Information on SCSI

This information is gleaned from from various freely available sources. Where I know who the author is, I have given credit. I make no promises as to the accuracy of this information. I don't write it, I just collect it. To the best of my knowledge, all of this information is public domain.

Various text files:
Frequently Asked Questions - comp.periphs.scsi (Internet)
Buyer's Guide - by Roy Neese
DRIVING US CRAZY, BUT FOR A REASON by Alan Brenden
Adaptec 1540/1542 hints by Darryl Okahata
WDSCS-ATXT

SCSI 2 documents:
ANSI Common Access Method rev 2.3
ANSI ATA rev 2.3
How to Get SCSI Standards
ASPI

Compiled and adapted by: Neil G. Rowland, Jr.

ASPI

ASPI is an emerging software standard for SCSI drivers. The spec is copyrighted by **Adaptec**, so I can't include it here. You can get download it yourself from Adaptec's Bulletin Board at (408)945-7727. The file name is **ASPIDOS.TXT**.

The basic idea is there's a device you can open and close via the usual INT 21h calls. You do an IOCTL to get the address of a driver routine. Then call this routine to do all SCSI operations, passing it a pointer to what's called an SRB (SCSI Request Block), that encodes the command.

ASPI is implemented by a TSR named **ASPI4DOS.EXE**.

- NGR

Copies of this proposal may be purchased from: BSR X3.***
Global Engineering, 2805 McGaw St, Irvine, CA 92714 X3T9.2/90-143
800-854-7179 714-261-1455

working draft proposed American National
Standard for Information Systems -

ATA (AT Attachment)

Rev 2.3 January 30, 1991

(Part of **ANSI ATA rev 2.3**)

Secretariat

Computer and Business Equipment Manufacturers Association (CBEMA)

Abstract: This standard defines the software interface between device drivers and the Host Bus Adapters or other means by which SCSI peripherals are attached to a host processor. The software interface defined provides a common interface specification for systems manufacturers, system integrators, controller manufacturers, and suppliers of intelligent peripherals.

This is an internal working document of X3T9.2, a Task Group of Accredited Standards Committee X3. As such this is not a completed standard. The contents are actively being modified by the X3T9.2 Task Group. This document is made available for review and comment only.

POINTS OF CONTACT:

John B. Lohmeyer
Chairman X3T9.2
NCR
3718 N Rock Rd
Wichita KS 67226

316-636-8703

I. Dal Allan
Vice-Chairman X3T9.2
ENDL
14426 Black Walnut Court
Saratoga CA 95070

408-867-6630

An electronic copy of this document is available from the SCSI Bulletin Board (316-636-8700).

This document has been prepared according to the style guide of the ISO (International Organization of Standards).

If this document was printed in a 2-up form directly from the printer, NOTEs had to be adjusted to fit into a half-page, which may have resulted in an imperfect representation of the format within the NOTE. This is most likely to occur if a series of NOTEs are mixed in without any line separation.

This document identifies all changes made since Rev 2.1 of June, 1990.

Foreword (This Foreword is not part of American National Standard X3.***- 199x.)

When the first IBM PC (Personal Computer) (tm) was introduced, there was no hard disk capability for storage. Successive generations of product resulted in the inclusion of a hard disk as the primary storage device. When the PC AT (tm) was developed, a hard disk was the key to system performance, and the controller interface became a de facto industry interface for the inclusion of hard disks in PC ATs.

The price of desktop systems has declined rapidly because of the degree of integration to reduce the number of components and interconnects required to build a product. A natural outgrowth of this integration was the inclusion of controller functionality into the hard disk.

In October 1988 a number of peripheral suppliers formed the Common Access Method Committee to encourage an industry-wide effort to adopt a common software interface to dispatch input/output requests to SCSI peripherals. Although this was the primary objective, a secondary goal was to specify what is known as the AT Attachment interface.

Suggestions for improvement of this standard will be welcome. They should be sent to the Computer and Business Equipment Manufacturers Association, 311 First Street N.W., Suite 500, Washington, DC 20001.

This standard was processed and approved for submittal to ANSI by the Accredited Standards Committee on Information Processing Systems, X3. Committee approval of this standard does not necessarily imply that all committee members voted for its approval. At the time it approved this standard, the X3 Committee had the following members:

X3 Committee list goes here:

Subcommittee X3T9 on I/O interfaces, which reviewed this standard, had the following members:

X3T9 Committee list goes here:

Task Group X3T9.2 on Lower-Level Interfaces, which completed the development of this standard, had the following members:

X3T9.2 Committee list goes here:

The initial development work on this standard was done by the CAM Committee.

The membership of the CAM Committee consisted of the following organizations:

Adaptec	Data Technology	NCR
AMD	Eastman Kodak	Olivetti
Apple	Emulex	Quantum
AT&T Bell Labs	Fujitsu uElectronics	Scientific Micro Systems
Caliper	Future Domain	Seagate
Cambrian Systems	Hewlett Packard	Sony
Cipher Data	IBM	Storage Dimensions
Cirrus Logic	Imprimis	Sun Microsystems
Columbia Data	Interactive Systems	Syquest Technology
CompuAdd	JVC	Sytron
Conner Peripherals	LMS OSD	Trantor
Dell Computer	Maxtor	Western Digital
Digital Equipment	Micropolis	
DPT	Miniscribe	

ANSI ATA - Table of Contents

(Part of **ANSI ATA rev 2.3**)

TABLE OF CONTENTS

FIGURES

TABLES

Information Processing Systems --

AT Attachment Interface

1. Scope

(Part of **ANSI ATA rev 2.3**)

1. Scope

This standard defines the AT Attachment Interface.

The CAM Committee was formed in October, 1988 and the first working document of the AT Attachment was introduced in March, 1989.

1.1 Description of Clauses

Clause 1 contains the Scope and Purpose.

Clause 2 contains Referenced and Related International Standards.

Clause 3 contains the General Description.

Clause 4 contains the Glossary.

Clause 5 contains the electrical and mechanical characteristics; covering the interface cabling requirements of the DC, data cables and connectors.

Clause 6 contains the signal descriptions of the AT Attachment Interface.

Clause 7 contains descriptions of the registers of the AT Attachment Interface.

Clause 8 describes the programming requirements of the AT Attachment Interface.

Clause 9 contains descriptions of the commands of the AT Attachment Interface.

Clause 10 contains an overview of the protocol of the AT Attachment Interface.

Clause 11 contains the interface timing diagrams.

Annex A is informative.

Annex B is informative.

2. References

(Part of **ANSI ATA rev 2.3**)

2. References

None.

3. General Description

(Part of **ANSI ATA rev 2.3**)

3. General Description

The application environment for the AT Attachment Interface is any computer which uses an AT Bus or 40-pin ATA interface.

The PC AT Bus (tm) is a widely used and implemented interface for which a variety of peripherals have been manufactured. As a means of reducing size and cost, a class of products has emerged which embed the controller functionality in the drive. These new products utilize the AT Bus fixed disk interface protocol, and a subset of the AT bus. Because of their compatibility with existing AT hardware and software this interface quickly became a de facto industry standard.

The purpose of the ATA standard is to define the de facto implementations.

Software in the Operating System dispatches I/O (Input/Output) requests via the AT Bus to peripherals which respond to direct commands.

3.1 Structure

This standard relies upon specifications of the mechanical and electrical characteristics of the AT Bus and a subset of the AT Bus specifically developed for the direct attachment of peripherals.

Also defined are the methods by which commands are directed to peripherals, the contents of registers and the method of data transfers.

4. Definitions and Conventions

(Part of **ANSI ATA rev 2.3**)

4. Definitions and Conventions

4.1 Definitions

For the purpose of this standard the following definitions apply:

4.1.1 ATA (AT Attachment): ATA defines a compatible register set and a 40-pin connector and its associated signals.

4.1.2 Data block: This term describes a data transfer, and is typically a single sector, except when declared otherwise by use of the Set Multiple command.

4.1.3 DMA (Direct Memory Access): A means of data transfer between peripheral and host memory without processor intervention.

4.1.4 Optional: This term describes features which are not required by the standard. However, if any feature defined by the standard is implemented, it shall be done in the same way as defined by the standard. Describing a feature as optional in the text is done to assist the reader. If there is a conflict between text and tables on a feature described as optional, the table shall be accepted as being correct.

4.1.5 PIO (Programmed Input/Output): A means of data transfer that requires the use of the host processor.

4.1.6 Reserved: Where this term is used for bits, bytes, fields and code values; the bits, bytes, fields and code values are set aside for future standardization, and shall be zero.

4.1.7 VU (Vendor Unique): This term is used to describe bits, bytes, fields, code values and features which are not described in this standard, and may be used in a way that varies between vendors.

4.2 Conventions

Certain terms used herein are the proper names of signals. These are printed in uppercase to avoid possible confusion with other uses of the same words; e.g., ATTENTION. Any lowercase uses of these words have the normal American- English meaning.

A number of conditions, commands, sequence parameters, events, English text, states or similar terms are printed with the first letter of each word in uppercase and the rest lowercase; e.g., In, Out, Request Status. Any lowercase uses of these words have the normal American-English meaning.

The American convention of numbering is used i.e., the thousands and higher multiples are separated by a comma and a period is used as the decimal point. This is equivalent to the ISO convention of a space and comma.

American:	ISO:
0.6	0,6
1,000	1 000
1,323,462.9	1 323 462,9

5. Interface Cabling Requirements

(Part of **ANSI ATA rev 2.3**)

5. Interface Cabling Requirements

5.1 Configuration

This standard provides the capability of operating on the AT Bus in a daisy chained configuration with a second drive that operates in accordance with these standards. One drive (selected as Drive 0) has been referred to as the master in industry terms and the second (selected as Drive 1) has been referred to as the slave (see Figure 5-3).

The designation as Drive 0 or Drive 1 is made by a jumper plug or switch on the drive.

Data is transferred in parallel (8 or 16 bits) either to or from host memory to the drive's buffer under the direction of commands previously transferred from the host. The drive performs all of the operations necessary to properly write data to, or read data from, the disk media. Data read from the media is stored in the drive's buffer pending transfer to the host memory and data is transferred from the host memory to the drive's buffer to be written to the media.

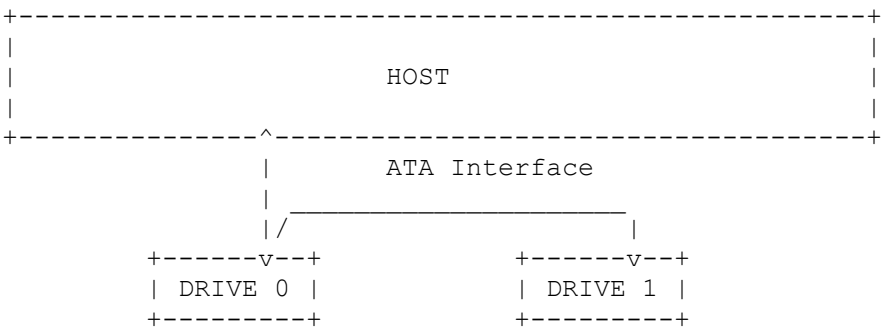


FIGURE 5-1: ATA INTERFACE TO EMBEDDED BUS PERIPHERALS

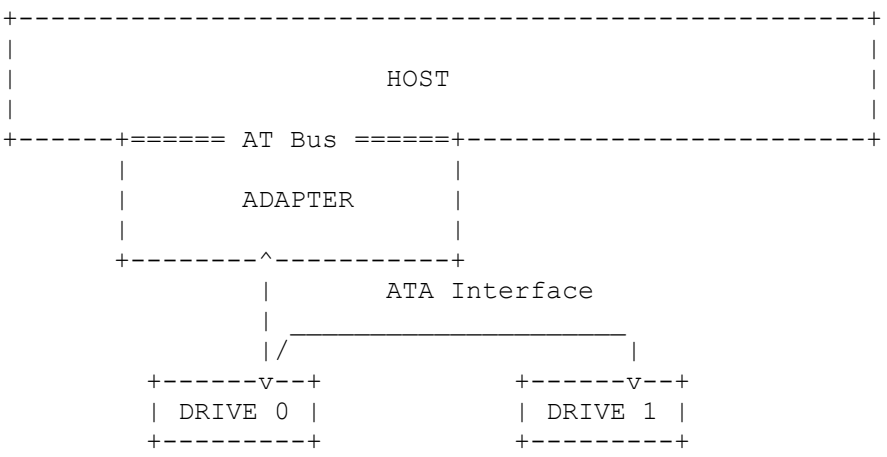
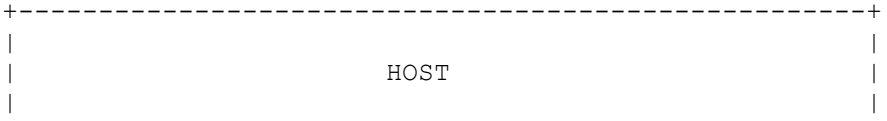


FIGURE 5-2: HOST BUS ADAPTER AND PERIPHERAL DEVICES



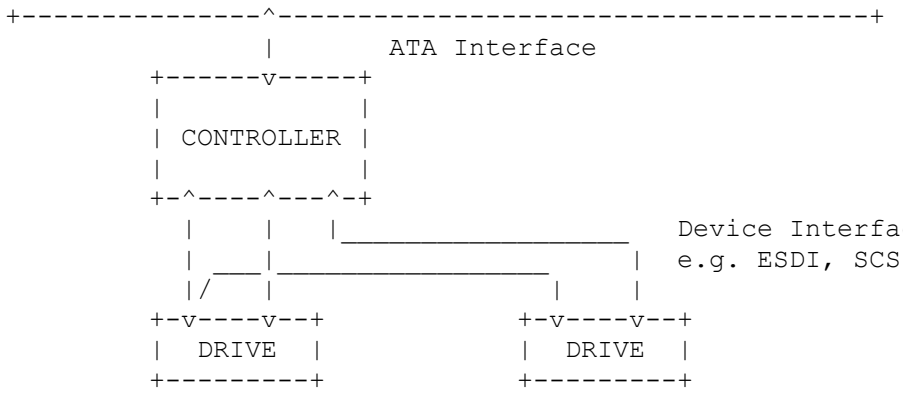


FIGURE 5-3: ATA INTERFACE TO CONTROLLER AND PERIPHERAL DEVICES

5.2 Addressing Considerations

In traditional controller operation, only the selected controller receives commands from the host following selection. In this standard, the register contents go to both drives (and their embedded controllers). The host discriminates between the two by using the DRV bit in the Drive/Head Register.

5.3 DC Cable and Connector

The drive receives DC power through a 4-pin or a low-power application 3-pin connector.

5.3.1 4-Pin Power

The pin assignments are shown in Table 5-1. Recommended part numbers for the mating connector to 18AWG cable are shown below, but equivalent parts may be used.

```
Connector (4 Pin)          AMP 1-480424-0 or equivalent.
Contacts (Loose Piece)     AMP 60619-4 or equivalent.
Contacts (Strip)           AMP 61117-4 or equivalent.
```

TABLE 5-1: DC INTERFACE

POWER LINE DESIGNATION	PIN NUMBER
+12 V	1-01
+12 V RETURN	1-02
+5 V RETURN	1-03
+5 V	1-04

5.3.2 3-Pin Power

The pin assignments are shown in Table 5-2. Recommended part numbers for the mating connector to 18AWG cable are shown below, but equivalent parts may be used.

```
Connector (3 Pin)          Molex 5484 39-27-0032 or equivalent.
```

TABLE 5-2: DC INTERFACE

POWER LINE DESIGNATION	PIN NUMBER

+12 V	1-01
Ground	1-02
+5 V	1-03

5.3.3 Device Grounding

System ground may be connected to a "quick-connect" terminal equivalent to:

Drive Connector Terminal	AMP 61664-1 or equivalent.
Cable Connector Terminal	AMP 62137-2 or equivalent.

Provision for tying the DC Logic ground and the chassis ground together or for separating these two ground planes is vendor specific.

5.4 I/O Connector

The I/O connector is a 40-pin connector as shown in Figure 5-4, with pin assignments as shown in Table 6-1.

The connector should be keyed to prevent the possibility of installing it upside down. A key is provided by the removal of Pin 20. The corresponding pin on the cable connector should be plugged.

The pin locations are governed by the cable plug, not the receptacle. The way in which the receptacle is mounted on the Printed Circuit Board affects the pin positions, and pin 1 should remain in the same relative position. This means the pin numbers of the receptacle may not reflect the conductor number of the plug. The header receptacle is not polarized, and all the signals are relative to Pin 20, which is keyed.

By using the plug positions as primary, a straight cable can connect drives. As shown in Figure 5-4, conductor 1 on pin 1 of the plug has to be in the same relative position no matter what the receptacle numbering looks like. If receptacle numbering was followed, the cable would have to twist 180 degrees between a drive with top-mounted receptacles, and a drive with bottom-mounted receptacles.

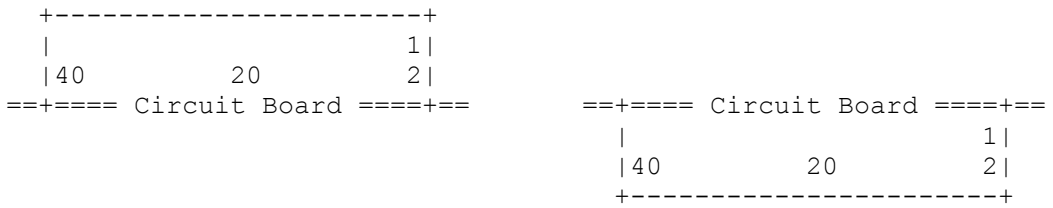


FIGURE 5-4: 40-PIN CONNECTOR MOUNTING

Recommended part numbers for the mating connector are shown below, but equivalent parts may be used.

Connector (40 Pin)	3M 3417-7000 or equivalent.
Strain relief	3M 3448-2040 or equivalent.
Flat Cable (Stranded 28 AWG)	3M 3365-40 or equivalent.
Flat Cable (Stranded 28 AWG)	3M 3517-40 (Shielded) or equivalent.

5.5 I/O Cable

The cable specifications affect system integrity and the maximum length that can be supported in any application.

TABLE 5-3: CABLE PARAMETERS

Cable length of 0.46m (18 inches) *	Min	Max
Driver IoL Sink Current	12mA	
Driver IoH Source Current		-400uA
Cable Capacitive Loading		200pF

* This distance may be exceeded in circumstances where the characteristics of both ends of the cable can be controlled.

6. Physical Interface

(Part of **ANSI ATA rev 2.3**)

6. Physical Interface

6.1 Signal Conventions

Signal names are shown in all upper case letters. Signals can be asserted (active, true) in either a high (more positive voltage) or low (less positive voltage) state. A dash character (-) at the beginning or end of a signal name indicates it is asserted at the low level (active low). No dash or a plus character (+) at the beginning or end of a signal name indicates it is asserted high (active high). An asserted signal may be driven high or low by an active circuit, or it may be allowed to be pulled to the correct state by the bias circuitry.

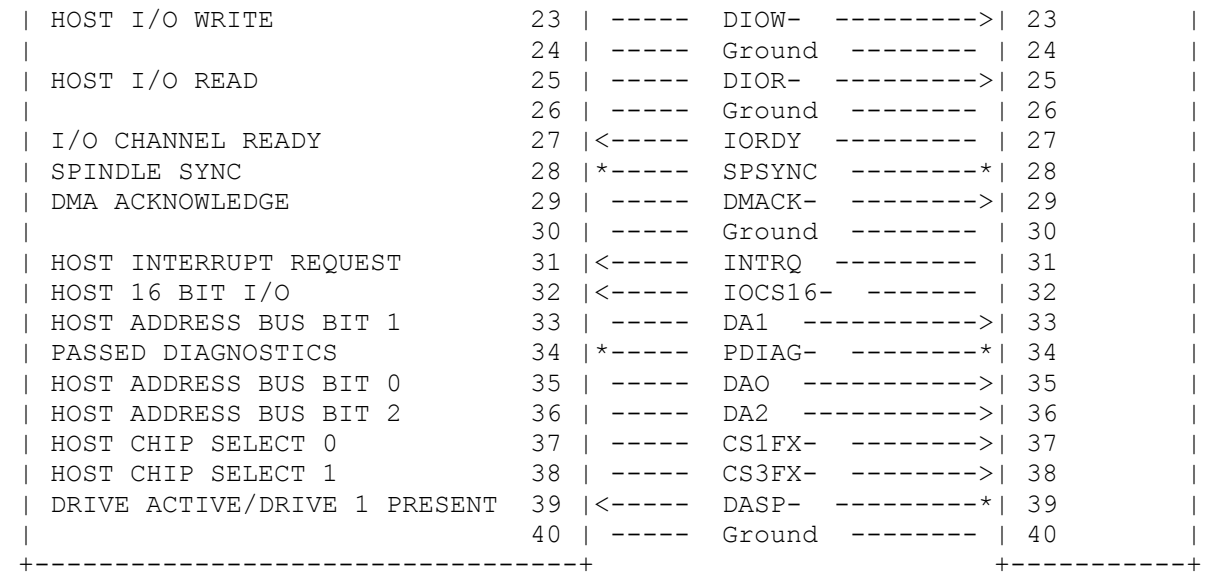
Control signals that are asserted for one function when high and asserted for another function when low are named with the asserted high function name followed by a slash character (/), and the asserted low function name followed with a dash (-) e.g. BITENA/BITCLR- enables a bit when high and clears a bit when low. All signals are TTL compatible unless otherwise noted. Negated means that the signal is driven by an active circuit to the state opposite to the asserted state (inactive, or false) or may be simply released (in which case the bias circuitry pulls it inactive, or false), at the option of the implementor.

6.2 Signal Summary

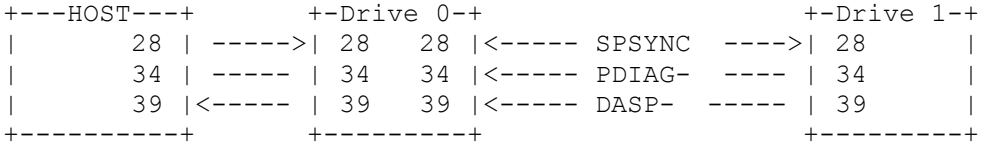
The physical interface consists of single ended TTL compatible receivers and drivers communicating through a 40-conductor flat ribbon nonshielded cable using an asynchronous interface protocol. The pin numbers and signal names are shown in Table 6-1. Reserved signals shall be left unconnected.

TABLE 6-1: INTERFACE SIGNALS

HOST I/O CONNECTOR				DRIVE I/O CONNECTOR
HOST RESET	1	----- RESET- -------->		1
	2	----- Ground --------		2
HOST DATA BUS BIT 7	3	<----- DD7 ---------->		3
HOST DATA BUS BIT 8	4	<----- DD8 ---------->		4
HOST DATA BUS BIT 6	5	<----- DD6 ---------->		5
HOST DATA BUS BIT 9	6	<----- DD9 ---------->		6
HOST DATA BUS BIT 5	7	<----- DD5 ---------->		7
HOST DATA BUS BIT 10	8	<----- DD10 ---------->		8
HOST DATA BUS BIT 4	9	<----- DD4 ---------->		9
HOST DATA BUS BIT 11	10	<----- DD11 ---------->		10
HOST DATA BUS BIT 3	11	<----- DD3 ---------->		11
HOST DATA BUS BIT 12	12	<----- DD12 ---------->		12
HOST DATA BUS BIT 2	13	<----- DD2 ---------->		13
HOST DATA BUS BIT 13	14	<----- DD13 ---------->		14
HOST DATA BUS BIT 1	15	<----- DD1 ---------->		15
HOST DATA BUS BIT 14	16	<----- DD14 ---------->		16
HOST DATA BUS BIT 0	17	<----- DD0 ---------->		17
HOST DATA BUS BIT 15	18	<----- DD15 ---------->		18
	19	----- Ground --------		19
	20	----- (keypin) ------		20
DMA REQUEST	21	<----- DMARQ ---------		21
	22	----- Ground --------		22



* Drive Intercommunication Signals



6.3 Signal Descriptions

The interface signals and pins are described in more detail than shown in Table 6-1. The signals are listed according to function, rather than in numerical connector pin order. Table 6-2 lists signal name mnemonic, connector pin number, whether input to (I) or output from (O) the drive, and full signal name.

TABLE 6-2: INTERFACE SIGNALS DESCRIPTION

Signal	Pin	I/O	
CS1FX-	37	I	Drive chip Select 0
CS3FX-	38	I	Drive chip Select 1
DA0	35	I	Drive Address Bus - Bit 0
DA1	33	I	- Bit 1
DA2	36	I	- Bit 2
DASP-	39	I/O	Drive Active/Drive 1 Present
DD0	17	I/O	Drive Data Bus - Bit 0
DD1	15	I/O	- Bit 1
DD2	13	I/O	- Bit 2
DD3	11	I/O	- Bit 3
DD4	9	I/O	- Bit 4
DD5	7	I/O	- Bit 5
DD6	5	I/O	- Bit 6
DD7	3	I/O	- Bit 7
DD8	4	I/O	- Bit 8
DD9	6	I/O	- Bit 9
DD10	8	I/O	- Bit 10

DD11	10	I/O	- Bit 11
DD12	12	I/O	- Bit 12
DD13	14	I/O	- Bit 13
DD14	16	I/O	- Bit 14
DD15	18	I/O	- Bit 15
DIOR-	25	I	Drive I/O Read
DIOW-	23	I	Drive I/O Write
DMACK-	29	I	DMA Acknowledge
DMARQ	21	O	DMA Request
INTRQ	31	O	Drive Interrupt
IOCS16-	32	O	Drive 16-bit I/O
IORDY	27	O	I/O Channel Ready
PDIAG-	34	I/O	Passed Diagnostics
RESET-	1	I	Drive Reset
SPSYNC	28	-	Spindle Sync
keypin	20	-	Pin used for keying the interface connector.

6.3.1 CS1FX- (Drive chip Select 0)

This is the chip select signal decoded from the host address bus used to select the Command Block Registers.

6.3.2 CS3FX- (Drive chip Select 1)

This is the chip select signal decoded from the host address bus used to select the Control Block Registers.

6.3.3 DA0-2 (Drive Address Bus)

This is the 3-bit binary coded address asserted by the host to access a register or data port in the drive.

6.3.4 DASP- (Drive Active/Drive 1 Present)

This is a time-multiplexed signal which indicates that a drive is active, or that Drive 1 is present. This signal shall be an open collector output and each drive shall have a 10K pull-up resistor.

During power on initialization or after RESET- is negated, DASP- shall be asserted by Drive 1 within 400 msec to indicate that Drive 1 is present.

Drive 0 shall allow up to 450 msec for Drive 1 to assert DASP-. If Drive 1 is not present, Drive 0 may assert DASP- to drive an activity LED.

DASP- shall be negated following acceptance of the first valid command by Drive 1 or after 31 seconds, whichever comes first.

Any time after negation of DASP-, either drive may assert DASP- to indicate that a drive is active.

NOTE: Prior to the development of this standard, products were introduced which did not time multiplex DASP-. Some used two jumpers to indicate to Drive 0 whether Drive 1 was present. If such a drive is jumpered to indicate Drive 1 is present it should work successfully with a Drive 1 which complies with this standard. If installed as Drive 1, such a

drive may not work successfully because it may not assert DASP- for a long enough period to be recognized. However, it would assert DASP- to indicate that the drive is active.

6.3.5 DD0-DD15 (Drive Data Bus)

This is an 8- or 16-bit bidirectional data bus between the host and the drive. The lower 8 bits are used for 8-bit transfers e.g. registers, ECC bytes.

6.3.6 DIOR- (Drive I/O Read)

This is the Read strobe signal. The falling edge of DIOR- enables data from a register or the data port of the drive onto the host data bus, DD0-DD7 or DD0- DD15. The rising edge of DIOR- latches data at the host.

6.3.7 DIOW- (Drive I/O Write)

This is the Write strobe signal. The rising edge of DIOW- clocks data from the host data bus, DD0-DD7 or DD0-DD15, into a register or the data port of the drive.

6.3.8 DMACK- (DMA Acknowledge) (Optional)

This signal shall be used by the host in response to DMARQ to either acknowledge that data has been accepted, or that data is available.

6.3.9 DMARQ (DMA Request) (Optional)

This signal, used for DMA data transfers between host and drive, shall be asserted by the drive when it is ready to transfer data to or from the host. The direction of data transfer is controlled by DIOR- and DIOW-. This signal is used in a handshake manner with DMACK- i.e. the drive shall wait until the host asserts DMACK- before negating DMARQ, and re-asserting DMARQ if there is more data to transfer.

When a DMA operation is enabled, IOCS16-, CS1FX- and CS3FX- shall not be asserted and transfers shall be 16-bits wide.

NOTE: ATA products with DMA capability require a pull-down resistor on this signal to prevent spurious data transfers. This resistor may affect driver requirements for drives sharing this signal in systems with unbuffered ATA signals.

6.3.10 INTRQ (Drive Interrupt)

This signal is used to interrupt the host system. INTRQ is asserted only when the drive has a pending interrupt, the drive is selected, and the host has cleared nIEN in the Device Control Register. If nIEN=1, or the drive is not selected, this output is in a high impedance state, regardless of the presence or absence of a pending interrupt.

INTRQ shall be negated by:

- assertion of RESET- or
- the setting of SRST of the Device Control Register, or
- the host writing the Command Register or
- the host reading the Status Register

NOTE: Some drives may negate INTRQ on a PIO data transfer completion, except on a single sector read or on the last sector of a multi-sector read.

On PIO transfers, INTRQ is asserted at the beginning of each data block to be transferred. A data block is typically a single sector, except when declared otherwise by use of the Set Multiple command. An exception occurs on Format Track, Write Sector(s), Write Buffer and Write Long commands - INTRQ shall not be asserted at the beginning of the first data block to be transferred.

On DMA transfers, INTRQ is asserted only once, after the command has completed.

6.3.11 IOCS16- (Drive 16-bit I/O)

Except for DMA transfers, IOCS16- indicates to the host system that the 16-bit data port has been addressed and that the drive is prepared to send or receive a 16-bit data word. This shall be an open collector output.

- When transferring in PIO mode, If IOCS16- is not asserted, transfers shall be 8-bit using DD0-7.
- When transferring in PIO mode, if IOCS16- is asserted, transfers shall be 16-bit using DD0-15. for 16-bit data transfers.
- When transferring in DMA mode, the host shall use a 16-bit DMA channel and IOCS16- shall not be asserted.

6.3.12 IORDY (I/O Channel Ready) (Optional)

This signal is negated to extend the host transfer cycle of any host register access (Read or Write) when the drive is not ready to respond to a data transfer request. When IORDY is not negated, IORDY shall be in a high impedance state.

6.3.13 PDIAG- (Passed Diagnostics)

This signal shall be asserted by Drive 1 to indicate to Drive 0 that it has completed diagnostics. A 10K pull-up resistor shall be used on this signal by each drive.

Following a power on reset, software reset or RESET-, Drive 1 shall negate PDIAG- within 1 msec (to indicate to Drive 0 that it is busy). Drive 1 shall then assert PDIAG- within 30 seconds to indicate that it is no longer busy, and is able to provide status. After the assertion of PDIAG-, Drive 1 may be unable to accept commands until it has finished its reset procedure and is Ready (DRDY=1).

Following the receipt of a valid Execute Drive Diagnostics command, Drive 1 shall negate PDIAG- within 1 msec to indicate to Drive 0 that it is busy and has not yet passed its drive diagnostics. If Drive 1 is present then Drive 0 shall wait for up to 5 seconds from the receipt of a valid Execute Drive Diagnostics command for Drive 1 to assert PDIAG-. Drive 1 should clear BSY before asserting PDIAG-, as PDIAG- is used to indicate that Drive 1 has passed its diagnostics and is ready to post status.

If DASP- was not asserted by Drive 1 during reset initialization, Drive 0 shall post its own status immediately after it completes diagnostics, and clear the Drive 1 Status Register to 00h. Drive 0 may be unable to accept commands until it has finished its reset procedure and is Ready (DRDY=1).

6.3.14 RESET- (Drive Reset)

This signal from the host system shall be asserted for at least 25 usec after voltage levels have stabilized during power on and negated thereafter unless some event requires that the drive(s) be reset following power on.

6.3.15 SPSYNC (Spindle Synchronization) (Optional)

This signal may be either input or output to the drive depending on a vendor- defined switch. If a drive is set to Master the signal is output, and if a drive is set to slave the signal is input.

There is no requirement that each drive implementation be plug-compatible to the extent that a multiple vendor drive subsystem be operable. Mix and match of different manufacturers drives is unlikely because rpm, sync fields, sync bytes etc need to be virtually identical. However, if drives are designed to match the following recommendation, controllers can operate drives with a single implementation.

There can only be one master drive at a time in a configuration. The host or the drive designated as master can generate SPSYNC at least once per rotation, but may be at a higher frequency.

SPSYNC received by a drive is used as the synchronization signal to lock the spindles in step. The time to achieve synchronization varies, and is indicated by the drive setting DRDY i.e. if the drive does not achieve synchronization following power on or a reset, it shall not set DRDY.

A master drive or the host generates SPSYNC and transmits it.

A slave drive does not generate SPSYNC and is responsible to synchronize its index to SPSYNC.

If a drive does not support synchronization, it shall ignore SPSYNC.

In the event that a drive previously synchronized loses synchronization, but is otherwise operational, it does not clear DRDY.

Prior to the introduction of this standard, this signal was defined as DALE (Drive Address Latch Enable), and used for an address valid indication from the host system. If used, the host address and chip selects, DAO through DA2, CS1FX-, and CS3FX- were valid at the negation of this signal and remained valid while DALE was negated, therefore, the drive did not need to latch these signals with DALE.

7. Logical Interface

(Part of **ANSI ATA rev 2.3**)

7. Logical Interface

7.1 General

7.1.1 Bit Conventions

Bit names are shown in all upper case letters except where a lower case n precedes a bit name. This indicates that when nBIT=0 (bit is zero) the action is true and when nBIT=1 (bit is one) the action is false. If there is no preceding n, then when BIT=1 it is true, and when BIT=0 it is false.

A bit can be set to one or cleared to zero and polarity influences whether it is to be interpreted as true or false:

True	BIT=1	nBIT=0
False	BIT=0	nBIT=1

7.1.2 Environment

The drives using this interface shall be programmed by the host computer to perform commands and return status to the host at command completion. When two drives are daisy chained on the interface, commands are written in parallel to both drives, and for all except the Execute Diagnostics command, only the selected drive executes the command. On an Execute Diagnostics command addressed to Drive 0, both drives shall execute the command, and Drive 1 shall post its status to Drive 0 via PDIAG-.

Drives are selected by the DRV bit in the Drive/Head Register (see 7.2.8), and by a jumper or switch on the drive designating it as either a Drive 0 or as Drive 1. When DRV=0, Drive 0 is selected. When DRV=1, Drive 1 is selected. When drives are daisy chained, one shall be set as Drive 0 and the other as Drive 1. When a single drive is attached to the interface it shall be set as Drive 0.

Prior to the adoption of this standard, some drives may have provided jumpers to indicate Drive 0 with no Drive 1 present, or Drive 0 with Drive 1 present.

Throughout this document, drive selection always refers to the state of the DRV bit, and the position of the Drive 0/Drive 1 jumper or switch.

7.2 I/O Register Descriptions

Communication to or from the drive is through an I/O Register that routes the input or output data to or from registers (selected) by a code on signals from the host (CS1FX-, CS3FX-, DA2, DA1, DA0, DIOR- and DIOW-).

The Command Block Registers are used for sending commands to the drive or posting status from the drive.

The Control Block Registers are used for drive control and to post alternate status.

Table 7-1 lists these registers and the addresses that select them.

```
Logic conventions are:   A = signal asserted
                         N = signal negated
```

x = does not matter which it is

TABLE 7-1: I/O PORT FUNCTIONS/SELECTION ADDRESSES

Addresses					Functions	
CS1FX-	CS3FX-	DA2	DA1	DA0	READ (DIOR-)	WRITE (DIOW-)
					Control Block Registers	
N	N	x	x	x	Data Bus High Imped	Not used
N	A	0	x	X	Data Bus High Imped	Not used
N	A	1	0	x	Data Bus High Imped	Not used
N	A	1	1	0	Alternate Status	Device Control
N	A	1	1	1	Drive Address	Not used
					Command Block Registers	
A	N	0	0	0	Data	Data
A	N	0	0	1	Error Register	Features
A	N	0	1	0	Sector Count	Sector Count
A	N	0	1	1	Sector Number	Sector Number
A	N	1	0	0	Cylinder Low	Cylinder Low
A	N	1	0	1	Cylinder High	Cylinder High
A	N	1	1	0	Drive/Head	Drive/Head
A	N	1	1	1	Status	Command
A	A	x	x	x	Invalid Address	Invalid Address

7.2.1 Alternate Status Register

This register contains the same information as the Status Register in the command block. The only difference being that reading this register does not imply interrupt acknowledge or clear a pending interrupt.

7	6	5	4	3	2	1	0
BSY	DRDY	DWF	DSC	DRQ	CORR	IDX	ERR

See 7.2.13 for definitions of the bits in this register.

7.2.2 Command Register

This register contains the command code being sent to the drive. Command execution begins immediately after this register is written. The executable commands, the command codes, and the necessary parameters for each command are listed in Table 9-1.

7.2.3 Cylinder High Register

This register contains the high order bits of the starting cylinder address for any disk access. At the end of the command, this register is updated to reflect the current cylinder number. The most significant bits of the cylinder address shall be loaded into the cylinder high Register.

NOTE: Prior to the introduction of this standard, only the lower 2 bits of this register were valid, limiting cylinder address to 10 bits i.e.

1,024 cylinders.

7.2.4 Cylinder Low Register

This register contains the low order 8 bits of the starting cylinder address for any disk access. At the end of the command, this register is updated to reflect the current cylinder number.

7.2.5 Data Register

This 16-bit register is used to transfer data blocks between the device data buffer and the host. It is also the register through which sector information is transferred on a Format Track command. Data transfers may be either PIO or DMA.

7.2.6 Device Control Register

The bits in this register are as follows:

7	6	5	4	3	2	1	0
x	x	x	x	1	SRST	nIEN	0

- SRST is the host software reset bit. The drive is held reset when this bit is set. If two disk drives are daisy chained on the interface, this bit resets both simultaneously. Drive 1 is not required to execute the DASP-handshake procedure.
- nIEN is the enable bit for the drive interrupt to the host. When nIEN=0, and the drive is selected, INTRQ shall be enabled through a tri-state buffer. When nIEN=1, or the drive is not selected, the INTRQ signal shall be in a high impedance state.

7.2.7 Drive Address Register

This register contains the inverted drive select and head select addresses of the currently selected drive. The bits in this register are as follows:

7	6	5	4	3	2	1	0
HiZ	nWTG	nHS3	nHS2	nHS1	nHS0	nDS1	nDS0

- HiZ shall always be in a high impedance state.
- nWTG is the Write Gate bit. When writing to the disk drive is in progress, nWTG=0.
- nHS3 through nHS0 are the one's complement of the binary coded address of the currently selected head. For example, if nHS3 through nHS0 are 1100b, respectively, head 3 is selected. nHS3 is the most significant bit.
- nDS1 is the drive select bit for drive 1. When drive 1 is selected and active, nDS1=0.
- nDS0 is the drive select bit for drive 0. When drive 0 is selected and active, nDS0=0.

NOTE: Care should be used when interpreting these bits, as they do not always represent the expected status of drive operations at the instant the status was put into this register. This is because of the

use of cacheing, translate mode and the Drive 0/Drive 1 concept with each drive having its own embedded controller.

7.2.8 Drive/Head Register

This register contains the drive and head numbers. The contents of this register define the number of heads minus 1, when executing an Initialize Drive Parameters command.

7	6	5	4	3	2	1	0
1	0	1	DRV	HS3	HS2	HS1	HS0

- DRV is the binary encoded drive select number. When DRV=0, Drive 0 is selected. When DRV=1, Drive 1 is selected.
- HS3 through HS0 contain the binary coded address of the head to be selected e.g. if HS3 through HS0 are 0011b, respectively, head 3 will be selected. HS3 is the most significant bit. At command completion, this register is updated to reflect the currently selected head.

7.2.9 Error Register

This register contains status from the last command executed by the drive or a Diagnostic Code.

At the completion of any command except Execute Drive Diagnostic, the contents of this register are valid when ERR=1 in the Status Register.

Following a power on, a reset, or completion of an Execute Drive Diagnostic command, this register contains a Diagnostic Code (see Table 9-2).

7	6	5	4	3	2	1	0
BBK	UNC	0	IDNF	0	ABRT	TK0NF	AMNF

- BBK (Bad Block Detected) indicates a bad block mark was detected in the requested sector's ID field.
- UNC (Uncorrectable Data Error) indicates an uncorrectable data error has been encountered.
- IDNF (ID Not Found) indicates the requested sector's ID field could not be found.
- ABRT (Aborted Command) indicates the requested command has been aborted due to a drive status error (Not Ready, Write Fault, etc.) or because the command code is invalid.
- TK0NF (Track 0 Not Found) indicates track 0 has not been found during a Recalibrate command.
- AMNF (Address Mark Not Found) indicates the data address mark has not been found after finding the correct ID field.
- Unused bits are cleared to zero.

7.2.10 Features Register

This register is command specific and may be used to enable and disable

features of the interface e.g. by the Set Features Command to enable and disable cacheing.

This register may be ignored by some drives.

Some hosts, based on definitions prior to the completion of this standard, set values in this register to designate a recommended Write Precompensation Cylinder value.

7.2.11 Sector Count Register

This register contains the number of sectors of data requested to be transferred on a read or write operation between the host and the drive. If the value in this register is zero, a count of 256 sectors is specified.

If this register is zero at command completion, the command was successful. If not successfully completed, the register contains the number of sectors which need to be transferred in order to complete the request.

The contents of this register may be defined otherwise on some commands e.g. Initialize Drive Parameters, Format Track or Write Same commands.

7.2.12 Sector Number Register

This register contains the starting sector number for any disk data access for the subsequent command. The sector number may be from 1 to the maximum number of sectors per track.

See the command descriptions for contents of the register at command completion (whether successful or unsuccessful).

7.2.13 Status Register

This register contains the drive status. The contents of this register are updated at the completion of each command. When BSY is cleared, the other bits in this register shall be valid within 400 nsec. If BSY=1, no other bits in this register are valid. If the host reads this register when an interrupt is pending, it is considered to be the interrupt acknowledge. Any pending interrupt is cleared whenever this register is read.

```
NOTE: If Drive 1 is not detected as being present, Drive 0 clears the Drive
      1 Status Register to 00h (indicating that the drive is Not Ready).
```

7	6	5	4	3	2	1	0
BSY	DRDY	DWF	DSC	DRQ	CORR	IDX	ERR

```
NOTE: Prior to the definition of this standard, DRDY and DSC were unlatched
      real time signals.
```

- BSY (Busy) is set whenever the drive has access to the Command Block Registers. The host should not access the Command Block Register when BSY=1. When BSY=1, a read of any Command Block Register shall return the contents of the Status Register. This bit is set by the drive (which may be able to respond at times when the media cannot be accessed) under the following circumstances:
  a) within 400 nsec after the negation of RESET- or after SRST has been set in the Device Control Register. Following acceptance of a reset it is recommended that BSY be set for no longer than 30 seconds by Drive 1 and

no longer than 31 seconds by Drive 0.

b) within 400 nsec of a host write of the Command Register with a Read, Read Long, Read Buffer, Seek, Recalibrate, Initialize Drive Parameters, Read Verify, Identify Drive, or Execute Drive Diagnostic command.

c) within 5 usecs following transfer of 512 bytes of data during execution of a Write, Format Track, or Write Buffer command, or 512 bytes of data and the appropriate number of ECC bytes during the execution of a Write Long command.

- DRDY (Drive Ready) indicates that the drive is capable of responding to a command. When there is an error, this bit is not changed until the Status Register is read by the host, at which time the bit again indicates the current readiness of the drive. This bit shall be cleared at power on and remain cleared until the drive is ready to accept a command.
- DWF (Drive Write Fault) indicates the current write fault status. When an error occurs, this bit shall not be changed until the Status Register is read by the host, at which time the bit again indicates the current write fault status.
- DSC (Drive Seek Complete) indicates that the drive heads are settled over a track. When an error occurs, this bit shall not be changed until the Status Register is read by the host, at which time the bit again indicates the current Seek Complete status.
- DRQ (Data Request) indicates that the drive is ready to transfer a word or byte of data between the host and the drive.
- CORR (Corrected Data) indicates that a correctable data error was encountered and the data has been corrected. This condition does not terminate a data transfer.
- IDX (Index) is set once per disk revolution.
- ERR (Error) indicates that an error occurred during execution of the previous command. The bits in the Error Register have additional information regarding the cause of the error.

8. Programming Requirements

(Part of **ANSI ATA rev 2.3**)

8. Programming Requirements

8.1 Reset Response

A reset is accepted within 400 nsec after the negation of RESET- or within 400 nsec after SRST has been set in the Device Control Register.

When the drive is reset by RESET-, Drive 1 shall indicate it is present by asserting DASP- within 400 msec, and DASP- shall remain asserted for 30 seconds or until Drive 1 accepts the first command. See also **6.3.4** and **6.3.13**.

When the drive is reset by SRST, the drive shall set BSY=1.

See also **7.2.6.**

When a reset is accepted, and with BSY set:

a) Both drives perform any necessary hardware initialization
b) Both drives clear any previously programmed drive parameters
c) Both drives may revert to the default condition
d) Both drives load the Command Block Registers with their default values
e) If a hardware reset, Drive 0 waits for DASP- to be asserted by Drive 1
f) If operational, Drive 1 asserts DASP-
g) Drive 0 waits for PDIAG- to be asserted if Drive 1 asserts DASP-
h) If operational, Drive 1 clears BSY
i) If operational, Drive 1 asserts PDIAG-
j) Drive 0 clears BSY

No interrupt is generated when initialization is complete.

The default values for the Command Block Registers if no self-tests are performed or if no errors occurred are:

```
Error         = 01h       Cylinder Low  = 00h
Sector Count  = 01h       Cylinder High = 00h
Sector Number = 01h       Drive/Head    = 00h
```

The Error Register shall contain a Diagnostic Code (see Table 9.2) if a self-test is performed.

Following any reset, the host should issue an Initialize Drive Parameters command to ensure the drive is initialized as desired.

There are three types of reset in ATA. The following is a suggested method of classifying reset actions:

- Power On Reset: the drive executes a series of electrical circuitry diagnostics, spins up the HDA, tests speed and other mechanical parametrics, and sets default values.
- Hardware Reset: the drive executes a series of electrical circuitry diagnostics, and resets to default values.
- Software Reset: the drive resets the interface circuitry to default values.

8.2 Translate Mode

The cylinder, head and sector geometry of the drive as presented to the host may differ from the actual physical geometry. Translate mode is an optional and device specific means of mapping between the two.

8.3 Power Conditions

Optional power commands permit the host to modify the behavior of the drive in a manner which reduces the power required to operate.

TABLE 8-1: POWER CONDITIONS

Mode	SRST	BSY	DRDY	Interface Active	Media
Sleep	1	x	x	No	0
Standby	x	0	1	Yes	0
Idle	x	0	1	Yes	1
Active	x	x	x	Yes	1

1 = Active 0 = Inactive

The lowest power consumption occurs in Sleep mode. When in Sleep mode, the drive needs a Software Reset to be activated (see 9.18). The time to respond could be as long as 30 seconds or more.

In Standby mode the drive interface is capable of accepting commands, but as the media is not immediately accessible, it could take the drive as long as 30 seconds or more to respond.

In Idle mode the drive is capable of responding immediately to media access requests. A drive in Idle mode may take longer to complete the execution of a command because it may have to activate some circuitry.

In Active mode the drive is capable of responding immediately to media access requests, and commands complete execution in the shortest possible time.

Ready is not a power condition. A drive may post ready at the interface even though the media may not be accessible.

See specific power-related commands.

8.4 Error Posting

The errors that are valid for each command are defined in Table 8-1. It is not a requirement that all valid conditions be implemented. See **7.2.9** and **7.2.13**for the definition of the Error Register and Status Register bits.

TABLE 8-2: REGISTER CONTENTS

Error Register						Status Register				
BBK	UNC	IDNF	ABRT	TK0NF	AMNF	DRDY	DWF	DSC	CORR	ERR

Check Power Mode				V			V	V	V		V		
Execute Drive Diags			See **9.2**									V	
Format Track			V	V			V	V	V		V		
Identify Drive				V			V	V	V		V		
Idle				V			V	V	V		V		
Idle Immediate				V			V	V	V		V		
Initialize Drive Parms							V	V	V				
Recalibrate				V	V		V	V	V		V		
Read Buffer				V			V	V	V		V		
Read DMA	V	V	V	V		V	V	V	V	V	V		
Read Long	V	V	V	V		V	V	V	V	V	V		
Read Multiple	V	V	V	V		V	V	V	V	V	V		
Read Sector(s)	V	V	V	V		V	V	V	V	V	V		
Read Verify Sector(s)	V	V	V	V		V	V	V	V	V	V		
Seek			V	V			V	V	V		V		
Set Features				V			V	V	V		V		
Set Multiple Mode				V			V	V	V		V		
Sleep				V			V	V	V		V		
Standby				V			V	V	V		V		
Standby Immediate				V			V	V	V		V		
Write Buffer				V			V	V	V		V		
Write DMA	V		V	V			V	V	V		V		
Write Long	V		V	V			V	V	V		V		
Write Multiple	V		V	V			V	V	V		V		
Write Same	V		V	V			V	V	V		V		
Write Sector(s)	V		V	V			V	V	V		V		
Write Verify	V	V	V	V		V	V	V	V	V	V		
Invalid Command Code				V			V	V	V		V		

V = valid on this command

9. Command Descriptions

(Part of **ANSI ATA rev 2.3**)

9. Command Descriptions

Commands are issued to the drive by loading the pertinent registers in the command block with the needed parameters, and then writing the command code to the Command Register.

The manner in which a command is accepted varies. There are three classes (see Table 9-1) of command acceptance, all predicated on the fact that to receive a command, BSY=0:

- Upon receipt of a Class 1 command, the drive sets BSY within 400 nsec.
- Upon receipt of a Class 2 command, the drive sets BSY within 400 nsec, sets up the sector buffer for a write operation, sets DRQ within 700 usec, and clears BSY within 400 nsec of setting DRQ.
- Upon receipt of a Class 3 command, the drive sets BSY within 400 nsec, sets up the sector buffer for a write operation, sets DRQ within 20 msec, and clears BSY within 400 nsec of setting DRQ.

NOTE: DRQ may be set so quickly on Class 2 and Class 3 that the BSY transition is too short for BSY=1 to be recognized.

The drive shall implement all mandatory commands as identified by an M, and may implement the optional commands identified by an O, in Table 9-1. V indicates a Vendor Specific command code.

If a new command is issued to a drive which has an uncompleted command (subsequently referred to as Old_Command) in progress, the drive shall immediately respond to the new command (Subsequently referred to as New_Command), even if execution of the Old_Command could have been completed.

There shall be no indication given to the system as to the status of the Old_Command which was being executed at the time the New_Command was issued.

TABLE 9-1: COMMAND CODES AND PARAMETERS

Class			Command Code	Parameters Used FR	SC	SN	CY	DH
1	Check Power Mode	O	98h E5h		y			D
1	Execute Drive Diagnostic	M	90h					D*
2	Format Track	M	50h	*	y		y	y
1	Identify Drive	O	ECh					D
1	Idle	O	97h E3h		y			D
1	Idle Immediate	O	95h E1h					D
1	Initialize Drive Parameters	M	91h		y			y
1	Recalibrate	M	1xh					D
1	Read Buffer	O	E4h					D
1	Read DMA (w/retry)	O	C8h		y	y	y	y
1	Read DMA (w/o retry)	O	C9h		y	y	y	y
1	Read Multiple	O	C4h		y	y	y	y
1	Read Sector(s) (w/retry)	M	20		y	y	y	y
1	Read Sector(s) (w/o retry)	M	21		y	y	y	y
1	Read Long (w/retry) See **9.13**	M	22		y	y	y	y
1	Read Long (w/o retry) See **9.13**	M	23		y	y	y	y
1	Read Verify Sector(s) (w/retry)	M	40		y	y	y	y

1	Read Verify Sector(s) (w/o retry)	M	41		y	y	y	y
1	Seek	M	7xh			y	y	y
1	Set Features	O	EFh	y				D
1	Set Multiple Mode	O	C6h		y			D
1	Set Sleep Mode	O	99h E6h					D
1	Standby	O	96h E2h		y			D
1	Standby Immediate	O	94h E0h					D
2	Write Buffer	O	E8h					D
3	Write DMA (w/retry)	O	CAh		y	y	y	y
3	Write DMA (w/o retry)	O	CBh		y	y	y	y
3	Write Multiple	O	C5h	*	y	y	y	y
3	Write Same	O	E9h	y	y	y	y	y
2	Write Sector(s) (w/retry)	M	30	*	y	y	y	y
2	Write Sector(s) (w/o retry)	M	31	*	y	y	y	y
2	Write Long (w/retry)	M	32	*	y	y	y	y
2	Write Long (w/o retry)	M	33	*	y	y	y	y
3	Write Verify	O	3Ch	*	y	y	y	y
	Vendor Unique	V	9Ah					
	Vendor Unique	V	C0-C3h					
	Vendor Unique	V	8xh					
	Vendor Unique	V	F0h-FFh					
	Reserved: All remaining codes							

CY = Cylinder Registers	SC = Sector Count Register
DH = Drive/Head Register	SN = Sector Number Register
FR = Features Register (see command descriptions for use)	
y - the register contains a valid parameter for this command. For the Drive/Head Register, y means both the drive and head parameters are used.	
D - only the drive parameter is valid and not the head parameter.	
D* - Addressed to Drive 0 but both drives execute it.	
* - Maintained for compatibility (**see 7.2.9**)	

9.1 Check Power Mode

This command checks the power mode.

If the drive is in, going to, or recovering from the Standby Mode the drive shall set BSY, set the Sector Count Register to 00h, clear BSY, and generate an interrupt.

If the drive is in the Idle Mode, the drive shall set BSY, set the Sector Count Register to FFh, clear BSY, and generate an interrupt.

9.2 Execute Drive Diagnostic

This command shall perform the internal diagnostic tests implemented by the drive. See also **6.3.4** and **6.3.13**. The DRV bit is ignored. Both drives, if present, shall execute this command.

If Drive 1 is present:

- Drive 0 waits up to 5 seconds for Drive 1 to assert PDIAG-.
- If Drive 1 has not asserted PDIAG-, indicating a failure, Drive 0 shall append 80h to its own diagnostic status.
- Both drives shall execute diagnostics.

- If Drive 1 diagnostic failure is detected when Drive 0 status is read, Drive 1 status is obtained by setting the DRV bit, and reading status.

If there is no Drive 1 present:

- Drive 0 posts only its own diagnostic results.
- Drive 0 clears BSY, and generates an interrupt.

The Diagnostic Code written to the Error Register is a unique 8-bit code as shown in Table 9-2, and not as the single bit flags defined in **7.2.9**.

If Drive 1 fails diagnostics, Drive 0 "ORs" 80h with its own status and loads that code into the Error Register. If Drive 1 passes diagnostics or there is no Drive 1 connected, Drive 0 "ORs" 00h with its own status and loads that code into the Error Register.

TABLE 9-2: DIAGNOSTIC CODES

Code	
01h	No error detected
02h	Formatter device error
03h	Sector buffer error
04h	ECC circuitry error
05h	Controlling microprocessor error
8xh	Drive 1 failed

9.3 Format Track

The implementation of the Format Track command is vendor specific. The actions may be a physical reformatting of a track, initializing the data field contents to some value, or doing nothing.

The Sector Count Register contains the number of sectors per track.

The track address is specified in the Cylinder High and Cylinder Low Registers, and the number of sectors is specified in the Sector Count Register. When the command is accepted, the drive sets the DRQ bit and waits for the host to fill the sector buffer. When the sector buffer is full, the drive clears DRQ, sets BSY and begins command execution.

The contents of the sector buffer shall not be written to the media, and may be either ignored or interpreted as follows:

DD15 ---- DD0			DD15 ---- DD0		
First Sector	Descriptor	: : : : : :	Last Sector	Descriptor	Remainder of buffer filled with zeros

One 16-bit word represents each sector, the words being contiguous from the start of a sector. Any words remaining in the buffer after the representation of the last sector are filled with zeros. DD15-8 contain the sector number. If an interleave is specified, the words appear in the same sequence as they appear on the track. DD7-0 contain a descriptor value defined as follows:

00h - Format sector as good

20h - Unassign the alternate location for this sector
40h - Assign this sector to an alternate location
80h - Format sector as bad

NOTE: Some users of the ATA drive expect the operating system partition table to be erased on a Format command. It is recommended that a drive which does not perform a physical format of the track, write a data pattern of all zeros to the sectors which have been specified by the Format Track command.

NOTE: It is recommended that implementors resassign data blocks which show repeated errors.

9.4 Identify Drive

The Identify Drive command enables the host to receive parameter information from the drive. When the command is issued, the drive sets BSY, stores the required parameter information in the sector buffer, sets DRQ, and generates an interrupt. The host then reads the information out of the sector buffer. The parameter words in the buffer have the arrangement and meanings defined in Table 9-3. All reserved bits or words shall be zero.

TABLE 9-3: IDENTIFY DRIVE INFORMATION

Word	
0	General configuration bit-significant information:
	15 0 reserved for non-magnetic drives
	14 1=format speed tolerance gap required
	13 1=track offset option available
	12 1=data strobe offset option available
	11 1=rotational speed tolerance is > 0.5%
	10 1=disk transfer rate > 10 Mbs
	9 1=disk transfer rate > 5Mbs but <= 10Mbs
	8 1=disk transfer rate <= 5Mbs
	7 0 reserved for removable cartridge drive
	6 1=fixed drive
	5 1=spindle motor control option implemented
	4 1=head switch time > 15 usec
	3 1=not MFM encoded
	2 1=soft sectored
	1 1=hard sectored
	0 0=reserved
1	Number of fixed cylinders
2	reserved
3	Number of heads
4	Number of unformatted bytes per track
5	Number of unformatted bytes per sector
6	Number of sectors per track
7-9	Vendor Unique
10-19	Serial number (20 ASCII characters, 0000h=not specified)
20	Buffer type
21	Buffer size in 512 byte increments (0000h=not specified)
22	# of ECC bytes passed on Read/Write Long cmds (0000h=not spec'd)
23-26	Firmware revision (8 ASCII characters, 0000h=not specified)
27-46	Model number (40 ASCII characters, 0000h=not specified)
47	15-8 Vendor Unique
	7-0 00h = Read/Write Multiple commands not implemented

```
|       |       xxh = Maximum number of sectors that can be transferred    |
|       |             per interrupt on Read and Write Multiple commands    |
|   48  | 0000h = cannot perform doubleword I/O                            |
|       | 0001h = can perform doubleword I/O                               |
|   49  | Capabilities                                                     |
|       | 15-9  0=reserved                                                 |
|       |    8  1=DMA Supported                                           |
|       |  7-0  Vendor Unique                                              |
|   50  |   reserved                                                       |
|   51  | 15-8  PIO data transfer cycle timing mode                        |
|       |  7-0  Vendor Unique                                              |
|   52  | 15-8  DMA data transfer cycle timing mode                        |
|       |  7-0  Vendor Unique                                              |
| 53-127|   reserved                                                       |
|128-159| Vendor Unique                                                    |
|160-255|   reserved                                                       |
+-------+------------------------------------------------------------------+
```

The fields described in 9.4.1 through 9.4.5 are not affected by the Initialize Drive Parameters command.

9.4.1 Number of fixed cylinders

The number of translated cylinders in the default translation mode.

9.4.2 Number of heads

The number of translated heads in the default translation mode.

9.4.3 Number of unformatted bytes per track

The number of unformatted bytes per translated track in the default translation mode.

9.4.4 Number of unformatted bytes per sector

The number of unformatted bytes per sector in the default translation mode.

9.4.5 Number of sectors per track

The number of sectors per track in the default translation mode.

9.4.6 Serial Number

The contents of this field are right justified and padded with spaces (20h).

9.4.7 Buffer Type

The contents of the field are determined by the manufacturer.

0000h = not specified.
0001h = a single ported single sector buffer which is not capable of simultaneous data transfers to or from the host and the disk.
0002h = a dual ported multi-sector buffer capable of simultaneous data transfers to or from the host and the disk.
0003h = a dual ported multi-sector buffer capable of simultaneous transfers with a read cacheing capability.

0004-FFFFh = reserved

These codes are typically not used by the operating system, however, they are useful for diagnostic programs which perform initialization routines e.g. a different interleave may be desirable for 0001h vs 0002h or 0003h.

9.4.8 Firmware Revision

The contents of this field are left justified and padded with spaces (20h).

9.4.9 Model Number

The contents of this field are left justified and padded with spaces (20h).

9.4.10 PIO data transfer cycle timing mode

The PIO transfer timing for each ATA device falls into categories which have unique parametric timing specifications. To determine the proper device timing category, compare the Cycle Time specified in Figure 11-1 with the contents of this field. The value returned in Bits 15-8 should fall into one of the categories specified in Figure 11-1, and if it does not, then Mode 0 shall be used to serve as the default timing.

9.4.11 DMA data transfer cycle timing mode

The DMA transfer timing for each ATA device falls into categories which have unique parametric timing specifications. To determine the proper device timing category, compare the Cycle Time specified in Figure 11-3 with the contents of this field. The value returned in Bits 15-8 should fall into one of the categories specified in Figure 11-3, and if it does not, then Mode 0 shall be used to serve as the default timing.

9.5 Idle

This command causes the drive to set BSY, enter the Idle Mode, clear BSY, and generate an interrupt. The interrupt is generated even though the drive may not have fully transitioned to Idle Mode.

If the drive is already spinning, the spinup sequence is not executed.

If the Sector Count Register is non-zero then the automatic power down sequence shall be enabled and the timer begins counting down immediately. If the Sector Count Register is zero then the automatic power down sequence shall be disabled.

9.6 Idle Immediate

This command causes the drive to set BSY, enter the Idle Mode, clear BSY, and generate an interrupt. The interrupt is generated even though the drive may not have fully transitioned to Idle Mode.

9.7 Initialize Drive Parameters

This command enables the host to set the number of sectors per track and the number of heads minus 1, per cylinder. Upon receipt of the command, the drive sets BSY, saves the parameters, clears BSY, and generates an interrupt.

The only two register values used by this command are the Sector Count Register which

specifies the number of sectors per track, and the Drive/Head Register which specifies the number of heads minus 1. The DRV bit designates these values to Drive 0 or Drive 1, as appropriate.

The sector count and head values are not checked for validity by this command. If they are invalid, no error will be posted until an illegal access is made by some other command.

9.8 Recalibrate

This command moves the read/write heads from anywhere on the disk to cylinder 0. Upon receipt of the command, the drive sets BSY and issues a seek to cylinder zero. The drive then waits for the seek to complete before updating status, clearing BSY and generating an interrupt.

If the drive cannot reach cylinder 0, a Track Not Found error is posted.

9.9 Read Buffer

The Read Buffer command enables the host to read the current contents of the drive's sector buffer. When this command is issued, the drive sets BSY, sets up the sector buffer for a read operation, sets DRQ, clears BSY, and generates an interrupt. The host then reads up to 512 bytes of data from the buffer.

The Read Buffer and Write Buffer commands shall be synchronized such that sequential Write Buffer and Read Buffer commands access the same 512 bytes within the buffer.

9.10 Read DMA

This command executes in a similar manner to the Read Sectors command except for the following:

- the host initializes a slave-DMA channel prior to issuing the command
- data transfers are qualified by DMARQ and are performed by the slave-DMA channel
- the drive issues only one interrupt per command to indicate that data transfer has terminated and status is available.

Any unrecoverable error encountered during execution of a Read DMA command results in the termination of data transfer at the sector where the error was detected. The sector in error is not transferred. The drive generates an interrupt to indicate that data transfer has terminated and status is available. The error posting is the same as that of the Read Sectors command.

9.11 Read Long

The Read Long command performs similarly to the Read Sectors command except that it returns the data and the ECC bytes contained in the data field of the desired sector. During a Read Long command, the drive does not check the ECC bytes to determine if there has been a data error. Only single sector read long operations are supported.

The transfer of the ECC bytes shall be 8-bits wide.

9.12 Read Multiple Command

The Read Multiple command performs similarly to the Read Sectors command. Interrupts are

not generated on every sector, but on the transfer of a block which contains the number of sectors defined by a Set Multiple command.

Command execution is identical to the Read Sectors operation except that the number of sectors defined by a Set Multiple command are transferred without intervening interrupts. DRQ qualification of the transfer is required only at the start of the data block, not on each sector.

The block count of sectors to be transferred without intervening interrupts is programmed by the Set Multiple Mode command, which shall be executed prior to the Read Multiple command.

When the Read Multiple command is issued, the Sector Count Register contains the number of sectors (not the number of blocks or the block count) requested.

If the number of requested sectors is not evenly divisible by the block count, as many full blocks as possible are transferred, followed by a final, partial block transfer. The partial block transfer shall be for n sectors, where

n = Remainder (Sector Count / Block Count)

If the Read Multiple command is attempted before the Set Multiple Mode command has been executed or when Read Multiple commands are disabled, the Read Multiple operation shall be rejected with an Aborted Command error.

Disk errors encountered during Read Multiple commands are posted at the beginning of the block or partial block transfer, but DRQ is still set and the data transfer shall take place as it normally would, including transfer of corrupted data, if any.

The contents of the Command Block Registers following the transfer of a data block which had a sector in error are undefined. The host should retry the transfer as individual requests to obtain valid error information.

Subsequent blocks or partial blocks are transferred only if the error was a correctable data error. All other errors cause the command to stop after transfer of the block which contained the error. Interrupts are generated when DRQ is set at the beginning of each block or partial block.

9.13 Read Sector(s)

This command reads from 1 to 256 sectors as specified in the Sector Count register. A sector count of 0 requests 256 sectors. The transfer begins at the sector specified in the Sector Number Register. See **10.1** for the DRQ, IRQ and BSY protocol on data transfers.

If the drive is not already on the desired track, an implied seek is performed. Once at the desired track, the drive searches for the appropriate ID field.

If retries are disabled and two index pulses have occurred without error free reading of the requested ID, an ID Not Found error is posted.

If retries are enabled, up to a vendor specific number of attempts may be made to read the requested ID before posting an error.

If the ID is read correctly, the data address mark shall be recognized within a specified

number of bytes, or the Address Mark Not Found error is posted.

DRQ is always set prior to data transfer regardless of the presence or absence of an error condition.

At command completion, the Command Block Registers contain the cylinder, head, and sector number of the last sector read.

If an error occurs, the read terminates at the sector where the error occurred. The Command Block Registers contain the cylinder, head, and sector number of the sector where the error occurred.

The flawed data is pending in the sector buffer.

9.14 Read Verify Sector(s)

This command is identical to the Read Sectors command, except that DRQ is never set, and no data is transferred to the host. See **10.3** for protocol.

When the requested sectors have been verified, the drive clears BSY and generates an interrupt. Upon command completion, the Command Block Registers contain the cylinder, head, and sector number of the last sector verified.

If an error occurs, the verify terminates at the sector where the error occurs. The Command Block Registers contain the cylinder, head, and sector number of the sector where the error occurred. The Sector Count Register shall contain the number of sectors not yet verified.

9.15 Seek

This command initiates a seek to the track and selects the head specified in the command block. The drive need not be formatted for a seek to execute properly. See **10.3** for protocol. The drive shall not set DSC=1 until the action of seeking has completed. The drive may return the interrupt before the seek is completed.

If another command is issued to the drive while a seek is being executed, the drive sets BSY=1, waits for the seek to complete, and then begins execution of the command.

9.16 Set Features

This command is used by the host to establish the following parameters which affect the execution of certain drive features:

```
44h  Vendor unique length of ECC on Read Long/Write Long commands
55h  Disable read look-ahead feature
66h  Disable reverting to power on defaults
AAh  Enable read look-ahead feature
BBh  4 bytes of ECC apply on Read Long/Write Long commands
CCh  Enable reverting to power on defaults
```

See **10.3** for protocol. If the value in the register is not supported or is invalid, the drive posts an Aborted Command error.

At power on, or after a hardware reset, the default mode is the same as that represented by AAh, BBh, and CCh. A setting of 66h allows settings for read lookahead, number of ECC bytes and multiple count which may have been modified since power on to remain at the

same setting after a software reset.

9.17 Set Multiple Mode

This command enables the drive to perform Read and Write Multiple operations and establishes the block count for these commands. See **10.3** for protocol.

The Sector Count Register is loaded with the number of sectors per block. Drives shall support block sizes of 2, 4, 8, and 16 sectors, if their buffer size is at least 8,192 bytes, and may also support other block sizes. Upon receipt of the command, the drive sets BSY=1 and checks the Sector Count Register.

If the Sector Count Register contains a valid value and the block count is supported, the value is loaded for all subsequent Read Multiple and Write Multiple commands and execution of those commands is enabled. If a block count is not supported, an Aborted Command error is posted, and Read Multiple and Write Multiple commands are disabled.

If the Sector Count Register contains 0 when the command is issued, Read and Write Multiple commands are disabled.

At power on, or after a hardware reset, the default mode is Read and Write Multiple disabled. If Disable Default has been set in the Features Register then the mode remains the same as that last established prior to a software reset, otherwise it reverts to the default of disabled.

9.18 Sleep

This command is the only way to cause the drive to enter Sleep Mode. The drive is spun down, and when it is stopped, BSY is cleared, an interrupt is generated, and the interface becomes inactive.

The only way to recover from Sleep mode without a reset or power on, is for the host to issue a software reset.

A drive shall not power on in Sleep Mode nor remain in Sleep Mode following a reset sequence. If the drive is already spun down, the spin down sequence is not executed.

9.19 Standby

This command causes the drive to enter the Standby Mode. See **10.3** for protocol. The drive may return the interrupt before the transition to Standby Mode is completed.

If the drive is already spun down, the spin down sequence is not executed.

If the Sector Count Register is non-zero then the automatic power down sequence shall be enabled and the timer will begin counting down when the drive returns to Idle mode. If the Sector Count Register is zero then the automatic power down sequence shall be disabled.

9.20 Standby Immediate

This command causes the drive to enter the Standby Mode. See **10.3** for protocol. The drive may return the interrupt before the transition to Standby Mode is completed.

If the drive is already spun down, the spin down sequence is not executed.

9.21 Write Buffer

This command enables the host to overwrite the contents of the drive's sector buffer with any data pattern desired. See **10.2** for protocol.

The Read Buffer and Write Buffer commands shall be synchronized within the drive such that sequential Write Buffer and Read Buffer commands access the same 512 bytes within the buffer.

9.22 Write DMA

This command executes in a similar manner to Write Sectors except for the following:

- the host initializes a slave-DMA channel prior to issuing the command
- data transfers are qualified by DMARQ and are performed by the slave-DMA channel
- the drive issues only one interrupt per command to indicate that data transfer has terminated and status is available.

Any error encountered during Write DMA execution results in the termination of data transfer. The drive issues an interrupt to indicate that data transfer has terminated and status is available in the Error Register. The error posting is the same as that of the Write Sectors command.

9.23 Write Multiple Command

This command is similar to the Write Sectors command. The drive sets BSY within 400 nsec of accepting the command, and interrupts are not presented on each sector but on the transfer of a block which contains the number of sectors defined by Set Multiple.

Command execution is identical to the Write Sectors operation except that the number of sectors defined by the Set Multiple command are transferred without intervening interrupts. DRQ qualification of the transfer is required only at the start of the data block, not on each sector.

The block count of sectors to be transferred without intervening interrupts is programmed by the Set Multiple Mode command, which shall be executed prior to the Read Multiple command.

When the Write Multiple command is issued, the Sector Count Register contains the number of sectors (not the number of blocks or the block count) requested.

If the number of requested sectors is not evenly divisible by the block count, as many full blocks as possible are transferred, followed by a final, partial block transfer. The partial block transfer is for n sectors, where

n = Remainder (Sector Count / Block Count)

If the Write Multiple command is attempted before the Set Multiple Mode command has been executed or when Write Multiple commands are disabled, the Write Multiple operation shall be rejected with an aborted command error.

Disk errors encountered during Write Multiple commands are posted after the attempted disk write of the block or partial block transferred. The Write command ends with the sector in error, even if it was in the middle of a block. Subsequent blocks are not transferred in the

event of an error. Interrupts are generated when DRQ is set at the beginning of each block or partial block.

The contents of the Command Block Registers following the transfer of a data block which had a sector in error are undefined. The host should retry the transfer as individual requests to obtain valid error information.

9.24 Write Same

This command executes in a similar manner to Write Sectors except that only one sector of data is transferred. The contents of the sector are written to the medium one or more times.

NOTE: The Write Same command allows for initialization of part or all of the medium to the specified data with a single command.

If the Features Register is 22h, the drive shall write that part of the medium specified by the sector count, sector number, cylinder and drive/head registers. If the Features Register contains DDh, the drive shall initialize all the user accessible medium. If the register contains a value other than 22h or DDh, the command shall be rejected with an aborted command error.

The drive issues an interrupt to indicate that the command is complete. Any error encountered during execution results in the termination of the write operation. Status is available in the Error Register if an error occurs. The error posting is the same as that of the Write Sectors command.

9.25 Write Long

This command is similar to the Write Sectors command except that it writes the data and the ECC bytes directly from the sector buffer; the drive does not generate the ECC bytes itself. Only single sector Write Long operations are supported.

The transfer of the ECC bytes shall be 8-bits wide.

9.26 Write Sector(s)

This command writes from 1 to 256 sectors as specified in the Sector Count Register (a sector count of zero requests 256 sectors), beginning at the specified sector. See **10.1** for the DRQ, IRQ and BSY protocol on data transfers.

If the drive is not already on the desired track, an implied seek is performed. Once at the desired track, the drive searches for the appropriate ID field.

If retries are disabled and two index pulses have occurred without error free reading of the requested ID, an ID Not Found error is posted.

If retries are enabled, up to a vendor specific number of attempts may be made to read the requested ID before posting an error.

If the ID is read correctly, the data loaded in the buffer is written to the data field of the sector, followed by the ECC bytes. Upon command completion, the Command Block Registers contain the cylinder, head, and sector number of the last sector written.

If an error occurs during a write of more than one sector, writing terminates at the sector where the error occurs. The Command Block Registers contain the cylinder, head, and sector

number of the sector where the error occurred. The host may then read the command block to determine what error has occurred, and on which sector.

9.27 Write Verify

This command is similar to the Write Sectors command, except that each sector is verified immediately after being written. The verify operation is a read without transfer and a check for data errors. Any errors encountered during the verify operation are posted. Multiple sector write verify commands write all the requested sectors and then verify all the requested sectors before generating the final interrupt.

10. Protocol Overview

(Part of **ANSI ATA rev 2.3**)

10. Protocol Overview

Commands can be grouped into different classes according to the protocols followed for command execution. The command classes with their associated protocols are defined below.

For all commands, the host first checks if BSY=1, and should proceed no further unless and until BSY=0. For most commands, the host will also wait for DRDY=1 before proceeding. Those commands shown with DRDY=x can be executed when DRDY=0.

Data transfers may be accomplished in more ways than are described below, but these sequences should work with all known implementations of ATA drives.

10.1 PIO Data In Commands

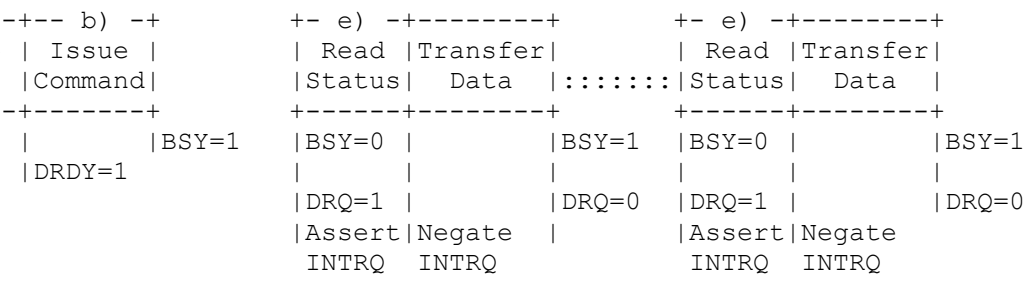
This class includes:

- Identify Drive
- Read Buffer
- Read Long
- Read Sector(s)

Execution includes the transfer of one or more 512 byte (>512 bytes on Read Long) sectors of data from the drive to the host.

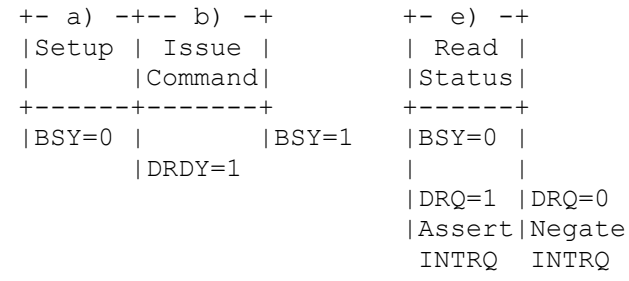
a) The host writes any required parameters to the Features, Sector Count, Sector Number, Cylinder and Drive/Head registers.
b) The host writes the command code to the Command Register.
c) The drive sets BSY and prepares for data transfer.
d) When a sector of data is available, the drive sets DRQ and clears BSY prior to asserting INTRQ.
e) After detecting INTRQ, the host reads the Status Register, then reads one sector of data via the Data Register. In response to the Status Register being read, the drive negates INTRQ.
f) The drive clears DRQ. If transfer of another sector is required, the drive also sets BSY and the above sequence is repeated from d).

10.1.1 PIO Read Command



If Error Status is presented, the drive is prepared to transfer data, and it is at the host's discretion that the data is transferred.

10.1.2 PIO Read Aborted Command



Although DRQ=1, there is no data to be transferred under this condition.

10.2 PIO Data Out Commands

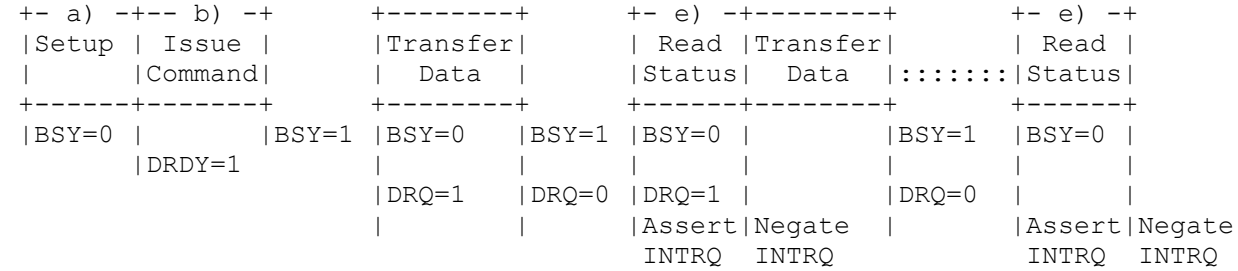
This class includes:

- Format
- Write Buffer
- Write Long
- Write Sector(s)

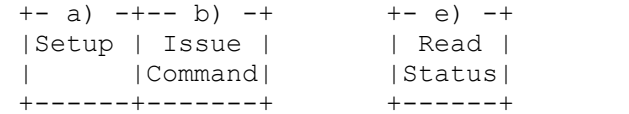
Execution includes the transfer of one or more 512 byte (>512 bytes on Write Long) sectors of data from the drive to the host.

a) The host writes any required parameters to the Features, Sector Count, Sector Number, Cylinder and Drive/Head registers.
b) The host writes the command code to the Command Register.
c) The drive sets DRQ when it is ready to accept the first sector of data.
d) The host writes one sector of data via the Data Register.
e) The drive clears DRQ and sets BSY.
f) When the drive has completed processing of the sector, it clears BSY and asserts INTRQ. If transfer of another sector is required, the drive also sets DRQ.
g) After detecting INTRQ, the host reads the Status Register.
h) The drive clears the interrupt.
i) If transfer of another sector is required, the above sequence is repeated from d).

10.2.1 PIO Write Command



10.2.2 PIO Write Aborted Command



```
|BSY=0 |        |BSY=1   |BSY=0 |
       |DRDY=1           |      |
       |                 |Assert|Negate
                          INTRQ  INTRQ
```

10.3 Non-Data Commands

This class includes:

- Execute Drive Diagnostic (DRDY=x)
- Idle
- Initialize Drive Parameters (DRDY=x)
- Read Power Mode
- Read Verify Sector(s)
- Recalibrate
- Seek
- Set Features
- Set Multiple Mode
- Standby

Execution of these commands involves no data transfer.

a) The host writes any required parameters to the Features, Sector Count, Sector Number, Cylinder and Drive/Head registers.
b) The host writes the command code to the Command Register.
c) The drive sets BSY.
d) When the drive has completed processing, it clears BSY and asserts INTRQ.
g) The host reads the Status Register.
h) The drive negates INTRQ.

10.4 Miscellaneous Commands

This class includes:

- Read Multiple
- Sleep
- Write Multiple
- Write Same

The protocol for these commands is contained in the individual command descriptions.

10.5 DMA Data Transfer Commands (Optional)

This class comprises:

- Read DMA
- Write DMA

Data transfers using DMA commands differ in two ways from PIO transfers:

- data transfers are performed using the slave-DMA channel
- no intermediate sector interrupts are issued on multi-sector commands

Initiation of the DMA transfer commands is identical to the Read Sector or Write Sector commands except that the host initializes the slave-DMA channel prior to issuing the

command.

The interrupt handler for DMA transfers is different in that:

- no intermediate sector interrupts are issued on multi-sector commands
- the host resets the DMA channel prior to reading status from the drive.

The DMA protocol allows high performance multi-tasking operating systems to eliminate processor overhead associated with PIO transfers.

a) Command Phase
   1) Host initializes the slave-DMA channel
   2) Host updates the Command Block Registers
   3) Host writes command code to the Command Register
b) Data Phase - the register contents are not valid during a DMA Data Phase.
   1) The slave-DMA channel qualifies data transfers to and from the drive with DMARQ
c) Status Phase
   1) Drive generates the interrupt to the host
   2) Host resets the slave-DMA channel
   3) Host reads the Status Register and Error Register

10.5.1 Normal DMA Transfer

```
+--------------+-------+      +---------------------+  +---------+------+
|Initialize DMA|Command|      |  DMA Data Transfer  |  |Reset DMA|Status|
+--------------+-------+      +---------------------+  +---------+------+
|BSY=0                 |BSY=1 |BSY=x                    |BSY=1    |BSY=0
                              |DRQ=x                    |nIEN=0
```

10.5.2 Aborted DMA Transfer

```
+--------------+-------+      +-------------+  +---------+------+
|Initialize DMA|Command|      |    DMA Data |  |Reset DMA|Status|
+--------------+-------+      +-------------+  +---------+------+
|BSY=0                 |BSY=1 |BSY=x           |BSY=1    |BSY=0
                              |DRQ=1           |nIEN=0
```

10.5.3 Aborted DMA Command

```
+--------------+-------+      +---------+------+
|Initialize DMA|Command|      |Reset DMA|Status|
+--------------+-------+      +---------+------+
|BSY=0                 |BSY=1 |BSY=1    |BSY=0
                              |nIEN=0
```

11. Timing

(Part of **ANSI ATA rev 2.3**)

11. Timing

11.1 Deskewing

The host shall provide cable deskewing for all signals originating from the controller. The drive shall provide cable deskewing for all signals originating at the host.

11.2 Symbols

Certain symbols are used in the timing diagrams. These symbols and their respective definitions are listed below.

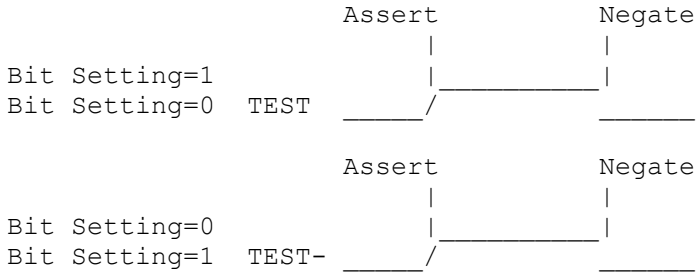
```
 / or    -  signal transition (asserted or negated) *
 < or >    -  data transition (asserted or negated)
 XXXXXX    -  undefined but not necessarily released
 .  .  .   -  the "other" condition if a signal is shown with no change
   #n      -  used to number the sequence in which events occur e.g. #a, #b
   _ _ __
__/_ _/    -  a degree of uncertainty as to when a signal may be asserted


__ _ _
   __   -  a degree of uncertainty as to when a signal may be negated
```

* All signals are shown with the Asserted condition facing to the top of the page. The negated condition is shown towards the bottom of the page relative to the asserted condition.

11.3 Terms

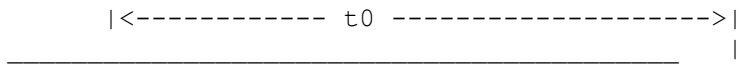
The interface uses a mixture of negative and positive signals for control and data. The terms asserted and negated are used for consistency and are independent of electrical characteristics.

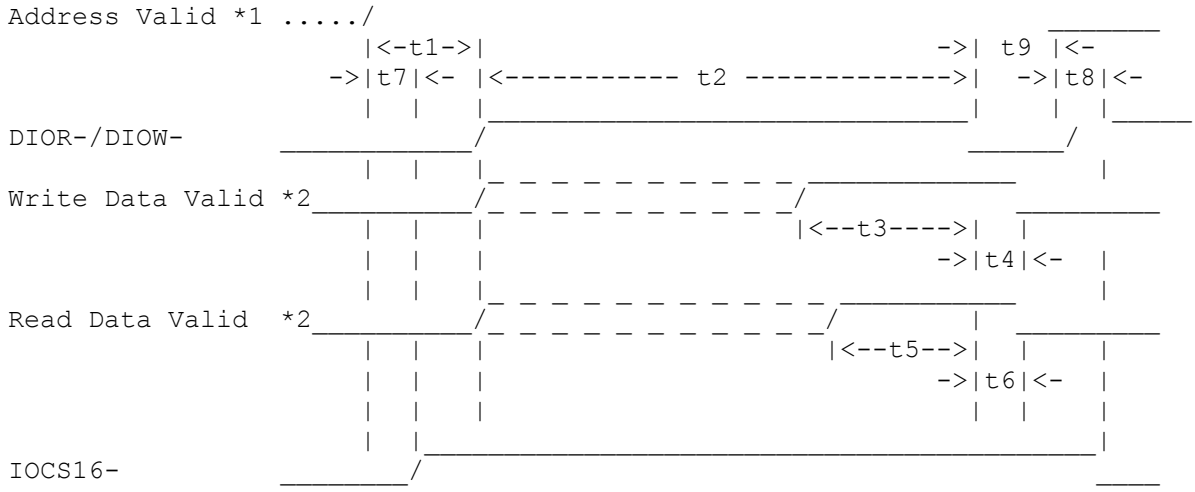
In all timing diagrams, the lower line indicates negated, and the upper line indicates asserted e.g. the following illustrates the representation of a signal named TEST going from negated to asserted and back to negated, based on the polarity of the signal.



11.4 Data Transfers

Figure 11-1 defines the relationships between the interface signals for both 16-bit and 8-bit data transfers.

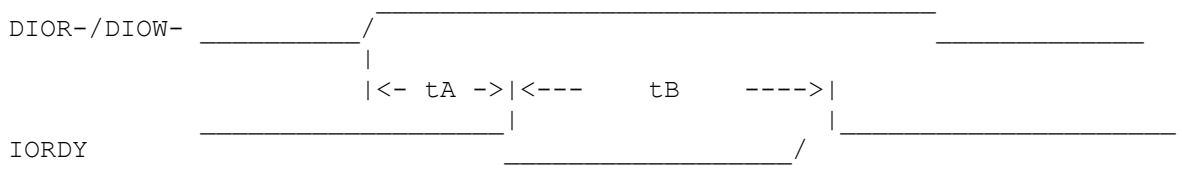




*1 Drive Address consists of signals CS1FX-, CS3FX- and DA2-0
*2 Data consists of DD0-15 (16-bit) or DD0-7 (8-bit)

	PIO Timing Parameters		Mode 0 nsec	Mode 1 nsec	Mode 2 nsec
t0	Cycle Time	(Min)	600	383	240
t1	Address Valid to DIOR-/DIOW- Setup	(Min)	70	50	30
t2	DIOR-/DIOW- 16-bit	(Min)	165	125	100
	Pulse Width 8-bit	(Min)	290	290	290
t3	DIOW- Data Setup	(Min)	60	45	30
t4	DIOW- Data Hold	(Min)	30	20	15
t5	DIOR- Data Setup	(Min)	50	35	20
t6	DIOR- Data Hold	(Min)	5	5	5
t7	Addr Valid to IOCS16- Assertion	(Max)	90	50	40
t8	Addr Valid to IOCS16- Negation	(Max)	60	45	30
t9	DIOR-/DIOW- to Address Valid Hold	(Min)	20	15	10

FIGURE 11-1: PIO DATA TRANSFER TO/FROM DRIVE

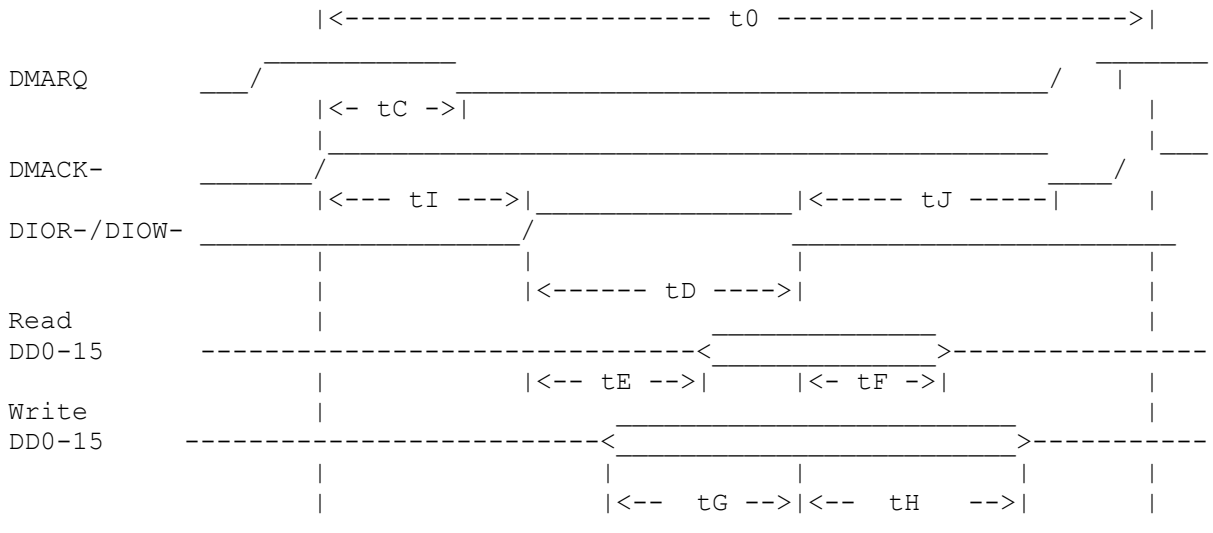


Label	Description	Min	Max	Units
tA	IORDY Setup time	-	35	nsecs
tB	IORDY Pulse Width	-	1,250	nsecs

WARNING: The use of IORDY for data transfers is a system integration issue which requires control of both ends of the cable.

FIGURE 11-2: IORDY TIMING REQUIRMENTS

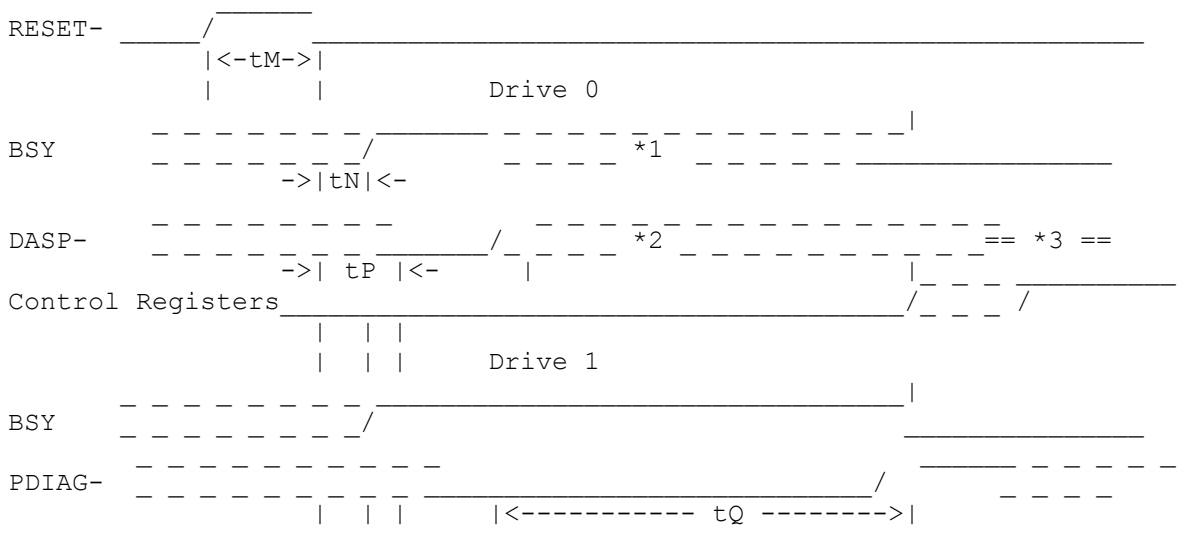
- -

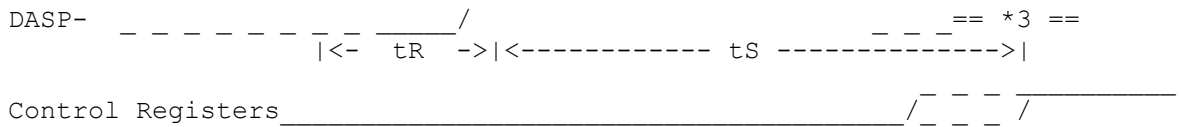


DMA Timing Parameters		Mode 0 nsec	Mode 1 nsec	Mode 2 nsec
t0	Cycle Time (Min)	960	480	240
tC	DMACK to DMREQ Delay (Max)	200	100	80
tD	DIOR-/DIOW- 16-bit (Min)	480	240	120
tE	DIOR- Data Setup (Min)	250	150	50
tF	DIOR- Data Hold (Min)	5	5	5
tG	DIOW- Data Setup (Min)	250	100	35
tH	DIOW- Data Hold (Min)	50	30	20
tI	DMACK to DIOR-/DIOW- Setup (Min)	0	0	0
tJ	DIOR-/DIOW- to DMACK Hold (Min)	0	0	0

FIGURE 11-3: DMA DATA TRANSFER

11.5 Power On and Hard Reset





*1 Drive 0 can set BSY=0 if Drive 1 not present
*2 Drive 0 can use DASP- to indicate it is active if Drive 1 is not present
*3 DASP- can be asserted to indicate that the drive is active

Label		Units	
tM	(Min)	25	usec
tN	(Max)	400	nsec
tP	(Max)	1	msec
tQ	(Max)	30	secs
tR Drive 0	(Max)	450	msec
tR Drive 1	(Max)	400	msec
tS	(Max)	30.5	secs

FIGURE 11-4 RESET SEQUENCE

Annex A: Diagnostic and Reset Considerations

(Part of **ANSI ATA rev 2.3**)

Annex A: Diagnostic and Reset Considerations
(informative).

This annex describes the following timing relationships during:

```
 a) Power On and Hardware Resets
    - One drive
    - Two drives
 b) Software Reset
    - One drive
    - Two drives
 c) Diagnostic Command Execution
    - One drive
    - Two drives
    - Two drives - Drive 1 failed
```

The timing assumes the following:

- o DASP- is asserted by Drive 1 and received by Drive 0 at power-on or hardware reset to indicate the presence of Drive 1. At all other times it is asserted by Drive 0 and Drive 1 to indicate when a drive is active.

- o PDIAG- is asserted by Drive 1 and detected by Drive 0. It is used by Drive 1 to indicate to Drive 0 that it has completed diagnostics and is ready to accept commands from the Host (BSY bit is cleared). This does not indicate that the drive is ready, only that it can accept commands. This line may remain asserted until the next reset occurs or an Execute Diagnostic command is received.

- o Unless indicated otherwise, all times are relative to the event that triggers the operation (RESET-, SRST=1, Execute Diagnostic Command).

A.1 Power On and Hardware Resets

A.1.1 Power On and Hardware Resets - One Drive

- Host asserts RESET- for a minimum of 25 usec.
- Drive 0 sets BSY within 400 nsecs after RESET- is negated.
- Drive 0 negates DASP- within 1 msec after RESET- negated.
- Drive 0 performs hardware initialization
- Drive 0 may revert to its default condition
- Drive 0 waits 1 msec then samples for at least 450 msec for DASP- to be asserted from Drive 1.
- Drive 0 clears BSY when ready to accept commands (within 31 seconds).

A.1.2 Power On and Hardware Resets - Two Drives

- Host asserts RESET- for a minimum of 25 usec.
- Drive 0 and Drive 1 set BSY within 400 nsec after RESET- negated.
- DASP- is negated within 1 msec after RESET- is negated.

A.1.2.1 Drive 1

- Drive 1 negates PDIAG- before asserting DASP-.
- Drive 1 asserts DASP- within 400 msecs after RESET- (to show presence).
- Drive 1 performs hardware initialization and executes its internal diagnostics.
- Drive 1 may revert to its default condition
- Drive 1 posts diagnostic results to the Error Register
- Drive 1 clears BSY when ready to accept commands.
- Drive 1 asserts PDIAG- to indicate that it is ready to accept commands (within 30 seconds from RESET-).
- Drive 1 negates DASP- after the first command is received or negates DASP- if no command is received within 30 seconds after RESET-.

A.1.2.2 Drive 0

- Drive 0 performs hardware initialization and executes its internal diagnostics.
- Drive 0 may revert to its default condition
- Drive 0 posts diagnostic results to the Error Register
- After 1 msec, Drive 0 waits at least 450 msec for DASP- to be asserted (from Drive 1). If DASP- is not asserted, no Drive 1 is present (see POWER- ON RESET - One Drive operation).
- Drive 0 waits up to 31 seconds for Drive 1 to assert PDIAG-. If PDIAG- is not asserted, Drive 0 sets Bit 7=1 in the Error Register.
- Drive 0 clears BSY when ready to accept commands (within 31 seconds).

A.2 Software Reset

A.2.1 Software Reset - One Drive

- Host sets SRST=1 in the Device Control Register.
- Drive 0 sets BSY within 400 nsec after detecting that SRST=1.
- Drive 0 performs hardware initialization and executes its internal diagnostics.
- Drive 0 may revert to its default condition.
- Drive 0 posts diagnostic results to the Error Register.
- Drive 0 clears BSY when ready to accept commands (within 31 seconds).

A.2.2 Software Reset - Two Drives

- Host sets SRST=1 in the Device Control Register.
- Drive 0 and Drive 1 set BSY within 400 nsec after detecting that SRST=1.
- Drive 0 and Drive 1 perform hardware initialization.
- Drive 0 and Drive 1 may revert to their default condition.

A.2.2.1 Drive 1

- Drive 1 negates PDIAG- within 1 msec.
- Drive 1 clears BSY when ready to accept commands.
- Drive 1 asserts PDIAG- to indicate that it is ready to accept commands (within 30 seconds).

A.2.2.2 Drive 0

- Drive 0 waits up to 31 seconds for Drive 1 to assert PDIAG-.

- Drive 0 clears BSY when ready to accept commands (within 31 seconds).

A.3 Diagnostic Command Execution

A.3.1 Diagnostic Command Execution - One Drive (Passed)

- Drive 0 sets BSY within 400 nsec after the Execute Diagnostic command was received.
- Drive 0 performs hardware initialization and internal diagnostics.
- Drive 0 resets Command Block registers to default condition.
- Drive 0 posts diagnostic results to the Error Register
- Drive 0 clears BSY when ready to accept commands (within 6 seconds).

A.3.2 Diagnostic Command - Two Drives (Passed)

- Drive 0 and Drive 1 set BSY within 400 nsec after the Execute Diagnostic command was received.

A.3.2.1 Drive 1

- Drive 1 negates PDIAG- within 1 msec after command received.
- Drive 1 performs hardware initialization and internal diagnostics.
- Drive 1 resets the Command Block registers to their default condition.
- Drive 1 posts diagnostic results to the Error Register
- Drive 1 clears BSY when ready to accept commands.
- Drive 1 asserts PDIAG- to indicate that it is ready to accept commands (within 5 seconds).

A.3.2.2 Drive 0

- Drive 0 performs hardware initialization and internal diagnostics.
- Drive 0 resets the Command Block registers to their default condition.
- Drive 0 waits up to <5 seconds for Drive 1 to assert PDIAG-.
- Drive 0 posts diagnostic results to the Error Register
- Drive 0 clears BSY when ready to accept commands (within 6 seconds).

A.3.3 Diagnostic Command Execution - One Drive (Failed)

- Drive 0 sets BSY within 400 nsec after Diagnostic command received.
- Drive 0 performs hardware initialization and internal diagnostics.
- Drive 0 resets Command Block registers to default condition.
- Drive 0 posts a Diagnostic Code to the Error Register indicating a failure.
- Drive 0 clears BSY when ready to accept commands (within 6 seconds)

A.3.4 Diagnostic Command Execution - Two Drives (Drive 1 Failed)

- Drive 0 and Drive 1 set BSY within 400 nsec after Diagnostic command received.

A.3.4.1 Drive 1

- Drive 1 negates PDIAG- within 1 msec after command received.
- Drive 1 performs hardware initialization and internal diagnostics.
- Drive 1 resets the Command Block registers to their default condition.
- Drive 1 posts a Diagnostic Code to the Error Register indicating failure.

- Drive 1 clears BSY.
- Drive 1 does not assert PDIAG-, indicating that it failed diagnostics.

A.3.4.2 Drive 0

- Drive 0 performs hardware initialization and internal diagnostics.
- Drive 0 resets the Command Block registers to their default condition.
- Drive 0 waits 6 seconds for Drive 1 to assert PDIAG- but PDIAG- is not asserted by Drive 1.
- Drive 0 posts a Diagnostic Code to the Error Register setting Bit 7=1 to indicate that Drive 1 failed diagnostics.
- Drive 0 clears BSY when ready to accept commands (within 6 seconds).

NOTE: The 6 seconds referenced above is a host-oriented value.

Annex B: Diagnostic and Reset Considerations

(Part of **ANSI ATA rev 2.3**)

Annex B: Diagnostic and Reset Considerations
(informative).

B.1 Power on and hardware reset (RESET-)

DASP- is read by Drive 0 to determine if Drive 1 is present. If Drive 1 is present Drive 0 will read PDIAG- to determine when it is valid to clear BSY and whether Drive 1 has powered on or reset without error, otherwise Drive 0 clears BSY whenever it is ready to accept commands. Drive 0 may assert DASP- to indicate drive activity.

B.2 Software reset

If Drive 1 is present Drive 0 will read PDIAG- to determine when it is valid to clear BSY and whether Drive 1 has reset without any errors, otherwise Drive 0 will simply reset and clear BSY. DASP- is asserted by Drive 0 (and Drive 1 if it is present) in order to indicate drive active.

B.3 Drive Diagnostic Command

If Drive 1 is present, Drive 0 will read PDIAG- to determine when it is valid to clear BSY and if Drive 1 passed or failed the Execute Drive Diagnostic command, otherwise Drive 0 will simply execute its diagnostics and then clear BSY. DASP- is asserted by Drive 0 (and Drive 1 if it is present) in order to indicate the drive is active.

B.4 Truth Table

In all the above cases: Power on, RESET-, software reset, and the Execute Drive Diagnostics command the Drive 0 Error Register is calculated as follows:

Drive 1 Present?	PDIAG- Asserted?	Drive 0 Passed	Error Register
Yes	Yes	Yes	01h
Yes	Yes	No	0xh
Yes	No	Yes	81h
Yes	No	No	8xh
No	(not read)	Yes	01h
No	(not read)	No	0xh

Where x indicates the appropriate Diagnostic Code for the Power on, RESET-, software reset, or drive diagnostics error.

B.5 Power On or Hardware Reset Algorithm

1) Power on or hardware reset
2) The hardware should automatically do the following:
   a) Set up the hardware to post both Drive 0 and Drive 1 status
   b) Set the Drive 0 Status Register to 80h (set BSY and clear all the other status bits)
   c) Set the Drive 1 Status Register to 80h (set BSY and clear all the other status bits)
3) Read the single Drive 0/Drive 1 jumper and note its state
4) Perform any remaining time critical hardware initialization including starting the spin up of the disk if needed

5) If Drive 1
   a) Negate the PDIAG- signal
   b) Set up PDIAG- as an output
   c) Assert the DASP- output
   d) Set up DASP- as an output if necessary
   e) Set up the hardware so it posts Drive 1 status only and continue to post 80h for Drive 1 status

   NOTE: all this must happen within 400 msec after power on or RESET-

   If Drive 0
   a) Set up PDIAG- as an input
   b) Release DASP- and set up DASP- as an input
   c) Test DASP- for 450 msec or until DASP- is asserted by Drive 1
   d) If DASP- is asserted within 450 msec
      i) Note that Drive 1 is present
      ii) Set up the hardware so it posts Drive 0 status only and continue to post 80h for the Drive 0 status

      If DASP- is not asserted within 450 msec
      i) note that Drive 1 is not present
   e) Assert DASP- to indicate drive activity

6) Complete all the hardware initialization needed to get the drive ready, including:
   a) Set the Sector Count Register to 01h
   b) Set the Sector Number Register to 01h
   c) Set the Cylinder Low Register to 00h
   d) Set the Cylinder High Register to 00h
   e) Set the Drive/Head Register to 00h

7) If Drive 1 and power on, or RESET- is valid
   a) Set the Error Register to Diagnostic Code 01h
   b) Set the Drive 1 Status Register to 00h
   c) Assert PDIAG-

   NOTE: All this must happen within 5 seconds of power on or the negation of RESET-

   If Drive 1 and power on or RESET- bad
   a) Set the Error Register to the appropriate Diagnostic Code
   b) Set the Drive 1 Status Register to 00h

   NOTE: All this must happen within 5 seconds of power on or the negation of RESET-

   If Drive 0, power on or RESET- valid, and a Drive 1 is present
   a) Test PDIAG- for 6 seconds or until PDIAG- is asserted by Drive 1
   b) If PDIAG- is asserted within 6 seconds
      i) Set the Error Register to Diagnostic Code 01h
   c) If PDIAG- is not asserted within 6 seconds
      i) Set the Error Register to 81h
   d) Set the Drive 0 Status Register to 00h

   If Drive 0, power on or RESET- bad, and a Drive 1 is present
   a) Test PDIAG- for 6 seconds or until PDIAG- is asserted by Drive 1
   b) If PDIAG- is asserted within 6 seconds
      i) Set the Error Register to the appropriate Diagnostic Code

   c) If PDIAG- is not asserted within 6 seconds
      i) Set the Error Register to 80h + the appropriate code
   d) Set the Drive 0 Status Register to 00h

   If Drive 0, power on or RESET- valid, and no Drive 1 is present
   a) Set the Error Register to Diagnostic Code 01h
   b) Set the Drive 1 Status Register to 00h
   c) Set the Drive 0 Status Register to 00h

```
   If Drive 0, power on or RESET- bad, and no Drive 1 is present
        a) Set the Error Register to the appropriate Diagnostic Code
        b) Set the Drive 1 Status Register to 00h
        c) Set the Drive 0 Status Register to 00h
8) Finish spin up if needed
9) If Drive 1
        a) Set the Drive 1 Status Register to 50h
        b) Negate DASP- if a command is not received within 30 seconds
   If Drive 0 and a Drive 1 is present
        a) Set the Drive 0 Status Register to 50h
        b) Negate DASP-
   If Drive 0 and no Drive 1 is present
        a) Leave the Drive 1 Status Register 00h
        b) Set the Drive 0 Status Register to 50h
        c) Negate DASP-
```

B.6 Software Reset Algorithm

```
1) The software reset bit is set
2) If Drive 1
        a) The hardware should set BUSY in the Drive 1 Status Register
        b) Negate the PDIAG- signal
        NOTE: this must happen within 1 msec of the software reset
   If Drive 0 and Drive 1 is present
           a) The hardware should set BUSY in the Drive 0 Status Register
   If Drive 0 and there is no Drive 1 the hardware should:
           a) Set BUSY in the Drive 0 Status Register
        b) Set the Drive 1 Status Register to 80h
3) Assert DASP-
4) Finish all the hardware initialization needed to place the drive in reset
5) Wait for the software reset bit to clear
6) Finish all hardware initialization needed to get the drive ready
   to receive any type of command from the host including:
        a) Set the Sector Count Register to 01h
        b) Set the Sector Number Register to 01h
        c) Set the Cylinder Low Register to 00h
        d) Set the Cylinder High Register to 00h
        e) Set the Drive/Head Register to 00h
7) If Drive 1 and reset valid
        a) Set the Error Register to Diagnostic Code 01h
        b) Set the Drive 1 Status Register to 50h
        c) Assert PDIAG-
        NOTE:  All this must happen within 5 seconds of the clearing of
               the software reset bit
   If Drive 1 and reset bad
        a) Set the Error Register to the appropriate Diagnostic Code
        b) Set the Drive 1 Status Register to 50h
        NOTE:  All this must happen within 5 seconds of the clearing of
               the software reset bit
   If Drive 0, reset valid, and a Drive 1 is present
           a) Test PDIAG- for 6 seconds or until PDIAG- is asserted by
           Drive 1
           b) If PDIAG- is asserted within 6 seconds
                i)  Set the Error Register to Diagnostic Code 01h
        c) If PDIAG- is not asserted within 6 seconds
                i)  Set the Error Register to 81h
        d) Set the Drive 0 Status Register to 50h
```

```
   If Drive 0, reset bad, and a Drive 1 is present
           a) Test PDIAG- for 31 seconds or until PDIAG- is asserted by
           Drive 1
           b) If PDIAG- is asserted within 31 seconds
                i)  Set the Error Register to the appropriate Diagnostic Code

        c) If PDIAG- is not asserted within 31 seconds
                i)  Set the Error Register to 80h + the appropriate code
        d) Set the Drive 0 Status Register to 50h
   If Drive 0, reset valid, and no Drive 1 is present
           a) Set the Error Register to Diagnostic Code 01h
        b) Set the Drive 1 Status Register to 00h
           c) Set the Drive 0 Status Register to 50h
   If Drive 0, reset bad, and no Drive 1 is present
           a) Set the Error Register to the appropriate Diagnostic Code
        b) Set the Drive 1 Status Register to 00h
           c) Set the Drive 0 Status Register to 50h
```

B.7 Diagnostic Command Algorithm

```
1) The diagnostics command is received
2) If Drive 1
        a) The hardware should set BUSY in the Drive 1 Status Register
        b) Negate the PDIAG- signal
        NOTE: this must happen within 1 msec after command acceptance
   If Drive 0 and Drive 1 is present
           a) The hardware should set BUSY in the Drive 0 Status Register
   If Drive 0 and there is no Drive 1 the hardware should
           a) Set BUSY in the Drive 0 Status Register
        b) Set BUSY in the Drive 1 Status Register
3) Assert DASP-
4) Perform all the drive diagnostics and note their results
5) Finish all the hardware initialization needed to get the drive ready
   to receive any type of command from the host including:
        a) Set the Sector Count Register to 01h
        b) Set the Sector Number Register to 01h
        c) Set the Cylinder Low Register to 00h
        d) Set the Cylinder High Register to 00h
        e) Set the Drive/Head Register to 00h
6) If Drive 1 and passed
        a) Set the Error Register to Diagnostic Code 01h
        b) Set the Drive 1 status to 50h
        c) Assert PDIAG-
        NOTE:  All this must happen within 5 seconds of the acceptance
               of the diagnostic command
   If Drive 1 and did not pass
           a) Set the Error Register to the appropriate Diagnostic Code
        b) Set the Drive 1 status to 50h
        NOTE:  All this must happen within 5 seconds of the acceptance
               of the diagnostic command
   If Drive 0, passed, and a Drive 1 is present
           a) Test PDIAG- for 6 seconds or until PDIAG- is asserted by
           Drive 1
           b) If PDIAG- is asserted within 6 seconds
                i)  Set the Error Register to Diagnostic Code 01h
        c) If PDIAG- is not asserted within 6 seconds
                i)  Set the Error Register to 81h
```

d) Set the Drive 0 status to 50h
e) Issue interrupt to the host

If Drive 0, did not pass, and a Drive 1 is present
a) Test PDIAG- for 6 seconds or until PDIAG- is asserted by Drive 1
b) If PDIAG- is asserted within 6 seconds
i) Set the Error Register to the appropriate Diagnostic Code

c) If PDIAG- is not asserted within seconds
i) Set the Error Register to 80h + the appropriate code
d) Set the Drive 0 Status Register to 50h
e) Issue interrupt to the host

If Drive 0, passed, and no Drive 1 is present
a) Set the Error Register to Diagnostic Code 01h
b) Set the Drive 1 Status Register to 00h
c) Set the Drive 0 Status Register to 50h
d) Issue interrupt to the host

If Drive 0, did not pass, and no Drive 1 is present
a) Set the Error Register to the appropriate Diagnostic Code
b) Set the Drive 1 Status Register to 00h
c) Set the Drive 0 Status Register to 50h
d) Issue interrupt to the host

Copies of this proposal may be purchased from: BSR X3.***
Global Engineering, 2805 McGaw St, Irvine, CA 92714 X3T9.2/90-186
800-854-7179 714-261-1455

working draft proposed American National
Standard for Information Systems -

SCSI-2 Common Access Method

Transport
and
SCSI Interface Module

Rev 2.3 February 25, 1991

(Part of **ANSI Common Access Method rev 2.3**)

Secretariat

Computer and Business Equipment Manufacturers Association (CBEMA)

Abstract: This standard defines the software interface between device drivers and the Host Bus Adapters or other means by which SCSI peripherals are attached to a host processor. The software interface defined provides a common interface specification for systems manufacturers, system integrators, controller manufacturers, and suppliers of intelligent peripherals.

This is an internal working document of X3T9.2, a Task Group of Accredited Standards Committee X3. As such this is not a completed standard. The contents are actively being modified by the X3T9.2 Task Group. This document is made available for review and comment only.

POINTS OF CONTACT:

John B. Lohmeyer
Chairman X3T9.2
NCR
3718 N Rock Rd
Wichita KS 67226

316-636-8703

I. Dal Allan
Vice-Chairman X3T9.2
ENDL
14426 Black Walnut Court
Saratoga CA 95070

408-867-6630

An electronic copy of this document is available from the SCSI Bulletin Board (316-636-8700).

This document has been prepared according to the style guide of the ISO (International Organization of Standards).

If this document was printed in a 2-up form directly from the printer, NOTEs had to be adjusted to fit into a half-page, which may have resulted in an imperfect representation of the format within the NOTE. This is most likely to occur if a series of NOTEs are mixed in without any line separation.

> This revision contains recommended changes by Jerry Armstrong and the DOS OSD working group. You will find that the DOS OSD has been extensively revised.
>
> John Gallant also submitted a list of change requests, most of which are included. Please look them over carefully, and be sure to advise Dal by fax at 408-867-2115 of any objections.

Foreword (This Foreword is not part of American National Standard X3.***-199x.)

In this standard, the Transport (XPT) and SCSI Interface Module (SIM) for the SCSI-2 Common Access Method is defined.

When the Small Computer System Interface (SCSI) was introduced, a large number of systems integrators included support in their operating systems. However, they were

parochial in implementation and a diverse set of strategies to support SCSI devices were implemented in software.

Some companies published their specifications and encouraged third-party suppliers to add new peripherals. Others failed to add support for SCSI or did not publish the specifications. An increasing demand developed for some common method to attach SCSI peripherals to a number of operating systems and a large range of computer systems. Much of this impetus stemmed from the growth in the desktop computing environment.

In October 1988 a number of peripheral suppliers formed the Common Access Method Committee to encourage an industry-wide effort to adopt a common software interface to despatch input/output requests to SCSI peripherals.

The primary objective was to define a set of software constructs and tables that would permit the manufacturers of host adapters to provide software or microcode to interpret requests in a common manner.

Out of the proposals made by a large number of contributors, the CAM Committee selected the best concepts and used them to develop the standard.

Some of the companies which contributed had designed their own methods to support SCSI devices, and for the most part set aside individual business considerations to foster the development and adoption of this standard.

Suggestions for improvement of this standard will be welcome. They should be sent to the Computer and Business Equipment Manufacturers Association, 311 First Street N.W., Suite 500, Washington, DC 20001.

This standard was processed and approved for submittal to ANSI by the Accredited Standards Committee on Information Procsessing Systems, X3. Committee approval of this standard does not necessarily imply that all committee members voted for its approval. At the time it approved this standard, the X3 Committee had the following members:

X3 Committee list goes here:

Subcommittee X3T9 on I/O interfaces, which reviewed this standard, had the following members:

X3T9 Committee list goes here:

Task Group X3T9.2 on Lower-Level Interfaces, which completed the development of this standard, had the following members:

X3T9.2 Committee list goes here:

The initial development work on this standard was done by the CAM Committee, an industry group formed for this purpose. The membership of the CAM Committee consisted of the following organizations:

Adaptec	Data Technology	NCR
AMD	Eastman Kodak	Olivetti
Apple	Emulex	Quantum
AT&T Bell Labs	Fujitsu uElectronics	Scientific Micro Systems

Caliper	Future Domain	Seagate
Cambrian Systems	Hewlett Packard	Sony
Cipher Data	IBM	Storage Dimensions
Cirrus Logic	Imprimis	Sun Microsystems
Columbia Data	Interactive Systems	Syquest Technology
CompuAdd	JVC	Sytron
Conner Peripherals	LMS OSD	Trantor
Dell Computer	Maxtor	Western Digital
Digital Equipment	Micropolis	
DPT	Miniscribe	

ANSI CAM - Table of contents

(Part of **ANSI Common Access Method rev 2.3**)

TABLE OF CONTENTS

FIGURES

TABLES

Information Processing Systems --

Common Access Method --

SCSI and Generic I/O

1. Scope

(Part of **ANSI Common Access Method rev 2.3**)

1. Scope

This standard defines the CAM (Common Access Method) for SCSI (Small Computer Systems Interface).

The purpose of this standard is to define a method whereby multiple environments may adopt a common procedure for the support of SCSI devices.

The CAM provides a structured method for supporting peripherals with the software (e.g. device driver) and hardware (e.g. host bus adapter) associated with any computer.

SCSI has provided a diverse range of peripherals for attachment to a wide range of computing equipment. Some system manufacturers have developed approaches for SCSI attachment which are widely followed, increasing the applications available for the attachment of SCSI peripherals. In markets where no standard method of attachment exists, however, variations between third party sellers has made it near-impossible for end users to rely on being able to attach more than one SCSI peripheral to one host bus adapter.

In an effort to broaden the application base for SCSI peripherals an ad hoc industry group of companies representing system integrators, controllers, peripherals, and semiconductors decided to address the issues involved.

The CAM Committee was formed in October, 1988 and the first working document of the XPT/SIM for SCSI I/O was introduced in October, 1989.

1.1 Description of Clauses

Clause 1 contains the Scope and Purpose.

Clause 2 contains Referenced and Related International Standards.

Clause 3 contains the General Description.

Clause 4 contains the Glossary.

Clause 5 describes the services provided by the XPT and SIM.

Clause 6 describes the facilities that use the Transport and SIM.

Clause 7 describes the ways that the Operating Systems support CAM and access the XPT.

Clause 8 contains the description of non-I/O functions supported by the XPT and SIM.

Clause 9 contains the description of I/O functions supported by the XPT and SIM.

Clause 10 contains the description of Target Mode functions supported by the XPT and SIM.

2. References

(Part of **ANSI Common Access Method rev 2.3**)

2. References

ISO DIS 10288 (ANSI X3.131-1990)
SCSI-2, Enhanced Small Computer Systems Interface

3. General Description

(Part of **ANSI Common Access Method rev 2.3**)

3. General Description

The application environment for CAM is any computer addressing a SCSI peripheral through a protocol chip on a motherboard, an ATA interface, or a Host Bus Adapter.

SCSI is a widely-used interface which provides common attachment for a variety of peripherals. Unfortunately, there is no common way to provide access to SCSI peripherals.

The purpose of the Common Access Method is to define a standard for the support of Host Bus Adapters and the like by device driver software.

Software in the Operating System dispatches I/O (Input/Output) requests to the SCSI peripherals in a number of different ways depending on the software architecture. The OSD (Operating System Dependencies) are defined in **Clause 6** for named software and hardware platforms.

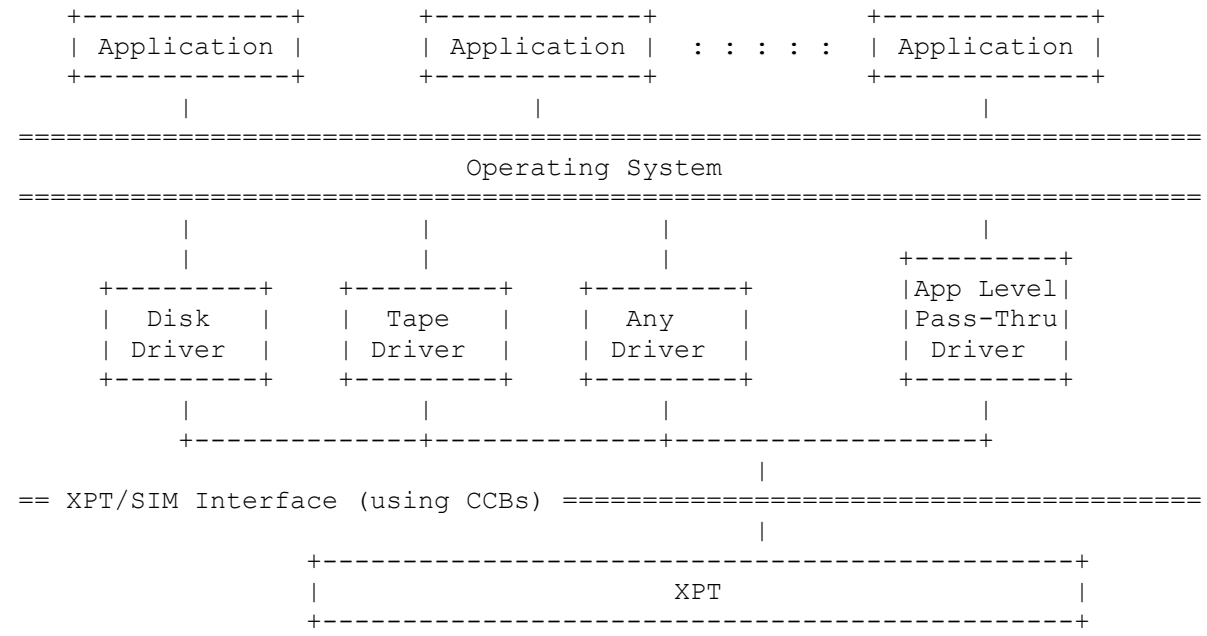
3.1 Environment

A model of the CAM usage environment is illustrated in Figure 3-1, where there may be multiple application and several device drivers attached to support the peripherals on the system.

Requests for SCSI I/O are made through the CAM Transport (XPT) interface. The XPT may execute them directly or pass them on to a lower level SIM for execution.

The XPT (Transport) function is illustrated as a separate element. In many applications, the XPT operations will be incorporated into a single module which integrates both XPT and SIM functionality. The logical separation between the two is maintained as there may be more than one SIM loaded.

A separate routing service may be provided by the operating system or the routing function can be achieved through chaining when multiple SIMs are loaded.



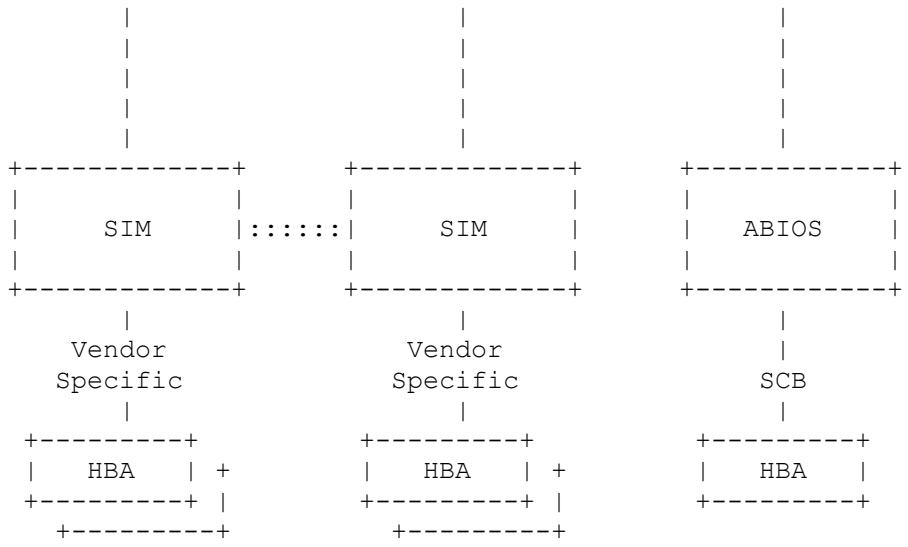


FIGURE 3-1 CAM ENVIRONMENT MODEL

3.2 Peripheral Driver Functions

Peripheral drivers provide the following functionality:

a) Interpretation of application or system level requests.
b) Mapping of application level requests to XPT/SIM Control Blocks.
c) Requesting of resources to initiate a CAM request.
   - CAM Control Blocks and supporting blocks that may be needed.
   - Buffer requirements.
d) Handling of exception conditions not managed transparently by SCSI e.g. Check Condition status, unexpected Bus Free, Resets etc).
e) Logging of exception conditions for maintenance analysis programs.
f) Format utility or services required by format utilities.
g) Establish parameters for HBA operation.
h) Set up routing of SCSI requests to the correct Path/Bus, target and LUN.
i) Initialization and configuration functions of a target not handled by a utility at installation and formatting time.
j) Establish a timeout value for a task and pass this value in the CCB.

3.3 XPT Functions

XPT services provide the following functionality to process CCBs:

a) Routing of the target CCB to the proper SIM.
b) OSD allocation of CCB resources e.g. Get_CCB, Free_CCB.
c) Maintenance of the SCSI Device Table. This consists of owning the table and servicing requests to read and write the table.
d) Providing properly formatted control blocks and priming the fields needed to accomplish a request.
e) Routing of asynchronous events back to peripheral driver.

3.4 SIM Functions

SIM services provide the following functionality to process CCBs:

a) Perform all interface functions to the SCSI HBA.

b) Manage or delegate, as required, all the SCSI HBA protocol steps.
c) Distinguish abnormal behavior and perform error recovery, as required.
d) Management of data transfer path hardware, including DMA circuitry and address mapping, and establish DMA resource requests (if necessary).
e) Queueing of multiple operations for different LUNs as well as the same LUN and assign tags for Tag Queueing (if supported).
f) Freeze and unfreeze the queue as necessary to accomplish queue recovery.
g) Assuring that the completed operation is posted back to the initiating device driver.
h) Management of selection, disconnection, reconnection, and data pointers of the SCSI HBA protocol.
i) Mechanisms to accept the selecting and sensing of the SCSI HBA functions supported.
j) Implement a timer mechanism, using values provided by the peripheral driver.

4. Definitions and Conventions

(Part of **ANSI Common Access Method rev 2.3**)

4. Definitions and Conventions

4.1 Definitions

For the purpose of this standard the following definitions apply:

4.1.1 Block: This defines an action to prevent access e.g. Busy.

4.1.2 CCB (CAM Control Block): The data structure provided by peripheral drivers to the XPT to control execution of a function by the SIM.

4.1.3 CDB (Command Descriptor Block): A block of information containing the SCSI opcode, parameters, and control bits for that operation.

4.1.4 DMA (Direct Memory Access): A means of data transfer between peripheral and host memory without processor intervention.

4.1.5 Freeze: This defines a software action to quiesce activity e.g. freeze the queue.

4.1.6 HBA (Host Bus Adapter): The hardware and microcode which provides the interface between system memory and the SCSI bus.

4.1.7 Lock: This defines a hardware action e.g. data cartridge in a removable media drive.

4.1.8 Nexus: A block of information containing the SCSI device, LUN, and Queue Tag Number (if any, as used in command queuing).

4.1.9 Null: A value which indicates that the contents of a field have no meanigng. This value is typically, though not necessarily, zero.

4.1.10 Optional: This term describes features which are not required by the standard. However, if any feature defined by the standard is implemented, it shall be done in the same way as defined by the standard. Describing a feature as optional in the text is done to assist the reader. If there is a conflict between text and tables on a feature described as optional, the table shall be accepted as being correct.

4.1.11 Reserved: Where this term is used for bits, bytes, fields and code values; the bits, bytes, fields and code values are set aside for future standardization. The default value shall be zero. The originator is required to define a Reserved field or bit as zero, but the receiver should not check Reserved fields or bits for zero.

4.1.12 SCB (Subsystem Control Block): The term defined by IBM to refer to an architecture to support SCSI Host Adapters.

4.1.13 SCSI (Small Computer Systems Interface): The I/O interface which this standard is designed to support.

4.1.14 SIM (SCSI Interface Module): A module designed to accept the CAM Control Blocks routed through the XPT in order to execute SCSI commands.

4.1.15 VU (Vendor Unique): This term is used to describe bits, bytes, fields, code values and features which are not described in this standard, and may be used in a way that varies between vendors.

4.1.16 XPT (Transport): A layer of software which peripheral drivers use to originate the execution of CAM functions.

4.2 Conventions

Within the tables, there is a Direction bit which indicates In or Out. The presumption is from the view of the peripheral driver i.e. information is Out to the SIM from the peripheral driver and In to the peripheral driver from the SIM.

Certain terms used herein are the proper names of signals. These are printed in uppercase to avoid possible confusion with other uses of the same words; e.g., ATTENTION. Any lower-case uses of these words have the normal American-English meaning.

A number of conditions, commands, sequence parameters, events, English text, states or similar terms are printed with the first letter of each word in uppercase and the rest lower-case; e.g., In, Out, Request Status. Any lower-case uses of these words have the normal American-English meaning.

The American convention of numbering is used i.e., the thousands and higher multiples are separated by a comma and a period is used as the decimal point. This is equivalent to the ISO convention of a space and comma.

American:	ISO:
0.6	0,6
1,000	1 000
1,323,462.9	1 323 462,9

5. Background

5. Background

SCSI (Small Computer Systems Interface) is a peripheral interface designed to permit a wide variety of devices to coexist. These peripherals are typically, but not necessarily, attached to the host by a single SCSI HBA (Host Bus Adapter).

5.1 Software

OS (Operating System) support for peripheral devices is normally achieved through peripheral drivers or utility programs. No single driver or program can reasonably support all possible SCSI peripherals, so separate drivers are needed for each class of installed SCSI device. These drivers need to be able to share the SCSI HBA hardware.

These drivers also have to work with a broad range of HBA hardware, from highly intelligent coprocessors to the most primitive, including a SCSI chip on a motherboard.

A standard SCSI programming interface layer is essential to insulate SCSI peripheral drivers and utilities from the HBA hardware implementation, and to allow multiple drivers to share a single SCSI hardware interface.

5.2 CAM (Common Access Method)

This standard describes the general definition of the CAM (Common Access Method). CAM functionality has been separated into a few major elements.

5.2.1 XPT (Transport)

The XPT (Transport) defines a protocol for SCSI peripheral drivers and programs to submit I/O requests to the HBA specific SIM module(s). Routing of requests to the correct HBA and posting the results of a request back to the driver are capabilities of the Transport.

5.2.2 SIM (SCSI Interface Module)

The SIM (SCSI Interface Module) manages HBA resources and provides a hardware-independent interface for SCSI applications and drivers i.e. the SIM is responsible to process and execute SCSI requests, and manage the interface to the HBA hardware.

There are no requirements on how the SIM is implemented, in RAM (Random Access Memory) or ROM (Read Only Memory), provided the XPT is properly supported. A ROM-based SIM may need a transparent (to the user) software layer to match the SIM-required services to the specific manner in which they are requested of the OS.

5.2.3 CCB (CAM Control Block)

The CAM Control Block is a data structure passed from the peripheral driver to the XPT. The contents of the data structure describe the action required and provides the fields necessary for successful processing of a request.

5.2.4 OSD (Operating System Dependent)

The system environment in which the CAM is operating is a function of the hardware platform and the Operating System being executed e.g. the byte ordering is different between an Intel-based and a Motorola-based machine, and the calling structure differs

greatly between Operating Systems.

Although the fields of a CCB may have a common meaning, the contents will vary by platform and OS. These dependencies cause differences in operation and implementation, but do not prevent interoperation on the same platform of two CAM modules implemented by different manufacturers.

The OSD issues are predominantly described in the XPT for each OS environment.

5.3 Principles of Operation

Ideally, a single XPT model would suffice for all OS environments for a single HBA, but this is impractical in light of the wide architectural differences between the various processor architectures.

Programming effort has been minimized by making the interfaces as similar as possible across OS platforms, and customizing the SIM for each HBA to maximize performance under each OS. HBAs vary widely in the capability and functions they provide so there may be an internal (transparent) interface to isolate hardware interface routines from routines which make use of OS resources.

In order to prevent each peripheral driver from having to scan the SCSI bus for devices at initialization, the XPT determines all installed SCSI devices and constructs an internal table. A XPT function is used by drivers and programs to access this table.

Peripheral drivers need to be developed with documentation provided by the operating system vendor in addition to that supplied by this standard.

Under Unix, the XPT and SIM would typically be compiled with the kernel at System Generation time, so that entry points would be resolved during linkage-editing.

Third party attachments may be supported without the need for a sysgen if suitable routing facilities are provided by the system vendor.

Under Novell, the XPT is supplied by Novell, and the SIM is implemented according to Novell documentation guidelines.

Under DOS, there is one logical XPT with one entry point, but it may consist of a number of separate modules (perhaps supplied for each HBA in the system).

Routing is a mechanism to support concurrent SIM modules being co-resident so that different HBAs can be mixed in the same system. This may be handled by chaining the XPT entry points, defining additional character devices, or by a specific routing entity.

Once the SIM is loaded, the peripheral drivers integrate each type of SCSI device into the OS through XPT, independent of the installed HBA hardware.

Under OS/2 the equivalent to a XPT function is supplied by Microsoft, and the SIM is implemented according to Microsoft LADDR documentation guidelines.

5.4 Requirements

System requirements addressed in defining the CAM include:

a) Device drivers and programs should be able to use any SCSI command, both

defined in SCSI-2 X3.131-1990 or Vendor Unique.

b) No assumptions on the size and format of transferred data.

c) Allowing all the capabilities of high end host adapters to be fully utilized and accommodate HBAs which do most of the SCSI processing on board (this precludes interfaces which expect to control SCSI phases).

d) Interpretation of sense data returned by SCSI devices shall be by the calling driver or program.

e) Fully re-entrant code.

NOTE: This is an obvious requirement for multitasking environments such as OS/2 or Unix but even in single tasking DOS applications, multithreaded I/O is required to achieve maximum performance. SCSI devices such as printers, communication ports and LAN interfaces are often serviced in the background under DOS. If an HBA cannot support multithreading, requests can be queued and serialized within the SIM module transparently to the XPT.

f) Support of multiple HBAs.

g) If optional features are not supported in a minimum functionality XPT and SIM, peripheral drivers shall be provided a means to determine what features are available.

h) Providing an initialization service so that the process of identifying the attached SCSI devices need not be repeated by each peripheral driver which loads in the system.

i) Providing a mechanism to abort I/O threads (at request of peripheral driver).

j) Ability to issue multiple I/O requests from one or more peripheral drivers to a single Target/LUN.

k) Providing peripheral drivers with a mechanism for allocating a Sense data area and for specifying the number of Sense bytes to be automatically requested on a CHECK CONDITION.

6. Transport

(Part of **ANSI Common Access Method rev 2.3**)

6. Transport

6.1 Accessing the XPT

The OS peripheral drivers access the XPT through a software call to a single entry point. The method for obtaining and using the entry point differs between operating systems.

The XPT is not involved in the reverse process to advise the peripheral driver of the completion of a request. The completion callback permits a direct return from the SIM to the peripheral driver (the exact method employed in callback is Operating System dependent).

The XPT is responsible to notify peripheral drivers of asynchronous events via the Asynchronous Callback mechanism

6.2 Initialization

The XPT is responsible for determining the interface configuration at power up initialization for the SIM drivers. Depending on the Operating System, the XPT may perform a scan of the attached SCSI peripherals automatically. See also the SCSI-2 X3.131-1990 Annex on Power Up Considerations.

The scan by the XPT/SIM would follow a pattern such as the following:

```
for all SCSI buses
     for all target IDs (excluding the initiator)
          find the device
          if device exists
               for all LUN's
                    use Inquiry command and save returned information
               end for
          end if
     end for
end for
```

6.3 Callback on Completion

Callback on Completion refers to the XPT/SIM making a call to the routine addressed by the Callback on Completion pointer in the CCB. The callback is used by a peripheral driver in much the same manner as a hardware interrupt.

Callback routines have the same privileges and restrictions as hardware interrupt service routines.

The Callback on Completion routine is called to indicate that the Requested I/O is complete. The specific address of the CCB completed is passed to the callback routine.

6.4 SCSI Request Queues

Queues are used in systems where there is a need to manage many outstanding requests. There are various types of queues and each has different support needs.

A SCSI request queue can occur in the following places:

- o in the SIM
- o in the Target/LUN
- o in the peripheral driver

The SIM keeps a queue of all the CCB requests from the various peripheral drivers that access a LUN.

A SCSI device may be able to keep a large queue using Tag Queues, or a simple queue of one element.

A peripheral driver can also keep a queue e.g. a simple elevator sort, if the LUN does not support tagged queuing.

6.4.1 The Target/LUN and the Peripheral Driver

The peripheral driver is responsible for maintaining the queue(s) internal to the Target/LUN.

The SIM, acting on behalf of the peripheral driver, sends the appropriate commands or messages to manage the Target/LUN queue(s).

When the Target/LUN has completed an operation, the peripheral driver is advised by the SIM via a callback or by checking CAM status for completion.

The peripheral driver needs to be aware that there may be other peripheral drivers and other systems working with the same Target/LUN.

6.4.2 The SIM

The SIM maintains a queue for each LUN which is logically shared by all peripheral drivers. The queue may support tagged commands. Queue priority shall be supported.

6.4.3 SIM Queuing

6.4.3.1 SIM Queue Priority

When SIM Queue Priority=1, the SIM places the CCB at the head of the queue for the LUN, instead of at the end. One use of this CAM flag is during error handling. If the queue is frozen and a CCB with SIM Queue Priority=1 is received, the CCB shall be placed at the head of the queue and the queue remains frozen. When the SIM queue is released, any CCBs with SIM Queue Priority=1 are executed atomically, and in LIFO sequence.

To force step-by-step execution, the peripheral driver can set SIM Queue Freeze=1, so that when the queue is released and a CCB with SIM Queue Priority=1 is executed, the queue is re-frozen by the SIM at completion.

6.4.3.2 Tag Recognition

To support tagged queueing recognition the SIM maintains a reference between the CCB pointers and the Queue Tags for a LUN. By this means, the SIM can handle both the queue tag resource allocation and reconnection of the I_T_L_Q nexus (see SCSI-2 X3.131-1990) for the CCB from a peripheral driver.

The peripheral driver is required to allow the SIM/XPT to handle the assignment of the queue tag ID for the request. The SIM assigns unique TAG IDs to the Target/LUN operation based on

its internal reference table.

When a LUN that supports tagged queuing reconnects to the Initiator (SIM/HBA pair), it will send the SIMPLE QUEUE TAG message with the queue tag value for the I_T_L_Q nexus. Using the returned queue tag ID, the SIM restores what is necessary to complete the SCSI transaction. The queue tag ID is freed by the SIM at the completion of the SCSI request.

6.4.3.3 Error conditions and Queues within the Subsystem

The SIM shall place its internal queue for a LUN into the frozen state for any status other than Request Completed Without Error and Request in Progress. When a LUN's queue is in the frozen state, the SIM shall not dispatch any CCBs to that LUN. Peripheral drivers can still send CCBs to the SIM for the LUN, or any other LUN. Any new CCBs received by the SIM shall be placed at the end of the queue, unless SIM Queue Priority=1 forces them to the head,

Following a Check Condition or Command Terminated status, the target's LUN queue is also frozen, and all other tagged commands stay in the queue until the allegiance condition is cleared. The allegiance condition is either cleared by an incoming command or following the return of sense data for the same initiator.

Since the SIM is the initiator, the SIM's internal queue shall go into a frozen state so that the pending sense information in the LUN will not be discarded. The SIM holds it's internal LUN queue in the frozen state until a Release SIM Queue CCB is received.

Using the Callback on Completion pointer contained in the CCB the SIM returns control of the CCB to the peripheral driver along with CAM Status indicating the frozen queue condition and other information.

The peripheral driver acts upon the information returned via the CCB. In the event that there is not a valid pointer in the callback field, the peripheral driver that originated the CCB shall retain responsibility for the CCB by watching the CAM Status field. The setting of the Autosense bit in the CAM flags does not affect how the SIM handles freezing the SIM's internal queue i.e. the Request Sense command issued by the SIM to recover status for Autosense does not release the SIM queue.

If the peripheral driver has to perform recovery with the LUN, a CCB can be placed at the head of the queue by setting SIM Queue Priority=1, and the SIM queue released. If the peripheral driver has other pending CCBs in the queue which it does not want to be sent to the LUN (depending on the cause of the Check Condition), then it can use a CAM Flag to freeze the queue upon completion of the CCB at the head of the queue. A SIM may reject a CCB with SIM Queue Freeze=1 if the queue is not frozen at the time the CCB is received.

6.5 SIM Handling of SCSI Resets

The CAM shall not define support for the "Soft Reset" SCSI option, but implementors may use the services of the SIM to provide vendor-specific support.

Following a SCSI Bus Reset, the SIM shall:

```
a) Block Path IDs to the reset bus i.e. new CCBs are rejected with status of
   CAM Busy.
b) Return all outstanding CCBs with status of SCSI Reset.
c) Unblock all Path IDs for the bus.
d) Call: xpt_async(opcode=reset,
                   path_id=bus that was reset,
```

```
                    target_id=-1,
                    lun=-1,
                    buffer_ptr=null,
                    data_cnt=0
e) Resume normal processing of CCBs.
```

6.6 Asynchronous Callback

In an event such as a SCSI Bus Reset or an Asynchronous Event Notification the XPT has to be able to make a callback to the peripheral driver(s), even though there may be no CCBs active for the peripheral driver(s).

Callback routines have the same privileges and restrictions as hardware interrupt service routines.

During system startup and driver initialization, the peripheral driver should register an Asynchronous Callback routine for all the SCSI devices with which it is working. In order for a peripheral driver to receive asynchronous callbacks, it shall issue a Set Asynchronous Callback CCB with the Asynchronous Event fields set to 1 for those events the peripheral driver wishes to be notified of through an asynchronous callback. The peripheral driver is required to explicitly register for the path IDs, targets, and LUNs. The use of a wildcard is not supported for the Set Asynchronous Callback CCB.

It is required that the Asynchronous Callback field be filled in with the callback routine address if any of the Asynchronous Events Enabled bits are set. The peripheral driver can de-register its Asynchronous Callback for a particular SCSI device by issuing the Set Asynchronous Callback CCB with the Events field cleared to zero and the Callback pointer containing the Callback Routine address of the peripheral driver issuing the request. All XPTs must provide the capability for any SIM to support asynchronous callback, but a given SIM does not have to support each (or any) of the Asynchronous Events Enabled bits.

Upon detection of a supported enabled event, the SIM shall do the following once for each detected event:

```
a) Classify the event:  determine the opcode which is the same as the encoded
   bit number of the Asynchronous Events Enabled.
b) Format the associated data within an internal, to the SIM, local buffer,
   e.g. the sense data received from an AEN.
   NOTE: This is a multiple processor "lock" point.
c) Perform the XPT reverse routing required by the event. The SIM will call
   the Async Callback entry point in the XPT:

   long xpt_async(opcode, path_id, target_id, lun, buffer_ptr, data_cnt)
```

All of the arguments, other than the pointer, are long values of 32 bits. The value of -1 in Path, Target and LUN can be used as a wild card. A null buffer pointer value and a count of 0 are valid for opcodes that do not require any data transfer.

NOTE: This call to the XPT is a multiple processor "lock" point.

Using the Path ID, Target, and LUN information from the xpt_async() call, the XPT scans its internal tables looking for "matches" with what the peripheral drivers had registered for using the Set Async Callback CCB (see 8.2.4). When a match is found, either exactly or with a wild card of "-1," the XPT shall copy the data for the opcode, if available, into the area reserved by the peripheral driver and then call the peripheral driver's Async Callback routine.

The arguments to the peripheral driver's Async Callback routine are the same as the xpt_async() routine though:

- the buffer_ptr value shall be the peripheral driver's buffer
- the data_cnt shall either be what the XPT had to transfer from the SIM's buffer or the limit of the peripheral driver's buffer.

Almost all of the information relating to the different opcodes can be included in the Path ID, Target and LUN arguments. The only opcodes that require an additional buffer area are AEN, Load SIM and Unload SIM. Table 6-1 lists the opcodes and the expected data requirements for the number of bytes to be transferred.

TABLE 6-1 ASYNC CALLBACK OPCODE DATA REQUIREMENTS

Event	Opcode	Path ID	Target	LUN	Data Cnt
Unsol. SCSI Bus Reset	0x0001	Valid	n/a	n/a	n/a
Unsol. Reselection	0x0002	Valid	Valid	Valid	n/a
reserved	0x0004				
SCSI AEN	0x0008	Valid	Valid	Valid	Min. 22
Sent BDR to Target	0x0010	Valid	Valid	n/a	n/a
SIM Module Loaded	0x0020	XPT ID	n/a	n/a	Min. 1
SIM Module Unloaded	0x0040	XPT ID	n/a	n/a	Min. 1
New Devices Found	0x0080	Valid	n/a	n/a	n/a

The AEN data requirements are a minimum of 22 bytes of buffer space. This space includes the 4 bytes required by the AEN Data Format and 18 bytes defined by the Sense Data Format (see SCSI-2 X3.131-1990).

The Load SIM and Unload SIM data requirements are a minimum of 1 byte. This byte contains the Path ID for the SIM. This Path ID is different that the path_id argument. The argument contains the unique XPT ID of 0xFF. The XPT ID is the ID used by the peripheral driver to register for async notification.

If there is valid data placed in the generic data buffer by the XPT/SIM, the peripheral driver is required to save or discard that data before returning control to the XPT/SIM.

6.7 Autosense

Autosense causes sense data to be retrieved automatically if a Check Condition is reported in the SCSI Status field. On a Check Condition, a SCSI Request Sense command is constructed and sent to the same target. The location and amount of the Sense data is specified in the Sense Info Buffer Pointer and Length fields respectively of the SCSI I/O Request CCB. If the length field is 0 or the buffer field is Null, the Request Sense command shall still be issued, but with a data allocation length of 0 (this should only be done by the peripheral driver when it is not interested in the sense information).

After completing the Request Sense sequence the CAM Status and SCSI Status fields contain the status of the original command (which caused the Check Condition).

The target can return fewer than the number of Sense bytes requested. This is not reported as an error, and Sense Status shall be flagged as valid.

6.8 Loadable Modules

Some operating system environments provide the ability to load or unload software drivers, thus peripheral drivers or SIM modules can be loaded dynamically. In such systems, the XPT module (typically supplied by the OS vendor) is either part of the system or must be loaded first.

The XPT, as part of a loadable OS, exports it's "label," which is to used as a reference by the other loadable modules. The XPT manages the loading of SIMs and provides the common access point for peripheral drivers to register a loaded or unloaded SIM.

When a peripheral driver is loaded, it can go through it's initialization process (see OSD initialization), call the XPT initalization point and then query the XPT for the HBAs that are present in the system and targets that have been identified as being on the SCSI channels.

When a SIM is loaded, the SIM and XPT have to work together to get the SIM entered into the internal tables and have the SIM initialized.

The SIM shall call the XPT once for each supported bus in order to obtain the Path ID for that bus.

long xpt_bus_register(CAM_SIM_ENTRY *)

The argument is the pointer for the data structure defining the entry points for the SIM. The value returned is the assigned Path ID; a value of -1 indicates that registration was not successful.

The SIM shall call the XPT once to de-register the bus for a given Path ID:

long xpt_bus_deregister(path_id)

The argument is the Path ID for the bus being de-registered. A return value of zero indicates the bus is no longer registered, any other value indicates the call was unsuccessful.

When the XPT is called it will update it's internal tables and then call the sim_init(path_id) function pointed to by the CAM_SIM_ENTRY structure. The initialization for the loaded SIM is no different than for a SIM statically included in the kernel at boot time. After the SIM has gone through the initialization process the XPT shall scan the SCSI bus in order to update its internal tables containing Inquiry information.

Peripheral drivers can request to be informed when a SIM is registered or de-registered via the Async Callback feature (see 6.6 and 8.2.4).

The CAM_SIM_ENTRY table is used to define the entry points for the SIMs.

```
typedef struct

   long (*sim_init)();      /* pointer to the SIM init routine */
   long (*sim_action)();    /* pointer to the SIM CCB go routine */
 CAM_SIM_ENTRY;
```

7. OSD (Operating System Dependent) Operation

7. OSD (Operating System Dependent) Operation

7.1 UNIX Operating System

The CAM subsystem is intended to provide a set of services for third-party vendors.

There are several sets of modules for Unix:

- peripheral drivers that are device class specific
- a configuration_driver for initialization
- the XPT
- SIMs that are HBA-specific

Each member of these sets is treated as a UNIX driver and is linked into the kernel. The XPT and configuration_driver (which is responsible for initialization) are OS-vendor specific; other drivers may come from any source.

At kernel configuration and link time the cam_conftbl[] is created and contains entry points for the SIMs, which are used by the XPT.

The cam_conftbl[] is used by the XPT/configuration_driver to call routines and pass CAM parameters between them e.g. the Path ID contained in the CCB created by the peripheral driver is used to index into the cam_conftbl[]. The entry point for the selected SIM, sim_action() is called with a pointer to the CCB as an argument.

The cam_edt[] data structure is used and created during the initialization process to contain the necessary information of all the targets found on all the HBAs during the init sequence.

The CAM Flags used are as described in Table 9-2.

7.1.1 Initialization

The initialization of the XPT and SIMs is under the control of the configuration_driver.

Due to the different Unix-based systems (BSD and System V), there is no common initialization process that can control the order of calls to the peripheral driver's and configuration_driver's init() routines. It is necessary to make sure that the subsystem is initialized before any requests can be serviced from the peripheral drivers. Due to this constraint when the peripheral driver's initialization routines are called the driver shall call the xpt_init() routine. If the subsystem is not yet initialized, the XPT shall call the configuration_driver to formally initialize the subsystem. Once the subsystem is set up, either from a previous xpt_init call or the configuration_driver being called, all subsequent xpt_init calls shall simply return.

When the configuration_driver is called for initialization, it uses the cam_conftbl[] entry structures. The configuration_driver makes the init() routine calls, to the XPT, and to each SIM in turn, allowing them to initialize. The initialization routine for the SIM is called with its Path ID as the argument. Interrupts shall be disabled or blocked by the configuration_driver during the initialization process.

After the initialization process has been completed, the configuration_driver obtains information about each SIM, HBA, and target device detected, and maintains a table, the cam_edt[], of these devices. The information is obtained by using CCBs through the CAM

interface.

Once the CAM subsystem is initialized and the cam_edt[] set, the peripheral drivers can use the subsystem. This allows them to determine what devices are known and make appropriate memory allocations and resource requests of the XPT.

The SCSI-2 Inquiry command shall be issued to all Target/LUNs on the attached interfaces, and shall contain an allocation length of 36 bytes, which is sufficient to transfer the device information and the product information. The EVPD and Page code fields in the Inquiry command shall be set to 0. It is assumed that the responding devices will return the Inquiry data, even though the device may not be ready for other commands. A limited number of retries will be done for devices that return Busy Status following the Inquiry command. If the retry limit is reached, the status of the device in the XPT will be set to "Not Found". The Inquiry command shall be the only command issued by the XPT to the devices during initialization.

7.1.2 Accessing the XPT

7.1.2.1 From the Peripheral Driver

The XPT provides functions to obtain CAM system resources for the peripheral driver. These functions are used to allocate and free CCB resources and to allocate and free DMA resources.

There are two routines used in the handling the CCB resources. The two routines are:

```
  CCB *xpt_ccb_alloc() and
  void xpt_ccb_free(CCB *):

 - The xpt_ccb_alloc() routine returns a pointer to the allocated CCB. The
peripheral driver can now use this CCB for it's SCSI/XPT requests.
 - The xpt_ccb_free() routine takes a pointer to the CCB that the
peripheral driver has finished with, and can now be returned to the CAM
subsystem CCB pool.
 - The pointer to the CCB returned from the xpt_ccb_alloc() call shall be
large enough to contain any of the possible XPT/SIM function request CCBs.
 - The CCB can only be used i.e. sent to the XPT, once. Once the CCB has
completed it shall be returned using the xpt_ccb_free() routine.
```

All returned status information is obtained at the callback point via the CAM and SCSI status fields.

7.1.2.2 From the SIM

The SIMs obtain requests from the XPT as they are passed across from the peripheral driver, via a routine included in the SIM's configuration information. The field in the configuration table is declared as "void (* sim_action)(CCB *)." The XPT does not modify CCBs or CDBs. The XPT shall intercept those CCBs which must be redirected to the configuration_driver (Get Device Type, and Set Device Type).

7.1.3 Callback on Completion

The Callback on Completion field in the CCB is a structure that is platform specific, but always contains at least a callback function pointer, named cbfcnp, and declared as "void (*cbfcnp)(CCB *)." The argument to cbfcnp shall be the address to the CCB.

The Disable Callback on Completion feature is not supported.

7.1.4 Pointer Definition in the UNIX Environment

Pointers in the CAM environment are treated as any other pointer in a given UNIX implementation. For the 80386 platforms, pointers are 32-bit virtual addresses into a flat address space.

7.1.5 Request Mapping Information

This field is expected to contain a pointer to the buf structure that the SCSI I/O CCB was created for. This copy of the buf structure pointer, bp, is used by the SIM to get to the I/O mapping information needed to access the data buffers allocated by the application program. A value of NULL is allowed if there is no need for the SIM to map the data buffer addresses i.e. data count is zero, the buffer is internal to the kernel, or the addresses are physical.

7.1.6 XPT Interface

The XPT interface provides functions that peripheral drivers and SIM modules can access in order to transfer information and process user requests. The following defines the entry points, and describes the required arguments and return values.

7.1.6.1 Functions for Peripheral Driver Support

a) long xpt_init()

This routine is called by the peripheral driver to request that the XPT and sub-layers be initialized. Once the sub-layers are initialized any subsequent calls by other peripheral drivers shall quickly return.

There are no arguments and the return code is either Success or Failure.

b) CCB *xpt_ccb_alloc()

This routine is used whenever a peripheral driver needs a CCB (the common data structure for processing SCSI requests). It returns a pointer to the allocated CCB which the peripheral driver can now use as the CCB for it's SCSI/XPT requests. The returned CCB shall be properly initialized for use as a SCSI I/O Request CCB. The SIM Private Data area shall have been already set up to be used by the XPT and SIM, and shall not be modified by the peripheral driver. It is recommended that the CCB be returned to the XPT following its use, and that CCBs not be re-used.

There are no arguments and the return value is a pointer to an initialized CCB.

c) void xpt_ccb_free(CCB *)

This routine takes a pointer to the CCB that the peripheral driver has finished with so it can be returned to the CAM subsystem CCB pool.

The argument is the pointer to the CCB to be freed, there is no return code.

d) long xpt_action(CCB *)

All CAM/SCSI CCB requests to the XPT/SIM are placed through this function call. All returned CAM status information is obtained at the callback point via the CAM and SCSI status fields.

The argument is a pointer to the CCB, and the return code is either Success or Failure.

7.1.6.2 Functions for SIM Module Support

a) See 6.8 for loadable module support:

long xpt_bus_register(CAM_SIM_ENTRY *)

long xpt_bus_deregister(path_id)

b) long xpt_async(opcode, path_id, target_id, lun, buffer_ptr, data_cnt)

The SIM calls this routine to inform the XPT that an async event has occured and that there may be peripheral drivers which need to be informed.

- The opcode, path_id, target_id, lun, and data_cnt arguments are long 32-bit values.
- The path_id, target_id, and lun define a nexus for the Async Callback.
- The opcode contains the value for what has happened.
- The buffer_ptr and data_cnt are used to inform the XPT where and how much data is associated with the opcode.

The return code is either Success or Failure.

7.1.7 SIM Interface

The SIM interface provides functions to the XPT, and should never be accessed directly by the peripheral driver. Each vendor's SIM should provide a publicly-defined entry structure such as CAM_SIM_ENTRY cse_vendorname.

The following defines the entry points, and describes the required arguments and return values.

a) long sim_init(pathid)

This routine is called by the XPT to request that the SIM be initialized. There are no arguments and the return code is either Success or Failure.

b) long sim_action(CCB *)

All CCB requests to the SIM are placed through this function call. All returned CAM status information is obtained at the callback point via the CAM and SCSI status fields.

The argument is a pointer to the CCB, and the return code is either Success or Failure.

7.2 Novell Operating System

Novell NetWare 386 drivers are called NLMs (NetWare Loadable Modules). These modules are registered and linked dynamically with NetWare 386: they are registered after the server is running and may be unloaded at any time.

The NetWare 386 CAM subsystem consists of 3 sets of NLMs:

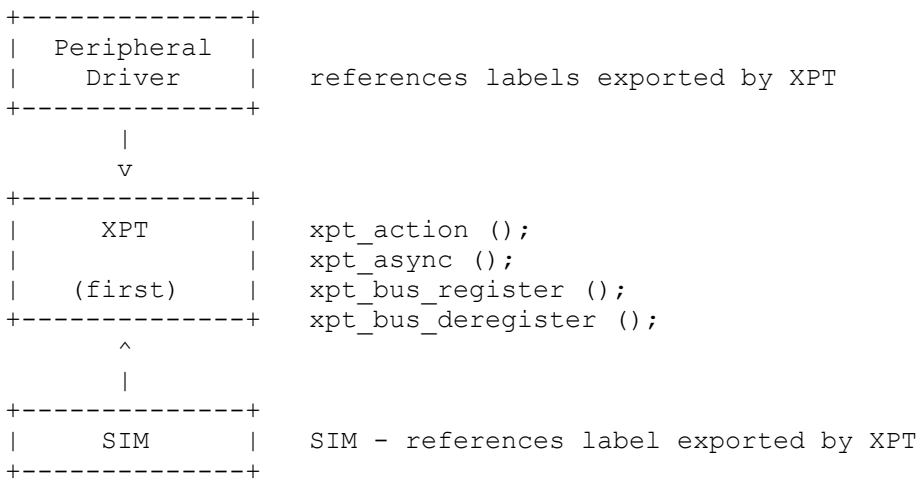
- peripheral drivers (NLMs) that are device class specific
- the XPT router and SIM maintenance NLM
- SIM NLMs that are HBA-specific

The peripheral drivers and SIMs communicate with the XPT through labels exported by the XPT when it is registered.

The CAM Flags used are as described in Table 9-2.

7.2.1 Initialization

As the Novell dynamic linker will not allow an NLM to register if it makes references to a label it cannot resolve, the order in which the NLMs register is important. The XPT module exports four entry points when it is registered, and both peripheral drivers and SIM modules make references to them. The XPT shall be registered first, after which either peripheral drivers or SIMs may be registered.



For an overview of SIM registration with the XPT see 6.8. For an overview of peripheral driver registration with the XPT see 6.6 and 8.2.4.

When NetWare 386 loads a SIM, it shall call the initialization routine specified in the Novell linker definition file. At this point the SIM can perform its initialization functions.

As part of initialization the SIM shall call the xpt_bus_register function once for each HBA it will support, to register the address of its entry point with the XPT and to get a path ID for each HBA from the XPT. The XPT then adds this SIM to its internal tables so it can route requests to the new SIM. The XPT also notifies all peripheral drivers that registered an asynchronous callback routine with the XPT (with the SIM Module Registered bit set), that a new path ID exists. Upon receiving this message the peripheral drivers can check for new devices on this path.

When NetWare 386 loads a peripheral driver, the initialization routine specified in the linker definition file shall be called. At this time,the driver needs to determine which, if any, SIMs are registered.

The peripheral driver sends a Path Inquiry CCB to each path to determine if a SIM is registered. If a valid response is returned the peripheral driver checks for devices that it will

support on that path. If the peripheral driver supports any devices on this path, it shall register an asynchronous callback routine and specify the SIM registration in the opcode field so that if the SIM is de-regitered, the peripheral driver shall be notified. In addition, a peripheral driver should also register for SIM registration to alert the driver of the need to locate devices on a newly added SIM module.

7.2.2 De-Registration

When a SIM de-registers, it shall call the xpt_bus_deregister() function once for each path the SIM supports. The XPT then calls every peripheral driver that has registered an asynchronous callback routine with the SIM Module De-Registered bit set on this path. Peripheral drivers then notify NetWare 386 that the drives on this path are in an inactive state. The XPT will then remove the path from its internal tables and block further peripheral driver requests on this path.

If a peripheral driver de-registers, it needs to notify the XPT module so that the dependency tables can be updated. This is done by registering an asynchronous callback routine with the opcode set to zero. The XPT will then remove this driver from its callback tables.

The XPT can only be unloaded after all peripheral drivers and SIM modules have been de-registered. NetWare 386 will not allow an NLM to unload if it has exported labels that other NLMs are using. As all SIM and peripheral drivers refer to labels exported by the XPT, NetWare 386 will not allow the XPT to unload until all the SIMs and peripheral drivers have been unloaded, at which point there is nothing left for the XPT to support and it can be safely de-registered.

7.2.3 Accessing the XPT

NetWare 386 allows an NLM to export functions which NLMs registered at a later time can reference. An NLM calls an exported function in the same way it calls any other function. The C language calling convention is assumed. In order for communication between the peripheral drivers, XPT, and SIM modules to work correctly the names of the XPT entry points have to be constant.

The entry points in the XPT module are:

- `xpt_action ()` accepts CAM blocks from the peripheral driver and routes them to the correct SIM
- `xpt_async ()` is used by the SIM module to notify the XPT when an asynchronous event occurs.
- `xpt_bus_register ()` is used to register the SIM with the XPT and obtain a Path ID.
- `xpt_bus_deregister ()` is used to unload the SIM associated with the Path ID.

7.2.4 Hardware Registration

The SIM module needs to do the actual registration of the host adapter with NetWare. Since only one SIM may support a given host adapter this prevents any hardware options from being registered twice. The SIM does not register any devices with NetWare, only the hardware options used by the card e.g. interrupt line, base address, DMA etc.

Interrupts generated by the host adapter will be handled by the SIM module, so the SIM must also register its interrupt service routine with NetWare.

A peripheral driver registers a logical card with NetWare 386 for each path_id it supports.

This logical card uses no hardware resources, but does have entry points IO and IOCTL requests from NetWare. The peripheral driver also reports the devices that it will support to NetWare.

The XPT does not register any hardware or devices with NetWare 386. It loadsas a driver, but does not register any IOPOLL or IOCTL entry points.

7.2.5 Miscellaneous

It is the responsibility of the peripheral driver to allocate memory for its CCB blocks. Normally the peripheral driver needs to keep one CCB structure for each device it will support, so the memory can be allocated in the DiskStructure provided by NetWare 386 when a device is added to the system.

Since fast disk channels are essential for a NetWare 386 server, peripheral drivers should never poll the CAM status field to wait for completion. The driver should send the CCB to the XPT module and then either do more work, or exit immediately. The SIM module will call the function whose address is in the callback field of the CCB block when the request is finished. The callback function runs at interrupt level, so it cannot call any NetWare 386 routines that are "blocking" or the file server will abend. See the Novell Disk Driver manual for details on blocking and non-blocking levels.

7.3 DOS (Disk Operating System)

Under DOS, a software interrupt is used to access any of the XPT or SIM functions, which are combined into a single module.

The routing functions of the XPT are performed by the DOS concept of "interrupt vector chaining." During execution, an XPT/SIM module determines if a particular CCB is one that it should handle. If not, it routes the CCB to the previous "owner" of the interrupt vector.

The CAM flags used by the DOS XPT/SIM are described in Table 9-2.

7.3.1 Initialization

During initialization, the XPT/SIM modules should be loaded as character device drivers.

As character device drivers are required by DOS to have unique names, the 8-character device name should be "$$CAMxxx", where xxx is the ASCII decimal numeric value of the lowest path ID supported by this XPT/SIM module.

The programming examples in this clause are used to assist the reader's understanding. Implementations do not need to use the same code, but they are required to accomplish the same goals.

7.3.1.1 Multiple XPTs

The pseudocode for the XPT initialization sequence is as follows:

```
Get INT 4Fh interrupt vector;
Save this address for chaining;
IF there is a CAM XPT already installed (see 7.3.2.1)
     Perform PATH INQUIRY (Path ID=0FFh) to get Highest Path ID;
     First Path ID = Highest Path ID + 1;
ELSE
```

```
          First Path ID = 0;
     END IF;
     Count number of Path IDs needed;
     IF no HBAs to support (Count = 0)
          Exit initialization without installing driver;
     END IF;
     Set INT 4Fh interrupt vector to point to CAM entry point;
     Save Highest Path ID used (First Path ID + Count - 1);
     Set character device name to "$$CAMxxx",
          where xxx=First Path ID;
     Perform all necessary HBA initialization;
     FOR each SCSI Bus supported:
          FOR each SCSI ID (excluding initiator)
               IF device exists
                    FOR each LUN
                         Perform INQUIRY to get PDT for table;
                    END FOR;
               END IF;
          END FOR;
     END FOR;
```

7.3.1.2 Device Table Handling

The XPT/SIM is only required to keep the peripheral device type of the devices connected to the supported SCSI bus(es).

7.3.2 Accessing the XPT

There are various mechanisms used to access XPT or SIM functions from peripheral drivers or application programs.

7.3.2.1 Testing for the presence of the XPT/SIM

Peripheral drivers and applications can check for the presence of an XPT/SIM module by performing a "check install" function such as:

On entry:
- AX = 8200h
- CX = 8765h
- DX = CBA9h

On return:
- AH = 0 (if successful)
- CX = 9ABCh
- DX = 5678h
- ES:DI = address of character string "SCSI_CAM"
- All other registers unaffected.

The following routine checks for the presence of an XPT/SIM module. It returns a value of 1 if a module is found and a value of 0 if not found.

```
CHK_FOR_CAM     PROC    NEAR
                MOV     CX,8765H            ; load l.s.w. of signature
                MOV     DX,0CBA9H           ; load m.s.w. of signature
                MOV     AX,8200H            ; load "check install" code
                INT     4FH                 ; perform "check install"
```

```
                CMP     AH,0                    ; function supported?
                JNE     NOT_THERE               ; if not, no xpt/sim
                CMP     DX,5678H                ; check m.s.w. of signature
                JNE     NOT_THERE               ; if invalid, no xpt/sim
                CMP     CX,9ABCH                ; check l.s.w. of signature
                JNE     NOT_THERE               ; if invalid, no xpt/sim
                CLD                             ; set direction flag
                MOV     CX,8                    ; load string length
                MOV     SI,OFFSET SCSI_CAM      ; get string address
           REPE CMPSB                           ; compare strings
                JNE     NOT_THERE               ; if strings differ, no xpt/sim
                MOV     AX,1                    ; load "found" status
                RET                             ; return to caller
NOT_THERE:      MOV     AX,0                    ; load "not found" status
                RET                             ; return to caller
CHK_FOR_CAM     ENDP
SCSI_CAM        DB      'SCSI_CAM'              ; string to find
```

7.3.2.2 Sending a CCB to the XPT

Once it is determined that an XPT/SIM module is present, the peripheral driver or application can access the XPT/SIM functions by sending a CCB to the XPT/SIM:

On entry:
ES:BX = address of the CCB
AX = 8100h

On return:
AH = 0 if successful
= 1 if invalid CCB address (segment=offset=0)
All other registers unaffected.

NOTE: The SIM may complete and return control to the location pointed to by the Callback on Completion field in the CCB before the software interrupt returns.

The following routine sends a CCB to the XPT/SIM module. It returns a value of 0 if successful and 1 if not.

```
SEND_CCB        PROC    NEAR
                MOV     AX,8100H                ; load "send ccb" function
                MOV     ES,SEGMENT CCB          ; load segment of ccb
                MOV     BX,OFFSET CCB           ; load offset of ccb
                INT     4FH                     ; call xpt/sim module
                SHR     AX,8                    ; put return code in al
                RET                             ; return to caller
SEND_CCB        ENDP
```

7.3.3 Callback on Completion

When an I/O operation has completed, a XPT/SIM module shall make a FAR call to the routine which had its address passed in the Callback on Completion field of the CCB. The first 4 bytes of this field are used to indicate the routine's address in the Intel Segment:Offset format. When the callback is made, hardware interrupts shall be disabled and ES:BX shall point to the completed CCB.

7.3.4 Asynchronous Callbacks

There are some differences in the DOS XPT/SIM implementation of Asynchronous Callbacks as compared with the description in 6.6.

The DOS XPT/SIM does not support the SIM Module Loaded and SIM Module Unloaded opcodes reported by the XPT/SIM module when the Asynchronous Callback Routine is called.

The Set Async Callback CCB is held by the XPT/SIM until it is "de-registered." This is accomplished by sending another Set Async Callback CCB to the XPT/SIM with all of the Asynchronous Event Enables reset and the address of the original Set Async Callback CCB in the Peripheral Driver Buffer Pointer field. At that point the original CCB shall be dequeued and both CCBs shall be returned to the peripheral driver or application.

> NOTE: There is an implication here that a peripheral driver or application which wishes to be notified when the specified asynchronous event occurs, has to register separately with each path ID.

The Peripheral Driver Buffer Pointer and Size of Allocated Peripheral Buffer fields in the Set Async Callback CCB are considered as Private Data by the XPT/SIM, to be used for CCB queuing.

When an Asynchronous event occurs that is enabled by the bits in the Asynchronous Event Enables field of the Set Async Callback CCB, the virtual address specified by the Asynchronous Callback Pointer field shall be called with the following registers:

On entry:

AH = opcode as specified in Table 6-1.
AL = path ID that generated the callback.
DH = target ID that caused event (if applicable).
DL = LUN that caused event (if applicable).
CX = data byte count (if applicable).
ES:BX = address of data buffer (if applicable).

On return:

All registers shall be preserved.

It is the responsibility of the peripheral driver or application to copy any or all required data out of the data buffer into a local buffer before returning from the Asynchronous Callback routine.

7.3.5 Pointer Definition

All pointers shall be passed to the XPT/SIM as segment:offset type virtual addresses.

7.4 OS/2 (Operating System 2)

Microsoft has documented LADDR as a generic I/O interface which supports many device interfaces, not only SCSI. The control blocks and their method of operation are defined in OS/2 Technical Reference Manuals which are available from Microsoft.

The OS/2 equivalent to the SIM is a BID (Bus Interface Driver).

The OS/2 equivalent to the CCB is an SRB (SCSI Request Block).

The CCB and the SRB share many common fields. The fields in the CCB are designated as OSD if they vary between OS/2 and other operating systems.

For further information on how peripheral drivers use the CCB/SRB and other SIM/BID capabilities of OS/2, it is necessary to use information available from Microsoft.

The CAM Flags used are as described by LADDR documentation.

8. CAM Control Blocks

8. CAM Control Blocks

The CCBs used by drivers and applications to request functions of the XPT and SIM have a common header, as shown in Table 8-1.

TABLE 8-1 CAM CONTROL BLOCK HEADER

Size	Dir	
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)

The sequence of the fields in the data structures will be consistent between vendors, but not necessarily the binary contents. The size and definition of the fields in the data structures can vary beween operating systems and hardware platforms, but the vendors are expected to provide compiler definitions which can be used by third-party attachments.

Several fields in the CCB are pointers, and their meaning is dependent on the OS which is being supported. In general, these pointers are interpreted as either virtual or physical addresses.

Additional bytes beyond the CCB Header are dependent on the Function Code.

Most SCSI messages are handled transparently by the SIM, but in some cases, the peripheral driver has been given the ability to force the SIM to issue a message. Table 8-2 summarizes the message support.

TABLE 8-2 SUPPORT OF SCSI MESSAGES

Abort	Discretely supported by function codes
Abort Tag	Discretely supported by function codes
Bus Device Reset	Discretely supported by function codes
Clear Queue	Not Supported
Command Complete	Transparently supported by SIM
Disconnect	Transparently supported by SIM *
Identify	Transparently supported by SIM
Ignore Wide Residue	Transparently supported by SIM
Initiate Recovery	Not Supported
Initiator Detected Error	Transparently supported by SIM
Linked Command Complete	Transparently supported by SIM
Message Parity Error	Transparently supported by SIM
Message Reject	Transparently supported by SIM
Modify Data Pointer	Transparently supported by SIM
No Operation	Transparently supported by SIM

Queue Tag Messages	
Head of Queue Tag	Discretely supported by function codes
Ordered Queue Tag	Discretely supported by function codes
Simple Queue Tag	Discretely supported by function codes
Release Recovery	Not Supported
Restore Pointers	Transparently supported by SIM
Save Data Pointers	Transparently supported by SIM
Synch Data Transfer Request	Transparently supported by SIM *
Terminate I/O Process	Discretely supported by function codes
Wide Data Transfer Request	Transparently supported by SIM

* Issuing this message influenced by peripheral driver via CAM flags

8.1 CCB Header

The Function Codes used to identify the XPT service being requested are listed in Table 8-3.

8.1.1 CAM Control Block Length

See **9.1.1.**

8.1.2 XPT Function Code

TABLE 8-3 XPT FUNCTION CODES

Code	
00-0F	Common Functions
00h	NOP
01h	Execute SCSI I/O (see 9.x)
02h	Get Device Type
03h	Path Inquiry
04h	Release SIM Queue
05h	Set Async Callback
06h	Set Device Type
07-0F	reserved
10-1F	SCSI Control Functions
10h	Abort SCSI command
11h	Reset SCSI Bus
12h	Reset SCSI Device
13h	Terminate I/O Process
14-1F	reserved
20h	Engine Inquiry (see 11.x)
21h	Execute Engine Request
22-2F	reserved
30-3F	Target Mode (see 10.x)
30h	Enable LUN
31h	Execute Target I/O
32-3F	reserved
40-7F	reserved
80-FF	Vendor Unique

If a Function Code which is not supported is issued to the XPT, the XPT shall complete the request and post CAM Status of Invalid Request.

8.1.3 CAM Status

See **9.1.3**.

8.1.4 Path ID

See **9.1.4**.

8.1.5 CAM Flags

The CAM Flags qualify the Function to be executed, and vary by Function Code.
See **9.1.5**.

8.2 Function Codes

8.2.1 Get Device Type

This function is executed at driver initialization in order to identify the targets they are intended to support e.g. A CD ROM driver can scan each Target/LUN address on each installed HBA to look for the CD ROM device type.

TABLE 8-4 GET DEVICE TYPE CCB

Size	Dir	Get Device Type
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)
4	O	Inquiry Data Pointer
1	I	Peripheral Device Type of Target/LUN

The information on attached SCSI devices is gathered at power on by the XPT (to eliminate the need for each driver to duplicate the effort of scanning the SCSI bus for devices).

The Peripheral Device Type is a 1-byte representation of Byte 0 of SCSI Inquiry Data i.e. bits 7-5=000.

If the Inquiry Data Pointer contains a value other than Null, it is a pointer to a buffer in the peripheral driver's data space large enough to hold the 36 bytes of Inquiry data associated with the Target/LUN. The data shall be copied from the internal tables of the XPT to the peripheral driver's buffer.

This function shall return non-zero CAM Status.
- CAM Status of Request Completed Without Error indicates that the specified device is installed and the Peripheral Device Type field is valid.

- CAM Status of SCSI Device Not Installed indicates that the Peripheral Device Type field is not valid.
- CAM Status of Invalid Path ID indicates that the Path ID is invalid.

Drivers are always able to use SCSI I/O requests to check for devices which may not have been found at power up.

8.2.2 Path Inquiry

This function is used to get information on the installed HBA hardware, including number of HBAs installed. To obtain further information on any other HBAs attached, this function can be issued for each HBA.

If the Path ID field of the CCB has a value of FFh on a PATH INQUIRY request, then the only field that shall be valid upon return to the caller is the Highest Path ID Assigned field. In addition, this field shall not be valid if the Path ID field in the CCB contains a value other than FFh.

TABLE 8-5 PATH INQUIRY CCB - Part 1 of 2

Size	Dir	Path Inquiry
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)
	I	Features Supported
1		Version Number
		00-07h Prior to Rev 1.7
		08h Implementation Version 1.7
		09-FFh Rev No e.g. 23h = 2.3
1		SCSI Capabilities
		7 Modify Data Pointers
		6 Wide Bus 32
		5 Wide Bus 16
		4 Synchronous Transfers
		3 Linked Commands
		2 reserved
		1 Tagged Queueing
		0 Soft Reset
1		Target Mode Support
		7 Processor Mode
		6 Phase Cognizant Mode
		5-0 reserved
1		Miscellaneous
		7 0=Scanned Low to High
		1=Scanned High to Low
		6 0=Removables included in scan
		1=Removables not included

			5	1=Inquiry data not kept by XPT
			4-0	reserved

TABLE 8-5 PATH INQUIRY CCB - Part 2 of 2

		HBA capabilities	
2	I		Engine count
14	I		Vendor Unique
4	I	Size of Private Data Area	
4	I	Asynchronous Event capabilities	
		31-24	Vendor Unique
		23- 8	reserved
			7 New Devices found during rescan
			6 SIM module De-Registered
			5 SIM module Registered
			4 Sent Bus Device Reset to Target
			3 SCSI AEN
			2 reserved
			1 Unsolicited Reselection
			0 Unsolicited SCSI Bus Reset
1	I	Highest Path ID Assigned	
1	I	SCSI Device ID (of Initiator)	
1		reserved	
1		reserved	
16	I	Vendor ID of SIM-supplier	
16	I	Vendor ID of HBA-supplier	
4	O	OSD Usage	

In some Operating System environments it may be possible to dynamically load and unload SIMs, so Path IDs may not be consecutive from 0 to the Highest Path ID Assigned.

The Path ID value of FFh is assigned as the address of the XPT.

The SCSI Capabilities field is a duplicate of the Byte 7 field in Inquiry Data Format.

The OSD Usage Pointer field is provided for OS-specific or platform-specific functions to be executed by the SIM. The contents of this field are vendor-specific and are not defined by this standard.

In some environments, the Private Data value returned may be zero because the OSD has central allocation of private data requirements, or it is a fixed size as defined by the OSD vendor.

See the vendor specification for the definition of Vendor Unique HBA capabilities peculiar to a particular HBA implementation.

The Asynchronous Event capabilities indicate what reasons cause the SIM to generate an asynchronous event.

```
This function shall return non-zero CAM Status.
 - CAM Status of Request Completed Without Error indicates that the other
   returned fields are valid.
 - CAM Status of Invalid Path ID indicates that the specified Path ID is not
```

installed.

8.2.3 Release SIM Queue

This function is provided so that the peripheral driver can release a frozen SIM queue for the selected LUN (see 6.4.3.3).

TABLE 8-6 RELEASE SIM QUEUE

Size	Dir	Release SIM Queue
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)

This function shall return CAM status of Request Completed Without Error.

8.2.4 Set Async Callback

This function is provided so that a peripheral driver can register a callback routine for the selected Bus/Target/LUN nexus.

TABLE 8-7 SET ASYNC CALLBACK CCB

Size	Dir	Set Async Callback
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)
4	O	Asynchronous Event Enables
		31-24 Vendor Unique
		23- 8 reserved
		7 New Devices found during rescan
		6 SIM module De-Registered
		5 SIM module Registered
		4 Sent Bus Device Reset to Target
		3 SCSI AEN
		2 reserved
		1 Unsolicited Reselection

			0 Unsolicited SCSI Bus Reset
4	O	Asynchronous Callback Pointer	
4	O	Peripheral Driver Buffer Pointer	
1	O	Size of Allocated Peripheral Buffer	

This function shall return:

- CAM Status of Request Completed Without Error indicates that the registration of the callback routine was accepted.
- CAM Status of Request Completed with Error indicates that the registration was rejected (possibly due to invalid parameter settings).

8.2.5 Set Device Type

This function requires the XPT to add the Target ID, LUN and peripheral type to the table of attached peripherals built during CAM initialization.

TABLE 8-8 SET DEVICE TYPE CCB

Size	Dir	Set Device Type
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)
1	O	Peripheral Device Type of Target/LUN

The SIM does not check the validity of the information supplied by the peripheral driver. This function shall return non-zero CAM Status.

NOTE: Blind insertion of device type information may corrupt the table, and results would be unpredictable.

- CAM Status of Request Completed Without Error indicates that the specified information was inserted into the table of SCSI devices.
- CAM Status of Request Completed with Error indicates a problem e.g. not enough room in the table to add the device information.

8.3 SCSI Control Functions

8.3.1 Abort XPT Request

This function requests that an XPT request be aborted by identifying the CCB associated with the request. It should be issued on any I/O request that has not completed that the driver wishes to abort. Success of the Abort function is never assured. This request does not necessarily result in an Abort message being issued over SCSI.

TABLE 8-9 ABORT XPT REQUEST CCB

Size	Dir	Abort XPT Request
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)
4	O	CCB to be Aborted Pointer

This function shall return CAM Status of Request Completed Without Error or Unable to Abort Request.

The actual failure or success of the Abort operation is indicated by the CAM Status eventually returned in the CCB specified.

8.3.2 Reset SCSI Bus

This function is used to reset the specified SCSI bus. This function should not be used in normal operation. This request shall always result in the SCSI RST signal being asserted (see 6.4.3.3 and 6.5).

TABLE 8-10 RESET SCSI BUS CCB

Size	Dir	Reset SCSI Bus
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)

This function shall return CAM Status of Request Completed Without Error.

The actual failure or success of the Reset SCSI Bus is indicated by the Asynchronous Callback information.

8.3.3 Reset SCSI Device

This function is used to reset the specified SCSI target. This function should not be used in normal operation, but if I/O to a particular device hangs up for some reason, drivers can

abort the I/O and Reset the device before trying again. This request shall always result in a Bus Device Reset message being issued over SCSI (see 6.4.3.3 and 6.5).

TABLE 8-11 RESET SCSI DEVICE CCB

Size	Dir	Reset SCSI Device
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)

This function shall return CAM Status of Request Completed Without Error.

The actual failure or success of the Reset SCSI Device is indicated by the Asynchronous Callback information.

8.3.4 Terminate I/O Process Request

This function requests that an XPT I/O request be terminated by identifying the CCB associated with the request. It should be issued on any I/O request that has not completed that the driver wishes to terminate. Success of the Terminate I/O Process is never assured. This request does not necessarily result in a Terminate I/O Process message being issued over SCSI.

TABLE 8-12 TERMINATE I/O PROCESS REQUEST CCB

Size	Dir	Terminate I/O Process Request
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)
4	O	CCB to be Aborted Pointer

This function shall return CAM Status of Request Completed Without Error.

The actual failure or success of the Terminate I/O Process operation is indicated by the CAM Status eventually returned in the CCB specified.

9. Execute SCSI I/O

9. Execute SCSI I/O

The most commonly executed request of the SIM is an I/O command, as defined in the CCB with a Function Code of Execute SCSI I/O.

9.1 CAM Control Block to Request I/O

Peripheral drivers should make all of their SCSI I/O requests using this function, which is designed to take advantage of all features of SCSI which can be provided by virtually any HBA/SIM combination. The CCB is as defined in Table 9-1.

This function will typically return with CAM Status of zero indicating that the request was queued successfully. Function completion can be determined by polling for non-zero status or through use of the Callback on Completion field.

TABLE 9-1 SCSI I/O REQUEST CCB

Size	Dir	SCSI I/O Request
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)
4	O	Peripheral Driver Pointer
4	O	Next CCB Pointer
4	O	Request Mapping Information (OSD)
4	O	Callback on Completion
4	O	SG List/Data Buffer Pointer
4	O	Data Transfer Length
4	O	Sense Info Buffer Pointer
1	O	Sense Info Buffer Length
1	O	CDB Length
2	O	Number of Scatter/Gather entries
4		reserved (OSD)
1	I	SCSI Status
3		reserved (OSD)
4	I	Residual Length
12	O	CDB
		- - - - - OSD - - - - -
4	O	Timeout Value
4	O	Message Buffer Pointer (Target-only)
2	O	Message Buffer Length (Target-only)
2	O	VU Flags
1	O	Tag Queue Action
3		reserved
n	O	Private Data

9.1.1 Address of this CCB

Pointer containing the Physical address of this CCB.

9.1.2 Callback on Completion

This is an OSD field which contains the method by which the SIM is to return to the caller. In some applications it is a pointer, but in others the location of the Callback on Completion routine may be a fixed location and the CCB would contain an argument. See the OSD-specific considerations in **Clause 6**. The address of the Completed CCB shall be passed on the stack to inform the peripheral driver which CCB has completed.

9.1.3 CAM Control Block Length

This field contains the length in bytes of the CCB, including this field and the Address of this CCB in the total.

9.1.4 CAM Flags

This field contains bit settings as described in Table 9-2 to indicate special handling of the requested function.

TABLE 9-2 CAM FLAGS (OSD)

Size	Bits	CAM Flags (OSD)
1	7-6	Direction
		00= reserved
		01=In
		10=Out
		11=No Data Transfer
	5	1=Disable Autosense
	4	1=Scatter/Gather
	3	1=Disable Callback on Comp
	2	1=Linked CDB
	1	1=Tagged Queue Action Enable
	0	1=CDB is a Pointer
1	7	1=Disable Disconnect
	* 6	1=Initiate Synchronous Transfers
	* 5	1=Disable Synchronous Transfers
	4	SIM Queue Priority
		1=Head insertion
		0=Normal (tail insertion)
	3	SIM Queue Freeze
	2	Engine Synchronize
	1-0	reserved
1	7	SG List/Data 0=Host 1=Engine
	6	CDB Pointer 0=VA 1=PA
	5	SG List/Data 0=VA 1=PA
	4	Sense Buffer 0=VA 1=PA
	3	Message Buffer 0=VA 1=PA
	2	Next CCB 0=VA 1=PA
	1	Callback on Comp 0=VA 1=PA
	0	reserved
1		Target Mode-Specific CAM Flags

```
|    |  7 | Data Buffer Valid                |
|    |  6 | Status Buffer Valid              |
|    |  5 | Message Buffer Valid             |
|    |  4 |    reserved                      |
|    |  3 | 1=Phase-Cognizant Mode           |
|    |  2 | 1=Target CCB Available           |
|    |  1 | 1=Disable AutoDisconnect         |
|    |  0 | 1=Disable AutoSave/Restore       |
+----+----+----------------------------------+
 * These bits are mutually exclusive
```

9.1.4.1 Byte 1 Bits

7-6 Direction - These encoded bits identify the direction of data movement during the data transfer phase, though when used in conjunction with Engine processing, they have a little diffferent meaning (see 11).
- a setting of 01 indicates a Read operation (data transfer from target to initiator).
- a setting of 10 indicates a Write operation (data transfer from initiator to target).
- a setting of 11 indicates there is to be no data transfer.

5 Disable Autosense - When set to 1 this bit disables autosense.

4 Scatter/Gather - when set to 1 this bit indicates that data is not to be transferred to/from a single location in memory but to/from several. In this case the Data Buffer Pointer refers to a list of addresses and length in bytes at each address to which the data is to be transferred. The format of the SG List is defined in Table 9-3.

TABLE 9-3 SCATTER GATHER LIST

```
+----+
|Size|
+----+-----------------+
|  4 | Data Address 1  |
|  4 | Data Length 1   |
|  4 | Data Address 2  |
|  4 | Data Length 2   |
|    |      :          |
|  4 | Data Address n  |
|  4 | Data Length n   |
+----+-----------------+
```

3 Disable Callback on Completion - When set to 1 the peripheral driver does not want the SIM to callback automatically when the request is completed. This implies that the caller will be polling for a non-zero CAM Status (which indicates successful completion or termination of the request).

2 Linked CDB - When set to 1 this CDB is a linked command. If this bit is set, then the Control field in the CDB shall have bit 0=1. If not, the results are unpredictable. See 9.2.

1 Tag Queue Actions are to be enabled.

0 If the CDB is identified as a Pointer, the first four bytes of the CDB field contain a pointer to the location of the CDB.

9.1.4.2 Byte 2 Bits

7 When Disable Disconnect=1 the Disconnect capability of SCSI is disabled. The default of 0 sets bit 6=1 in the SCSI Identify MSG (which indicates

that the initiator has the ability to disconnect and reconnect.
6 When Initiate Synchronous Transfers=1 the SIM shall negotiate Synchronous transfers, and wherever possible execute synchronous transfers.
5 When Disable Synchronous Transfers=1 the SIM shall negotiate Asynchronous transfers (if previously negotiated Synchronous). If unable to negotiate Synchronous or negotiation has not yet been attempted, the SIM shall not initiate negotiation.
4 When SIM Queue Priority=1 the SIM shall place this CCB at the head of the Target/LUN internal queue to be the next operation sent to the Target/LUN by the SIM.
3 When SIM Queue Freeze=1 the SIM shall place its internal Target/LUN queue into the frozen state. Upon callback, the CAM Status for this CCB shall have the SIM Queue Freeze flag set. This bit should only be set for SIM error recovery and should be used in conjuntion with the SIM Queue Priority bit and the Release SIM Queue command.
2 The Engine Synchronize=1 is used in conjunction with the In or Out setting to flush any residual bits before terminating engine processing (see 11).

9.1.4.3 Byte 3 Bits

The Pointer fields are set up to have one characteristic. If a bit is set to 1 it means the pointer contains a Physical Address. If set to 0 it means the pointer contains a Virtual Address. If the SIM needs an address in a different form to that provided, it should be converted by the SIM (using OSD facilities) and stored in Private Data.

9.1.4.4 Byte 4 Bits

The Target Mode Only Flags are only active on Enable LUN or Execute Target I/O commands.

7-5 The Buffer Valid bits identify which buffers have contents. In the event that more than one bit is set, they shall be transferred in the sequence of Data Buffer, Status, Message Buffer.
3 Phase-Cognizant Mode - if target operations are supported, when set to 1, the SIM shall operate in Phase-Cognizant Mode, otherwise it shall operate in Processor Mode.
2 Target CCB Available - when set to 1 this bit indicates that the XPT/SIM can use this CCB to process this request. A value of 0 indicates that this CCB is not available to the XPT/SIM.
1 AutoDisconnect - when set to 1 this bit disables AutoDisconnect. The default of 0 causes the XPT/SIM to automatically disconnect, if the Identify message indicates DiscPriv is set.
0 AutoSave - when set to 1 this bit disables AutoSave. The default of 0 causes the XPT/SIM to automatically to send a Save Data Pointer message on an AutoDisconnect.

9.1.5 CAM Status

This field is returned by the SIM after the function is completed. A zero status indicates that the request is still in progress or queued. CAM Status is defined in Table 9-4.

If Autosense information is available, the code returned shall be incremented by 80h e.g. 04h indicates an error occurred, and 84h indicates that an error occurred and Autosense information is available for analysis.

TABLE 9-4 CAM STATUS

+------+--+

00h	Request in progress
01h	Request completed without error
02h	Request aborted by host
03h	Unable to Abort Request
04h	Request completed with error
05h	CAM Busy
06h	Invalid Request
07h	Invalid Path ID
08h	SCSI device not installed
09h	Unable to Terminate I/O Process
0Ah	Target Selection Timeout
0Bh	Command Timeout
0Ch	reserved
0Dh	Message Reject received
0Eh	SCSI Bus Reset Sent/Received
0Fh	Uncorrectable Parity Error Detected
10h	Autosense Request Sense Cmd Failed
11h	No HBA detected
12h	Data OverRun/UnderRun
13h	Unexpected Bus Free
14h	Target bus phase sequence failure
15h	CCB Length Inadequate
16h	Cannot Provide Requested Capability
17h	Bus Device Reset Sent
18h	Terminate I/O Process
19-37h	reserved
	Target Mode Only Status
38h	Invalid LUN
39h	Invalid Target ID
3Ah	Function not Implemented
3Bh	Nexus not Established
3Ch	Invalid Initiator ID
3Dh	SCSI CDB Received
3Eh	LUN Already Enabled
3Fh	SCSI bus Busy
+40H	to indicate that SIM Queue is frozen
+80h	to indicate that Autosense is valid

- 00h Request in progress: the request is still in process.
- 01h Request completed without error: the request has completed and no error condition was encountered.
- 02h Request aborted by host: the request was aborted by the peripheral driver.
- 03h Unable to Abort Request: the SIM was unable to abort the request as instructed by the peripheral driver.
- 04h Request completed with error: the request has completed and an error condition was encountered.
- 05h CAM Busy: CAM unable to accept request at this time.
- 06h Invalid Request: the request has been rejected because it is invalid.
- 07h Invalid Path ID indicates that the Path ID is invalid.
- 08h SCSI device not installed: Peripheral Device Type field is not valid.
- 09h Unable to Terminate I/O Process: the SIM was unable to terminate the request as instructed by the peripheral driver.
- 0Ah Target Selection Timeout: The target failed to respond to selection.
- 0Bh Command Timeout: the specified command did not complete within the

timer value specified in the CCB.
- 0Dh Message Reject received: The SIM received a Message Reject MSG.
- 0Eh SCSI Bus Reset Sent/Received: The SCSI operation was terminated at some point because the SCSI bus was reset.
- 0Fh Uncorrectable Parity Error Detected: An uncorrectable SCSI bus parity error was detected. When this occurs, the SIM sends the ABORT message to the target.
- 10h Autosense Request Sense Command Failed: The SIM attempted to obtain sense data and failed.
- 11h No HBA detected: HBA no longer responding to SIM (assumed to be a hardware problem).
- 12h Data OverRun: target transferred more data bytes than peripheral driver indicated in the CCB.
- 13h Unexpected Bus Free: an unexpected Bus Free condition occurred.
- 14h Target Bus Phase Sequence Failure: the target failed to operate in a proper manner according to X3.131-1990 e.g. it went to the Message Out phase after the initiator asserted ATN.
- 15h CCB Length Inadequate: More private data area is required in the CCB.
- 16h Cannot Provide Requested Capability: Resources are not available to provide the capability requested (in the CAM Flags).
- 17h Bus Device Reset Sent: this CCB was terminated because a Bus Device Reset was sent to the target.
- 18h Terminate I/O Process: this CCB was terminated because a Terminate I/O Process was sent to the target.
- 38h Invalid LUN indicates that the LUN specified is outside the supported range of the SCSI bus.
- 39h Invalid Target ID indicates that the Target ID does not match that used by the HBA specified by the Path ID field.
- 3Ah Function Not Implemented indicates that Target Mode is not supported.
- 3Bh Nexus not Established: There is currently no connection established between the specified Target ID and Target LUN and any initiator.
- 3Ch Invalid Initiator ID: The initiator ID specified is outside the valid range that is supported.
  NOTE: This status can also be returned if the target tries to reselect an initiator other than the one to which it was previously connected.
- 3Dh SCSI CDB Received: Indicates that the target has been selected and that the SCSI CDB is present in the CCB.
- 3Eh LUN Already Enabled: The LUN identified in Enable LUN was previously enabled.
- 3Fh SCSI bus Busy: The SIM failed to win arbitration for the SCSI Bus during several different bus free phases.

9.1.6 CDB

This field either contains the SCSI CDB (Command Descriptor Block), or a pointer to the CDB, to be dispatched.

9.1.7 CDB Length

This field contains the length in bytes of the CDB.

9.1.8 Data Transfer Length

This field contains the length in bytes of the data to be transferred.

9.1.9 Function Code

See 8.1.2.

9.1.10 LUN

This field identifies the SCSI LUN to which this CCB is being directed.

9.1.11 Message Buffer Length (Target-only)

This field contains the length in bytes of the field which is to be used to hold Message information in the event that the Peripheral Drivers needs to issue any MSGs. This field is exclusive to Target Mode operation.

9.1.12 Message Buffer Pointer (Target-only)

This field contains a pointer to buffer containing Messages. This pointer is only valid for use in Target Mode.

9.1.13 Next CCB Pointer

This field contains a pointer to the next command block in a chain of command blocks. A value of 0 indicates the last command block on the chain. This field is used for the linking of commands.

9.1.14 Number of Scatter/Gather entries

This field contains the number of entries in the SG List.

9.1.15 Path ID

The Path ID specifies the SCSI port on the installed HBA to which the request is addressed. Path IDs are assigned by the XPT, begin with zero, and need not be consecutive. The Path ID of FFh is assigned for the XPT. An HBA may have more than one SCSI port. A SIM may support more than one HBA.

9.1.16 Peripheral Driver Pointer

This field contains a pointer which is for the exclusive use of the Peripheral Driver, which use is not defined by this standard.

9.1.17 Private Data

This field is used to contain whatever fields the CAM Module needs to execute the request. As such it constitutes a scratchpad of working space needed by the SIM and/or the XPT. The size of this area is an OSD as it may differ between SIMs and XPTs by environment or by vendor implementation. The device driver is responsible to query the XPT and ensure that enough Private Data area is available to the SIM and/or XPT.

9.1.18 Request Mapping Information (OSD)

This field is a pointer to an OSD dependent data structure which is associated with the original I/O request.

9.1.19 Residual Length

This field contains the difference in twos complement form of the number of data bytes transferred by the HBA compared with the number of bytes requested by the CCB.

9.1.20 SCSI Status

This field contains the status byte returned by the SCSI target after the command is completed.

9.1.21 Sense Info Buffer Length

This field contains the length in bytes of the field which is to be used to hold Sense data in the event that a Request Sense is issued.

9.1.22 Sense Info Buffer Pointer

This field contains a pointer to the data buffer for Request Sense data. This pointer will only be used if a Check Condition occurs while performing the specified command.

9.1.23 SG List/Data Buffer Pointer

This field contains a pointer to either the data buffer to which data is to be transferred, or to the SG List which contains the list of scatter/gather addresses to be used for the transfer.

9.1.24 Tagged Queue Action

SCSI provides the capability of tagging commands to force execution in a specific sequence, or of letting the target optimize the sequence of execution to improve performance. This function provides a similar capability. For a description of the tagged command queueing philosophy see SCSI-2 X3.131-1990.

When the Queue Action Enable bit in the CAM Flags is set, the CDB issued by the SIM shall be associated with the Queue Action specified as:

20h = Simple Tag Request
21h = Head of Queue Tag Request
22h = Ordered Queue Tag Request

9.1.25 Target ID

This field identifies the SCSI target which is to be selected for execution of the CCB request.

9.1.26 Timeout Value

This field contains the maximum period in seconds that a request can remain outstanding. If this value is exceeded then the CAM Status shall report the timeout condition. A value of 00h in the CCB means the peripheral driver accepts the SIM default timeout. A value of F...Fh in the CCB specifies an infinite period.

9.1.27 VU Flags

The uses for this field are defined in the vendor specification.

9.2 Command Linking

The SIM supports SCSI's ability to link commands in order to guarantee the sequential execution of several requests. This function requires that both the HBA and the involved target(s) support the SCSI Link capability.

To utilize linking, a chain of CCBs is built with the Next CCB Pointer being used to link the CCBs together. The CAM Flag Link bit shall be set in all CCBs but the last in the chain. When a SCSI target returns the Linked Command Complete message, the next CCB is processed, and its associated CDB is dispatched.

Any Check Condition returned by the target on a linked command shall break the chain.

10. Target Mode (Optional)

10. Target Mode (Optional)

If a Target Mode function is specified by a CCB and this functionality is not provided by a particular SIM implementation, then a CAM Status of Function Not Implemented shall be returned in the CCB.

The Target Mode functionality causes the HBA associated with the specified SCSI link to be set up so that it may be selected as a target i.e. when an HBA is operating in Target mode, it is responding to other HBAs on the same SCSI cable.

There are two different modes of target operation, either or both of which may be supported by the XPT/SIM as defined by the Target Mode Support flags in the Path Inquiry CCB.

- Processor mode
- Phase-Cognizant mode

Processor mode permits an application to register itself as a LUN and provide a set of one or more CCBs that the XPT/SIM can use for receiving and sending data. In this mode, when the adapter is selected and the XPT/SIM receives an Identify message for a LUN that has registered as a Processor LUN, the XPT/SIM will accept any processor device commands (Inquiry, Request Sense, Send, Receive) and, using one of the available CCB's, process the SCSI command through completion.

Upon disconnection, the SIM calls back on completion to let the application know that the CCB has been processed. From the time that the application registers itself until the time a command has completed, there is no callabck to the application.

In summary, Processor applications get called back only after the SCSI command has been completely processed, and leaves all phase handling and SCSI command processing nuances to the XPT/SIM and the previously registered CCB's.

Phase-Cognizant mode permits an application tighter control over what takes place when a SCSI command is received by the SIM. When a Phase-Cognizant application registers itself and a command is received, the XPT/SIM does an immediate Callback on Completion after placing the SCSI command in an available CCB. The Phase-Cognizant application is responsible to set up data, message, status fields and CAM-Flags in the CCB and re-issue the CCB with an Execute Target I/O function code so that the XPT/SIM knows which phases it should execute. The "callback-reissue CCB" cycle may happen multiple times before a command completes execution.

In summary, Phase Cognizant applications get a callback immediately after the SCSI command block is received and is expected to instruct the XPT/SIM as to which phases to go through to perform the command.

10.1 Enable LUN

The specified Target ID shall match that returned by the HBA Inquiry Function for the HBA. The specified LUN is the one enabled for selection, and if the HBA is to respond as an additional LUN, another Enable LUN is required.

In addition to providing a hook into the application, this function is intended to provide an area that the XPT/SIM can use as working space when the HBA is selected.

TABLE 10-1 ENABLE LUN CCB

Size	Dir	Enable LUN CCB
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)
2	O	Group 6 Vendor Unique CDB Length
2	O	Group 7 Vendor Unique CDB Length
4	O	Pointer to Target CCB List
2	O	Number of Target CCBs

If the Number of Target CCBs is zero, then Target Mode is disabled, otherwise the Pointer to Target CCB List refers to a list of addresses of CCBs to which the data is to be transferred (see Table 10-2).

TABLE 10-2 TARGET CCB LIST

Size	Target CCB List
4	CCB Address 1
4	CCB Address 2
	:
4	CCB Address n

The XPT/SIM shall place the pointer to the CCB, or the pointer to the list of CCBs, in a list until the specified Target ID and LUN is selected on the SCSI link specified by the Path ID field. While the request is being held, the CAM Status field of the Target CCB, shall be set to Request in Progress. The application is required to poll on the CAM status field of the Target CCB or provide a Completion Callback routine through the Target CCB.

The XPT/SIM shall keep an indication of whether a single CCB or list of CCBs was provided on the Enable LUN service.

The XPT/SIM shall set the following in each Target CCB when they are first provided:

- CAM Status to Request In Progress
- CAM Flags shall be the same as those in the Enable LUN CCB
- CAM Flags shall set the Target CCB Available as needed

Within the Target CCB provided, the following information shall be present and valid,

- CAM Flag information including AutoDisconnect and AutoSave.
- CDB field is valid for the Command Blocks that may be received. That is either CDBs are embedded in the CCB, or a pointer to a CDB area is provided in the CDB field.

- The Group 6 and 7 Vendor Unique CDB Length fields contain the number of bytes a target application expects to receive for it's vendor unique command set. The previous item shall go hand-in-hand with this requirement. The Group 6 and 7 Vendor Unique CDB Length fields shall be retained for each LUN enabled.

If the target application supports Vendor Unique Command Blocks, then the CDB field of the CCB shall reflect the nature and size of those Vendor Unique Command Blocks. Ample space shall be provided to contain the CDBs that may be received. If a CDB greater than the size of the CDB field is desired, then the CDB field shall contain a pointer to a CDB.

To disable the selection of a specific LUN, the application performs an Enable LUN with a zero value for the Number of Target CCBs.

If a LUN is disabled, after having been enabled, then the Inquiry data and the Vendor Unique CDB Length data shall be cleared.

The XPT/SIM shall prevent a nexus being established between an initiator and a specified LUN that has been disabled. If there is a pre-existing nexus, then Invalid Request shall be returned.

This function shall return non-zero CAM Status.

- CAM Status of Request Completed Without Error indicates that the Enable LUN was completed successfully.
- CAM Status of Invalid Request indicates that there is currently a nexus established with an initiator that shall be terminated, first.
- CAM Status of Invalid Path ID indicates that the Path ID is invalid.
- CAM Status of Invalid Target ID indicates that the Target ID does not match that used by the HBA specified by the Path ID field.
- CAM Status of Invalid LUN indicates that the LUN specified is outside the supported range of the SCSI bus.
- CAM Status of Function Not Implemented indicates that Target Mode is not supported by this implementation of CAM.

10.2 Phase Cognizant Mode

10.2.1 Target Operation of the HBA

When the HBA is selected, the XPT/SIM automatically sets the HBA to the Message Out phase to receive the Identify, Synchronous Data, and other messages that may be sent by the Initiator. The XPT/SIM response to these messages shall be as defined in X3.131-1990.

The LUN shall be extracted from the Identify Message and the appropriate CCB shall be extracted from the list of CCBs being held by the XPT/SIM.

If the LUNTAR bit (or any of the reserved bits) of the Identify Message is set to 1, then the XPT/SIM shall send a MESSAGE REJECT message back to the initiator.

If no CCBs are being held by the XPT/SIM for a Target ID, then the XPT/SIM shall not respond to the selection of that Target ID.

If CCBs are being held by the XPT/SIM, and the LUN indicated by the Identify Message does not have a CCB provided by an application, then the XPT/SIM shall provide the following support:

a) If an Inquiry command is sent to this LUN, then the XPT/SIM shall respond with Inquiry Data that indicates "Logical Unit Not Supported."
b) If any other command (except Request Sense) is sent to this LUN, then the XPT/SIM shall respond with a Check Condition.
c) If a Request Sense command is sent to this LUN after a Check Condition status is sent, then the XPT/SIM shall respond with sense data that indicates "Logical Unit Not Supported".

The XPT/SIM shall scan the CAM Flags in the CCB(s) provided with Enable LUN.
If none of them have the Target CCB Available bit set, the XPT/SIM shall request the SCSI CDB and post BSY status. The XPT/SIM shall not modify the SCSI CDB(s) in the CCB(s).

After processing the CDB from a Target CCB, the target application shall set CCB Available in the CAM Flags, which allows the application to pass the same CCB back to the XPT/SIM upon Callback on Completion (this prevents the possibility that the XPT/SIM could use the CCB on selection). The setting of the Target Available bit could be done at the Callback on Completion after the Exectute Target I/O which trasnmits SCSI Status.

If a target application sets Target Available upon recognizing that a CDB has been received and uses a different CCB to perform the data transfer, there is a lower likelihood of a BSY response to the initiator when a CCB is not available.

The Disable Disconnect bit in the CAM Flags field shall be updated to indicate the state of the DiscPriv bit in the Identify message that was received from the initiator. If the DiscPriv bit was set in the Identify Message, then the Disable Disconnect bit shall be cleared, and vice-versa. NOTE: The default state of the Disable Disconnect bit in the CAM Flags is cleared, implying that disconnect is enabled.

The Target ID field shall be set to the ID of the initiator that performed the selection. This field can then be used by subsequent functions, such as reselect, to determine the Initiator's ID.

Once the initial Message Out Phase is complete, the XPT/SIM automatically sets the HBA to the Command Out Phase to request the SCSI CDB. After receiving the SCSI CDB bytes, the XPT/SIM shall set the CAM status field to CAM Status of SCSI CDB received, and clear the CCB Available bit in the CAM Flags.

Upon completion of the data phase, the XPT/SIM shall send the appropriate SCSI status and Command Complete and then disconnect from the bus. The XPT/SIM shall then post the required CAM Status in the CCB, or Callback on Completion.

If the Group Code of the Operation Code of the Command Block is Vendor Unique the XPT/SIM shall ensure that only the indicated number of command bytes are received. If the required number of bytes are exceeded or not transferred, then the XPT/SIM shall return a status of Check Condition, the Sense Key in the Sense Buffer shall be set to Illegal Request, and the Additional Sense Key and Qualifier shall be set to Command Phase Error.

If the DiscPriv bit in the Identify message was set, which results in the Disable Disconnect bit of the CAM Flags being cleared, and the Disable AutoDisconnect bit of the CAM Flags field is cleared, the XPT/SIM shall automatically disconnect upon receipt of the command block. The subsequent invocation of the Execute Target I/O function shall perform an automatic reselect when it is invoked.

If a BUS DEVICE RESET message is received at any time, the XPT/SIM shall set the CAM

Status field to SCSI Bus Reset Sent/Received for any CCB being held (through Enable LUN), or that is active in the XPT/SIM.

If a SCSI Bus Reset occurs the asynchronous callback and bus reset mechanism defined for initiator mode shall be followed.

The SIM shall reject any CCB which has a Timeout Value of other than infinity.

10.2.2 Execute Target I/O

If the Data Valid bit is set, the XPT/SIM shall enter the data phase indicated by the direction bit in the CAM Flags field (ie. DATA IN or DATA OUT). It shall send/receive data to/from the buffer(s) indicated in the CCBs Scatter Gather List or Data Pointer.

If the Status Valid bit is set, the XPT shall send the status byte specified in the SCSI Status field to the current initiator and then send the Command Complete Message.

If the Message Valid bit is set, the XPT shall enter the Message phase and transfer the contents of the Message buffer.

The XPT/SIM shall receive and respond to any messages resulting from ATN being asserted by the initiator, in addition to any messages it sends to the initiator.

The XPT/SIM shall be able to execute all the phases indicated by the Buffer Valid bits of the CAM Flags, within a single invocation of the Execute Target I/O i.e. if more than one bit is set, the order of execution of the phases shall be data, status, and message.

If the Data Buffer Valid and Status Buffer Valid bits of the CAM Flags are both set for an invocation of Execute Target I/O, the AutoDisconnect and AutoSave features shall be disabled.

If the Disable AutoDisconnect bit of the CAM Flags is cleared, and the Disable Disconnect of the CAM Flags bit is cleared, then the XPT/SIM shall disconnect on the completion of the data transfer.

If the Disable AutoSave bit of the CAM Flags is cleared, then the XPT/SIM shall send a Save Data Pointers message to the initiator prior to disconnect.

The XPT/SIM shall perform an automatic reselect if the XPT/SIM had disconnected after the receipt of the CDB, or had disconnected upon completion of a previous Execute Target I/O (within the same I/O process).

Upon the last Execute Target I/O, the target application should consider setting the Disable AutoSave bit, which shall disable the sending of the Save Data Pointers.

This function typically returns with CAM Status of zero indicating that the request was executed successfully. Function completion can be determined by polling for non-zero status or through use of the Callback on Completion field.

10.3 Processor Mode

10.3.1 CCB Acceptance

In Processor mode, the Target CCB List shall contain at least one pre-built CCB that the SIM can use when it responds to selection. The Target CCBs that are supported by the SIM

include CDBs for the following commands:

- Inquiry
- Receive
- Request Sense
- Send

The SIM shall verify that the CCBs in the Target CCB List contain supported commands, valid data buffers etc.

Any invalid CCB in the list shall be rejected and the LUN shall not be enabled.

10.3.2 Target Operation of the HBA

When the target HBA is selected, it shall automatically request the CDB.

The SIM shall search the Target CCB List to find a matching CDB. If a matching CDB is found, it shall verify that Target CCB Available=1, and use the contents of the data buffers to process the command received. The SIM shall clear Target CCB Available, and if the peripheral driver wants the CCB to be re-used it is responsible to set Target CCB Available=1,

Upon completion of the CDB received, the SIM shall report CAM Status in the CCB and call back the peripheral driver.

If the Target CCB List has no CCBs with Target CCB Available=1, but matches were found, the SIM shall send Busy Status to the Initiator.

If the Target CCB List contained no matching CCBs, then the SIM shall return Check Condition to the Initiator. Upon receipt of a Request Sense command, the SIM shall return a Sense code of "Invalid CDB" to the Initiator.

If an Inquiry CDB is received but there is no Inquiry CDB in one of the CCBs in the Target CCB List then the SIM shall return Inquiry Data of "LUN Not Supported" to the Initiator. NOTE: A CCB to respond to an Inquiry CDB should be provided in every Target CCB List.

If an Inquiry CDB is and there is an Inquiry CDB in one of the CCBs in the Target CCB List then the SIM shall return the Inquiry information provided by the data buffer pointer. The SIM does not clear Target CCB Available or call back as it is a placeholder of consistent information.

11. HBA Engines

(Part of **ANSI Common Access Method rev 2.3**)

11. HBA Engines

An engine is a hardware device implemented in an HBA to perform time-intensive functions not available on target devices. Generally, these engines are required to process data prior to building a CDB and submitting to the device. There may be more than one engine in a HBA.

One use of engines is to compress data. In this mode, a device driver first submits data to the engine. Once the engine has completed processing the data, an Execute SCSI CCB can be built for the SCSI transfer.

The engine model allows for the addressing of buffer memory located on the HBA. The buffer addressing appears to the host as contiguous space. Using this model, it is possible to submit multiple requests until the engine buffer is full. Once the full condition is met, an Execute SCSI CCB can be built.

When the full condition occurs (as defined by the Destination Data Length equalling the Destination Data Maximum Length), the amount of unprocessed source data is reported in the Source Residual Length. The residual data may then be re-submitted at a later time.

11.1 Engine Inquiry

This function is used to learn information on the engines installed in the HBA hardware. This function is issued for each engine.

TABLE 11-1 ENGINE INQUIRY CCB

Size	Dir	Engine Inquiry
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)
2	O	Engine Number
1	I	Engine Type
		0=Buffer Memory
		1=Lossless Compression
		2=Lossy Compression
		3=Encryption
		4-FF reserved
1	I	Engine Algoritm ID
		0=Vendor Unique
		1=LZ1 Variation 1 (STAC)
		2=LZ2 Variation 1 (HP DCZL)
		3=LZ2 Variation 2 (Infochip)
		4-FF reserved
4	I	Engine Memory Size

```
+----+---+--------------------------------------+
```

The Engine Type reports the generic function the addressed engine number is capable of supporting.

The Engine Algorithm ID reports the specific capability the addressed engine supports.

The amount of buffer memory provided for an engine is reported in the Engine Memory Size.

This function shall return non-zero CAM Status.

- CAM Status of Request Completed Without Error indicates that the other returned fields are valid.
- CAM Status of Invalid Request indicates that the specified Engine Number is not installed.

11.2 Execute Engine Request (Optional)

To accomodate buffering associated with the engine, the CAM Flag SG List/Data set to 1=Engine is used to specify that the normal Data Buffer Pointer is actually a physical address in the buffer space of the engine.

There are four modes associated with engine processing established by CAM Flags:

- A Direction setting of Out is used to Encrypt or Compress the data
- A Direction setting of Out is used to Decrypt or Decompress the data
- Synchronize is used in conjunction with In or Out to flush any residual bits prior to terminating engine processing.

Certain types of engines change the size of data as part of their operation e.g. the purpose of compression engines is to reduce the size of data prior to transmission over SCSI. As such, the Execute Engine Request CCB allows the engine to report the size of the resultant data.

TABLE 11-1 EXECUTE ENGINE REQUEST CCB

Size	Dir	Execute Engine Request
		- - - - - OSD - - - - -
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
1		reserved
		- - - - Common - - - - -
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags (OSD)
4	O	Peripheral Driver Pointer
4	O	reserved (OSD)
4	O	Request Mapping Information (OSD)
4	O	Callback on Completion
4	O	SG List/Data Buffer Pointer
4	O	Data Transfer Length
4	O	Engine Buffer Data Pointer
1		reserved (OSD)

1			reserved (OSD)
2	O		Number of Scatter/Gather entries
4	O		Destination Data Maximum Length
4	I		Destination Data Length
4	I		Source Residual Length
12			reserved (OSD)
			- - - - - OSD - - - - -
4	O		Timeout Value
4			reserved
2	O		Engine Number
2	O		VU Flags
1			reserved
3			reserved
n	O		Private Data

This function will typically return with CAM status of zero indicating that the request was queued successfully. Function completion can be determined by polling for non-zero status or through use of the Callback on Completion field.

Annex A. Physical/Logical Translation in 80x86 Environment

(Part of **ANSI Common Access Method rev 2.3**)

ANNEX

Annex A. Physical/Logical Translation in 80x86 Environment

A.1 OSD Formatting of Disk Drives

The DOS physical address to/from logical block address conversion algorithms to map SCSI disks into int 13h Head-Cylinder-Sector format vary widely between suppliers of software to support third party disks.

The following "C" routines have been adopted by CAM as representing the most efficient utlization of capacity. The following code is ANSI "C" that can be compiled using the Microsoft C compiler, version 5.1.

a) SETSIZE converts a Read Capacity value to int 13h Head-Cylinder-Sector requirements. It minimizes the value for number of heads and maximizes the number of cylinders. This will support rather large disks before the number of heads will not fit in 4 bits (or 6 bits). This algorithm also minimizes the number of sectors that will be unused at the end of the disk while allowing for very large disks to be accomodated. This algorithm does not use physical geometry.

b) LTOP does logical to physical conversion

c) PTOL does physical to logical conversion

d) MAIN is a test routine for a, b and c.

A.1.1 SETSIZE

```
*/
typedef unsigned int UINT;
typedef unsigned long ULNG;
/*
 *  Convert from logical block count to Cylinder, Sector and Head (int 13)
 */

 int setsize(ULNG capacity,UINT *cyls,UINT *hds,UINT *secs)


  UINT rv = 0;
  ULNG heads, sectors, cylinders, temp;

  cylinders = 1024L;                /* Set number of cylinders to max value */
  sectors = 62L;                    /* Max out number of sectors per track */

  temp = cylinders * sectors;       /* Compute divisor for heads */
  heads = capacity / temp;          /* Compute value for number of heads */
  if (capacity % temp)             /* If no remainder, done! */
    heads++;                        /* Else, increment number of heads */
    temp = cylinders * heads;       /* Compute divisor for sectors */
    sectors = capacity / temp;      /* Compute value for sectors per track */
    if (capacity % temp)           /* If no remainder, done! */
      sectors++;                    /* Else, increment number of sectors */
```

```
      temp = heads * sectors;       /* Compute divisor for cylinders */
      cylinders = capacity / temp;  /* Compute number of cylinders */


  if (cylinders == 0) rv=1;         /* Give error if 0 cylinders */

  *cyls = (UINT) cylinders;         /* Stuff return values */
  *secs = (UINT) sectors;
  *hds  = (UINT) heads;
  return(rv);
```

A.1.2 LTOP

```
/*
 *  logical to physical conversion
 */

void ltop(ULNG block,UINT hd_count,UINT sec_count,UINT *cyl,UINT *hd,UINT
*sec)


  UINT spc;
  spc = hd_count * sec_count;
  *cyl = block / spc;
  *hd =  (block % spc) / sec_count;
  *sec = (block % spc) % sec_count;
```

A.1.3 PTOL

```
/*
 *     Physical to logical conversion
 */

ULNG ptol(UINT cyl,UINT hd,UINT sec,UINT cyl_count,UINT hd_count,UINT
sec_count)


  ULNG cylsize;
  cylsize = sec_count * hd_count;
  return((cyl * cylsize) + (hd * sec_count) + sec);
```

A.2 Backwards Compatibility

The selection of a new algorithm for CAM solves the problem of future compatibility, but it does not solve the problem of the installed base. The following technique will permit a supplier to update the installed base to CAM-compliant operation but not require users to reforamt their drives.

A.2.1 ROM-based

The one sector that is independent of the algorithm is sector 00. Under DOS and many other Operating Systems this sector is used for the boot sector and contains the Partition Table for a fixed disk.

If the Partition Table is structured according to MS DOS and IBM DOS standards, partitions end on cylinder boundaries e.g.

Offset from start of Partition		Table entry
00h	Boot Indicator	80h
01h	Beginning or start head	01h
02h	beginning or start sector	01h
03h	Beginning or start cylinder	00h
04h	System indicator	04h
05h	Ending head	07h
06h	Ending sector	91h
07h	Ending cylinder	7Ch
08h	Starting sector (relative to beginning of disk)	
0Ch	Number of sectors in partition	

The ending head 07h indicates a device with 8 heads (0 to 7). The ending sector 91h contains 2 bits of high cylinder so it has to be masked to obtain ending sector = 11h (17 decimal).

To verify these values calculate:

Logical Ending sector (from Beginning Head, Cylinder, and Sector)

and compare it to:

(Starting Sector + Number of Sectors in Partition)

This leaves Number of Cylinders as the one unresolved parameter. This is obtained by:

Read Capacity divided by (Heads * Sectors).

All of this can be done by the BIOS in ROM or RAM. To be capable of booting from any drive or cartridge regardless of the algorithm used to partition and format the media, the BIOS would need to respond to int 13 function 8 with the head, sector, and cylinder values obtained from this information. In addition, the BIOS would need to use those values in its calculation from physical to logical sectors.

Example of Pseudocode:

```
For each Drive
     Read Boot Sector (LBA 0)
     Validate The Signature at end of Sector (55AA)
     Find Partion with largest Logical Start Cyl

        If No Partitions found
        Use Defaults
        Exit

     SECS = Ending Sector (from partition table)
     Heads = Ending Head+1  (from partition table)

     Logical_End = End_cyl * (End_head+1 * End_sector) +
                   (End_head * End_sector) + End_sector
```

```
Compare Logical_End to Starting_sec + Number_sec
   If not equal
   Use Defaults
   Exit

Cyls = Capacity / (End_head+1 * End_sector)
```

A.2.2 RAM-based

Under DOS it is possible to modify the code of the boot sector to accomplish bootability. Access to other partitions is dependent on the device driver to do a translation.

This method is a patch just prior to jumping to code loaded in memory at segment 00 offset 7C00h.

```
PUSH   AX           ; Save registers used in patch
PUSH   DX
MOV    AH,08        ; set function code = 8 get drive parameters
INT    13           ; do INT 13 call
INC    DH           ; inc head number to convert from zero based
MOV    [7C1A],DH    ; fix value of heads in BPB table
AND    CL,3F        ; Mask off non-sector information
MOV    [7C18],CL    ; fix value of sectors in BPB table
POP    DX
POP    AX           ; Restore registers used in patch
JMP    7C00         ; jump to partition boot loader

01B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 80 01
01C0  01 00 06 07 91 7C 11 00-00 00 57 52 01 00 00 00
01D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
01E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
01F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 55 AA
```

7C00	7C03	7C0B	7C0D	7C0E
	Name	Bytes/	Sectors/	Reserved
jump nop	I B M 4 . 0	Sector	Cluster	Sectors
EB 3C 90	49 42 4D 20 20-34 2E 30	00 02	04	01 00

7C10	7C11	7C13	7C15	7C16	7C18	7C1A
# FATs	# DIR entries	# Log'l Sectors	Media Descrip	# FAT Sectors	# Sectors	# Heads
02	00 02	00 00	F8	55 00	11 00	08 00

Annex B: Target Application Examples

(Part of **ANSI Common Access Method rev 2.3**)

Annex B: Target Application Examples

[Mike Roche to prepare an update which is consistent with Section 10.]

The following are examples of how a Target Application can operate the Target Mode capabilities defined in Section 9.

B.1 Initialization Sequence with Single Target CCB provided

```
- fill Target CCB with required info
  targetCCB.callbackPointer = callback routine address
- fill Enable CCB with the required information
  enableCCB.functionCode = function code for enable lun
  enableCCB.targetid = the id of the target
  enableCCB.targetLun = the lun to enable
  enableCCB.group6VULength = vendor unique length for Group 6 (IF required)
  enableCCB.group7VULength = vendor unique length for Group 7 (IF required)
  enableCCB.targetCCBListLength = 0
  enableCb.targetCCBPointer = &targetCCB
- Enable LUN (&enableCCB)
- EXIT
```

B.2 Initialization Sequence with Multiple Target CCBs provided

```
- fill Target CCB #1 with required info
  target1CCB.callbackPointer = callback routine address #1
- fill Target CCB #2 with required info
  target2CCB.callbackPointer = callback routine address #2
  target2CCB.camStatus = request completed by target application
- fill Target CCB #n with required info
  targetnCCB.callbackPointer = callback routine address #n
  targetnCCB.camStatus = request completed by target application
- targetCCBList [0] = pointer to target1CCB
- targetCCBList [1] = pointer to target2CCB
- targetCCBList [n] = pointer to targetnCCB

NOTE:  where targetCCBList is an array of pointers

- fill enable CCB with the required information
  enableCCB.functionCode = function code for enable lun
  enableCCB.targetid = the id of the target
  enableCCB.targetLun = the lun to enable
  enableCCB.group6VULength = vendor unique length for Group 6 (IF required)
  enableCCB.group7VULength = vendor unique length for Group 7 (IF required)
  enableCCB.targetCCBListLength = 4 * number of target CCBs
  enableCb.targetCCBPointer = &targetCCBList
- Enable LUN (&enableCCB)
- EXIT
```

B.3 Application Sequence with single Execute Target I/O

```
- IF targetCCB.camStatus != SCSI CDB Received THEN EXIT, OR
  callback from XPT/SIM
- process SCSI CDB field in targetCCB
```

```
- fill targetCCB with required information
  targetCCB.functionCode = function code for execute target io
  targetCCB.camFlags = data phase and status phase
  targetCCB.dataBufferPointerLength = length of data
  targetCCB.dataBufferPointer = pointer to data buffer
  targetCCB.SCSIStatus = whatever status is appropriate
- Execute Target IO (&targetCCB)
/* return target CCB to pool           */
- targetCCB.camStatus = request in progress
- EXIT
```

B.4 Application Sequence with multiple Execute Target I/O

```
- IF targetCCB.camStatus != SCSI CDB Received THEN EXIT, OR
  callback from XPT/SIM
- process SCSI CDB field in targetCCB
- loop until all data transferred
      fill targetCCB with required information
      targetCCB.functionCode = function code for execute target io
      targetCCB.camFlags = data phase
      targetCCB.dataBufferPointerLength = length of data
      targetCCB.dataBufferPointer = pointer to data buffer

      IF (last data block)
         targetCCB.camFlags = data phase AND status phase
         targetCCB.SCSIStatus = whatever status is appropriate

      Execute Target IO (&targetCCB)
- end loop
/* return target CCB to pool           */
- targetCCB.camStatus = request in progress
- EXIT
```

Annex C: Unix OSD Data Structures

(Part of **ANSI Common Access Method rev 2.3**)

Annex C: Unix OSD Data Structures

```
/* ------------------------------------------------------------------------ */

/* cam.h                Version 1.05                    Nov. 07, 1990 */

/*  This file contains the definitions and data structures for the CAM
    Subsystem interface.  The contents of this file should match the
    data structures and constants that were specified in the CAM document,
    CAM/89-003 Rev 2.2.

*/

/* ------------------------------------------------------------------------ */

/* Defines for the XPT function codes, Table 8-2 in the CAM spec. */

/* Common function commands, 0x00 - 0x0F */
#define XPT_NOOP        0x00    /* Execute Nothing */
#define XPT_SCSI_IO     0x01    /* Execute the requested SCSI IO */
#define XPT_GDEV_TYPE   0x02    /* Get the device type information */
#define XPT_PATH_INQ    0x03    /* Path Inquiry */
#define XPT_REL_SIMQ    0x04    /* Release the SIM queue that is frozen */
#define XPT_SASYNC_CB   0x05    /* Set Async callback parameters */
#define XPT_SDEV_TYPE   0x06    /* Set the device type information */

/* XPT SCSI control functions, 0x10 - 0x1F */
#define XPT_ABORT       0x10    /* Abort the selected CCB */
#define XPT_RESET_BUS   0x11    /* Reset the SCSI bus */
#define XPT_RESET_DEV   0x12    /* Reset the SCSI devies, BDR */
#define XPT_TERM_IO     0x13    /* Terminate the I/O process */

/* Target mode commands, 0x30 - 0x3F */
#define XPT_EN_LUN      0x30    /* Enable LUN, Target mode support */
#define XPT_TARGET_IO   0x31    /* Execute the target IO request */

#define XPT_FUNC        0x7F    /* TEMPLATE */
#define XPT_VUNIQUE     0x80    /* All the rest are vendor unique commands */

/* ------------------------------------------------------------------------ */

/* General allocation length defines for the CCB structures. */

#define IOCDBLEN        12      /* Space for the CDB bytes/pointer */
#define VUHBA           16      /* Vendor Unique HBA length */
#define SIM_ID          16      /* ASCII string len for SIM ID */
#define HBA_ID          16      /* ASCII string len for HBA ID */
#define SIM_PRIV        50      /* Length of SIM private data area */

/* Structure definitions for the CAM control blocks, CCB's for the subsystem.
*/

/* Common CCB header definition. */
typedef struct ccb_header
```

```
    struct ccb_header *my_addr; /* The address of this CCB */
    u_short cam_ccb_len;        /* Length of the entire CCB */
    u_char cam_func_code;       /* XPT function code */
    u_char cam_status;          /* Returned CAM subsystem status */
    u_char cam_path_id;         /* Path ID for the request */
    u_char cam_target_id;       /* Target device ID */
    u_char cam_target_lun;      /* Target LUN number */
    u_long cam_flags;           /* Flags for operation of the subsystem */
 CCB_HEADER;

/* Common SCSI functions. */

/* Union definition for the CDB space in the SCSI I/O request CCB */
typedef struct

    u_char *cam_cdb_ptr;              /* Pointer to the CDB bytes to send */
    u_char cam_cdb_bytes[ IOCDBLEN ]; /* Area for the CDB to send */
 CDB_UN;

/* Get device type CCB */
typedef struct

    CCB_HEADER cam_ch;                /* Header information fields */
    u_char cam_pd_type;               /* Periph device type from the TLUN */
    char *cam_inq_data;               /* Ptr to the inquiry data space */
 CCB_GETDEV;

/* Path inquiry CCB */
typedef struct

    CCB_HEADER cam_ch;                /* Header information fields */
    u_long cam_feature_flags;         /* Supported features flags field */
    u_char cam_version_num;           /* Version number for the SIM/HBA */
    u_char cam_hba_inquiry;           /* Mimic of INQ byte 7 for the HBA */
    u_char cam_target_sprt;           /* Flags for target mode support */
    u_char cam_hba_misc;              /* Misc HBA feature flags */
    u_char cam_vuhba_flags[ VUHBA ];  /* Vendor unique capabilities */
    u_long cam_sim_priv;              /* Size of SIM private data area */
    u_long cam_async_flags;           /* Event cap. for Async Callback */
    u_char cam_hpath_id;              /* Highest path ID in the subsystem */
    u_char cam_initiator_id;          /* ID of the HBA on the SCSI bus */
    char cam_sim_vid[ SIM_ID ];       /* Vendor ID of the SIM */
    char cam_hba_vid[ HBA_ID ];       /* Vendor ID of the HBA */
    u_char *cam_osd_usage;            /* Ptr for the OSD specific area */
 CCB_PATHINQ;

/* Release SIM Queue CCB */
typedef struct

    CCB_HEADER cam_ch;                /* Header information fields */
 CCB_RELSIM;

/* SCSI I/O Request CCB */
typedef struct

    CCB_HEADER cam_ch;                /* Header information fields */
    u_char *cam_pdrv_ptr;             /* Ptr used by the Peripheral driver */
```

```
    CCB_HEADER *cam_next_ccb;         /* Ptr to the next CCB for action */
    void (*cam_cbfcnp)();             /* Callback on completion function */
    u_char *cam_data_ptr;             /* Pointer to the data buf/SG list */
    u_long cam_dxfer_len;             /* Data xfer length */
    u_char *cam_sense_ptr;            /* Pointer to the sense data buffer */
    u_short cam_sense_len;            /* Num of bytes in the Autosense buf */
    u_char cam_cdb_len;               /* Number of bytes for the CDB */
    u_short cam_sglist_cnt;           /* Num of scatter gather list entries */
    u_char cam_scsi_status;           /* Returned scsi device status */
    long cam_resid;                   /* Transfer residual length: 2's comp */
    CDB_UN cam_cdb_io;                /* Union for CDB bytes/pointer */
    u_long cam_timeout;               /* Timeout value */
    u_char *cam_msg_ptr;              /* Pointer to the message buffer */
    u_short cam_msgb_len;             /* Num of bytes in the message buf */
    u_short cam_vu_flags;             /* Vendor unique flags */
    u_char cam_tag_action;            /* What to do for tag queuing */
    u_char cam_sim_priv[ SIM_PRIV ];  /* SIM private data area */
 CCB_SCSIIO;

/* Set Async Callback CCB */
typedef struct

    CCB_HEADER cam_ch;                /* Header information fields */
    u_long cam_async_flags;           /* Event enables for Callback resp */
    void (*cam_async_func)();         /* Async Callback function address */
    u_char *pdrv_buf;                 /* Buffer set aside by the Per. drv */
    u_char pdrv_buf_len;              /* The size of the buffer */
 CCB_SETASYNC;

/* Set device type CCB */
typedef struct

    CCB_HEADER cam_ch;                /* Header information fields */
    u_char cam_dev_type;              /* Val for the dev type field in EDT */
 CCB_SETDEV;

/* SCSI Control Functions. */

/* Abort XPT Request CCB */
typedef struct

    CCB_HEADER cam_ch;                /* Header information fields */
    CCB_HEADER *cam_abort_ch;         /* Pointer to the CCB to abort */
 CCB_ABORT;

/* Reset SCSI Bus CCB */
typedef struct

    CCB_HEADER cam_ch;                /* Header information fields */
 CCB_RESETBUS;

/* Reset SCSI Device CCB */
typedef struct

    CCB_HEADER cam_ch;                /* Header information fields */
 CCB_RESETDEV;
```

```
/* Terminate I/O Process Request CCB  */
typedef struct

    CCB_HEADER cam_ch;                /* Header information fields */
    CCB_HEADER *cam_termio_ch;        /* Pointer to the CCB to terminate */
 CCB_TERMIO;

/* Target mode structures. */

typedef struct

    CCB_HEADER cam_ch;
    u_short cam_grp6_len;             /* Group 6 VU CDB length */
    u_short cam_grp7_len;             /* Group 7 VU CDB length */
    u_short cam_ccb_listcnt;          /* Count of Target CCBs in the list */
    u_char *cam_ccb_listptr;          /* Pointer to the target CCB list */
 CCB_EN_LUN;

/* ---------------------------------------------------------------------- */

/* Defines for the CAM status field in the CCB header. */

#define CAM_REQ_INPROG          0x00  /* CCB request is in progress */
#define CAM_REQ_CMP             0x01  /* CCB request completed w/out error */
#define CAM_REQ_ABORTED         0x02  /* CCB request aborted by the host */
#define CAM_UA_ABORT            0x03  /* Unable to Abort CCB request */
#define CAM_REQ_CMP_ERR         0x04  /* CCB request completed with an err */
#define CAM_BUSY                0x05  /* CAM subsystem is busy */
#define CAM_REQ_INVALID         0x06  /* CCB request is invalid */
#define CAM_PATH_INVALID        0x07  /* Path ID supplied is invalid */
#define CAM_DEV_NOT_THERE       0x08  /* SCSI device not installed/there */
#define CAM_UA_TERMIO           0x09  /* Unabel to Terminate I/O CCB req */
#define CAM_SEL_TIMEOUT         0x0A  /* Target selection timeout */
#define CAM_CMD_TIMEOUT         0x0B  /* Command timeout */
#define CAM_MSG_REJECT_REC      0x0D  /* Message reject received */
#define CAM_SCSI_BUS_RESET      0x0E  /* SCSI bus reset sent/received */
#define CAM_UNCOR_PARITY        0x0F  /* Uncorrectable parity error occured */
#define CAM_AUTOSENSE_FAIL      0x10  /* Autosense: Request sense cmd fail */
#define CAM_NO_HBA              0x11  /* No HBA detected Error */
#define CAM_DATA_RUN_ERR        0x12  /* Data overrun/underrun error */
#define CAM_UNEXP_BUSFREE       0x13  /* Unexpected BUS free */
#define CAM_SEQUENCE_FAIL       0x14  /* Target bus phase sequence failure */
#define CAM_CCB_LEN_ERR         0x15  /* CCB length supplied is inadaquate */
#define CAM_PROVIDE_FAIL        0x16  /* Unable to provide requ. capability */
#define CAM_BDR_SENT            0x17  /* A SCSI BDR msg was sent to target */
#define CAM_REQ_TERMIO          0x18  /* CCB request terminated by the host */

#define CAM_LUN_INVALID         0x38  /* LUN supplied is invalid */
#define CAM_TID_INVALID         0x39  /* Target ID supplied is invalid */
#define CAM_FUNC_NOTAVAIL       0x3A  /* The requ. func is not available */
#define CAM_NO_NEXUS            0x3B  /* Nexus is not established */
#define CAM_IID_INVALID         0x3C  /* The initiator ID is invalid */
#define CAM_CDB_RECVD           0x3E  /* The SCSI CDB has been received */
#define CAM_SCSI_BUSY           0x3F  /* SCSI bus busy */

#define CAM_SIM_QFRZN           0x40  /* The SIM queue is frozen w/this err */
#define CAM_AUTOSNS_VALID       0x80  /* Autosense data valid for targit */
```

```
/* ---------------------------------------------------------------------- */

/* Defines for the CAM flags field in the CCB header. */

#define CAM_DIR_RESV       0x00000000 /* Data direction (00: reserved) */
#define CAM_DIR_IN         0x00000040 /* Data direction (01: DATA IN) */
#define CAM_DIR_OUT        0x00000080 /* Data direction (10: DATA OUT) */
#define CAM_DIR_NONE       0x000000C0 /* Data directino (11: no data) */
#define CAM_DIS_AUTOSENSE  0x00000020 /* Disable auto sence feature */
#define CAM_SCATTER_VALID  0x00000010 /* Scatter/gather list is valie */
#define CAM_DIS_CALLBACK   0x00000008 /* Disable callback feature */
#define CAM_CDB_LINKED     0x00000004 /* The CCB contains a linked CDB */
#define CAM_QUEUE_ENABLE   0x00000002 /* SIM queue actions are enabled */
#define CAM_CDB_POINTER    0x00000001 /* The CDB field contains a pointer */

#define CAM_DIS_DISCONNECT 0x00008000 /* Disable disconnect */
#define CAM_INITIATE_SYNC  0x00004000 /* Attempt Sync data xfer, and SDTR */
#define CAM_DIS_SYNC       0x00002000 /* Disable sync, go to async */
#define CAM_SIM_QHEAD      0x00001000 /* Place CCB at the head of SIM Q */
#define CAM_SIM_QFREEZE    0x00000800 /* Return the SIM Q to frozen state */

#define CAM_CDB_PHYS       0x00400000 /* CDB pointer is physical */
#define CAM_DATA_PHYS      0x00200000 /* SG/Buffer data ptrs are physical */
#define CAM_SNS_BUF_PHYS   0x00100000 /* Autosense data ptr is physical */
#define CAM_MSG_BUF_PHYS   0x00080000 /* Message buffer ptr is physical */
#define CAM_NXT_CCB_PHYS   0x00040000 /* Next CCB pointer is physical */
#define CAM_CALLBCK_PHYS   0x00020000 /* Callback func ptr is physical */

#define CAM_DATAB_VALID    0x80000000 /* Data buffer valid */
#define CAM_STATUS_VALID   0x40000000 /* Status buffer valid */
#define CAM_MSGB_VALID     0x20000000 /* Message buffer valid */
#define CAM_TGT_PHASE_MODE 0x08000000 /* The SIM will run in phase mode */
#define CAM_TGT_CCB_AVAIL  0x04000000 /* Target CCB available */
#define CAM_DIS_AUTODISC   0x02000000 /* Disable autodisconnect */
#define CAM_DIS_AUTOSRP    0x01000000 /* Disable autosave/restore ptrs */

/* ---------------------------------------------------------------------- */

/* Defines for the SIM/HBA queue actions.  These value are used in theSCSI I/O
CCB, for the queue action field. [These values should match thedefines from
some other include file for the SCSI message phases.  We maynot need these
definitions here. ] */

#define CAM_SIMPLE_QTAG         0x20           /* Tag for a simple queue */
#define CAM_HEAD_QTAG           0x21           /* Tag for head of queue */
#define CAM_ORDERED_QTAG        0x22           /* Tag for ordered queue */

/* ---------------------------------------------------------------------- */

/* Defines for the timeout field in the SCSI I/O CCB.  At this time a valueof
0xF-F indicates a infinite timeout.  A value of 0x0-0 indicates that theSIM's
default timeout can take effect. */

#define CAM_TIME_DEFAULT        0x00000000     /* Use SIM default value */
#define CAM_TIME_INFINITY       0xFFFFFFFF     /* Infinite timout for I/O */
```

```
/* ---------------------------------------------------------------------- */

/* Defines for the Path Inquiry CCB fields. */

#define CAM_VERSION             0x22  /* Binary value for the current ver */

#define PI_MDP_ABLE             0x80  /* Supports MDP message */
#define PI_WIDE_32              0x40  /* Supports 32 bit wide SCSI */
#define PI_WIDE_16              0x20  /* Supports 16 bit wide SCSI */
#define PI_SDTR_ABLE            0x10  /* Supports SDTR message */
#define PI_LINKED_CDB           0x08  /* Supports linked CDBs */
#define PI_TAG_ABLE             0x02  /* Supports tag queue message */
#define PI_SOFT_RST             0x01  /* Supports soft reset */

#define PIT_PROCESSOR           0x80  /* Target mode processor mode */
#define PIT_PHASE               0x40  /* Target mode phase cog. mode */

#define PIM_SCANHILO            0x80  /* Bus scans from ID 7 to ID 0 */
#define PIM_NOREMOVE            0x40  /* Removable dev not included in scan */

/* ---------------------------------------------------------------------- */

/* Defines for Asynchronous Callback CCB fields. */

#define AC_FOUND_DEVICES        0x80  /* During a rescan new devies found */
#define AC_SIM_DEREGISTER       0x40  /* A loaded SIM has de-registered */
#define AC_SIM_REGISTER         0x20  /* A loaded SIM has registered */
#define AC_SENT_BDR             0x10  /* A BDR message was sent to target */
#define AC_SCSI_AEN             0x08  /* A SCSI AEN has been received */
#define AC_UNSOL_RESEL          0x02  /* A unsolicited reselection occured */
#define AC_BUS_RESET            0x01  /* A SCSI bus RESET occured */

/* ---------------------------------------------------------------------- */

/* Typedef for a scatter/gather list element. */

typedef struct

    u_long cam_sg_address;              /* Scatter/Gather address */
    u_long cam_sg_count;                /* Scatter/Gather count */
 SG_ELEM;

/* ---------------------------------------------------------------------- */

/* Unix OSD defines and data structures. */

#define INQLEN  36              /* Inquiry string length to store. */

/* General Union for Kernal Space allocation.  Contains all the possible
CCBstructures.  This union should never be used for manipulating CCB's its
onlyuse is for the allocation and deallocation of raw CCB space. */

typedef union

    CCB_SCSIIO    csio;         /* Please keep this first, for debug/print
*/
    CCB_GETDEV    cgd;
```

```
    CCB_PATHINQ   cpi;
    CCB_RELSIM    crs;
    CCB_SETASYNC  csa;
    CCB_SETDEV    csd;
    CCB_ABORT     cab;
    CCB_RESETBUS  crb;
    CCB_RESETDEV  crd;
    CCB_TERMIO    ctio;
 CCB_SIZE_UNION;

/* The typedef for the Async callback information.  This structure is used
tostore the supplied info from the Set Async Callback CCB, in the EDT table.
*/

typedef struct

    u_short cam_event_enable;           /* Event enables for Callback resp */
    void (*cam_async_func)();           /* Async Callback function address */
    u_long cam_async_blen;              /* Length of "information" buffer */
    u_char *cam_async_ptr;              /* Address for the "information */
 ASYNC_INFO;

/* The CAM_SIM_ENTRY definition is used to define the entry points forthe SIMs
contained in the SCSI CAM subsystem.  Each SIM file willcontain a declaration
for it's entry.  The address for this entry willbe stored in the cam_conftbl[]
array along will all the other SIMentries. */

typedef struct

    void (*sim_init)();         /* Pointer to the SIM init routine */
    void (*sim_action)();       /* Pointer to the SIM CCB go routine */
 CAM_SIM_ENTRY;

/* The CAM EDT table, this structure contains the device information forall
the devices, SCSI ID and LUN, for all the SCSI busses in the system. */

typedef struct

    long cam_tlun_found;      /* Flag for the existence of the target/LUN */
    ASYNC_INFO cam_ainfo;     /* Async callback info for this B/T/L */
    u_long cam_owner_tag;     /* Tag for the peripheral driver's ownership */
    char cam_inq_data[ INQLEN ];/* storage for the inquiry data */
 CAM_EDT_ENTRY;

/* ---------------------------------------------------------------------- */
```

Annex D: Operating System Vendor Documentation

(Part of **ANSI Common Access Method rev 2.3**)

Annex D: Operating System Vendor Documentation

D.1 Documentation

The following lists those operating system vendors which have agreed to supply information to third party vendors on the support of SCSI devices. This is not a complete list of all vendors that support CAM.

```
[ This is a partial list and as it is the addresses of the CAM members
  it is unlikely to be the right address to contact for information
  and documentation. Would those listed please advise correct addresses
  and department and advise the phone number and fax number? Also, any
  that I should have included, please add yourselves to the list by
  sending Dal a fax at 408-867-2115. ]
```

AT&T BELL LABS
1100 E Warrenville Rd
Naperville
IL 60566

MICROSOFT
POB 97017
Redmond
WA 98073

INTERACTIVE SYSTEMS
2401 Colorado Ave
Santa Monica
CA 90404

NOVELL
122 E 1700 S
Provo
UT 84606

DIGITAL EQUIPMENT
110 Spit Brook Rd
ZKO3-3/T79
Nashua
NH 03062

SUN MICROSYSTEMS
2550 Garcia Ave Bdg 15
Mountain View
CA 94043

IBM
MS 5226
PO Box 1328
Boca Raton
FL 33429

D.2 DOS Background

During the development of XPT/SIM by the CAM Committee, several approaches to support SCSI under DOS were implemented by vendors. Some were proprietary and required licensing agreements while others were parochial. Of the latter, some provided documentation upon request in order to encourage their adoption.

Subsequent to the XPT/SIM being defined across multiple operating systems, IBM made information available on the attachment of SCSI devices using the 4Bh/80h interrupt, which assumes that there is an SCB (Subsystem Control Block) data structure. SCBs are unlike CCBs.

The IBM introduction created a de facto implementation, and IBM has provided a software license at no charge to those companies which write peripheral drivers to support SCBs. The SCBs and their method of operation are defined in Technical Reference Manuals which are available from IBM.

The CAM approach is oriented towards using the CCB data structure to provide a common approach across multiple operating systems. Although it may have been possible to use SCBs to provide this capability, full technical information was not provided early enough in the development cycle.

DRIVING US CRAZY, BUT FOR A REASON
by Alan Brenden

In the early days of the PC, there wasn't much involved with the decision-making process when a new hard disk was to be bought or repaired. The first hard disks used Seagate's ST506 technology and that was your choice.

Times and technology have changed and today's high-performance systems make it neccessary to match the needs of the system to the storage technology. In this article I will try and explain to you just what are these drives that drive us crazy. - ESDI, IDE, and SCSI, when and why.

ST506/412 (MFM & RLL)
Original the ST506 drives used an encoding method know as Modified Frequency Modulation (MFM). As the need for bigger drives evolved a new encoding method w as developed to pack data tighter together. This method is know as Run Length Limited (RLL). This method involves looking at groups of 16bits rather then each individual bit. This achieves a kind of compression of the data that allows roughly 50% more on a disk then MFM. The trade off was that you needed a higher grade of media and timing is more critical. Prices for media have dropped in the last 3 years and RLL drives have just about wiped MFM drives from the market place. ESDI, SCSI, and IDE also use a type of RLL encoding. ST506/412 drives have 2 cables, a 34 pin control cable and a 20 pin data cable. ST506 MFM has a data transfer rate of 625K bytes per second and a storage capacity of 5MB - 100MB. ST506 RLL has a data transfer rate of 937K bytes per second and a storage capacity of 30MB-200MB.

ESDI
ESDI (Enhanced Small-Device Interface) was developed to allow faster transfer rates and high disk capacities. Greater intelligence reduces the amount of communication between the drive and the controller. The transfer of data between the drive and the controller uses a pulse code that isn't required to return to zero between pulses, as does ST506. This is know as Non Return to Zero (NRZ) and increases data transfer. ESDI uses the same cables as the ST506 but can never be mixed. It is CPU controlled and is suitable for single tasking environments. ESDI has a data transfer rate of 1-3M bytes per second and a storage capacity of 80MB - 2GB. One controller can handle up to 2 drives with multiple controllers possible.

IDE
As the name implies, IDE (Integrated Drive Electronics) combines both the disk and the controller on the same unit. Only a simple interface is needed and typically it is built directly into the motherboard. If the interface is not built into the motherboard a simple paddle-board is used and because so little electronics are needed an additional serial and parallel port is sometime included. It transfers only data and doesn't need to send format and sector information as does ESDI. Therefore the data transfer rate can be 3-4 times faster then ESDI.

The IDE drive is not a device level interface and has the ability to lie to the BIOS and give the logical appearance of a known device type, while physically it may be totally different. You won't see bad tracks on an IDE drive because the drive hides them. Because of this you can not low level format an IDE drive without specific utilities for that drive. IDE uses a single 40 pin cable. It is limited to a 2 foot cable and 2 addresses with no termination needed. The first drive is configured as the master and the second as the slave. IDE has a transfer rate of .625MB-2MB bytes per second and a storage capacity of 20MB-500MB.

SCSI

SCSI (Small Computer Systems Interface), pronounced "scuzzy", is a more general version of the IDE interface. SCSI hard disks boast the fastest transfer rates of all the discussed technologies, with SCSI 2 having a transfer rate up to 40M bytes per second. SCSI implements 2 ways of boosting transfer rates, fast and wide. FAST SCSI doubles the clock speed, and WIDE SCSI increases the bus width. SCSI also implements other performance features, including controller based RAM caching and tag command queuing. By queuing commands the SCSI controller can free up the CPU to do other tasks while it finishes its task. SCSI also has the ability to transfer data to another SCSI device without CPU involvement. SCSI uses a single 50 pin cable with devices daisy chained together and terminated on both ends. Seven devices can be installed per controller with up to four controllers. SCSI 1 has a data transfer rate of 1-5M bytes per second and a storage capacity of 20MB-1.5GB. SCSI 2 has a data transfer rate of 1-40M bytes per second and a storage capacity of 40MB-3GB.

WHICH IS BEST?

Performance isn't without price. Many applications don't need the performance of SCSI, it is by far the most expensive. IDE or ESDI will usually suffice for most applications. IDE being the cheapest of the three. SCSI has the added advantage of the greatest expandability, so if you need SCSI the money is well spent.

Archive-name: scsi-faq
Last-modified: 5/13/93
Version: @(#)scsi.faq 1.5

SCSI FAQ:
Frequently Asked Questions for comp.periphs.scsi

Table of contents:

Part of **FAQ**

====
QUESTION: **Is it possible for two computers to access the same SCSI disks?**
ANSWER From: burke@seachg.uucp (Michael Burke)
====

Yes, two (or more) systems can be on the same scsi bus as scsi disk and tape drives. As long as the scsi requirements are met - cable lengths, termination and type - the devices can share the scsi bus.

The question should be - Are there any O/S' that will allow the sharing of file systems? It would not make sense for two hosts to go about treating shared disks as if they owned the device. Data would be destroyed pretty quickly.

On the issue of tape devices, however, O/S' tend to give exclusive usage to an application. In this way, tape drives can be shared much more easily.

Disks can be best shared by having two (or more) partitions on a disk. Each host "owning" it's own file system.

Part of **FAQ**

====
QUESTION: **Where can I get SCSICNTL.EXE and other Adaptec files?**
ANSWER From: randy@psg.com (Randy Bush)
and Timothy Hu timhu@ico.isc.com
====

New files from Roy as follows:

ftp.psg.com:~/pub/adaptec/...

```
-rw-rw-r--  1 randy    staff      110689 Feb 25 00:29 SCSICNTL.EXE.Z
-rw-rw-r--  1 randy    staff      368640 Feb 25 00:27 adse.dd
-rw-rw-r--  1 randy    staff        1959 Feb 25 00:25 adse.dd.readme
-rw-rw-r--  1 randy    staff       17896 Feb 25 00:37 list
-rw-rw-r--  1 randy    staff       99545 Feb 25 00:20 os2drv.zip
-rw-rw-r--  1 randy    staff       70801 Feb 25 00:20 scsi_drv.Z
-rw-rw-r--  1 randy    staff       66508 Feb 25 00:24 scsi_drv.readm
-rw-rw-r--  1 randy    staff      118697 Feb 25 00:17 update.pkg.Z
```

You can get the ASPI specs from Adaptec's Bulletin Board (408)945-7727.

Part of **FAQ**

====
QUESTION: **What kinds of Optical Drives are available?**
ANSWER From: joungwoo@mensa.usc.edu (John Kim)
====

As I promised I am posting the summary of what I learned about 128mb optical drives through many kind replies and some effort on my part. The purpose of this informal survey was to aid people (starting from myself) in deciding on which 128mb optical drive to buy.

When I posted my questions, it was done only on comp.mac.sys.hardware and forgot to do the same also on comp.arch.storage and comp.periphs.scsi where are less traffic than c.m.s.h. However, as a Macintosh owner myself, this survey was biased toward the Mac world and the mail order houses mentioned specializes in Mac-related products, although the below mentioned optical drives might be usable also with non-Mac platforms (Sun, NeXT, PC-compatibles).

My questions were:

o what kind of drive you bought from whom at what price
o what drive mechanism (MOST, Epson, Fujitsu, Sony, ...) it uses
o how fast it is in terms of average seek time & data transfer rate
o how noisy the drive is
o how large and heavy the drive is
o what drive formatting program (eg, FWB or Silver Lining) you use what its goods/bads
o the quality of the service of the seller (mail order company, retail store, etc.)

Summary

In general, these days, some magneto-optical (MO) drives seem to be almost as fast as (if not faster than) ordinary hard drives (HD). The access time of fastest 128MB MO drives (around 30ms) are slower than average HD's access time (15ms) but the transfer rate seems to be about the same (764KBytes/sec) or not much slower. The advantages of the MO drives over the HDs are that your storage space is almost limitless, expandable at a relatively cheap price ($40/120MB = 34 cents/MB) compared to $1/MB rate of HDs or that of SyQuest drives, and the life of the media is very long (they say it's 30 years or rewritable 100,000 times.)

Fijitsu 128 REM Portable: At this moment, to my knowledge, 128 MByte optical drives based on Fujitsu mechanism seems to be the fastest, roughly having average seek time of 30ms and average transfer rate of 768KBytes/sec. These are the most recently introduced; when I called DGR Technology (800-235-9748) and MacProducts USA (800-MAC-USA1), both of which are in Texas, about a week ago they were taking orders for them at the lowest price ($999) (advertised in MacWorld April '93 issue). DGR didn't have them in stock yet and were expecting to have them in quantity in 10 weeks; MacProducts told me it will take 4 to 5 weeks before I get my order. Call them again now -- the situation may have improved. Another good thing about this Fujitsu drive is that it is more compact in size than previous 128mb optical drives, ie, "portable". I don't know how Fujitsu mechanism (FM) is different from Epson mechanism (EM) and how FM provides a similar performance at a cheaper price in a smaller frame. Maybe using split-head implementation to make the read-write head lighter? Could anybody post info on this? One person tells me that the eject mechanism is

too strong, sometimes shooting the cartridge out making them land on the floor. He says Fujitsu told him that the FM's coming out in April will have gentler eject.

Epson: The next fastest (or maybe just about the same speed) are Epson mechanism (EM) drives, having average access time of 34ms and transfer rate of 768KB/sec. These achieve faster speed compared to other old mechanisms by having a higher rpm (3600rpm vs. past 2400 rpm). MacDirect (selling products labelled NuDesign, 800-621-8467, in Chicago, IL) and DGR advertises to carry them and currently sells them at $1098 and these are available right now. Folks who used these seemed to be very satisfied.

Slower ones: Other mechanisms (Sony, Panasonic, etc.) seem to have been dominating the optical drive market before FM and EM's advent. These have a typical access rate of ~45ms. I don't know if now there are new implementations that make them perform better then FM and EM. Maybe someone can tell us.

My current plan is to wait several weeks and buy an FM ($999) when they get to have them in stock. Since MacDirect has been the leader in providing cheapest price they may start selling FMs at a competitive price soon. At the moment MacDirect doesn't seem to carry FMs yet. MacProducts USA is also a good place to look for good prices. These three places (DGR, Mac Products USA and MacDirect) are all advertised in MacWorld and MacUser.

Noise Level: One thing to consider might be noise of the drive. Different mechanisms may have typical noise level, but one thing sure is that different resellers/companys' drive's noise level differs even for the same drive mechanism, eg, Sony. It looks like different casing produces different noise levels? (Could someone confirm/disconfirm this aspect?) Base on the report in Nov '92 issue of MacWorld, the noise level of MacDirect, MacProduct and DGR 128mb MO drives seem to be OK or quite quiet.

This issue of MacWorld deals with removable media drives (optical drives of various capacity, SyQuest, Bernoulli and Flopticals) and you can get some idea on what the differences among different drive mechanisms are.

Price of Media: Usual price per 128mb cartridge was $49 - $59. But ClubMac (800-258-2650, in California) was the cheapest place to buy -- $39/128mb cartridge. If you live in CA, due to sales tax, Computer Design & Graphic Systems (800-741-6227) (in Ft. Myers, Florida) might be your cheapest: $40.

Formatting Software: Another thing to consider is what kind of media formatting software you will use. All companys (or mail order places) seemed to provide for free formatting program with their drives. I don't have the details on this. But an inefficient formatting can result in slow drive performance. The most popular one used to be FWB's Hard Disk Toolkit but Anubis (advertised to improve performance up to 35% [compared to what?]) is beginning to be used also. I don't know if all formatting program and the drive hardware allows to have read and write verify off but by having these turned off you can obtain significant speed boost at the risk of less secure data transfer. MacWorld's report warned that drives from some companys don't let you turn on/off the verify. In the worst case, some come with verify off and no option to toggle it back to ON.

256mb MO drives: In general these have better transfer rate (1.23MB/sec) and a little slower access time (35ms). I feel that this capacity will soon be the next standard. These drives are able to also read/write 128mb cartridges and 256mb will soon be new ANSI and ISO standard. I once heard from a saleperson at a mail order place that these are not reliable yet and he saw many they sold came back with complaints. This may be a non-general instance on a typical drive mechanism (seems to be MOST mechanism). Personally, I feel 128mb is accomodating enough for personal usage at home unless you are dealing with very large

data files (eg, large graphic images).

Part of **FAQ**

====
QUESTION: **Where can I get FTP/download SCSI documents?**
ANSWER From: news@mgse.UUCP (News Administator)
====

Last Changed: Thu Sep 24 23:31:09 CDT 1992 (New BBS Phone number)

This is a periodic posting of information about some of the archives at ftp.cs.tulane.edu and the available files from the SCSI-BBS, including SCSI, ESDI, IPI, and Fiber Channel documents from the standards committees.

These files are available for FTP from ftp.cs.tulane.edu in the directory pub/scsi. Files are stored in file areas as they are found in the BBS with each area having a file named 'files.bbs' that tells what each file is. The file pub/scsi/index.Z list each file area, it's descriptions and it's files.

Thanks to John Lohmeyer of NCR, a majority of the SCSI related files from the SCSI BBS are now available for anonymous ftp. These files were sent to me by Mr. Lohmeyer at his expense so that more people would have access to them. The SCSI BBS (719-574-0424) contains a large amount of data relating to SCSI, and ESDI as well as SCSI-2, IPI, and Fiber Channel, as well as the last revision of the SCSI-1 standard before it went to publication by ANSI.

Most of the files in the SCSI archive are either archived with the ZIP utility or compressed with the 'compress' program. Most of the text files are stored as Wordstar word processing files. PKzip for PC/MS-DOS is included in the archive to allow users to break up the .ZIP files, and the PC/MS-DOS binaries and .C source are also in the archive to convert the Wordstar documents to ASCII text.

Part of **FAQ**

====
QUESTION: **What is the telephone number of Archive Corporation?**
ANSWER From: jdp@caleb.UUCP (Jim Pritchett)
====
Archive Corporation (800) 537 2248
Tech Support (800) 227 6296

====
QUESTION: **What is the telephone number for Quantum?**
ANSWER From: paladin@world.std.com (Thomas G Schlatter)
====
Quantum:
BBS? (408) 434-1664
FAX (408) 943-0689
Tech support (408) 432-1100

====
QUESTION: **What is the telephone number for Seagate?**
ANSWER From: landis@sugs.tware.com (Hale Landis)
====

Here are the numbers for Seagate's Technical Support.

SeaBOARD - Bulletin Board System available 24 hours. Use 8 data bits, no parity, 1 stop bit (8-N-1).

```
USA/Canada  408-438-8771    9600 baud*
England     44-62-847-8011  9600 baud*
Germany     49-89-140-9331  2400 baud*
Singapore   65-292-6973     9600 baud*
Australia   61-2-756-2359   9600 baud*
```

* - Maximum baud rate supported.

SeaFAX 408-438-2620

Use a touch-tone phone to have information returned to you via FAX. Available 24 hours.

Technical Support Fax 408-438-8137

FAX your questions or comments 24 hours. Responses are sent between 8:00AM and 5:00PM PST Monday through Friday.

SeaFONE 408-438-8222

Provides recorded information 24 hours or talk to a technical specialist between 8:00AM to 5:00PM PST Monday through Friday.

SeaTDD 408-438-5382

Using a Telecommunications Device for the Deaf, you can send questions or comments 24 hours or have a dialog with a technical support specialist between 8:00AM and 5:00PM PST

Monday through Friday.

====
QUESTION: **What is the telephone number and address of Conner Peripherals?**
ANSWER From: ekrieger@quasar.hacktic.nl (Eric Krieger)
====

CONNER PERIPHERALS, Incorporated
3081 Zanker Road
San Jose CA 95134
BBS LINE: (408)456-4415 (V.32)
CONNER

WATTS LINE:
PAY LINE: (408)456-4500
FAX LINE:
(408)456-3200

====
QUESTION: **What is the address and telephone number of WANGTEK?**
ANSWER From: "Terry Kennedy, Operations Mgr" <uunet!spcvxa.spc.edu!TERRY>
====

Wangtek can be reached at:

WANGTEK Incorporated
41 Moreland Road
Simi Valley, CA 93065
(805) 583-5255 [voice]
(805) 583-8249 [FAX]
(805) 582-3370 [BBS]

WANGTEK-Europe
Unit 1A, Apollo House
Calleva Industrial Park
Aldermaston, Reading
RG7 4QW England
(44) 734-811463 [voice]
(44) 734-816076 [FAX]
851-848135 [telex]

====
QUESTION: **What is the number for NCR?**
ANSWER From: gkendall@ncr-mpd.FtCollinsCO.NCR.COM (Guy Kendall)
====

For data manuals for any NCR chips, please call 800-334-5454 or 719-630-3384.

Part of **FAQ**

====
QUESTION: **what is FAST SCSI?**
ANSWER From: kev@hpcpbla.bri.hp.com (Kevin Jones)
====

There are 2 handshaking modes on the SCSI bus, used for transferring data: ASYNCHRONOUS and SYNCHRONOUS. ASYNCHRONOUS is a classic Req/Ack handshake. SYNCHRONOUS is "sort of" Req/Ack, only it allows you to issue multiple Req's before receiving Ack's. What this means in practice is that SYNCHRONOUS transfers are approx 3 times faster than ASYNCHRONOUS.

SCSI1 allowed asynchronous transfers at up to 1.5 Mbytes/Sec and synchronous transfers at up to 5.0 Mbytes/Sec.

SCSI2 had some of the timing margins "shaved" in order that faster handshaking could occur. The result is that asynchronous transfers can run at up to 3.0 Mbytes/Sec and synchronous transfers at up to 10.0 Mbytes/Sec. The term "FAST" is generally applied to a SCSI device which can do syncrhonous transfers at speeds in excess of 5.0 Mbytes/Sec. This term can only be applied to SCSI2 devices since SCSI1 didn't have the timing margins that allow for FAST transfers.

Part of **FAQ**

====
QUESTION: **Where can I get SCSI documents?**
ANSWER From: kev@hpcpbla.bri.hp.com (Kevin Jones)
and jmatrow@donald.WichitaKS.NCR.COM (John Matrow
====

The only literature that I'm aware of is:

The SCSI specification: Available from:

Global Engineering Documents
15 Inverness Way East
Englewood Co 80112-5704
(800) 854-7179
SCSI-1: X3.131-1986
SCSI-2: X3.131-199x
SCSI-3 X3T9.2/91-010R4 Working Draft

(Global Engineering Documentation in Irvine, CA (714)261-1455??)

SCSI-1: Doc # X3.131-1986 from ANSI, 1430 Broadway, NY, NY 10018

IN-DEPTH EXPLORATION OF SCSI can be obtained from Solution Technology, Attn: SCSI Publications, POB 104, Boulder Creek, CA 95006, (408)338-4285, FAX (408)338-4374

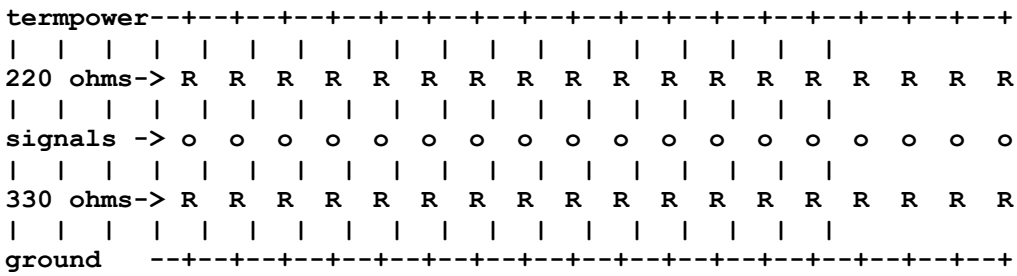
THE SCSI ENCYLOPEDIA and the SCSI BENCH REFERENCE can be obtained from ENDL Publishing, 14426 Black Walnut Ct., Saratoga, CA 95090, (408)867-6642, FAX (408)867-2115

SCSI: UNDERSTANDING THE SMALL COMPUTER SYSTEM INTERFACE was published by Prentice-Hall, ISBN 0-13-796855-8

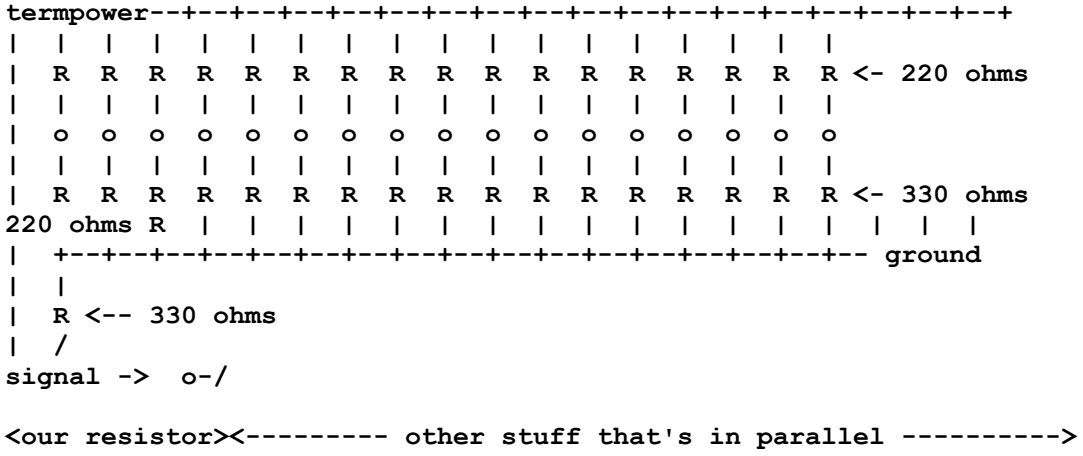
Part of **FAQ**

====
QUESTION: **SCSI terminators should measure 136 ohms?136 ohms?**
ANSWER From: stevel@coos.dartmouth.edu (Steve Ligett)
====

Yes, that is what you should measure. Let's see how that is so. The terminator contains 18 220-ohm resistors from signals to termpower, and 18 330-ohm resistors from those signals to ground. I've drawn that below:



When you measure from any one signal to termpower, you aren't measuring that resistor in isolation, you are measuring that resistor IN PARALLEL with the combination of the corresponding 330 ohm resistor plus 17 220+330 ohm resistor pairs in series. I've redrawn the schematic to make this easier to see:



<our resistor><--------- other stuff that's in parallel ---------->

We're trying to measure that one resistor from a signal to termpower, but there's a ton of other stuff in parallel. The resistance of that "stuff" is 330 + 550/17 ohms (the 330 ohm resistor, in series with a parallel combination of 17 550 ohm resistors). The general formula for the equivalent of two resistances in parallel is r1*r2/(r1+r2). Whipping out my trusty spreadsheet, I find that the "stuff" has a resistance of about 362 ohms, and that in parallel with 220 ohms is about 137 ohms.

Part of FAQ

====
QUESTION: **What are the pinouts for SCSI connectors?**
From: snively@scsi.Eng.Sun.COM (Bob Snively)
====

Originally dated May 23, 1990

The connector families described by the drawings have standard pin numberings which are described the same way by all vendors that I have encountered. The SCSI-2 specification identifies the standard numbering, using that convention. It happened to be documented by AMP, but all the vendors use the same convention.

The following diagrams have the outline drawings of connector sockets at the bottom. This is really for reference only, because the connector sockets and plugs are both specified as to their numbering and usually are labeled.

There are some minor problems in naming the microconnector conductor pairs, which I have corrected in the enclosed diagram. All the conductor pairs of the Mini-Micro (High Density) connector are in fact passed through on the cables. SCSI-2 defines the RSR (Reserved) lines as maybe ground or maybe open, but they are still passed through the cable. Most present standard SCSI devices will ground those lines.

---------- microSCSI to SCSI Diagram ----------

SCSI Connector Pinouts

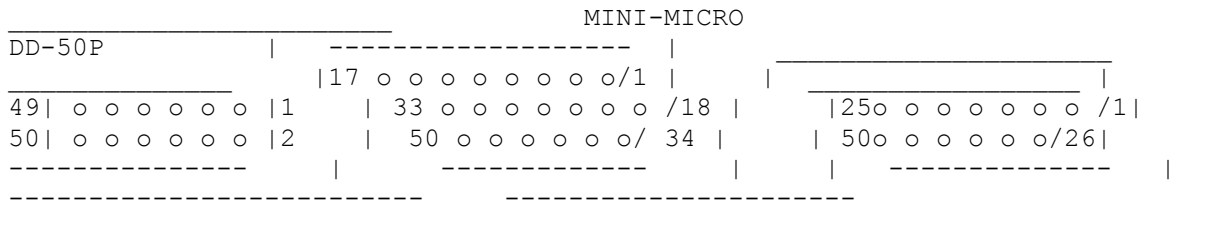
SCSI SIGNAL	DD-50P	MINI MICRO	DD-50SA		SCSI SIGNAL	DD-50P	MINI MICRO	DD-50SA
-DB(0)	2	26	34		GND	1	1	1
-DB(1)	4	27	2		GND	3	2	18
-DB(2)	6	28	19		GND	5	3	35
-DB(3)	8	29	36		GND	7	4	3
-DB(4)	10	30	4		GND	9	5	20
-DB(5)	12	31	21		GND	11	6	37
-DB(6)	14	32	38		GND	13	7	5
-DB(7)	16	33	6		GND	15	8	22
-DB(P)	18	34	23		GND	17	9	39
GND	20	35	40		GND	19	10	7
GND	22	36	8		GND	21	11	24
RSR	24	37	25		RSR	23	12	41
TERMPWR	26	38	42		OPEN	25	13	9
RSR	28	39	10		RSR	27	14	26
GND	30	40	27		GND	29	15	43
-ATN	32	41	44		GND	31	16	11
GND	34	42	12		GND	33	17	28
BSY	36	43	29		GND	35	18	45
-ACK	38	44	46		GND	37	19	13
-RST	40	45	14		GND	39	20	30
-MSG	42	46	31		GND	41	21	47
-SEL	44	47	48		GND	43	22	15
-C/D	46	48	16		GND	45	23	32

| -REQ | 48 | 49 | 33 | | GND | 47 | 24 | 49 |
| -I/O | 50 | 50 | 50 | | GND | 49 | 25 | 17 |

* NC = NOT CONNECTED

CONNECTOR TYPES:

DD-50SA



(VIEWED FROM FACE OF CONNECTOR - USE VENDOR NUMBERING SYSTEM AS SPECIFIED)

Part of **FAQ**

====
QUESTION: **what is the difference between SCSI-1 and SCSI-2?**
ANSWER From Dal Allen:
====

SCSI-1_versus_SCSI-2

In 1985, when the first SCSI standard was being finalized as an American National Standard, the X3T9.2 Task Group was approached by a group of manufacturers. The group wanted to increase the mandatory requirements of SCSI and to define further features for direct-access devices. Rather than delay the SCSI standard, X3T9.2 formed an ad hoc group to develop a working paper that was eventually called the Common Command Set (CCS). Many products were designed to this working paper.

In parallel with the development of the CCS working paper, X3T9.2 sought permission to begin working on an enhanced SCSI standard, to be called SCSI-2. SCSI-2 would include the results of the CCS working paper, caching commands, performance enhancement features, and whatever else X3T9.2 deemed worthwhile. While SCSI-2 was to go beyond the original SCSI standard (now referred to as SCSI-1), it was to retain a high degree of compatibility with SCSI-1 devices.

How is SCSI-2 different from SCSI-1?

1. Several options were removed from SCSI-1:

a. Single initiator option was removed.
b. Non-arbitrating Systems option was removed.
c. Non-extended sense data option was removed.
d. Reservation queuing option was removed.
e. The read-only device command set was replaced by the CD-ROM command set.
f. The alternative 1 shielded connector was dropped.

2. There are several new low-level requirements in SCSI-2:

a. Parity must be implemented.
b. Initiators must provide TERMPWR -- Targets may provide TERMPWR.
c. The arbitration delay was extended to 2.4 us from 2.2 us.
d. Message support is now required.

3. Many options significantly enhancing SCSI were added:

a. Wide SCSI (up to 32 bits wide using a second cable)
b. Fast SCSI (synchronous data transfers of up to 10 Mega-transfers per second -- up to 40 MegaBytes per second when combined with wide SCSI)
c. Command queuing (up to 256 commands per initiator on each logical unit)
d. High-density connector alternatives were added for both shielded and non- shielded connectors.
e. Improved termination for single-ended buses (Alternative 2)
f. Asynchronous event notification
g. Extended contingent allegiance
h. Terminate I/O Process messaging for time- critical process termination

4. New command sets were added to SCSI-2 including:

a. CD-ROM (replaces read-only devices)
b. Scanner devices
c. Optical memory devices (provides for write-once, read-only, and erasable media)
d. Medium changer devices
e. Communications devices

5. All command sets were enhanced:

a. Device Models were added
b. Extended sense was expanded to add:
+ Additional sense codes
+ Additional sense code qualifiers
+ Field replaceable unit code
+ Sense key specific bytes

c. INQUIRY DATA was expanded to add:
+ An implemented options byte
+ Vendor identification field
+ Product identification field
+ Product revision level field
+ Vital product data (more extensive product reporting)

d. The MODE SELECT and MODE SENSE commands were paged for all device types
e. The following commands were added for all device types:

+ CHANGE DEFINITION
+ LOG SELECT
+ LOG SENSE
+ READ BUFFER
+ WRITE BUFFER

f. The COPY command definition was expanded to include information on how to handle inexact block sizes and to include an image copy option. g. The direct-access device command set was enhanced as follows:

+ The FORMAT UNIT command provides more control over defect management
+ Cache management was added:
- LOCK/UNLOCK CACHE command
- PREFETCH command
- SYNCHRONIZE CACHE command
- Force unit access bit
- Disable page out bit

+ Several new commands were added:
- READ DEFECT DATA
- READ LONG
- WRITE LONG
- WRITE SAME

+ The sequential-access device command set was enhanced as follows:

- Partitioned media concept was added:
* LOCATE command
* READ POSITION command

- Several mode pages were added
- Buffered mode 2 was added
- An immediate bit was added to the WRITE FILEMARKS command

+ The printer device command set was enhanced as follows:
- Several mode pages defined:
* Disconnect/reconnect
* Parallel printer
* Serial printer
* Printer options

+ The write-once (optical) device command set was enhanced by:
- Several new commands were added:
* MEDIUM SCAN
* READ UPDATED BLOCK
* UPDATE BLOCK

- Twelve-byte command descriptor blocks were defined for several commands to accommodate larger transfer lengths.

==========

The following article was written by Dal Allan of ENDL in April 1990. It was published nine months later in the January 1991 issue of "Computer Technology Review". While it appeared in the Tape Storage Technology Section of CTR, the article is general in nature and tape-specific. In spite of the less than timely publication, most of the information is still valid.

It is reprinted here with the permission of the author. If you copy this article, please include this notice giving "Computer Technology Review" credit for first publication.

What's New in SCSI-2

Scuzzy is the pronunciation and SCSI (Small Computer System Interface) is the acronym, for the best known and most widely used ANSI (American National Standards Institute) interface.

Despite use of the term "Small" in its name, everyone has to agree that Scuzzy is large - in use, in market impact, in influence, and unfortunately, in documentation. The standards effort that began with a 20-page specification in 1980 has grown to a 600 page extravaganza of technical information.

Even before ANSI (American National Standards Institute) published the first run of SCSI as standards document in 1986, ASC (Accredited Standards Committee) X3T9.2 was hard at work on SCSI-2.

No technical rationale can be offered as to why SCSI-1 ended and SCSI-2 began, or as to why SCSI-2 ended and SCSI-3 began. The justification is much more simple - you have to stop sometime and get a standard printed. Popular interfaces never stop evolving, adapting, and

expanding to meet more uses than originally envisaged.

Interfaces even live far beyond their technological lifespan. SMD (Storage Module Drive) has been called technically obsolete for 5 years but every year there are more megabytes shipped on the SMD interface than the year before. This will probably continue for another year or so before the high point is reached, and it will at least a decade before SMD is considered to be insignificant.

If SCSI enhancements are cut off at an arbitrary point, what initiates the decision? Impatience is as good an answer as any. The committee and the market get sick of promises that the revision process will "end soon," and assert pressure to "do it now."

The SCSI-3 effort is actively under way right now, and the workload of the committee seems to be no less than it was a year ago. What is pleasant, is that the political pressures have eased.

There is a major difference between the standards for SCSI in 1986 and SCSI- 2 in 1990. The stated goal of compatibility between manufacturers had not been achieved in SCSI in 1986 due to a proliferation of undocumented "features."

Each implementation was different enough that new software drivers had to be written for each device. OEMs defined variations in hardware that required custom development programs and unique microcode. Out of this diversity arose a cry for commonality that turned into CCS (Common Command Set), and became so popular that it took on an identity of its own.

CCS defined the data structures of Mode Select and Mode Sense commands, defect management on the Format command and error recovery procedures. CCS succeeded because the goals were limited, the objectives clear and the time was right.

CCS was the beginning of SCSI-2, but it was only for disks. Tape and optical disks suffered from diversity, and so it was that the first working group efforts on SCSI-2 were focused on tapes and optical disks. However, opening up a new standards effort is like lifting the lid on Pandora's Box - its hard to stay focused on a single task. SCSI-2 went far beyond extending and consolidating CCS for multiple device types.

SCSI-2 represents three years of creative thought by some of the best minds in the business. Many of the new features will be useful only in advanced systems; a few will find their way into the average user's system. Some may never appear in any useful form and will atrophy, as did some original SCSI features like Extended Identify.

Before beginning coverage of "what's new in SCSI-2," it might be well to list some of the things that aren't new. The silicon chips designed for SCSI are still usable. No new features were introduced which obsolete chips. The cause of silicon obsolescence has been rapid market shifts in integrating functions to provide higher performance.

Similarly, initiators which were designed properly, according to SCSI in 1986, will successfully support SCSI-2 peripherals. However, it should be pointed out that not all the initiators sold over the last few years behaved according to the standard, and they can be "blown away "by SCSI-2 targets.

The 1986 standard allows either initiators or targets to begin negotiation for synchronous transfers, and requires that both initiators and targets properly handle the sequence. A surprisingly large percentage of SCSI initiators will fail if the target begins negotiation. This has not been as much of a problem to date as it will become in the future, and you know as

well as I do, that these non-compliant initiators are going to blame the SCSI-2 targets for being "incompatible."

Quirks in the 1986 standard, like 4 bytes being transferred on Request Sense, even if the requested length was zero have been corrected in SCSI-2. Initiators which relied on this quirk instead of requesting 4 bytes will get into trouble with a SCSI-2 target.

A sincere effort has been made to ensure that a 1986-compliant initiator does not fail or have problems with a SCSI-2 target. If problems occur, look for a non-compliant initiator before you blame the SCSI-2 standard.

After that little lecture, let us turn to the features you will find in SCSI-2 which include:

o Wide SCSI: SCSI may now transfer data at bus widths of 16 and 32 bits. Commands, status, messages and arbitration are still 8 bits, and the B-Cable has 68 pins for data bits. Cabling was a confusing issue in the closing days of SCSI-2, because the first project of SCSI-3 was the definition of a 16- bit wide P-Cable which supported 16-bit arbitration as well as 16-bit data transfers. Although SCSI-2 does not contain a definition of the P-Cable, it is quite possible that within the year, the P-Cable will be most popular non-SCSI-2 feature on SCSI-2 products. The market responds to what it wants, not the the arbitrary cutoffs of standards committees.

o Fast SCSI: A 10 MHz transfer rate for SCSI came out of a joint effort with the IPI (Intelligent Peripheral Interface) committee in ASC X3T9.3. Fast SCSI achieves 10 Megabytes/second on the A-Cable and with wider data paths of 16- and 32-bits can rise to 20 Megabytes/second and even 40 Megabytes/second. However, by the time the market starts demanding 40 Megabytes/second it is likely that the effort to serialize the physical interface for SCSI-3 will attract high-performance SCSI users to the Fiber Channel.

A word of caution. At this time the fast parameters cannot be met by the Single Ended electrical class, and is only suitable for Differential. One of the goals in SCSI-3 is to identify the improvements needed to achieve 10 MHz operation with Single Ended components.

o Termination: The Single Ended electrical class depends on very tight termination tolerances, but the passive 132 ohm termination defined in 1986 is mismatched with the cable impedance (typically below 100 ohms). Although not a problem at low speeds when only a few devices are connected, reflections can cause errors when transfer rates increase and/or more devices are added. In SCSI-2, an active terminator has been defined which lowers termination to 110 ohms and is a major boost to system integrity.

o Bus Arbitration, Parity and the Identify Message were options of SCSI, but are required in SCSI-2. All but the earliest and most primitive SCSI implementations had these features anyway, so SCSI-2 only legitimizes the de facto market choices. The Identify message has been enhanced to allow the target to execute processes, so that commands can be issued to the target and not just the LUNs.

o Connectors: The tab and receptacle microconnectors chosen for SCSI-2 are available from several sources. A smaller connector was seen as essential for the shrinking form factor of disk drives and other peripherals. This selection was one of the most argued over and contentious decisions made during SCSI-2 development.

o Rotational Position Locking: A rose by any other name, this feature defines synchronized spindles, so than an initiator can manage disk targets which have their spindles locked in a known relative position to each other. Synchronized disks do not all have to be at Index, they can be set to an offset in time relative to the master drive. By arraying banks of

synchronized disks, faster transfer rates can be achieved.

o Contingent Allegiance: This existed in SCSI-1, even though it was not defined, and is required to prevent the corruption of error sense data. Targets in the Contingent Allegiance state reject all commands from other initiators until the error status is cleared by the initiator that received the Check Condition when the error occurred.

Deferred errors were a problem in the original SCSI but were not described. A deferred error occurs in buffered systems when the target advises Good Status when it accepts written data into a buffer. Some time later, if anything goes wrong when the buffer contents are being written to the media, you have a deferred error.

o Extended Contingent Allegiance (ECA): This extends the utility of the Contingent Allegiance state for an indefinite period during which the initiator that received the error can perform advanced recovery algorithms.

o Asynchronous Event Notification (AEN): This function compensates for a deficiency in the original SCSI which did not permit a target to advise the initiator of asynchronous events such as a cartridge being loaded into a tape drive.

o Mandatory Messages: The list of mandated messages has grown:

Both	Target	Initiator
Identify	Abort	Disconnect
Message Reject	No Operation	Restore Pointer
Message Parity Error	Bus Device Reset	Save Data Pointer
	Initiator Detected Error	

o Optional messages have been added to negotiate wide transfers and Tags to support command queueing. A last-minute inclusion in SCSI-2 was the ability to Terminate I/O and receive the residue information in Check Condition status (so that only the incomplete part of the command need be re-started by the initiator).

o Command Queueing: In SCSI-1, initiators were limited to one command per LUN e.g. a disk drive. Now up to 256 commands can be outstanding to one LUN. The target is allowed to re-sequence the order of command execution to optimize seek motions. Queued commands require Tag messages which follow the Identify.

o Disk Cacheing: Two control bits are used in the CDB (Command Descriptor Block) to control whether the cache is accessed on a Read or Write command, and some commands have been added to control pre-fetching and locking of data into the cache. Users do not have to change their software to take advantage of cacheing, however, as the Mode Select/Mode Sense Cache page allows parameters to be set which optimize the algorithms used in the target to maximize cache performance. Here is another area in which improvements have already been proposed in SCSI-3, and will turn up in SCSI-2 products shipping later this year.

o Sense Keys and Sense Codes have been formalized and extended. A subscript byte to the Sense Code has been added to provide specifics on the type of error being reported. Although of little value to error recovery, the additional information about error causes is

useful to the engineer who has to analyze failures in the field, and can be used by host systems as input to prognostic analysis to anticipate fault conditions.

o Commands: Many old commands have been reworked and several new commands have been added.

o Pages: Some method had to be found to pass parameters between host and target, and the technique used is known as pages. The concept was introduced in CCS and has been expanded mightily in SCSI-2.

A number of new Common Commands have been added, and the opcode space for 10-byte CDBs has been doubled.

o Change Definition allows a SCSI-2 initiator to instruct a SCSI-2 target to stop executing according to the 1986 standard, and provide advanced SCSI- 2 features. Most SCSI-2 targets will power on and operate according to the 1986 standard (so that there is no risk of "disturbing" the installed initiators, and will only begin operating in SCSI-2 mode, offering access to the advanced SCSI-2 capabilities, after being instructed to do so by the initiator using the Change Definition command.

o The Mode Select and Mode Sense pages which describe parameters for operation have been greatly expanded, from practically nothing in 1986 to hundreds of items in SCSI-2. Whenever you hear of something being described as powerful and flexible tool, think complicated. Integrators are advised to be judicious in their selection of the pages they decide to support.

o the Inquiry command now provides all sorts of interesting data about the target and its LUNs. Some of this is fixed by the standard, but the main benefit may be in the Vendor Unique data segregated into the special designation of Vital Product Data, which can be used by integrators as a tool to manage the system environment.

o Select Log and Sense Log have been added so that the initiator can gather both historical (e.g. all Check Conditions) and statistical (e.g. number of soft errors requiring ECC) data from the target.

o Diagnostic capabilities have been extended on the Read/Write Buffer and Read/Write Long commands. The ways in which the target can manage bad blocks in the user data space have been defined further and regulated to reduce inconsistencies in the 1986 standard. A companion capability to Read Defect Data permits the initiator to use a standard method to be advised of drive defect lists.

o A new group of 12-byte command blocks has been defined for all optical devices to support the large volume sizes and potentially large transfer lengths. The Erase command has been added for rewritable optical disks so that areas on the media can be pre-erased for subsequent recording. Write Once disks need Media Scan, so that the user can find blank areas on the media.

o New command sets have been added for Scanners, Medium Changers, and CD ROMs.

All of this technical detail can get boring, so how about some "goodies" in SCSI-2 which benefit the common man and help the struggling engineer? First, and probably the best feature in SCSI-2 is that the document has been alphabetized. No longer do you have to embark on a hunt for the Read command because you cannot remember the opcode.

In the 1986 standard, everything was in numeric sequence, and the only engineers who

could find things easily were the microprogrammers who had memorized all the message and opcode tables. Now, ordinary people can find the Read command because it is in alphabetic sequence. This reorganization may sound like a small matter but it wasn't, it required a considerable amount of effort on the part of the SCSI-2 editors. It was well worth it.

Another boon is the introduction for each device class of models which describe the device class characteristics. The tape model was the most needed, because various tape devices use the same acronym but with different meanings or different acronyms for the same meaning.

The SCSI-2 tape model defines the terms used by SCSI-2, and how they correspond to the acronyms of the different tapes. For example, on a 9-track reel, End of Tape is a warning, and there is sufficient media beyond the reflective spot to record more data and a trailer. Not so on a 1/4" tape cartridge, End of Tape means out of media and no more data can be written. This sort of difference in terms causes nightmares for standardization efforts.

So there it is, a summary of what is in SCSI-2. Its not scary, although it is daunting to imagine plowing through a 600-page document. Time for a commercial here. The "SCSI Bench Reference" available from ENDL Publications (408-867-6642), is a compaction of the standard. It takes the 10% of SCSI-2 which is constantly referenced by any implementor, and puts it in an easy- to-use reference format in a small handbook. The author is Jeff Stai, one of the earliest engineers to become involved with SCSI implementation, and a significant contributor to the development of both the 1986 standard and SCSI-2.

SCSI-2 is not yet published as a standard, but it will be available later this year. Until then, the latest revision can be purchased from Global Engineering (800-854-7179).

Biography

Consultant and analyst I. Dal Allan is the founder of ENDL and publisher of the ENDL Letter and the "SCSI Bench Reference." A pioneer and activist in the development and use of standard interfaces, he is Vice Chairman of ASC X3T9.2 (SCSI) and Chairman of the SCSI-2 Common Access Method Committee.

Part of **FAQ**

====
QUESTION: **Is SYNCHRONOUS faster than ASYNCHRONOUS?**
QUESTION: **Is the 53C90 Faster than spec?**
From: kstewart@ncr-mpd.FtCollins.NCR.COM (Ken Stewart)
====

I've seen a few comments about our 54C90 being faster than spec. While I doubt the author was really complaining (I got twice as much as I paid for--sure makes me mad ;) I'd like to explain the situation.

Along the way, I'll also show that asynchronous is faster on short cables, while synchronous is faster on long cables. The cross-over point occurs somewhere around six feet--assuming that you have our 53C90 family devices at both ends of the cable. The reason has to do with the propagation delay of the cable; the turn around time of the silicon; and the interlocked nature of the asynchronous handshake.

1) We have measured propagation delays from various cables and found an average of 1.7 nanoseconds per foot, which is roughly 5.25 ns per meter.

2) The turn-around time is the amount of time the SCSI chip takes to change an output in response to an input. If REQ is an input then ACK is an output. Or if ACK is an input then REQ is an output. Typical turn-around time for the 53C90 is 40 nanoseconds.

3) The asynchronous transfer uses an interlocked handshake where a device cannot do the next thing until it receives positive acknowledgment that the other device received the last thing.

```
First REQ goes true                          /* driven by Target */
then ACK is permitted to go true             /* driven by Initiator */
then REQ is permitted to go false
then ACK is permitted to go false
```

Thus we have four "edges" propagating down the cable plus 4 turn-around delays. Asynchronous transfer requires 55 ns setup and no hold time (paragraph in 5.1.5.1 in SCSI-1 or SCSI-2) which gives an upper speed limit around 18 MB/s. A detailed analysis (assuming 53C90 family) shows that the setup time subtracts out. This is mostly because we are running at one-third the max rate, but also because setup for the next byte can begin anytime after ACK is received true or REQ is received false, depending on who is receiving. You can either take my word for it or draw the waveforms yourself. Thus, the asynchronous transfer reduces to:

```
(4 * 1.7 * 1) + (4 * 40ns) = 167 ns          /* 1 foot cable */
= 6 MB/s

(4 * 5.25 * 6) + (4 * 40ns) = 286 ns         /* 6 meter cable */
= 3.5 MB/s

(4 * 5.25 * 25) + (4 * 40ns) = 685 ns        /* 25 meter cable */
= 1.5 MB/s
```

note: cables longer than 6 meters require external differential transceivers which add delay and degrade the performance even more than indicated here.

Our simulations say that under very best conditions (fast silicon, low temperature, high voltage, zero length cable) we can expect more than 8 MB/s asynchronously. In the lab, I routinely measure 5 MB/s on 8 foot cables. So, if you were writing the data manual for this, how would YOU spec it?

The framers of the SCSI spec threw in synchronous mode to boost the performance on long cables. In synchronous mode, the sending device is permitted to send the next byte without receiving acknowledgment that the receiver actually received the last byte. Kind of a ship and pray method. The acknowledgment is required to come back sometime, but we just don't have to wait for it (handwave the offset stuff and the ending boundary conditions). In this mode any external transceivers add a time shift, but not a delay. So if you negotiate for 5 MB/s, you get 5MB/s regardless how long the cable is and regardless whether you are single-ended or differential. But you can't go faster than 5.5 MB/s, except in SCSI-2. Synchronous mode does have a hold time (unlike asynch) but again, setup and hold times subtract out. In SCSI-1 synchronous mode, the speed limit comes from the combined ASSERTION PERIOD + NEGATION PERIOD which is 90ns + 90ns = 180ns = 5.5 MB/s. Our 53C90 family doesn't quite hit the max, but we do guarentee 5.0 MB/s. In SCSI-2, anything above 5.0 MB/s is considered to be FAST. Here the maximum transfer rate is explicitly limited to 100 ns or 10MB/s; you don't have to read between the lines to deduce it.

Interesting tid-bit: given a SCSI-2 FAST period of 100 ns and a cable delay of 131 ns on a 25 meter cable, you can actually stack 1.31 bytes in the 8-bit cable. In FAST and WIDE SCSI you can stack 5.24 bytes in this copper FIFO. Hummm...

Part of **FAQ**

====
QUESTION: **What are the jumpers on my Conner drive?**
ANSWER From: ekrieger@quasar.hacktic.nl (Eric Krieger)
====

QUICK INSTALLATION GUIDE

SCSI

Most SCSI host adapters are compatible with Conner drives. Software drivers and installation instructions are provided with the host adapter.

The drives are shipped with SCSI ID set to 7. To select a different ID refer to the following:

Table A

ID	E-1	E-2	E-3
0	out	out	out
1	in	out	out
2	out	in	out
3	in	in	out
4	out	out	in
5	in	out	in
6	out	in	in
7	in	in	in

Table B

ID	E2	E3	E4
0	out	out	out
1	in	out	out
2	out	in	out
3	in	in	out
4	out	out	in
5	in	out	in
6	out	in	in
7	in	in	in

Parity is always ENABLED on the CP3200,CP30060,CP30080,CP30100. All other models, jumper E-4 to disable parity.

SCSI drive parameters:

Model	Hds	Cyl	Sec	Table	LED
CP2020	2	642	32	A	n/a
CP340	4	788	26	B	1
CP3020	2	622	33	A	1
CP3040	2	1026	40	A	1
CP3180	6	832	33	A	1
CP3100	8	776	33	A	1
CP30060	2	1524	39	A	2
CP30080	4	1053	39	A	2
CP30100	4	1522	39	A	2
CP30200	4	2119	49	A	2
CP3200	8	1366	38	A	2
CP3360	8	1806	49	A	2
CP3540	12	1806	49	A	2

LED 1
J-4 Pin 1 = +
Pin 2 = -

LED 2
J-1 Pin 3 = +
Pin 4 = -

Part of **FAQ**

====
QUESTION: **What are the jumpers for my Wangtek 5150 drive?**
ANSWER From: "Terry Kennedy, Operations Mgr" <uunet!spcvxa.spc.edu!TERRY>
====

First, the disclaimer: This is not an official representation of Wangtek or of my employer. This is info I've discovered by reading publicly avail- able reference material. When changing jumpers, always observe proper anti- static precautions and be sure you have the current configuration written down so you have a known starting point.

Ok. Here's the complete scoop on Wangtek 5150ES drives:

The current part number for a "generic" 5150ES is:

33685-201 (black faceplate)
33685-202 (beige faceplate)

These are referred to as the "ACA version" of the drive.

There are _many_ other part numbers for 5150ES drives. If you have one that isn't one of the above, it doesn't mean you have an old or an out of rev drive, it just means its a special version created for a distributor or OEM, or with different default jumper settings.

You can order the Wangtek 5150ES OEM Manual from Wangtek. It is part number 63045-001 Revision D.

There are 5 possible logic boards. Here are the jumper options for each:

Logic assembly #33678

(J10)
0 - SCSI unit LSB
1 - SCSI unit
2 - SCSI unit MSB
K - not documented

J32 - Diagnostic test connector, default is not installed
E1, F1 - SCSI termination power. E1 in = power from drive and to cable,
E1 out - power from cable. F1 = terminator power fuse, 1.5A FB.
Default is IN.
E2 - Chassis ground. E2 in jumpers logic to chassis ground. E2 out isolates through a .33 uFD capacitor. Default is IN.
E5 - Master oscillator enable. Test only. Must be IN.
E20 - Factory test. Must be OUT.
RP1, RP2, RP3 - SIP terminators. Default is IN, remove for no termination.

Logic assembly #30559

HDR1 - Factory testing. Setting depends on drive. Don't touch.
HDR2 - Factory testing. Defaults are pins 15-16, 17-18, 19-20. Don't touch.
HDR3 pin 1 - A-B enables buffered mode. B-C disables. Can be overridden by SCSI Mode

Select.
HDR3 pin 2, 3 - Default data format. Set to B-C for a 5150ES.
HDR3 pin 4 - parity enable. A-B enables, B-C disables.

(J10)
0 - SCSI unit LSB
1 - SCSI unit
2 - SCSI unit MSB
K - not documented

E1 - SCSI termination power. E1 in = power from drive and to cable,
E1 out - power from cable.
E2 - Chassis ground. E2 in jumpers logic to chassis ground. E2 out isolates through a .33 uFD capacitor. Default is IN.
E3 - Master oscillator enable. Test only. Must be IN.
E4 - Write test mode. Test only. Must be OUT.
E5 - Write oscillator enable. Test only. Must be IN.
E6 - Disable HDR2. Test only. Must be IN.
E7 - Microcontroller clock select. In for a 5150ES.
E8 - Write precomp select. Set on a per-drive basis. Don't touch.
E9 - RAM size. Don't touch.
E10 - Erase frequency. Don't touch.
RP2, RP3 - DIP and SIP terminators. Default is IN, remove for no termination.

Logic assembly #30600

HDR1 - Factory testing. Setting depends on drive. Don't touch.
HDR2 - Write precomp select. Set on a per-drive basis. Don't touch.
HDR3 pin 1, 2, 3 - SCSI device address. 1 is LSB, 3 is MSB. A-B=1, B-C=0
HDR3 pin 4 - Parity enable. IA-B is enabled.
HDR3 pin 5, 6 - Default data format. B-C for a 5150ES.
HDR3 pin 7 - Buffered mode select. A-B is enabled.
HDR3 pin 8 - Reserved. Must be OUT.
HDR4 - Write frequency select. Don't touch.
E1 - SCSI termination power. E1 in = power from drive and to cable,
E1 out - power from cable.
E2 - Chassis ground. E2 in jumpers logic to chassis ground. E2 out isolates through a .33 uFD capacitor. Default is IN.
E3 - Hard/soft reset. IN enables hard reset.
E4 - Write precomp select. Don't touch.
E5 - Clock speed. Don't touch.
E6 - Tape hole test. Don't touch.

Logic assembly #30552

HDR1 - Factory testing. Setting depends on drive. Don't touch.
HDR2 - Write precomp select. Set on a per-drive basis. Don't touch.
HDR3 pin 1, 2, 3 - SCSI device address. 1 is LSB, 3 is MSB. [Note - HDR3 pins 1-3 are duplicated at another location on the board]
HDR3 pin 4 - Parity enable. IN is enabled.
HDR3 pin 5, 6, 7, 8 - Default data format. 5,5 B-C, 7-8 A-B for a 5150ES.
HDR4 - Write frequency select. Don't touch.
E1 - SCSI termination power. E1 in = power from drive and to cable,

E1 out - power from cable.
E2 - Chassis ground. E2 in jumpers logic to chassis ground. E2 out isolates through a .33 uFD capacitor. Default is IN.
E3 - Hard/soft reset. IN enables hard reset.
E4 - Write precomp select. Don't touch.
E5 - Clock speed. Don't touch.
E6 - Tape hole test. Don't touch.

Logic assembly #30427

HDR1 - Factory testing. Setting depends on drive. Don't touch.
HDR2 - Write precomp select. Set on a per-drive basis. Don't touch.
HDR3 pin 1, 2, 3 - SCSI device address. 1 is LSB, 3 is MSB. A-B=1, B-C=0
HDR3 pin 4 - Parity enable. IA-B is enabled.
HDR3 pin 5, 6, 7, 8 - Default data format. 5,5 B-C, 7-8 A-B for a 5150ES.
E1, E3 - Factory test. Must be IN.
E2 - SCSI termination power. E2 in = power from drive and to cable,
E2 out - power from cable.
E4 - Chassis ground. E4 in jumpers logic to chassis ground. E4 out isolates through a .33 uFD capacitor. Default is IN.

Firmware - There are many flavors of firmware. I have seen the following parts:

24115-xxx
24144-xxx
21158-xxx

the -xxx suffix changes as the firmware is updated. According to the folks I spoke to at Wangtek, the standard firmware is the 21158. The latest version as of this writing is 21158-007. All of these will work with the Adaptec and GTAK.

The firmware options (as returned by a SCSI Identify) are on the end of the product string, which is "WANGTEK 5150ES SCSI ES41C560 AFD QFA STD" for the 21158-007 firmware. The 3-letter codes have the following meaning:

AFD - Automatic Format Detection - the drive will recognize the format (such as QIC-24, QIC-120, or QIC-150) that the tape was written in.

QFA - Quick File Access - the ability to rapidly locate a tape block, and to implement the "position to block" and "report block" SCSI commands. This is compatible with the Tandberg implementation.

STD - Standard feature set.

Part of **FAQ**

====
QUESTION: **What is CAM?**
ANSWER From: ctjones@bnr.ca (Clifton Jones)
====

Common Access Method.

It is a proposed ANSI standard to make it easier to program SCSI applications by encapsulating the SCSI functions into a standardized calling convention.

ANSWER From: landis@sugs.tware.com (Hale Landis)
====

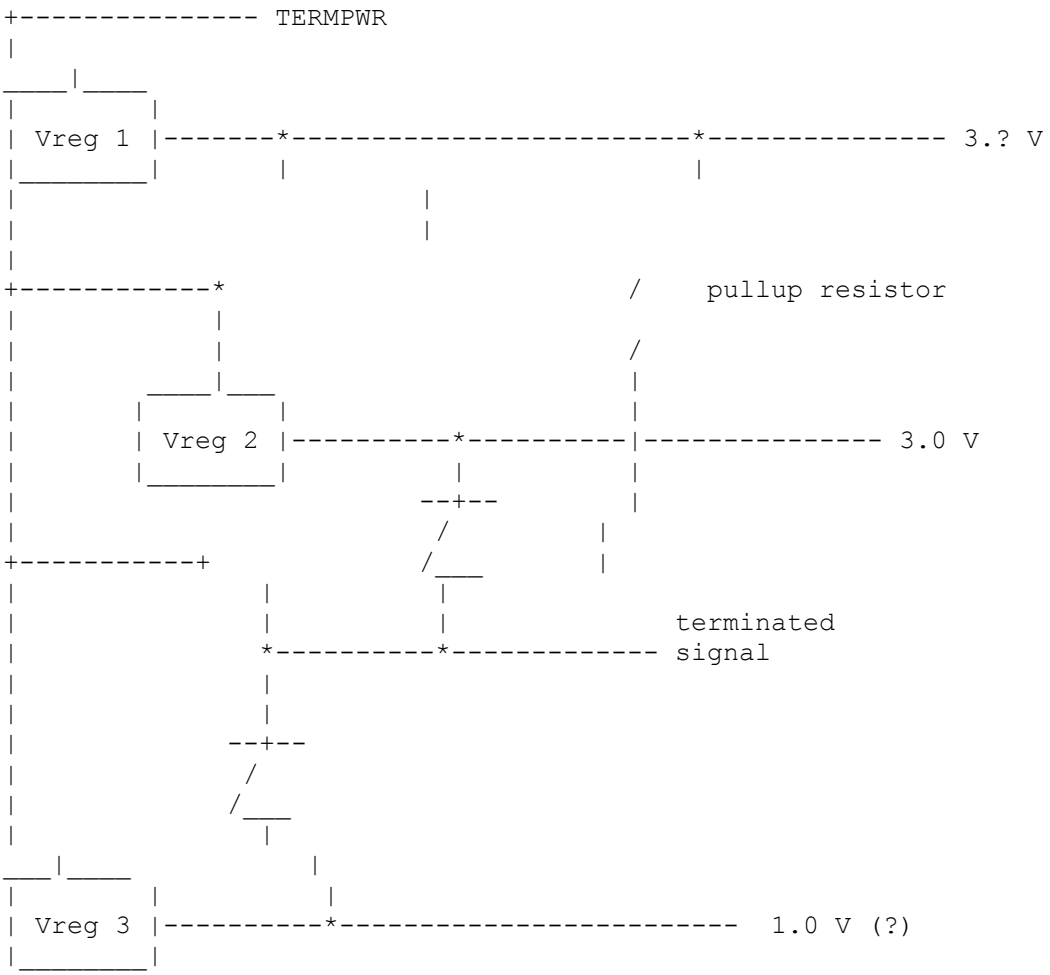
You may be able to get the CAM spec(s) from the SCSI BBS

See also: **ANSI Common Access Method rev 2.3**

Part of **FAQ**

====
QUESTION: **What is FPT (Termination)?**
ANSWER From: jvincent@bnr.ca (John Vincent)
====

FPT is actually really simple, I wish I had thought of it. What it does is use diode clamps to eliminate over and undershoot. The "trick" is that instead of clamping to +5 and GND they clamp to the output of two regulated voltages. This allows the clamping diodes to turn on earlier and is therefore better at eliminating overshoot and undershoot. The block diagram for a FPTed signal is below. The resistor value is probably in the 120 to 130 ohm range. The actual output voltages of the regulators may not be exaclty as I have shown them but ideally they are matched to the diode characteristics so that conduction occurs when the signal voltage is greater than 3.0 V or less than 0.5 V.



Part of **FAQ**

====
QUESTION: **What is Active Termination?**
ANSWER From: eric@telebit.com (Eric Smith)
and brent@auspex.com (Brent R. Largent)
====

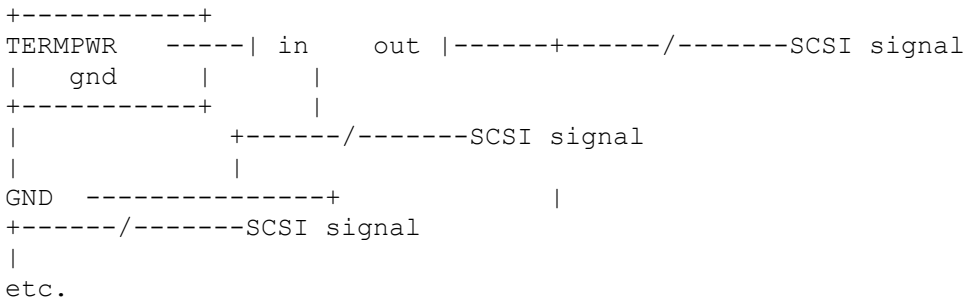
An active terminator actually has one or more voltage regulators to produce the termination voltage, rather than using resistor voltage dividers.

This is a passive terminator:

TERMPWR ------/------+------/----- GND
|
|
SCSI signal

Notice that the termination voltage is varies with the voltage on the TERMPWR line. One voltage divider (two resistors) is used for each SCSI signal.

An active terminator looks more like this (supply filter caps omitted):



Assuming that the TERMPWR voltage doesn't drop below the desired termination voltage (plus the regulator's minimum drop), the SCSI signals will always be terminated to the correct voltage level.

Several vendors have started making SCSI active terminator chips, which contain the regulator and the resistors including Dallas Semiconductor, Unitrode Integerated Circuits and Motorola

Part of **FAQ**

====
QUESTION: **Why Is Active Termination Better?**
ANSWER brent@auspex.com (Brent R. Largent)
====

Typical pasive terminators (resistors) fluctuate directly in relation to the TERM Power Voltage. Usually terminating resistors will suffice over short distances, like 2-3 feet, but for longer distances active termination is a real advantage. It reduces noise.

Active Termination provide numerous advantages:
- A logic bit can disconnect the termination
- Provides Negative Clamping on all signal lines
- Regulated termination voltage
- SCSI-2 spec recommends active termination on both ends of the scsi cable.
- Improved Resistance tolerences (from 1% to about 3%)

"Buyer's Guide" by Roy Neese
The following is intended for those who may have or are thinking about purchasing a SCSI subsytem.

THE HOST ADAPTER: THE FIRST PIECE OF THE PUZZLE

Host adapters come in many flavors. Some are well supported, others are not. Some are intelligent, some are not.

To decide on the correct host adapter to use is a difficult decision and one that should not be taken lightly. Unless you are prepared to continue investing in other adapters as your systems needs grow, you need to study and understand the different types of host adapters and the application they are going to be used in.

The simplest host adapters are cards that may have an INT13 At BIOS to support hard drives under MS-DOS. These host adapters usually have no UNIX support as it is very difficult to do a device driver that can be made to work reliably. Dumb cards such as these usually have timing constraints that make it almost impossible to have a UNIX driver.

The next level of host adapter goes to the other extreme. They are very intelligent and usually hide the SCSI bus from the operating system. These types of adapters have a wide base of support in software, as they can run in any environment with very little software effort. They are also expensive adapters, which may not be needed in the MS-DOS machine. Indeed, they may be overkill for a machine that is to be relegated to the DOS environment.

If you are going to be using MS-DOS and MS-DOS alone, then the low cost approach is not a bad one. But if you intend to use UNIX/Novell/OS2, then the low cost approach will be a poor one. These operating systems/environments are all multi-threaded. That is they can issue more than one command at a time. With an intelligent host adapter, this is easily done and managed by the host adapter. With a low cost board, the software to do this work must be driven by the main CPU, which will incur considerable command overhead.

Of course, there is also the support issue. You always would like to avoid the finger pointing that can occur when using products from several companies, when a bug arises. This is best done by buying a product that is supported by the operating system manufacturer. There are very few SCSI host adapters supported by the operating systems manufacturers. You need to ask them what they will support and what they won't. A little planning in this area may solve many potential problems for you.

Questions you should ask:
a) What operating environment will I be using?
b) Is the operating environment multi-threaded?
c) Am I going to be using several operating systems?
d) Do I care if support is from one operating system vendor?

SCSI DEVICES AND MANUFACTURER'S CLAIMED DATA RATES

SCSI DEVICES AND MANUFACTURER'S CLAIMED DATA RATES

Ignore the manufacturer's rated data rates. They are measured on the SCSI bus assuming that no data is being transferred to the device buffer from the media and they also assume the best case data transfer (no disconnects).

To get an idea of how a device might perform, look at the clock rate of the head. You will see a range from 7.5 Mhz to 24 Mhz. Some devices have multiple clock rates. These are bit zone recorded devices. They have more sectors on the outer tracks than they do on the inner tracks.

The data rate to the devices buffer is the rate of the clock on the head. A 7.5Mhz head moves data to the buffer at 750KB/sec and a 24Mhz clock on the head moves data at 2.4MBytes/sec.

Data rates to/from the buffer from/to the SCSI bus are regulated by the clock rate of the dual ported buffer (where applicable). If the buffer has a 4MHz clock, then the fastest the data can be shipped to/from the buffer is 4MBytes/sec, but if data is being shipped to/from the media from/to the buffer, then the rate that data can move across the SCSI bus is directly impacted. For instance, if the drive is putting data into the buffer at 2.0MBytes/sec and the buffer is a 4MHz part. The fastest data can be moved to the SCSI bus is 2.0MBytes/sec not 4MBytes/sec. This, of course, only applies to those devices that have a dual ported buffer. How the buffer full/empty ratios are used will also effect the data rate.

SCSI devices, for the most part, have programmable buffer full/empty ratios stored in a mode page of the device. NOTE: Not all devices allow a user to reprogram these parameters. These parameters affect when the device should and should not, get on the SCSI bus for a data transfer. Typically, a SCSI device will ask for the bus, on a read, when there is at least 1 sector in the data buffer ready to transfer. The transfer will start and the device will continue to put data into the buffer if possible, as data is taken out of the buffer. If the buffer falls below the 1 sector limit, then the device will disconnect from the bus and will not reconnect until the buffer has at least one sector's worth of information.

Questions you should ask:
a) How fast is the dual ported buffer?
b) What is the clock rate of the heads?
c) Is the device capable of synchronous transfers?
d) How is the buffer full/empty ratios calculated and implemented?

TO BUFFER OR NOT TO BUFFER?

TO BUFFER OR NOT TO BUFFER?

The size of the buffer has a lot to do with the overall performance of the SCSI device. Buffer sizes range from 16K to 256K. If the buffer is not dual ported, then that implies the device cannot allow data to be moved from/to the SCSI bus while the HD is moving data to/from the buffer and vice-versa.

If the buffer is just a buffer to smooth the data transfer, then the size of the buffer can be small if you have a host adapter capable of moving the data at the full rate of the SCSI bus. If the adapter is dumb and cannot move data very quickly, then a larger buffer is required to minimize the disconnects/reconnects on the SCSI bus. In this implementation, the size of the buffer will not impact nor help performance. Also, with a dumb adapter you will find it very difficult to maintain a 1:1 interleave with this type of buffering scheme. Typically, this is the poorest performing SCSI device.

If the device has a read ahead buffer, then sequential accesses will be much quicker. Although the more fragmented the file system the worse the worse the performance. In this case, the size is dependent on the overall implementation. If the read ahead will abort at the end of a track, then the buffer need not be larger than the largest track on a device. If the read ahead is aborted at a cylinder switch, then the buffer size should be large enough to accommodate the largest cylinder. For the most part, this is a good implementation.

If the device has a read ahead cache buffer, then this, like the read ahead buffer, will give good sequential accesses, but still poor performance on a very fragmented filesystem. As data in the buffer is recoverable, because this is a cache, some performance gains will be noticed in a multi-user environment. The same rules for the read ahead buffer above apply when it concerns the size of the buffer. A better performance implementation.

If the device has a segmented cache buffer, then this will yield the best performance available in all operating environments. It must be tunable so you can match the characteristics of the operating environment. The size of the buffer should not be less than 64K for this type of implementation to be effective, but it can be as large as the vendor chooses and the bigger the better. This is the best performance choice.

Questions you should ask:
a) What type of buffer algorithm do you use? (i.e. read-ahead, cache)
b) Is the buffer algorithm programmable?
c) What is the size of the buffer?
d) What is the clock (or resolution) of the buffer?
e) Is the buffer dual ported?

SCSI COMMAND OVERHEAD

SCSI COMMAND OVERHEAD

SCSI command overhead is a much discussed topic when users opt to go with SCSI and generally a very heated topic. In the past, the overhead was very high (on the order of 3 milli-seconds). Today SCSI overhead, for the device, is down to 1 milli-second and less. Some devices have multiple processors, one for running the SCSI bus and one for the device electronics. These type of devices have less overhead than a single processor device as they can do things in parallel. Be aware of the manufacturers specs. Though they don't lie, they publish the best case overhead times. SCSI command overhead, as measured by the manufacturer, is typically taken from the time a TEST UNIT READY command takes to complete. The problem with that is this command takes the shortest path through the vendor's firmware as it does not require any data transfers.

The best way to measure the overhead for a SCSI device is to issue a write command and then issue the same command again, to ensure the device is at the track it should be. Using a logic analyzer, measure the time from the last command byte to the ending status phase for the second write command. Subtract the data phase of the command and you will have a more believable idea of the overhead. The reason you want to use a write command is to eliminate any disk latency that a read command would generate.

Measuring overhead from the system level is virtually impossible, as you also have the host adapter overhead and then the bus and CPU rates come into play.

Questions you should ask:
a) What is the SCSI command overhead for your controller?
b) How is the overhead measured?
c) Does your device have multiple processors?
d) What is the clock rate of the processors?

BENCHMARKING MADE EASY??

BENCHMARKING MADE EASY??

Throw away CORETEST, or disregard the seek times and the size of therequest. Seek times on a SCSI device cannot be measured at the BIOS levelunless the benchmark can be told what physical parameters to use.

SCSI is a logical block interface and has no physical characteristics. Thenumber of heads, sectors, and cylinders at the BIOS level are all incorrect, sothe seek times that virtually all of the benchmark programs use will not yieldaccurate seek times.

You cannot use the manufacturer's seek rate as it is measured at themechanical level and not at the SCSI bus level. To test the seek times, youalmost have to write your own benchmark. If 'adaptex' was going to be aroundI would have a program that would do just that for SCSI devices and post it,but alas.

CORETEST also reads the same data over and over again, so a SCSI device thathas a read ahead buffer, will not be accurately measured.

In order to get a fair idea of the data rates that a SCSI device can yieldrequires you to measure data transfers in the following ways:

1) Sequentially
2) Random (1/3 stroke, preferred as you can do that at the BIOS level)
3) Read the same data over and over again.

Now that will take care of the single-threaded benchmark, but what UNIX/XENIXand other multi-threaded operating environments? While none of the currentbenchmarks really do a fair job of measuring the correct thing for theseenvironments, they can yield some useful information.

With a multi-threaded environment, you should not only consider the averageaccess time and the data rate, but more importantly. How does the SCSIimplementation I have chosen effect the overall throughput of my system? Whilein a single-threaded environment most devices will benchmark well, devicecharacteristics change dramatically in a multi-threaded mode.

This is due in part to the devices ability to efficiently arbitrate for theSCSI bus. Some devices may take as long as 3 milli-seconds to complete anarbitration cycle, while others take only 1 milli-second. The only way youcan judge the devices is via a logic analyzer. You are basically lookingfor the overall usage of the SCSI bus.

Questions you should ask:
a) How can I measure the average seek time?
b) What environment will this implementation be used in?
c) How can I best judge the performance in this environment?

PERSONAL PERFORMANCE RATING

PERSONAL PERFORMANCE RATING

If I had to rate, regardless of capacity, the best performing SCSI devicesthat I have used, the list would go:

```
Quantum         (3 1/2") 64K segmentable cache buffer, RLL 2.7, ZBR
                         async/sync
Quantum         (5 1/4") 64K segmentable circular cache buffer, RLL 2.7, async
Imprimis        (5 1/4") 32K read ahead buffer (*), RLL 2.7, ZBR, async/sync
Maxtor          (5 1/4") 45K read ahead buffer (*), RLL 2.7, async/sync
Micropolis      (5 1/4")
Priam           (5 1/4")
Imprimis        (3 1/2")
Maxtor          (3 1/2") 8K buffer, RLL 2.7, async
Seagate         (5 1/4") Basic buffer
Conner          (3 1/2") Basic buffer
Rodime          (3 1/2") Basic buffer

* Drives come with read ahead disabled.  It can only be activated via
  software

Recomended products to avoid

Cast/Newbury    Do not meet SCSI standard (Rev17B)
Microscience    Do not meet SCSI standard (Rev17B)

TERMS
ZBR     Zone Bit Recorded
```


Roy Neese
Adaptec Central Field Applications Engineer
UUCP @ texbell,attctc!cpe!adaptex!neese
merch!adaptex!neese

From senator-bedfellow.mit.edu!enterpoop.mit.edu!news.media.mit.edu!
mintaka.lcs.mit.edu!yale!yale.edu!spool.mu.edu!sdd.hp.com!col.hp.com!news.dtc.hp.com!
srgenprp!darrylo Fri Jan 29 19:04:24 EST 1993
Article: 49954 of comp.sys.ibm.pc.hardware
Newsgroups: comp.sys.ibm.pc.hardware
Path: senator-bedfellow.mit.edu!enterpoop.mit.edu!news.media.mit.edu!
mintaka.lcs.mit.edu!yale!yale.edu!spool.mu.edu!sdd.hp.com!col.hp.com!news.dtc.hp.com!
srgenprp!darrylo
From: darrylo@sr.hp.com (Darryl Okahata)
Subject: **Adaptec hints & tricks**
Sender: news@srgenprp.sr.hp.com (placeholder for future)
Message-ID: <C1Fxrs.G7v@srgenprp.sr.hp.com>
Date: Tue, 26 Jan 1993 03:02:16 GMT
Reply-To: darrylo@sr.hp.com
Organization: Hewlett-Packard / Center for Primal Scream Therapy
X-Newsreader: TIN [version 1.1 PL8.5]
Lines: 548

A few months back, I asked for help getting my Adaptec 1542 card working with Windows 3.1. Here's a summary of what I discovered, as well as other interesting information pertaining to the Adaptec 1542.

-- Darryl Okahata
Internet: darrylo@sr.hp.com

DISCLAIMER: this message is the author's personal opinion and does not constitute the support, opinion or policy of Hewlett-Packard or of the little green men that have been following him all day.

==========
$Id: adaptec.txt 1.8 1993/01/25 00:55:08 darrylo Rel darrylo $

Hints and Tips for the Adaptec 1540/1542 SCSI adapter

This document contains hints and tips for getting the Adaptec 1540/1542 SCSI adapter to work with various hardware and software packages. They are based upon my experiences with an Adaptec 1542A controller, and will, hopefully, help others. However, note that I cannot guarantee that the following will really help you (it works for me), and the information in this document could possibly cause you to lose some or all of your files on your hard disk.

IMPORTANT! BACK UP THE ENTIRE CONTENTS OF YOUR HARD DISK BEFORE TRYING ANYTHING BASED UPON INFORMATION IN THIS DOCUMENT.

Copyright 1993, by Darryl Okahata. This document may be freely copied for personal use only, and **may not be reprinted in a for-profit publication without the consent of the author**. Please note that I have no connection with Adaptec other than as a customer.

Topics covered in this document:

* **Sound cards**
* **Miscellaneous info**

Please note that parts of this document contain technical, and sometimes terse, descriptions of problems.

For reference:

Adaptec technical support:	(800) 959-7274
Adaptec BBS (2400/9600):	(408) 945-7727

Please send comments, corrections, etc. via email to me:

CompuServe:	75206,3074
Internet:	darrylo@sr.hp.com

Windows 3.1 enhanced mode

(Part of **Adaptec 1540/1542 hints**)

******* Windows 3.1 enhanced mode:**

The Windows 3.1 install program should automatically configure DOS and Windows for use with the Adaptec 1542. However, just in case something went wrong, I'm going to describe some of the changes needed to get Windows 3.1 working with the 1542. Also, you may have noticed that installing Windows 3.1 makes your PC run much slower, even when you're not running Windows; methods of speeding it up are discussed in the section called, "Windows 3.1 runs slowly".

*** MSDOS configuration:**

The Windows install program adds the SmartDrive disk cache to your CONFIG.SYS and AUTOEXEC.BAT files. If you follow the instructions, you'll notice that you'll need to use double-buffering with SmartDrive (this is the default setup). You'll also notice that your system runs much, much slower -- in both Windows *AND* MSDOS. See the section called, "Windows 3.1 runs slowly", for some ways of speeding your system up.

*** Windows configuration:**

To get the Adaptec 1542 to work with Windows, make sure that the "[388Enh]" section of the SYSTEM.INI file contains the entry:

```
VirtualHDIRQ=Off
```

I believe that the Windows install program automatically adds this entry to SYSTEM.INI, but I'm not sure. If this doesn't work for you, you might want to try adding some more lines:

```
VirtualHDIRQ=Off
SystemROMBreakPoint=false
EMMExclude=A000-CFFF
```

(You probably don't need the above lines, though.) The "SystemROMBreakPoint" entry is used to enable support for memory managers like QEMM/386MAX (only needed if you use such programs).

*** Windows 3.1 runs slowly:**

Once you do get Windows 3.1 running with the 1542, chances are that your system is running much slower than before. If it's not, it's probably because:

1. You happen to be using **ASPI4DOS.SYS** version 3.1 in your CONFIG.SYS file. Congratulations -- this appears to be a winning solution.

2. You are very lucky. Whether your luck will hold out remains to be seen ....

If your system is running much slower than before, this is almost definitely caused by Smartdrive with double-buffering. According to the Windows documentation, and the Microsoft technical note #Q81808 ("SMARTDrive Double Buffering Required with

ASPI4DOS.SYS"), you must use Smartdrive with double-buffering enabled. While this works, it really slows down your PC; I once estimated that this slowed my PC down by a factor of 5 (FIVE). As I consider this unacceptable, I looked for other solutions.

Unfortunately, you cannot just disable double-buffering. If you do, Windows 3.1 in enhanced mode will not work, and you might even destroy the contents of your hard disk by trying to run Windows 3.1. What you can do is one of the following:

1. Use other drivers that provide double-buffering. It is my opinion that the unbelievable slowness in Smartdrive is caused either by horribly inefficient double-buffering, or by a bug in Smartdrive.

2. Use a driver that provides "VDS" services ("VDS" stands for "Virtual DMA Services"). This is a standard, which is supported by Windows 3.1, that allows bus-mastering disk controllers (like the 1542) to work with Windows.

After trashing my hard disk countless times, I found the following solutions, none of which require using Smartdrive (note, however, that I am now getting occasional parity errors, which are probably *NOT* caused by these solutions, but might be -- see below). While the following does not require Smartdrive, using some kind of disk cache utility is strongly recommended, as this makes Windows run much, much faster:

1. If you do not have the **ASPI4DOS.SYS** driver, or you do not need ASPI functions (for controlling a CDROM, tape drive, more than two physical hard disks, etc.), you can add the SCSIHA.SYS driver to your CONFIG.SYS file, e.g.:

   DRIVER=c:.SYS /V386

   (Windows needs the "/V386" option.) This driver MUST be loaded into LOW memory (it cannot be loaded into high memory), and it occupies about 16-20K. As of November 1992, the SCSIHA.SYS driver could be obtained from the Adaptec BBS at (408)-945-7727 (hopefully, it's still there).

2. If you need ASPI functions and have the ASPI4DOS.SYS driver, version 3.0 or 3.0a, you can use both the ASPI4DOS.SYS and SCSIHA.SYS drivers in your CONFIG.SYS file, e.g.:

   DRIVER=c:DOS.SYS
   DRIVER=c:.SYS /V386

   Amazingly enough, the SCSIHA.SYS driver can also be loaded high (assuming you have DOS 5.0); I would have thought that this would crash my system, but it doesn't. I asked Adaptec's technical support about this, and they said that loading SCSIHA.SYS high should be fine as long as ASPI4DOS.SYS is loaded LOW.

   On my system, NOT using SCSIHA.SYS with ASPI4DOS 3.0a would occasionally cause Windows 3.1 to crash upon restarting or exiting Windows, with the additional result of a corrupted disk (some of my C:.GRP files would be corrupted). For me, these crashes usually occurred while making a different program from PROGMAN.EXE

the default Windows shell, and vice-versa. This is the reason SCSIHA.SYS may be necessary.

I have absolutely no idea if SCSIHA.SYS is necessary with versions of ASPI4DOS earlier than 3.0.

Note that many people can use ASPI4DOS 3.0 or 3.0a without SCSIHA.SYS; they do not seem to have any problems at all. I consider these people lucky. Others, like me, have had all sorts of problems.

3. In my opinion, the best, but not necessarily the easiest, solution is to upgrade to ASPI4DOS 3.1. The SCSIHA.SYS driver is no longer needed. Unfortunately, while you could get previous ASPI4DOS upgrades from the Adaptec BBS, the ASPI4DOS 3.1 driver is not available from the Adaptec BBS. As far as I know, there are only three ways to get a copy:

   * You can buy the new (as of November 1992) Adaptec EZ SCSI driver kit, which supposedly includes ASPI4DOS 3.1 as well as other drivers, such as CDROM drivers. I believe the list price is around $75.

   * If you already have a copy of an older version of ASPI4DOS, you can supposedly contact Adaptec to upgrade it to EZ SCSI for around $30.

   * A copy of ASPI4DOS 3.1 is included in Central Point PC Tools 8.0 for MSDOS. Note that the documentation and driver are stored in different directories. Note further that only ASPI4DOS is included; the CDROM drivers and drivers to support more than two hard disks are not included. This is where I obtained my copy of ASPI4DOS 3.1.

Note, however, that I am now getting occasional parity errors with Windows. In all probability, defective hardware in my PC is causing this, as I upgraded my motherboard just after I found the above solutions. However, because these parity errors occur only during disk accesses, there is a very small, but definite, possibility that the parity errors are driver-related (for example, changing the bus on/off timing for certain disk transfers might cause this). I've run various memory tests for hours at a time, and these tests have found no problems. This problem is probably caused by memory with marginal timing requirements, which cause parity errors during disk transfers (this is why the memory tests didn't find any problems -- the problems show up only under disk I/O). However, I'm mentioning this just in case it isn't a hardware problem.

Floppy-controller-based tape backup devices

(Part of **Adaptec 1540/1542 hints**)

******* Floppy-controller-based tape backup devices:**

There are two possible problems with using the Adaptec 1542 with a floppy-controller-based tape backup device, such as the Colorado Memory Systems Jumbo 250:

1. Tape backups/restores can take a very long time. The tape drive constantly starts, stops, starts, stops, etc.

2. Tape operations may be erratic, or encounter too many tape errors. (This problem might be caused by defective hardware on my 1542. However, I've heard of other people having similar problems, and so I'm mentioning this just in case it is not a hardware problem on my 1542.)

*** Tape backups/restores take a long time:**

If you have a floppy-controller-based tape backup device, you may have to adjust the Adaptec 1540/1542 "bus on/off timing" for best results when using the tape drive. Normally, while doing a tape backup or restore, the tape drive motor should be continuously running, with only an occasional pause. However, the default bus timing on the Adaptec 1540/1542 may cause the tape drive motor to start and stop, start and stop, every few seconds. This causes needless wear to the tape and tape drive (however, note that a dirty tape head or a defective tape drive can also cause this -- make sure your tape heads are clean). This also causes the tape backup or restore to take much, much longer than necessary.

The problem here is that these tape backups use the floppy DMA to transfer data in memory to/from the tape drive, and the Adaptec uses DMA to transfer data in memory to/from the hard disk. The floppy DMA needs to feed data to the tape drive at a certain rate; if the tape drive is not fed data quickly enough by the floppy DMA, the tape drive stops, rewinds a bit, and restarts (once enough data is eventually fed to it). The default bus timing on the Adaptec (which is really DMA timing) is "too large". For example, when a backup is done, data has to be transferred from a hard disk to memory, and then from memory to the tape. Because the default timing on the Adaptec "hogs" the memory too much (too much time is spent transferring data from a hard disk to memory), not enough time is spent transferring data from memory to the tape drive. As a result, the tape drive constantly starts and stops, because data is not fed to it quickly enough.

The solution is to change the Adaptec's bus on/off timing. The default factory setting is 11 microseconds on, and 5 microseconds off. The "bus on" timing needs to be lowered to 2-4 microseconds. This can be done in one of two ways:

* If you have **ASPI4DOS**, you can use the "/n" option. For example, I use a "bus on" timing of 4 microseconds, which means that I use the following line in my CONFIG.SYS file:

```
DEVICE=c:dos.sys /n4
```

Note that there is NO space between the "/n" and the "4".

* If you don't have ASPI4DOS, your only recourse is to try to find a program called "SETSCSI.EXE", which is very difficult to find. The

reason is that Adaptec, for reasons of their own, does not seem to want this widely distributed. I once asked someone who worked for Adaptec, and they asked me to not upload it anywhere. If you have anonymous ftp access to the Internet, you could try using archie to hunt down a copy; I believe that there are a couple of sites that have it. If you do find a copy, you run it like so:

setscsi -n:4

This adjusts the "bus on" timing to 4 microseconds. Running SETSCSI.EXE without any arguments resets the bus timing back to the factory defaults.

Note that it seems that you cannot use SETSCSI.EXE if you use ASPI4DOS; SETSCSI.EXE crashed my system if ASPI4DOS was loaded. I could use SETSCSI.EXE with SCSIHA.SYS, however.

Do not lower the "bus on" timing below 2 microseconds, or increase it above 11 microseconds. If you lower it too low, the hard disk throughput will suddenly drop; your system will feel slower. For me, 4 microseconds works fine. This value may work fine for you, or you may have to adjust it downwards a little.

Once you've lowered the "bus on" timing, tape backups and restores should run faster.

Also, do not experiment with the bus on/off times (with the other options that I have intentionally not described), unless you know what you are doing. Bad combinations can cause parity errors and worse, by starving memory refresh.

A program called BUSTIFIX.EXE exists on the Adaptec BBS. Unless this has been upgraded since I last checked (which has been a while), this is a self-extracting archive containing a batch file and a couple of other files. This batch file was supposed to allow one to set the bus on/off times for the 1540/1542 and others. However, when I tried running this program with my 1542A, my system crashed. At the time, I was running SCSIHA.SYS, and I didn't check to see if there was a conflict with it. Maybe this old program works only with the 1542B, although the docs say that it works with the 1542A?

*** Erratic tape operations or too many tape errors:**

This "problem" may or may not exist. Although it existed on my system, a hardware problem just on my particular 1542 could cause it. However, I've heard of other people having similar problems, and so I'm mentioning this just in case it isn't a hardware problem just on my 1542.

Symptoms of this "problem", which persists even after cleaning the tape head:

1. Backing up to tape encounters "unusable sector detected" errors, resulting in an aborted tape backup.

2. Tape backup works, but the tape compare fails.

3. The tape drive starts, stops, starts, stops, etc. much too often. Unlike the above-mentioned problem ("Tape backups/restores take a long time"), where the tape drive starts and stops every few seconds, this kind of starting/stopping occurs every few 10-20 seconds or so.

4. Fastback Plus 3.1 does not find/see any tape backup devices. Other programs, like Central Point Backup and the CMS Jumbo software (assuming that you have a CMS Jumbo 250 tape drive) can find/see the tape drive, but Fastback Plus 3.1 cannot.

5. Too many tape read errors.

Although I do not know what is causing this problem, I discovered that using a different floppy controller solves it. A few months ago, I upgraded my motherboard, which contained an integrated floppy controller. As I already had a floppy controller on the 1542, I initially disabled the motherboard floppy controller. After a while, I decided to try disabling the 1542 floppy controller and using the one on the motherboard. When I did this, the tape drive (a CMS Jumbo 250) reliability increased dramatically, and Fastback Plus 3.1 was suddenly able to find and use the tape drive.

I don't know if this was caused by a hardware problem on my 1542. On the one hand, the floppy drives worked great when they were attached to the 1542, which seems to say that there was nothing wrong with the 1542. On the other hand, the tape drive didn't work well attached to the 1542 floppy controller, but it did work when attached to a different controller; this could be an indication of a hardware problem on my 1542. I did change floppy drive cables, and so it is conceivable that the problem was in the cables. I don't know what the cause really is; however, if you're having similar problems, you might want to consider trying a new floppy controller.

Sound cards

(Part of **Adaptec 1540/1542 hints**)

******* Sound cards:**

Many popular sound cards can play or record digitized sound, and this is typically done using DMA. Like the tape drive DMA, the Adaptec's DMA can conflict with the sound card DMA. Unlike that of the tape DMA, this "conflict" usually manifests itself as a parity error (your system crashes with a parity error message). What happens is that, data is being transferred so quickly by the sound card and the Adaptec, memory refresh cannot occur quickly enough, which causes a parity error. Usually, getting a parity error means that there is a hardware problem with your system; in this case, however, the parity error is not a symptom of bad hardware.

I've found that such parity errors typically occur while recording digitized sound, and the chances of such errors increase as you increase the recording fidelity (e.g., higher sampling rate, recording in stereo, recording using 16-bits instead of 8, etc.).

Like the tape drive solution, the solution here is to lower the Adaptec's "bus on" timing. See the section on tape drives for information on how this is done. Note, however, that this may or may not solve the problem; it may only reduce the probability of a parity error. The software used to record digitized sound can greatly affect this problem (i.e., some software is inefficient). Disk caches, the speed of your hard disk, and the amount of disk fragmentation can also affect this.

Miscellaneous info

(Part of **Adaptec 1540/1542 hints**)

******* Miscellaneous info:**

This section contains miscellaneous hints, tips, and rumors. Much of it is merely information that I've heard or read about, and have not verified. I believe that the following information is correct, but I'm not sure. Use it at your own risk.

* With QEMM 6.00, 6.01, and 6.02, you need to specify the "DB=" parameter (e.g., "DB=2"), unless you are using the **ASPI4DOS** driver. If you don't, QEMM will crash/hang at bootup. Although the QEMM manual mentions this, the install program does not seem to detect that a 1542 is present and automatically add this option to the QEMM command line (at least, this occurred with the QEMM 6.00 install program -- I haven't tested any other version). Earlier versions of QEMM probably need this parameter, but I'm not sure (I've never used a version earlier than 6.00).

  If you use ASPI4DOS, you do not need to give QEMM the "DB=" parameter.

* Some or all versions of the 1542 do not support hard disks over one gigabyte in size. To support hard disks with capacities over 1GB, you need to get a new ROM BIOS from Adaptec. I'm not sure if this is still true of the latest 1542Bs being sold by Adaptec.

* To connect a CDROM drive to the 1542, you need a SCSI CDROM drive and some drivers. Note that some CDROM drives have proprietary interfaces (non-SCSI); these drives cannot be used with the 1542. You have three choices for CDROM drivers (I have no idea how well the following solutions work, or even if they work -- the following is secondhand information):

  1. You can buy Adaptec's EZ SCSI driver package, which lists for something like $75. If you already have older Adaptec drivers, you can supposedly upgrade to EZ SCSI for around $30. Contact Adaptec for details. The EZ SCSI package supposedly contains everything that you need.

  2. You can buy the CorelSCSI! driver package, which is made by the same people that make CorelDRAW! This package contains CDROM drivers, SCSI tape drivers, WORM drivers, etc. I do not know the list price, but I've seen this package sold for around $80-$90. Note that CorelSCSI! does not come with the ASPI4DOS driver, which is needed. If you do not already have ASPI4DOS, you may be better off getting Adaptec's EZ SCSI instead.

  3. [This method is obsolete, as the following drivers have been obsoleted by Adaptec's EZ SCSI kit, but I'm mentioning it in case someone already has these drivers.] You can use the drivers in the Adaptec ASW-1410 kit (ASPI4DOS) and the ASW-410 kit (ASPI CDROM drivers). You will have to get a copy of MSCDEX.EXE (a high-level CDROM driver), if it is not included in the ASW-410 kit, but this is available from several bulletin boards.

* To use a SCSI tape drive with the 1542, you need software that knows how to talk to a SCSI tape drive. Software that I've heard about are (again, like the above section on CDROM drives, I have no idea how well the following solutions work, or even if they work -- the following is secondhand information):

    1. Central Point PC Tools 8.0 for MSDOS supposedly supports a large number of SCSI tape drives. It comes with SCSI drivers (ASPI4DOS 3.1) as well as Central Point Backup.

    2. The CorelSCSI! driver package contains a SCSI tape backup program (see the above section on CDROM drives for more details). However, note that CorelSCSI! does not come with, but requires, ASPI4DOS.

* I've seen advertisements that sell the 1542 in three configurations:

    1. 1542 SCSI controller with hard disk ROM BIOS.
    2. 1542 SCSI controller w/BIOS and Adaptec ASPI drivers.
    3. 1542 SCSI controller w/BIOS, Adaptec ASPI drivers, and CorelSCSI! drivers/programs.

  I imagine that Adaptec now sells the 1542 in a fourth configuration:

    4. 1542 SCSI controller w/BIOS and EZ SCSI drivers (including ASPI drivers).

* Those people who use Unix might be interested in a version of GNU tar for MSDOS that talks to a SCSI tape drive via the ASPI4DOS driver (you need this driver before you can use this program). I've never used this version of GNU tar, but I've heard that it works (I don't know how well, though). If you have anonymous ftp access to the Internet, a copy can be found on wsmr-simtel20.army.mil and mirror sites:

```
PD1:<MSDOS.DSKUTL>
ASPIBIN.ZIP   67841 920131 Gnu Tar for SCSI tape drives, Adaptec 154xx
ASPIPAT.ZIP   21206 920131 Patches for ASPIBIN relative to Gnu Tar 1.10
ASPISRC.ZIP 221370 920131 Src for Gnu Tar for SCSI tape, Adaptec ctrlr
```

  I have no idea if a copy can be found on Compuserve; UNIXFORUM might have it, if any forum does.

* As far as MSDOS is concerned, the 1542A and the 1542B controllers are the same; with MSDOS, the 1542A should work as well as the 1542B. However, the hardware for these two boards is not 100% identical, and there is at least one (NON-MSDOS) program that initially did not work with a 1542A, but did work with a 1542B (BSD386 -- a 386 version of BSD Unix).

* In case anyone's curious, here's an edited copy of my CONFIG.SYS file:

```
FILES=40
BUFFERS=40
BREAK=ON
STACKS=10,256
```

```
DEVICE=c:dos.sys /d /n4
DEVICE=C:.SYS on RAM ROM DMA=32 ST:M X=F800-FFFF
DOS=HIGH,UMB
DEVICEHIGH=c:.sys
DEVICEHIGH=C:.EXE
shell = c:.com /p
```

Note that I'm using QEMM and ASPI4DOS 3.1. If I were using ASPI4DOS 3.0 or 3.0a, I'd probably have to use a CONFIG.SYS that looked like:

```
FILES=40
BUFFERS=40
BREAK=ON
STACKS=10,256
DEVICE=c:dos.sys /d /n4
DEVICE=C:.SYS on RAM ROM DMA=32 ST:M X=F800-FFFF
DOS=HIGH,UMB
DEVICEHIGH=c:.sys /V386
DEVICEHIGH=c:.sys
DEVICEHIGH=C:.EXE
shell = c:.com /p
```

If I weren't using ASPI4DOS, I'd probably use something that looked like:

```
FILES=40
BUFFERS=40
BREAK=ON
STACKS=10,256
DEVICE=c:.sys /V386
DEVICE=C:.SYS on RAM ROM DB=32 DMA=32 ST:M X=F800-FFFF
DOS=HIGH,UMB
DEVICEHIGH=c:.sys
DEVICEHIGH=C:.EXE
shell = c:.com /p
```

However, if I used a floppy-controller-based tape drive, or if I planned to record high-quality sound from a sound card, I would still need some way of changing the Adaptec's bus on/off times. The first two versions of CONFIG.SYS take care of this, but this last version doesn't.

How to Get SCSI Standards
(and other SCSI Books)

This file contains information on obtaining the various SCSI standards and other published information about SCSI. If you know of other published material on SCSI that should be included here, please leave a message on the SCSI BBS or send a Fax to John Lohmeyer 316-636-8889.

American National Standard (ANSI) for Small Computer System Interface (SCSI), X3.131-1986. This approved standard is available from:

American National Standards Institute
11 West 42nd Street
13th Floor
New York, NY 10036
Sales Department: (212) 642-4900

The Small Computer System Interface - 2 (SCSI-2) was approved by ANSI on August 31, 1990. Since then, the X3T9.2 committee has asked ANSI to return SCSI-2 for further editing. Revision 10d is in the works, but it is not finished yet. Revision 10c can be purchased from:

Global Engineering Documents
2805 McGaw
Irvine, CA 92714
(800) 854-7179 Outside USA and Canada: (714) 261-1455

Global has identified this document as X3.131-199x. (The latest revision is Rev 10c with a date of 3/9/90.)

If you are looking for a introduction to SCSI, try:

"SCSI: Understanding the Small Computer System Interface"
Written by NCR Corporation
Available from: Prentice Hall, Englewood Cliffs, NJ, 07632
Phone: (201) 767-5937 ISBN 0-13-796855-8

If you want an easy-to-read reference book on SCSI-2 that includes timing diagrams (but omits all command sets other than for disks, tapes, and processor devices), try:

"The SCSI Bench Reference"
ENDL Publications
14426 Black Walnut Court
Saratoga CA, 95070
(408) 867-6642

Also, coming soon from ENDL Publications, the "SCSI Encyclopedia". I've reviewed volume 1 and find it an excellent source of information about SCSI-2. Unfortunately, it is NOT inexpensive.

WDSCS-ATXT/WDATXT-FAAST
PC/XT or PC/AT SCSI Adapter

WDATXT-FASST KIT - An "unintelligent" SCSI host adapter that is compatible with the IBM XT, AT and compatible systems. It uses a 50 pin external SCSI bus "D" connector with a standard 50 pin internal SCSI cable. The WDATXT-FASST can be used as both a target and an initiator and it serves as an excellent tool for SCSI designers. It also provides a low cost alternative for end-users desiring to install a SCSI peripheral device such as a hard disk drive or a tape backup unit. The kit includes an 8-bit SCSI HBA board, manual, FASST software diskettes and an internal SCSI cable.

WE ARE IN THE PROCESS OF CONFIGURING A MORE COMPLETE USER GUIDE ON THIS PRODUCT
FOR THE BULLETIN BOARD. IN THE INTERIM, THE FOLLOWING JUMPER DEFINITIONS MAY BE HELPFUL.

WDSCS-ATXT
Jumper Settings

****
Jumper Block "W1"
****

The W1 jumper block is a spare register for the BIOS to use.

****
Jumper Block "W2"
****

X=Installed
0=Removed

SA8 7-8	SA7 5-6	SA6 3-4	SA5 1-2	I/O Port
X	X	X	X	200H
X	X	X	O	220H
X	X	O	X	240H
X	X	O	O	250H
X	O	X	X	280H
X	O	X	O	200H
X	O	O	X	2C0H
X	O	O	O	2E0H
O	X	X	X	300H
O	X	X	O	320H
O	X	O	X	340H
O	X	O	O	360H
O	O	X	X	380H
O	O	X	O	3A0H

O	O	O	X	3C0H
O	O	O	O	3E0H

****
Jumper Block "W3"
****

This jumper block is used to select the BIOS ROM Address location within the host system's memory space. The HBIC always assumes that host address bits BA18 (MA18) and BA19 (MA19) are logic "1", therefore, the starting address cannot be lower than location C0000H. The following defines the different jumper settings and is followed by an example:

MA19	MA18	MA17	MA16	MA15	MA14
1	1	------Jumpers----------			

Signals MA14-MA17 (BA14-BA17):

Jumper OFF = logic "1"
Jumper ON = logic "0"

EXAMPLE: ATXT BIOS ROM Address = C8000H

Jumpers	7-8	5-6	3-4	1-2
Address Bits		MA17	MA16	MA15	MA14
Jumper Settings	ON	ON	OFF	ON

****
Jumper Block "W4"
****

This jumper block is to select Interrupt Request channel. Only one jumper shall be used to select one Channel.

"W4":	1-2	3-4	5-6	7-8	9-10	11-12	
Channel:	BIRQ2	BIRQ3	BIRQ4	BIRQ5	BIRQ6	BIRQ7	
	ON	--	--	--	--	--	= IRQ2 SELECTED
	--	ON	--	--	--	--	= IRQ3 SELECTED

--	--	ON	--	--	--	= IRQ4 SELECTED
--	--	--	ON	--	--	= IRQ5 SELECTED
--	--	--	--	ON	--	= IRQ6 SELECTED
--	--	--	--	--	ON	= IRQ7 SELECTED

****

Jumper Block "W5"

****

W5 is used to tie together two different signal lines between Jumper Blocks "W1" and "W2".

"W5":	1-2	3-4	
	OFF	OFF	= Neither signal line tied.
	ON	OFF	= "W1" 9-10 tied to "W2" 1-2 only.
	OFF	ON	= "W1" 11-12 tied to "W2" 3-4 only.
	ON	ON	= Both signal lines tied.