table 3—6, Possible Error Codes

Code	Mnemonic	Definition
$06	BUSERR	SCSI bus error
$27	IOERROR	I/O Error
$28	NODRIVE	Target device not present on SCSI bus
$2D	BADBLOCK	Illegal Block Number
$2F	OFFLINE	Device not on-line; no disk in drive

WRITE BLOCK($02)

Description

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 bank $00 RAM at the address specified in the data buffer pointer.

Parameter list

- **Parameter Count (0):** 1 byte, set to 3 for this command
- **Unit Number (1):** 1 byte, in the range $00—$7E
- **Data Buffer Pointer (2,3)*:** 1 word, data buffer address
- **Block Number (4,5,6)*:** 3 bytes, logical address of target block

*These bytes are processed as a single parameter.

Parameter description

Unit Number:"This parameter contains the unit number of the target SCSI device, as loaded into the Zero–Page address$43.

Data Buffer Pointer:"This parameter contains the beginning address of the data buffer from which the target block is written. The first byte (byte 2) is the low byte, and the second byte (byte 3) is the high byte of the address.

Block Numbers:"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 the middle byte, and byte 6 the high byte.

Important: Logical–to–physical address translation is not performed by the Smartport; translation must be performed by the target device.

Error Codes:"Table 3–7 defines the possible error codes for the WRITE BLOCK command.

Table 3—7, Possible Error Codes

Code	Mnemonic	Definition
$06	BUSERR	SCSI bus error
$27	IOERROR	I/O Error
$28	NODRIVE	Target device not present on SCSI bus
$2B	NOWRITE	Target device/media Write-protected
$2D	BADBLOCK	Illegal Block Number
$2F	OFFLINE	Device not on-line; no media in drive

FORMAT ($03)

Description

The FORMAT command prepares all of 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.

Important: Operating System-specific information, such as bitmaps and catalogs, **is not** prepared by the FORMAT command.

Parameter list

- **Parameter Count (0):** 1 byte, set to 1 for this command
- **Unit Number (1):** 1 byte, in the range $00—$7E

Parameter description

Unit Number: This parameter contains the unit number of the target SCSI device, as loaded into the Zero-Page address$43.

Error Codes: Table 3-8 defines the possible error codes for the FORMAT command.

Table 3—8, Possible Error Codes

Code	Mnemonic	Definition
$06	BUSERR	SCSI bus error
$27	IOERROR	I/O Error
$28	NODRIVE	Target device not present on SCSI bus
$2B	NOWRITE	Target device/media Write-protected
$2F	OFFLINE	Device not on-line; no media in drive

CONTROL ($04)

Description

The CONTROL command writes a 512–byte control block to the device specified in the Unit Number parameter. The control block written to the target device contains a control command and a list of data required by the target device to execute that command.

Parameter list

- **Parameter Count (0):** 1 byte, set to 3 for this command
- **Unit Number (1):** 1 byte, in the range $00—$7E
- **Control List Pointers (2,3)*:** 1 word, data buffer low and high address
- **Control Code (4):** 1 byte, in the range $00—$FF

*These two bytes are processed as a single, one-word (16–bit) parameter.

Parameter description

Unit Number:"This parameter contains the unit number of the target SCSI device, as loaded into the Zero–Page addres $43. If the Unit Number is set to$00, all devices resident on the SCSI bus are targeted and a special set of control commands is used (see **Unit Number $00 Commands**, below).

Control List Pointer:"This parameter contains the beginning address of the data buffer used to store the control list information required 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.

Control Code:"This parameter contains the hexadecimal code number indicating which control command is being issued. With the exception of the mandatory control commands defined in Table 3–9, control commands are device specific. All Apple II SCSI devices must support the control commands defined in Table 3–9.

Table 3—9, Mandatory Control Commands

Code	Definition
$00	Device Reset
$01	Set device Control Block (DCB)
$02	Set newline status (character device only)
$03	Handle device interrupt

Code$00: this command orders a soft reset of the target device. The control list contains device-specific information required by the device controller to reinitialize itself.

Code$01: this command writes the DCB. The DCB is typically used to control the operating parameters of the device itself. Since the length of the DCB is device-dependent, Apple recommends the following procedure for altering the DCB:

1) Read in the DCB by using the STATUS $01 command

2) Make the desired changes

3) Write the modified DCB using this control command

Code$02: **NOT DEFINED**

Code$03: **NOT DEFINED**

Unit Number $00 Commands: A CONTROL command issued with a Unit Number of $00 affects all devices on the SCSI bus. This type of CONTROL command uses a special set of control commands.

Code$00: this command enables interrupt handling for the Smartport.

Code$01: this command disables interrupt handling for the Smartport.

Error Codes:"Table 3-10 defines the possible error codes for the CONTROL command.

Table 3—10, Possible Error Codes

Code	Mnemonic	Definition

$06	BUSERR	SCSI bus error
$21	BADCTL	Illegal Status Code
$22	BADCTLPARM	Invalid parameter list
$30—$3F	none	Device-specific error

INIT ($05)

Description

The INIT command forces the Smartport to reinitialize itself. All of the devices on the SCSI bus are hard reset, and new device numbers assigned, where necessary.

Parameter list

- **Parameter Count (0):** 1 byte, set to 1 for this command
- **Unit Number (1):** 1 byte, set to $00 (Smartport) for this command

Parameter description

Unit Number:"This parameter targets the Smartport.

Error Codes:"Table 3–11 defines the possible error codes for the INIT command.

Table 3—11, Possible Error Codes

Code	Mnemonic	Definition
$06	BUSERR	SCSI bus error
$28	NODRIVE	Target device not present on SCSI bus

OPEN ($06)

Description

The OPEN command opens a logical file on the target device for data I/O. This command is used for character devices only.

Parameter list

- **Parameter Count (0):** 1 byte, set to 1 for this command
- **Unit Number (1):** 1 byte, in the range $01—$7E

Parameter description

Unit Number:"This parameter contains the unit number of the target SCSI character device, as loaded into the Zero-Page address$43.

Error Codes:"Table 3–12 defines the possible error codes for the OPEN command.

Table 3—12, Possible Error Codes

Code	**Mnemonic**	**Definition**
$01	BADCMD	Illegal command
$06	BUSERR	SCSI bus error
$28	NODRIVE	Target device not present on SCSI bus

CLOSE ($07)

Description

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.

Parameter list

- **Parameter Count (0):** 1 byte, set to 1 for this command
- **Unit Number (1):** 1 byte, in the range $01—$7E

Parameter description

Unit Number:"This parameter contains the unit number of the target SCSI character device, as loaded into the Zero-Page address$43.

Error Codes:"Table 3-13 defines the possible error codes for the CLOSE command.

Table 3—13, Possible Error Codes

Code	Mnemonic	Definition
$01	BADCMD	Illegal command
$06	BUSERR	SCSI bus error
$28	NODRIVE	Target device not present on SCSI bus

READ ($08)

Description

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 is written into bank $00 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.

Parameter list

- **Parameter Count (0):** 1 byte, set to 4 for this command
- **Unit Number (1):** 1 byte, in the range $01—$7E
- **Data Buffer Pointer (2,3)*:** 1 word, data buffer address
- **Byte Count (4,5)*:** 2 bytes, number of bytes to read
- **Address Pointer (4,5,6)*:** 3 bytes, device-specific parameter

*These bytes are processed as a single parameter.

Parameter description

Unit Number:"This parameter contains the unit number of the target SCSI device, as loaded into the Zero-Page address$43.

Data Buffer Pointer:"This parameter contains the beginning address of the host data buffer to which the target bytes are written. The first byte (byte 2) is the low byte, and the second byte (byte 3) is the high byte of the address.

Byte Count:"This parameter contains the number of bytes to read for this command. Byte 4 is the low byte of this number, and byte 5 the high byte.

Address Pointer:"This parameter contains the addressing information required by the target device. This parameter is device-specific.

Important: Logical–to–physical address translation is not performed by the Smartport; translation must be performed by the target device.

Error Codes:"Table 3–14 defines the possible error codes for the READ command.

Table 3—14, Possible Error Codes

Code	Mnemonic	Definition
$06	BUSERR	SCSI bus error
$27	IOERROR	I/O Error
$28	NODRIVE	Target device not present on SCSI bus
$2B	NOWRITE	Target device/media Write–protected
$2D	BADBLOCK	Illegal Block Number
$2F	OFFLINE	Device not on–line; no media in drive

WRITE ($09)

Description

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 bank $00 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.

Parameter list

- **Parameter Count (0):** 1 byte, set to 4 for this command
- **Unit Number (1):** 1 byte, in the range $01—$7E
- **Data Buffer Pointer (2,3)*:** 1 word, data buffer address
- **Byte Count (4,5)*:** 2 bytes, number of bytes to read
- **Address Pointer (4,5,6)*:** 3 bytes, device-specific parameter

*These bytes are processed as a single parameter.

Parameter description

Unit Number:"This parameter contains the unit number of the target SCSI device, as loaded into the Zero-Page address$43.

Data Buffer Pointer:"This parameter contains the beginning address of the data buffer from which the target bytes are written. The first byte (byte 2) is the low byte, and the second byte (byte 3) is the high byte of the address.

Byte Count:"This parameter contains the number of bytes to write for this command. Byte 4 is the low byte of this number, and byte 5 the high byte.

Address Pointer:"This parameter contains the addressing information required by the target device. This parameter is device-specific.

Important: Logical-to-physical address translation is not performed by the Smartport; translation must be performed by the target device.

Error Codes:"Table 3-15defines the possible error codes for the WRITE command.

Table 3—15, Possible Error Codes

Code	Mnemonic	Definition
$06	BUSERR	SCSI bus error
$27	IOERROR	I/O Error
$28	NODRIVE	Target device not present on SCSI bus
$2B	NOWRITE	Target device/media Write-protected
$2D	BADBLOCK	Illegal Block Number
$2F	OFFLINE	Device not on-line; no media in drive

An example of a Smartport call

The following text is an example of a Smartport call in a program.

A—DRAFT NOTE: An example of an applicable Smartport call in a program is not available as of this writing.

Appendix A

Device Partioning

Device partioning is a means of setting up a SCSI block-type device for use by multiple operating systems by dividing it into a number of sections. These sections are called device partions.

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

Each of the partions on a device are defined in a software table called the Device Partion Map (DPM). The DPM is itself a partion, and is stored on the device it describes. The DPM is described below.

Device partioning currently supports hard disk drives only.

The Device Partion Map (DPM)

The DPM is a software table made up of a variable number of 1-block entries. The entries in the DPM are called Partion Descriptor Maps (PDMs). One PDM is contructed for each partion on the device, including the DPM partion 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 partions on the device. Figure A-1 shows the structure of the DPM.

Figure A—1
The DPM

The Partion Descriptor Map

The Partion Descriptor Map is a one-block wide entry in the DPM table. The PDM consists of a series of data fields that define the partion. Figure A-2 shows the structure of the PDM. The following sections define the PDM data fields.

Signature

This two-byte field is set to the PDM ID string, indicating that this block is a PDM.. The PDM ID string is $504D_{16}$.

DPM Block Count

This 32-bit field is set to the number of blocks in the Device Partion Map. because the PDM for the Device Partion Map may not be the first PDM on the device, this field must be included for every PDM.

Name

This field is filled with the *physical* name of the partion, which may differ from the *logical name* of the partion. This field is an ASCII string with a length of 32 bytes. Upper and lower-case characters are not distinguished.

Important: If this field is filled with a string of less than 32 bytes, the entry must be terminated with a **NUL** (binary zero) character.

This field may be left empty, provided the first byte is filled with the **NUL** character.

Type

This field is filled with an ASCII string describing the purpose or use of the partion, such as the name of the operating system that uses it. This field is an ASCII string with a length of 32 bytes. Upper and lower-case characters are not distinguished.

Important: If this field is filled with a string of less than 32 bytes, the entry must be terminated with a **NUL** (binary zero) character.

This field may be left empty, provided the first byte is filled with the **NUL** character.

All Type strings which begin with **"Apple"** are reserved for use by Apple Computer.

Start

This 32–bit field is set to the number of the first **physical** block allocated to the partion.

Size

This 32–bit field is set to the number of blocks allocated to the partion.

Data Start

This 32–bit field is set to the number of the first block allocated to the partion that contains valid data. This number may differ from the one set in the Start field due to the presence of bad blocks or blocks used for boot code, etc...

Data Size

This 32–bit field is set to the number of blocks allocated to the partion that contain valid data.

Status

This 32–bit field contains eight 1–bit status flags. These flags are defined as follows:

PDM Valid?

This flag is set to 1 for a valid PDM.

Allocated?

This flag is set to 1 if the partion has been allocated by an operating system or systems.

In Use?

This flag is set to 1 if the partion is currently being accessed. This flag is intended for use in multi-processor environments.

Boot OK?

This flag is set to 1 if the partion contains valid boot code.

Read OK?

This flag is set to 1 if the managment operating system for this partion allows reading of the partion. This flag does not apply to operating systems other than the operating system to which the partion is allocated.

Write OK?

This flag is set to 1 if the managment operating system for this partion allows writing to the partion. This flag does not apply operating systems other than the operating system to which the partion is allocated.

PIND Boot?

This flag is set to 1 if the boot code contained in the partion is position-independent. For some microprocessors, this flag indicates whether or not the LOAD AT and JUMP addresses must be adhered to.

OSSF

These flags are not defined; they are available for use by the operating system to which the partion is allocated.

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

●Boot Start

This 32–bit field contains the boot code starting block number.

●Boot Size

This 32–bit field is set to the number of bytes of boot code contained in the partion.

●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. C code for this routine is as follws:

```
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. Upper and lower–case characters are not distinguished.

Important: If this field is filled with a string of less than 16 bytes, the entry must be terminated with a **NUL** (binary zero) character.

This field may be left empty, provided the first byte is filled with the **NUL** character.

●Boot Arg

These four 32–bit fields contains arguments that are boot code–specific.

Creating device partions

To create a partion on a SCSI block device, you must create the PDM in the Device Partion Map. To create the PDM simpy use the Smartport commands to Write all of the necessary fields to the DPM.

Once the PDM is created, your partion is ready to be allocated to an operating system or systems; that is, it is a **valid** partion. Valid partions are available for allocation by any operating system.