ANSI Document X3T9.3 No. 185

SASI

SHUGART ASSOCIATES SYSTEM INTERFACE

(This document is subject to change without notice)

● Indicates changes from Revision C and D.

Rev. E 4/2/82

INTRODUCTION

This Shugart Associates System Interface defines a local I/O Bus specification which can be operated at data rates up to an estimated 1.5 megabytes per second, depending upon circuit implementation choices. The primary objective of the interface is to provide host computers with device independence, within a type of devices. Thus, disk drives, tape drives, printers and even communication devices, of different type, can be added to the host computer without requiring modifications to generic system hardware or software. Provision is made for the ready addition of non-generic features and functions.

The interface is designed for the compatible operation of a wide range of host and device capabilities, including low cost, low function capabilities, as well as higher cost, higher function capabilities. This gives the interface a great deal of generality.

The interface uses "logical" rather than "physical" addressing for all data structures. All data is addressed as logical blocks up to the maximum number of blocks in a device; and, each device can be interrogated to determine how many blocks it contains.

Provision is made for cable lengths up to 15 meters using differential drivers and receivers. An in-cabinet mounting using cable lengths up to six meters and single ended drivers and receivers is also available.

The interface protocol includes provision for the connection of multiple initiators (SASI bus devices capable of initiating an operation) and multiple targets (SASI bus devices capable of responding to a request to perform an operation). Arbitration (i.e., bus-contention logic) is built into the architecture of SASI. A logical priority system awards interface control to a SASI bus device that wins arbitration.

SCOPE

The scope of this document, at this level of revision, encompasses primarily rotating memory devices (disk drives). It consists of two specifications.

The specification in Part A (previously referred to as a hardware specification) is now called a "physical path" specification. It defines the physical components of the interface (as well as the function of each line). It contains all of the specifications associated with obtaining and maintaining control of the SASI bus. Additionally, it contains definitions of the message protocol which links the SASI bus devices to each other and controls the physical path between them.

The specification in Part B (previously referred to as the software command set) is herein referred to as the functional specification. It defines the operation of the interface in detail, which in turn defines the operation of the bus devices (controllers or hosts). It also defines the functions which are standard ... the functions which are optional ... and the responses to the execution of these functions.

SASI REVISION LIST

Revision Letter	Description	Date
-	First presentation to ANSI committee X3T9-3 (2 weeks following announcement in Electronic Design)	9-15-81
A	o Revised to reflect changes to meet NCR requirements	1-5-82
	o Further revised to add description of commands	1-12-82
B	Edited and improved for working meeting with NCR, Adoptec and Optimum in Sunnyvale	1-25-82
C	o Rewritten in preparation for presentation to ANSI	1-29-82
	o Addenda and Errata for Rev. C	2-3-82
D	Revised to make all known corrections in preparation for "working session" in Sunnyvale on March 3 and 4	2-19-82
E	Revised to incorporate changes resulting from "working session" in Sunnyvale in preparation for April 26 and 27 meeting in Phoenix.	4-1-82

PART A

PHYSICAL PATH SPECIFICATION

1.0 SCOPE

This is the physical path definition of the Shugart Associates System Interface (SASI), which is designed to provide an efficient method of communication between computers and peripheral input/output devices.

SASI is implemented using an eight-port bus which includes the following key features:

- Single or multiple host computer system.
- Bus contention handled via self-arbitration on a prioritized basis.
- Accomodation of multiple peripheral device types.
- Asynchronous communication of up to an estimated 1.5 MBytes/sec, depending upon circuit implementation choices.
- Multiple overlap of peripheral device operations.
- Direct copy between peripheral devices.
- Oriented toward intelligent peripheral devices.

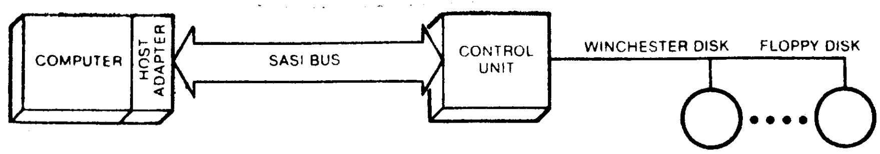
The practical application of this SASI physical path specification will require a higher level software command set document (see Part B).

2.0 SASI BUS

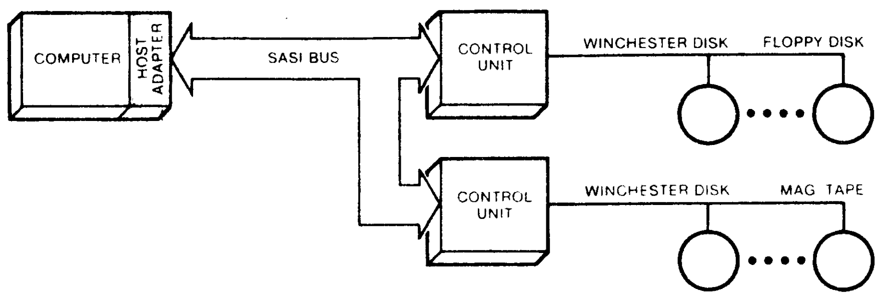
Communication on the SASI Bus is allowed between two SASI **BUS PORTS** at any given time. There is a maximum of eight (8) **BUS PORTS**. Each port is attached to a SASI DEVICE (e.g., peripheral device controller or host computer).

When two **SASI BUS DEVICES** communicate on the bus, one acts as an **INITIATOR** and the other acts as a **TARGET**. The **INITIATOR** (typically a host computer) starts an operation and the **TARGET** (typically a peripheral device controller) performs the operation. A **SASI BUS DEVICE** will usually have a fixed role as an **INITIATOR** or **TARGET**, but some may be able to assume either role.

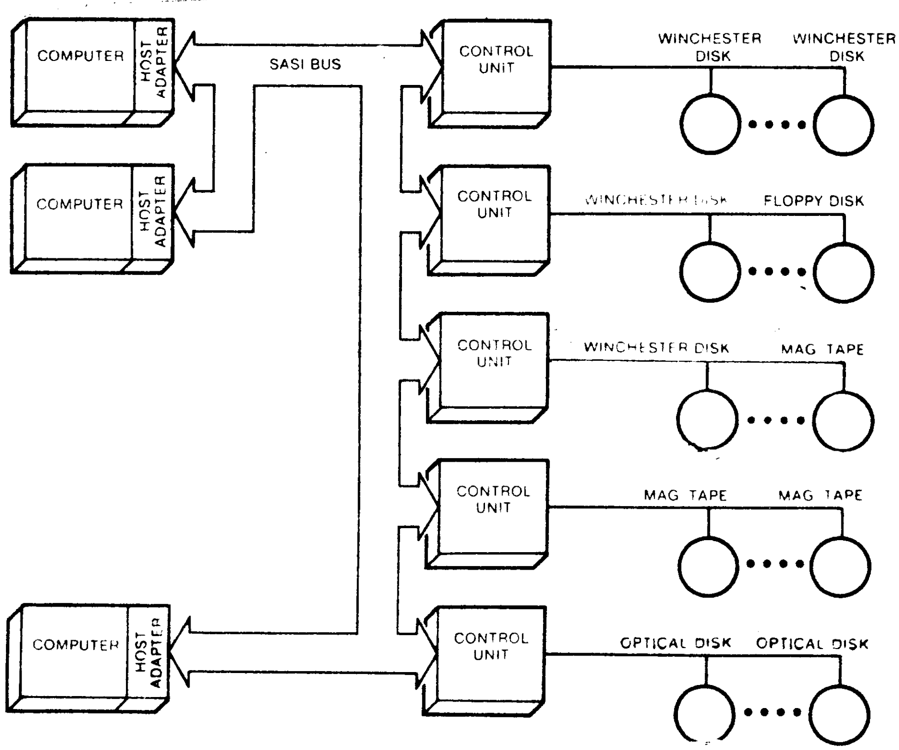
An **INITIATOR** may address up to eight (8) peripheral **I/O** devices that are connected to a **TARGET**. Three sample system configurations are shown in Figure 2.0.



SIMPLE SYSTEM



BASIC TWO CONTROL UNIT SYSTEM



COMPLEX SYSTEM

Up to 8 SASI **DEVICES** can be supported by the SASI bus. They can be any combination of host CPU's and intelligent controllers.

Figure 2.0 SAMPLE SASI CONFIGURATIONS

Certain bus functions are assigned to the **INITIATOR** and certain bus functions are assigned to the **TARGET.** The **INITIATOR** may arbitrate for the bus and select a particular **TARGET.** The **TARGET** may request the transfer of **COMMAND, DATA, STATUS** or other information on the bus, and in some cases it may arbitrate for the bus and reselect an **INITIATOR** for the purpose of continuing some operation.

Data transfers on the bus are asynchronous and follow a defined **REQUEST/ACKNOWLEDGE handshake** protocol. One eight-bit byte of information may be transferred with each handshake.

2.1 SASI PHYSICAL PATH PHILOSOPHY

Consider the system shown in Figure 2.0 where an **INITIATOR** and **TARGET** communicate on the **SASI** bus in order to execute some command (e.g., **READ** or **WRITE** data). Since it would be combersome to briefly desribe these functions at the detailed level of the bus, a higher level of description is used here. Only one of a number of possible partitions of the physical/functional interface is presented for illustration.

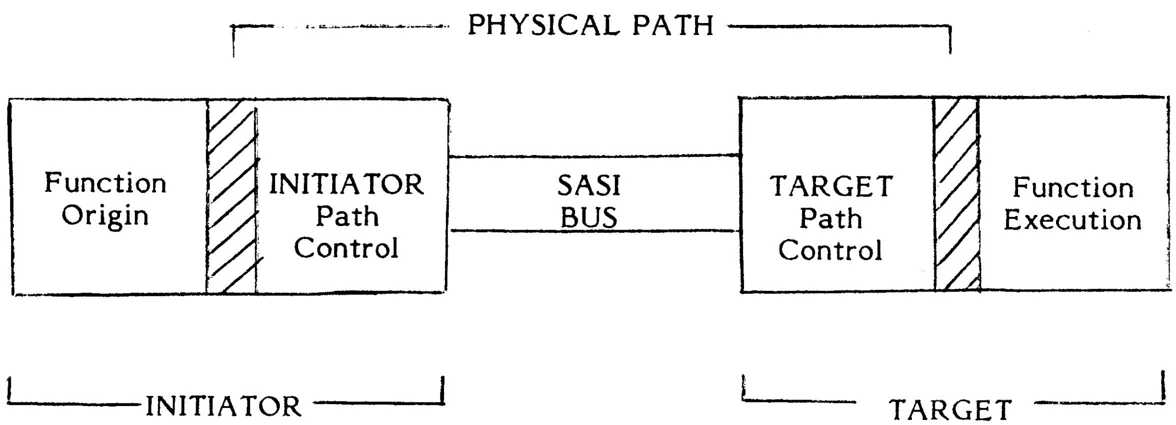


Figure 2.0 PHYSICAL PATH

2.1.1 Pointers

Note that in the **SASI** architecture there are three "conceptual" memory address pointers which are used to access three "conceptual" byte wide memory areas. The pointers, which as presented here, reside in the **INITIATOR** path control, are reserved for **COMMAND, DATA** and **STATUS.**

After the pointers are initially loaded by the **INITIATOR,** their movement is under strict control of the **TARGET.** When the **TARGET** transfers a byte of information to or from the three areas, the corresponding pointer increments.

2.1.2 Typical Functions (External to Path Control)

Listed below are some typical functions that affect the physical path but come from outside the physical path boundary.

- **Establish Path** — This function allows the **INITIATOR** to set up the physical path (the physical and logical connection between the **INITIATOR** and a particular peripheral device for the purpose of executing some peripheral device command. This may involve arbitration to gain ownership of the **SASI** bus. The "Establish Path" function requires the peripheral device address (i.e., **SASI TARGET** bus address and **LUN** within that address). Also required are the three pointers to the **COMMAND, DATA** and **STATUS** areas. A saved copy of these pointers may also be required (see "Reestablish Path" and "Restore State" functions).

- **Get Command** — This function allows a **TARGET** to fetch a **COMMAND** from the area pointed to by the **COMMAND** pointer.

- **Get Data and Send Data** — These functions allow the **TARGET** to transfer data to or from the area pointed to by the **DATA** pointer.

- **Send Status** — This allows the **TARGET** to send **STATUS** information for a command to the area pointed to by the **STATUS** pointer.

- **End of Command** — This function is sent by the **TARGET** to indicate that the current command terminated and that valid status has been sent. (Note that the current command may be linked to another command.)

- **End Path** — The **TARGET** uses this function to completely clear the physical path with regards to the currently attached peripheral device. A new command can thus be accepted for this device.

- **Break Path** — This function is done by the **TARGET** to temporarily break the physical path so that others may use the **SASI** bus.

- **Reestablish Path** — The **TARGET** uses this function to establish a physical path connection that was temporarily broken by the "Break Path" function. An indication of the **INITIATOR**'s **SASI** bus address and the peripheral device address are required. **SASI** bus arbitration is also required. The **INITIATOR** must restore the **COMMAND, DATA** and **STATUS** pointers to their last saved values.

- **End of Link** — This function is sent by the **TARGET** to indicate that the current command has terminated but that it was linked to another command. The initiator is then allowed to set up its **COMMAND, DATA** and **STATUS** pointers to the initial state of the next command.

- **Save State** — The **TARGET** uses this function to direct the **INITIATOR** to save a copy of the current (active) **COMMAND, DATA** and **STATUS** pointers for the currently connected peripheral device.

- **Restore State** — The **TARGET** uses this function to direct the **INITIATOR** to load the current (active) **COMMAND, DATA** and **STATUS** pointers with the last saved values for the currently connected peripheral device..

2.2 BUS SIGNALS

There are nine (9) control signals and nine (9) data signals (including parity), as listed below:

- BSY
- SEL
- C/D
- I/O
- MSG
- REQ
- ACK
- ATN
- RST
- DB (7-0, P) (Data Bus)

These signals are described below:

- **BSY (BUSY)** — An "or-tied" signal which generally indicates when the bus is being used.

- **SEL (SELECT)** — An "or-tied" signal used by an **INITIATOR** to select a **TARGET** or by a **TARGET** to reselect an **INITIATOR.**

- **C/D (CONTROL/DATA)** — A signal driven by a **TARGET;** it generally indicates whether **CONTROL** or **DATA** information is on the data bus. Assertion indicates **CONTROL.**

- **I/O (INPUT/OUTPUT)** — A signal driven by a **TARGET** which controls the direction of data movement on the data bus with respect to an **INITIATOR.** Assertion indicates **INPUT** to the **INITIATOR.**

- **MSG (MESSAGE)** — A signal driven by a **TARGET** during the **MESSAGE** phase.

- **REQ (REQUEST)** — A signal driven by a **TARGET** to indicate a request for a **REQ/ACK** data transfer handshake.

- **ACK (ACKNOWLEDGE)** — A signal driven by an **INITIATOR** to indicate an acknowledgement for a **REQ/ACK** data transfer handshake.

- **ATN (ATTENTION)** — A signal driven by an **INITIATOR** to indicate the **ATTENTION** condition.

- **RST (RESET)** — An "or-tied" signal which indicates the **RESET** condition.

- **DB(7-0,P) (DATA BUS)** — Eight data bit signals, plus a parity bit signal which form a **DATA BUS. DB(7)** is the most significant bit and has the highest priority during **ARBITRATION.** Bit number, significance and priority decrease downward to **DB(0).**

  Data parity **DB(P)** is odd. The use of parity is a system option (i.e., a system is configured so that all **SASI DEVICES** on a bus generate parity and have parity detection enabled, or all **SASI DEVICES** have parity detection disabled). Parity is not guaranteed valid during the **ARBITRATION** phase.

Each of the eight data signals **DB(7)** through **DB(0)** is uniquely assigned as a **TARGET** or **INITIATOR**'s own **SASI BUS** address (i.e., **SASI DEVICE ID**). This **SASI DEVICE ID** would normally be assigned and strapped in a **SASI DEVICE** during system configuration.

During **ARBITRATION**, a **SASI DEVICE** that desires the use of the **SASI BUS** asserts its assigned data bit (**SASI DEVICE ID**) but leaves the other data bits in the passive (non-driven) state. Thus, a **SASI DEVICE** may use one data bit driver for its **SASI DEVICE ID** during **ARBITRATION** and a different driver when driving the complete data bus in other phases of the bus operation.

2.3 BUS PHASES

The bus has eight (8) distinct operational phases:

- **BUS FREE** Phase
- **ARBITRATION** Phase
- **SELECTION** Phase
- **RESELECTION** Phase
- **COMMAND** Phase
- **DATA** Phase
- **STATUS** Phase
- **MESSAGE** Phase

(COMMAND, DATA, STATUS and MESSAGE: These phases are collectively termed the **INFORMATION TRANSFER** phases.)

The bus can never be in more than one phase at any given time. Unless otherwise noted, the following descriptions assume that bus signals which are not mentioned will not be asserted.

2.3.1 Bus Free Phase

Purpose:

The **BUS FREE** phase is used to indicate that no **SASI DEVICE** is actively using the bus and that the bus is available for subsequent users.

Description:

The **BUS FREE** phase is created by the deassertion and passive release of all bus signals.

SASI DEVICES shall detect the **BUS FREE** phase by the simultaneous (within a **DESKEW DELAY**) condition of both **SEL** and **BSY** not asserted while the **RESET** condition is not active.

During the **BUS FREE** phase, all active **SASI DEVICES** shall immediately deassert and passively release all bus signals (within a **BUS CLEAR DELAY**) after the **BSY** and **SEL** signals are deasserted from the bus.

2.3.2 Arbitration Phase

Purpose:

The **ARBITRATION** phase allows one **SASI DEVICE** to gain control of the bus so that this device can assume the role of an **INITIATOR** or **TARGET**.

Option:

Implementation of the **ARBITRATION** phase is a system option. Systems with no **ARBITRATION** phase can have only one **INITIATOR**. The **ARBITRATION** phase is required for systems which use the **RESELECTION** phase.

Description:

After a **SASI DEVICE** that wants to arbitrate for the bus detects the **BUS FREE** phase it waits a minimum of **BUS FREE DELAY** and a maximum of **BUS SET DELAY** in order to assert **BSY** and its own **SASI DEVICE ID** on the bus. (The time required for the device to detect the **BUS FREE** phase should be included in the measurement of the amount of time that the device waits.)

Note: The **SASI DEVICE ID** is asserted on the **DATA BIT** signal that corresponds to the device's unique **BUS ADDRESS**. All other **DATA BUS** drivers in this **SASI DEVICE** must be passive. Data parity is not guaranteed valid during **ARBITRATION**.

Any **SASI DEVICE** that is arbitrating will immediately clear itself from arbitration (within a **BUS CLEAR DELAY** time) by deasserting its **BSY** and **ID** signals if **SEL** is asserted by any other **SASI DEVICE** that has won arbitration.

After an **ARBITRATION DELAY** (measured from the assertion of **BSY**) the **SASI DEVICE** that is arbitrating examines the **DATA** bus. If a higher priority **SASI DEVICE ID** is on the bus (**DB(7)** = highest) then the **SASI DEVICE** clears itself from **ARBITRATION** by deasserting its **BSY** and **ID** signals. If the **SASI DEVICE** determines that its own **ID** is the highest asserted, then it wins **ARBITRATION** and asserts SEL; (after the assertion of **SEL** the **SASI DEVICE** must wait a minimum of two **BUS SETTLE DELAYS** before changing the assertion of any bus signals).

2.3.3 Selection Phase

Purpose:

The **SELECTION** phase allows an **INITIATOR** to select a **TARGET** for the purpose of initiating some **TARGET** function(s), (e.g. read or write data).

Description:

Note: All during the **SELECTION** phase the **I/O** signal is not asserted so that this phase can be distinguished from the **RESELECTION** phase.

In systems where the **ARBITRATION** phase is not implemented, the INITIATOR first detects the **BUS FREE** phase and then waits a minimum of **BUS SETTLE DELAY**. Then the **INITIATOR** asserts the **DATA BUS** with both the desired **TARGET's ID** and its own **INITIATOR ID** (see option note below). After two **DESKEW DELAYS** the **INITIATOR** asserts **SEL**.

In systems with **ARBITRATION** implemented, the **BSY** and **SEL** signals will be asserted by an **INITIATOR** when going from the **ARBITRATION** phase to the **SELECTION** phase. Also, a minimum of two **BUS SETTLE DELAYS** will have been waited. The **INITIATOR** then asserts the **DATA BUS** with both the desired **TARGET's ID** and its own **INITIATOR ID** (see option note below). After these assertions, the **INITIATOR** waits at least two **DESKEW DELAYS** and then deasserts **BSY**. The **INITIATOR** then waits a **BUS SETTLE DELAY** before looking for a response from the **TARGET**.

In all systems, the selected **TARGET** detects the simultaneous (within a **DESKEW DELAY**) condition of **SEL** and its own **SASI DEVICE ID** asserted, and both **BSY** and **I/O** not asserted. The selected **TARGET** may sample the **DATA BUS** in order to try to determine the **SASI DEVICE ID** of the **INITIATOR** that is doing the selecting (see option note below). The selected **TARGET** then responds by asserting **BSY**. (Note that in systems with parity implemented, the **TARGET** will not respond to its **SASI DEVICE ID** if bad parity is detected on the bus. Note also that if more than two **SASI DEVICE IDs** are on the bus, the **TARGET** should consider this an illegal condition and should not respond to the selection.)

At least two **DESKEW DELAYS** after the **INITIATOR** detects **BSY** sent from the **TARGET**, it deasserts **SEL** and may change the **DATA** signals.

Option:

Initiators that do not implement the **RESELECTION** phase are allowed the option of asserting only one **BUS DEVICE ID** (the **TARGET**'s **ID**) during **SELECTION.**

2.3.4 Reselection Phase

Purpose:

The **RESELECTION** phase allows a **TARGET** to reconnect to an **INITIATOR** for the purpose of continuing some operation that was previously started by the **INITIATOR** but was interrupted by the **TARGET**; (i.e., the **TARGET** disconnected by allowing a **BUS FREE** phase to occur before the operation was complete).

Option Note:

RESELECTION can only be used in systems that have **ARBITRATION** implemented.

Description:

After the **TARGET** has gone through the **ARBITRATION** phase, it will be asserting **BSY** and **SEL** and will have waited a minimum of two **BUS SETTLE DELAYS.** The **TARGET** then asserts the **I/O** signal and also asserts the **DATA BUS** with the desired **INITIATOR**'s **ID** and its own **TARGET ID.** After these assertions the **TARGET** waits at least two **DESKEW DELAYS** and then deasserts **BSY.** The **TARGET** then waits a **BUS SETTLE DELAY** before looking for a response from the **INITIATOR.**

The reselected **INITIATOR** detects the simultaneous (within a **DESKEW DELAY**) condition of **SEL, I/O** and its own **SASI DEVICE ID** asserted, and **BSY** not asserted. The reselected **INITIATOR** may sample the **DATA BUS** to determine the **SASI DEVICE ID** of the **TARGET** that is doing the **RESELECTION.** The reselected **INITIATOR** then responds by asserting **BSY.** (Note that in systems with parity implemented the **INITIATOR** will not respond to a **DEVICE ID** that has bad parity. Note also that if more than two **SASI DEVICE ID**'s are on the bus, the **INITIATOR** should consider this an illegal condition and should not respond to the **RESELECTION.)**

After the **TARGET** detects **BSY** sent from the **INITIATOR,** the **TARGET** will: (1) also assert **BSY** and continue the assertion until it is done using the bus; and (2) wait at least two **DESKEW DELAYS** and then deasserts **SEL** and possibly change the **I/O** and **DATA** signals.

The reselected **INITIATOR** detects the deassertion of **SEL** and releases its assertion of **BSY.**

2.3.5 Information Transfer Phases (COMMAND, DATA, STATUS and MESSAGE Phases)

Common Notes:

The **COMMAND, DATA, STATUS** and **MESSAGE** phases can all be grouped together as the **INFORMATION TRANSFER** phases because they are all used to transfer data or control information via the **DATA BUS.** The actual contents of the information is beyond the scope of this section.

The **C/D, I/O** and **MSG** signals are used to distinguish between the different **INFORMATION TRANSFER** phases. See Table 2.0.

TABLE 2.0 INFORMATION TRANSFER PHASES

SIGNAL					
MSG	C/D	I/O	PHASE NAME	DIRECTION OF INFORMATION XFER	COMMENT
0	0	0	DATA OUT PHASE	(INIT to TARG)	DATA PHASES
0	0	1	DATA IN PHASE	(INIT from TARG)	
0	1	0	COMMAND PHASE	(INIT to TARG)	
0	1	1	STATUS PHASE	(INIT from TARG)	
1	0	0	*		
1	0	1	*		
1	1	0	MSG OUT PHASE	(INIT to TARG)	MESSAGE PHASES
1	1	1	MSG IN PHASE	(INIT from TARG)	

Notes: 0 = **SIGNAL DEASSERTION,** 1 = **SIGNAL ASSERTION.**
INIT = **INITIATOR, TARG** = **TARGET.**
* = Not used.

The **INFORMATION TRANSFER** phases use one or more **REQ/ACK** handshakes to control the data transfer. Each **REQ/ACK** allows the transfer of one byte of data. The **REQ/ACK** handshake starts with the **TARGET** asserting the **REQ** signal. The **INITIATOR** responds by asserting the **ACK** signal. The **TARGET** then deasserts the **REQ** signal. The **INITIATOR** again responds by deasserting the **ACK** signal.

If the **I/O** signal is asserted, data will be **INPUT** into the **INITIATOR** from the **TARGET.** The **TARGET** shall guarantee that valid data is available on the bus at the **INITIATOR**'s port at least a **DESKEW DELAY** before the assertion of **REQ** is at the **INITIATOR**'s port. The data shall remain valid until the assertion of **ACK** by the **INITIATOR.** It shall be the **TARGET**'s responsibility to compensate for the maximum **CABLE SKEW** and the skew of its own drivers.

If the **I/O** signal is **NOT** asserted, data will be **OUTPUT** from the **INITIATOR** into the **TARGET**. The **INITIATOR** shall guarantee that valid data is available on the bus at the **TARGET**'s port at least a **DESKEW DELAY** before the assertion of **ACK** is at the **TARGET**'s port. Valid data shall remain on the bus until the **TARGET** deasserts **REQ**. It shall be the **INITIATOR**'s responsibility to compensate for the maximum **CABLE SKEW** and the skew of its own drivers.

During each **INFORMATION TRANSFER** phase the **BSY** line shall remain asserted and the **SEL** line shall remain deasserted. Additionally, during each **INFORMATION TRANSFER** phase the **TARGET** shall continuously envelope the **REQ/ACK** handshake(s) with the **C/D**, **I/O** and **MSG** signals in such a manner that these control signals are valid for a **BUS SETTLE DELAY** before the **REQ** of the first handshake and remain valid until the deassertion of **ACK** at the end of the last handshake.

2.3.6 Command Phase

Purpose:

The **COMMAND** phase allows the **TARGET** to request command information from the **INITIATOR**.

Description:

The **TARGET** asserts the **C/D** signal and deasserts the **I/O** and **MSG** signals during the **REQ/ACK** handshake(s) of this phase.

2.3.7 Data Phase

The **DATA** phase is a term that encompasses both the **DATA IN** phase and the **DATA OUT** phase.

2.3.7.1 Data In Phase

Purpose:

The **DATA IN** phase allows the **TARGET** to request that data be **INPUT** to the **INITIATOR** from the **TARGET**.

Description:

The **TARGET** asserts the **I/O** signal and deasserts the **C/D** and **MSG** signals during the **REQ/ACK** handshake(s) of this phase.

2.3.7.2 Data Out Phase

Purpose:

The **DATA OUT** phase allows the **TARGET** to request that data be **OUTPUT** from the **INITIATOR** to the **TARGET**.

Description:

The **TARGET** deasserts the **C/D, I/O** and **MSG** signals during the **REQ/ACK** handshake(s) of this phase.

2.3.8 Status Phase

Purpose:

The **STATUS** phase allows the **TARGET** to request that status information be sent from the **TARGET** to the **INITIATOR.**

Description:

The **TARGET** asserts **C/D** and **I/O** and it deasserts the **MSG** signal during the **REQ/ACK** handshake(s) of this phase.

2.3.9 Message Phase

The **MESSAGE** phase is a term that encompasses the **MESSAGE IN,** and **MESSAGE OUT** phases.

2.3.9.1 Message In Phase

Purpose:

The **MESSAGE IN** phase allows the **TARGET** to request that **MESSAGES** be **INPUT** to the **INITIATOR** from the **TARGET.**

Description:

The **TARGET** asserts **C/D, I/O** and **MSG** during the **REQ/ACK** handshake(s) of this phase.

2.3.9.2 Message Out Phase

Purpose:

The **MESSAGE OUT** phase allows the **TARGET** to request that a **MESSAGE** be **OUTPUT** from the **INITIATOR** to the **TARGET.** The **TARGET** may invoke this phase at its convenience only in response to the **ATTENTION** condition created by the **INITIATOR.**

Description:

In response to the **ATTENTION** condition, the **TARGET** asserts **C/D** and **MSG** and deasserts the **I/O** signal during the **REQ/ACK** handshake(s) of this phase. (See **ATTENTION** condition description.)

2.3.10 Signal Restrictions Between Phases:

When the **BUS** is between two phases, the following restrictions shall apply to the bus signals:

- The **BSY, SEL, REQ** and **ACK** signals shall not change.
- The **C/D, I/O, MSG** and **DATA** signals may change.
- The **ATN** and **RST** signals may change as defined under the descriptions for the **ATTENTION** and **RESET** conditions.

2.4 BUS CONDITIONS

The bus has two asynchronous conditions:

- **ATTENTION** Condition.
- **RESET** Condition.

These conditions cause certain **BUS DEVICE** actions and can alter the bus phase sequence.

2.4.1 Attention Condition

Purpose:

The **ATTENTION** condition allows an **INITIATOR** to inform a **TARGET** that the **INITIATOR** has a **MESSAGE** ready. The **TARGET** may get this message at its convenience by performing a **MESSAGE OUT** phase.

Description:

The **INITIATOR** creates the **ATTENTION** condition by asserting **ATN** at any time except during the **ARBITRATION** or **BUS FREE** phases.

The **TARGET** may respond with the **MESSAGE OUT** phase.

The **INITIATOR** may keep **ATN** asserted if more than one byte is to be transferred.

The **INITIATOR** deasserts the **ATN** signal during: (1) the **RESET** condition, or when the bus goes to a **BUS FREE** phase, or (2) while the **REQ** signal is asserted and the **ACK** signal is not yet asserted during the last **REQ/ACK** handshake of a **MESSAGE OUT** phase.

2.4.3 Reset Condition

Purpose:

The **RESET** condition is used to immediately clear all **SASI DEVICES** from the bus and to reset these devices and their associated equipment (as required).

Description:

Note: This condition takes precedence over all other phases and conditions.

The **RESET** condition can occur at any time.

Any **SASI DEVICE** (whether active or not) can create the **RESET** condition. The **RESET** condition should be used with caution because of its possible effects.

The **RESET** condition is created by the assertion of the **RST** signal.

When the **RESET** condition exists, all **SASI DEVICES** will immediately (within a **BUS CLEAR DELAY**) deassert and passively release all bus signals except **RST** itself. In addition all **SASI DEVICES** and their associated equipment shall be reset to initial conditions (as required).

The **RESET** condition shall be on for a minimum of a **RESET HOLD TIME**.

During the **RESET** condition, no bus signal except **RST** is guaranteed to be in a valid state.

Regardless of what bus phase may have been interrupted, following the **RESET** condition the bus shall go to a **BUS FREE** phase and then start a normal phase sequence.

2.5 PHASE SEQUENCING

The order in which phases are used on the bus follows a prescribed sequence.

In all systems, the **RESET** condition can interrupt any phase and is always followed by the **BUS FREE** phase. Also, any other phase can be followed by the **BUS FREE** phase.

In systems where the **ARBITRATION** phase is not implemented, the allowable sequencing is shown in Figure 2.1. The normal progression would be from the **BUS FREE** phase to **SELECTION**, and from **SELECTION** to one or more of the **INFORMATION TRANSFER** phases (**COMMAND**, **DATA**, **STATUS** or **MESSAGE**).

In systems where the **ARBITRATION** phase is implemented, the allowable sequencing is shown in Figure 2.2. The normal progression would be from the **BUS FREE** phase to **ARBITRATION**, from **ARBITRATION** to **SELECTION** or **RESELECTION**, and from **SELECTION** or **RESELECTION** to one or more of the **INFORMATION TRANSFER** phases (**COMMAND, DATA, STATUS** or **MESSAGE**).

There are no restrictions on the sequencing between **INFORMATION TRANSFER** phases. A phase may even follow itself (e.g., a **DATA** phase may be followed by another **DATA** phase).

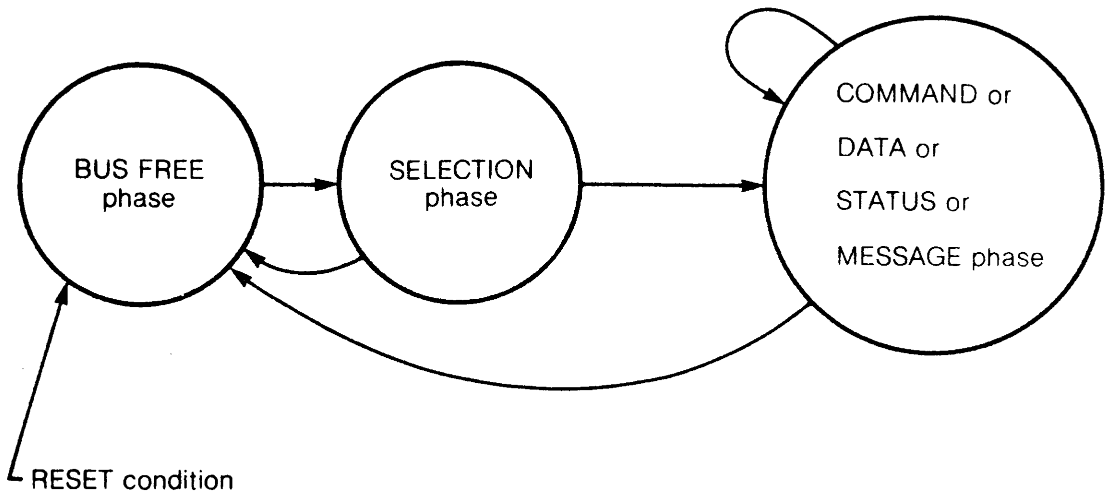


Figure 2.1 PHASE SEQUENCING
(For systems with no arbitration)

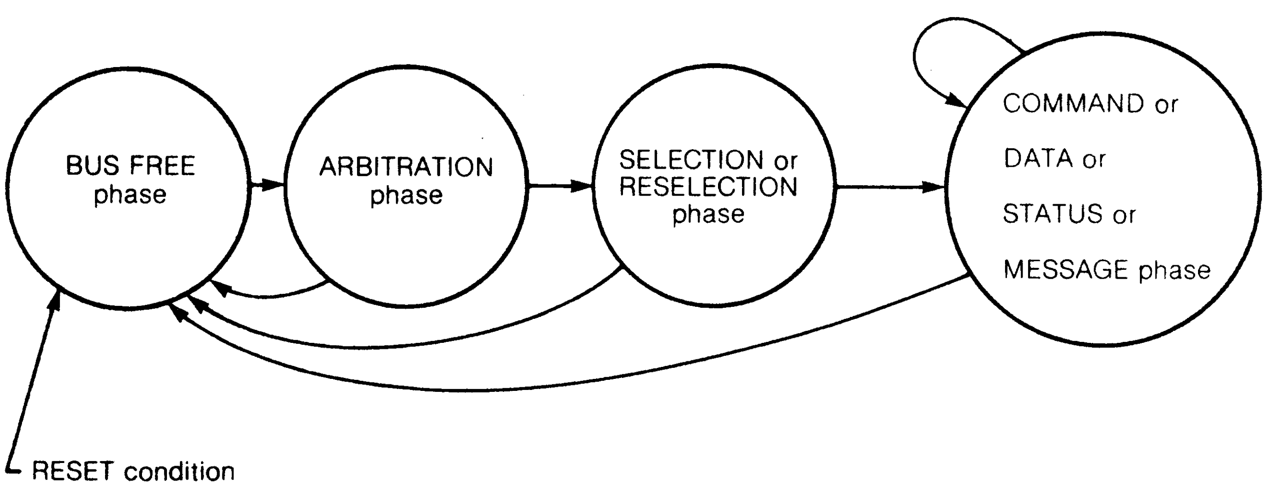


Figure 2.2 PHASE SEQUENCING
(For systems with arbitration)

2.6 SIGNAL ASSERTIONS

(To be specified.)

2.7 MESSAGE SYSTEM

The message system allows communication between an **INITIATOR** and **TARGET** for the purpose of physical path management.

2.7.1 Messages

The messages are listed along with their coded values (in HEX) and their definitions.

COMMAND COMPLETE (00) — Sent from a **TARGET** to an **INITIATOR** to indicate that the execution of a command (or series of linked commands) has terminated and that valid status has been sent to the **INITIATOR**.

Note 1: This command may have been executed successfully or unsuccessfully as indicated in the status.

EXTENDED MESSAGE FOLLOWS (01) — Sent from either the **INITIATOR** or **TARGET** to indicate that a multiple byte message will follow. (Note that no extended messages are currently defined, but that the first byte of an extended message has been reserved as a length indicator for the number of bytes to follow. A value of zero indicates 256 bytes.)

SAVE STATE (02) — Sent from a **TARGET** to direct an **INITIATOR** to save a copy of the pointers which define the present state of the physical path for the currently attached **LUN**. (See SASI PHYSICAL PATH PHILOSOPHY description.)

RESTORE STATE (03) — Sent from a **TARGET** to direct an **INITIATOR** to restore its pointers to the most recently saved state for the currently attached **LUN**. (See SASI PHYSICAL PATH PHILOSOPHY description.)

DISCONNECT (04)

Sent from a TARGET to inform an **INITIATOR** that the present physical path is going to be broken (the **TARGET** will disconnect by releasing **BSY**), but that a later reconnect will be required in order to complete the current operation.

TRANSMISSION ERROR (05)

Sent from an **INITIATOR** to inform a **TARGET** that an **INITIATOR** detected retryable error has occured since the last time the state of the physical path was saved.

Note: Commonly, this is for a data parity error.

SELECTIVE RESET (06)

Sent from an **INITIATOR** to direct a **TARGET** to abort the current command. No further action is required for this command (i.e., no command retry or status sent). If the **TARGET** is busy or reserved with another **INITIATOR**, the **TARGET** should respond with **MESSAGE REJECT**.

MESSAGE REJECT (07)

Sent from either the **INITIATOR** or **TARGET** to indicate that the last message it received was inappropriate or has not been implemented.

Note: In order to indicate its intentions of sending this message, the **INITIATOR** must assert the **ATN** signal prior to its release of ACK for the **REQ/ACK** handshake of the message that will be rejected. This provides an interlock so that the **TARGET** can determine which message will be rejected.

NO OPERATION (08)

Sent from an **INITIATOR** in response to a **TARGET**'s request for a message when the **INITIATOR** does not currently have any other valid message to send.

MESSAGE PARITY ERROR (09)

Sent from either the **INITIATOR** or **TARGET** to indicate that the last message it received had a parity error.

Note: In order to indicate its intentions of sending this message, the **INITIATOR** must assert the **ATN** signal prior to its release of **ACK** for the **REQ/ACK** handshake of the message that has the parity error. This provides an interlock so that the **TARGET** can determine which message has the parity error.

LINKED COMMAND COMPLETE (0A)

Sent from a **TARGET** to an **INITIATOR** to indicate that the execution of a linked command has completed and that status has been sent. The **INITIATOR** is then allowed to set up the pointers for the initial state for the next linked command.

LINKED COMMAND COMPLETE (WITH FLAG) (0B)

Sent from a **TARGET** to an **INITIATOR** to indicate that the execution of a linked command (with the **FLAG** bit set) has completed and that status has been sent. The **INITIATOR** is then allowed to set up the pointers for the initial state of the next linked command. Typically the **FLAG** would cause an interrupt in the **INITIATOR**.

IDENTIFY (80 to FF)

This message can be sent by either the **INITIATOR** or **TARGET**. It is used to establish the physical path connection between an **INITIATOR** and **TARGET** for a particular **LUN**.

Bit 7	This bit is always set to distinguish this message from the others.
Bit 6	This bit is only set by the **INITIATOR**. When it is set, it indicates that the **INITIATOR** has the ability to accommodate disconnection and reconnection.
Bits 5 - 3	Reserved.
Bits 2 - 0	These bits specify an **LUN** address in a **TARGET**.

2.7.2 Message Protocol

All systems are required to implement the **COMMAND COMPLETE** message. A functional system can be constructed without using any of the other messages if some alternate means is provided for specifying the **LUN** (e.g., some bits in the command).

SASI DEVICES indicate their ability to accommodate more than the **COMMAND COMPLETE** message by asserting or responding to the **ATN** signal. The **INITIATOR** indicates this by creating the **ATTENTION** condition before it releases BSY when going through the **SELECTION** phase. The **TARGET** indicates its ability to accommodate more messages by responding to the **ATTENTION** condition with the **MESSAGE OUT** phase after going through the **SELECTION** phase.

The first message sent by the **INITIATOR** after the **SELECTION** phase is the **IDENTIFY** message. This allows the establishment of the physical path for a particular **LUN** specified by the **INITIATOR.** After **RESELECTION,** the **TARGET**'s first message is also **IDENTIFY.** This allows the physical path to be established for the **TARGET**'s specified **LUN.**

Whenever a physical path is established in an **INITIATOR** that can accommodate disconnection and reconnections, the **INITIATOR** must assure that the present state of the physical path is equal to the saved copy for that particular **LUN.**

2.8 TIMING

Unless otherwise indicated, the delay time measurements for each **SASI DEVICE** are calculated from signal conditions existing at that device's own **SASI BUS PORT.** Thus, normally these measurements need not consider delays in the bus cable.

- **ARBITRATION DELAY** (1.7 us)

  The minimum time a **SASI DEVICE** must wait from asserting **BSY** for arbitration until the data bus can be examined to see if arbitration has been won. There is no maximum time.

- **BUS CLEAR DELAY** (650 ns)

  The maximum time that a **SASI DEVICE** can take to stop driving all bus signals after:

  (1) The release of **BSY** when going to the **BUS FREE** phase.

  (2) Another **SASI DEVICE** asserts **SEL** during the **ARBITRATION** phase.

- **BUS FREE DELAY** (100 ns)

  The minimum time that a **SASI DEVICE** must wait from its detection of the **BUS FREE** phase until its assertion of **BSY** when going to the **ARBITRATION** phase.

- **BUS SET DELAY** (1.1 us)

  The maximum time a **SASI DEVICE** can take from its detection of the **BUS FREE** phase to its assertion of **BSY** and its **ID** for the purpose of **ARBITRATION.**

- **BUS SETTLE DELAY** (450 ns)

- **CABLE SKEW** (10 ns)

  The maximum difference in propagation time allowed between any two **SASI BUS** signals when measured between any two **SASI BUS PORTS.**

- **DESKEW DELAY** (45 ns)

- **RESELECTION RESPONSE TIME** (system dependent, not specified here)

  The maximum time that a **TARGET** must wait for a **BSY** response from an **INITIATOR** before timing out during a **RESELECTION** phase.

- **RESET HOLD TIME** (25 us)

  The minimum time for which **RST** is asserted. There is no maximum.

- **SELECTION RESPONSE TIME** (system dependent, not specified here)

  The maximum time that an **INITIATOR** must wait for a **BSY** response from a **TARGET** before timing out during a **SELECTION** phase.

2.9 PHYSICAL DESCRIPTION

SASI devices are daisy-chained together using a common cable. Both ends of the cable are terminated. All signals are common between all SASI devices. Two driver/receiver options are available:

- Single-ended drivers and receivers, which allow a maximum cable length of six meters (primarily for in-cabinet interconnection).

- Differential drivers and receivers, which allow a maximum cable length of fifteen meters (primarily for interconnection outside of a cabinet).

2.9.1 Cable Requirements (Single-Ended Option)

A fifty (50) conductor flat cable (or twisted pair flat cable) shall be used. The maximum cable length shall be 6.0 meters.

Each **SASI BUS PORT** shall have a 0.1 meter maximum stub length of any conductor when measured from the bus cable connector.

Bus termination may be internal to the **SASI BUS DEVICES** that are at the ends of the bus cable.

The cable pin assignment is shown in Figure 2.4.

Connector Requirements:

The connector shall be a fifty (50) conductor flat cable connector which consists of two rows of 25 pins on 100 mil centers. The 3M "Scotchflex" #3425-3000 cable connector will satisfy this requirement.

2.9.2 Cable Requirements (Differential Option)

A fifty conductor flat cable or twisted pair cable shall be used. The maximum cable length shall be 15 meters.

Each **BUS PORT** shall have a 0.2 meter maximum stub length of any conductor when measured from the bus cable connector.

Bus termination may be internal to the **BUS DEVICES** that are at the ends of the bus cable.

The cable pin assignment is shown in Figure 2.5.

Connector Requirements:

The connector shall be a fifty (50) conductor flat cable connector or equivalent for round cable. The connector shall consist of two rows of 25 pins on 100 mil centers.

SIGNAL	PIN NUMBER	
-DB(0)	2	
-DB(1)	4	
-DB(2)	6	
-DB(3)	8	
-DB(4)	10	
-DB(5)	12	
-DB(6)	14	
-DB(7)	16	
-DB(P)	18	
---	20	↑
---	22	
---	24	not
---	26	
---	28	used
---	30	↓
-ATN	32	
-SPARE	34	for future use **(TERMINATE AS A SIGNAL LINE)**
-BSY	36	
-ACK	38	
-RST	40	
-MSG	42	
-SEL	44	
-C/D	46	
-REQ	48	
-I/O	50	

Note: All signals are low true. All odd pins are connected to ground.

Figure 2.4 PIN ASSIGNMENTS (Single-Ended Option)

SIGNAL	PIN NUMBER		SIGNAL	
*SHIELD GROUND	1	2	GROUND	
+DB(0)	3	4	-DB(0)	
+DB(1)	5	6	-DB(1)	
+DB(2)	7	8	-DB(2)	
+DB(3)	9	10	-DB(3)	
+DB(4)	11	12	-DB(4)	
+DB(5)	13	14	-DB(5)	
+DB(6)	15	16	-DB(6)	
+DB(7)	17	18	-DB(7)	
+DB(P)	19	20	-DB(P)	
	21	22		
	23	24		
	25	26		not used ●
	27	28		
+ATN	29	30	-ATN	
+SPARE	31	32	-SPARE	(Terminate as signal)
+BSY	33	34	-BSY	
+ACK	35	36	-ACK	
+RST	37	38	-RST	
+MSG	39	40	-MSG	
+SEL	41	42	-SEL	
+C/D	43	44	-C/D	
+REQ	45	46	-REQ	
+I/O	47	48	-I/O	
GROUND	49	50	GROUND	

*Optional shield ground on some cables.

Figure 2.5 PIN ASSIGNMENTS (Differential Option)

2.10 ELECTRICAL DESCRIPTION

Note: For these measurements, bus termination is assumed to be external to the **SASI DEVICE**. A typical **SASI DEVICE** would have the provision for allowing optional internal termination.

2.10.1 Single-Ended Option

All signals are low true.

All assigned signals are terminated with 220 ohms to +5 volts (nominal) and 330 ohms to ground at each end of the cable. See Figure 2.3.

All signals use open collector or three-state drivers as noted in Section 2.6.

Each signal driven by a **SASI DEVICE** shall have the following output characteristics when measured at the device's **SASI BUS PORT** connection:

True = Signal Assertion = 0.0 VDC to 0.5 VDC
Minimum driver output capability = 48 ma (sinking) @ 0.5 VDC

False = Signal Non-assertion = 2.5 VDC to 5.25 VDC

The open collector driver 7438 will satisfy this requirement.

Each signal received by a **SASI DEVICE** shall have the following input characteristics when measured at the device's **SASI BUS PORT** connection:

True = Signal Assertion = 0.0 VDC to 0.8 VDC
Maximum total input load = -0.4 ma @ 0.4 VDC

False = Signal Non-assertion = 2.0 VDC to 5.25 VDC

The 74LS14 receiver with hysteresis will satisfy this requirement.

2.10.2 Differential Option

All bus signals consist of two lines denoted **SIGNAL (+)** AND **SIGNAL (-)**. A signal is **ASSERTED** when **SIGNAL (+)** is more positive than **SIGNAL (-)**, while a signal is **NON-ASSERTED** when **SIGNAL (-)** is more positive than **SIGNAL (+)**. All assigned signals are terminated at each end of the cable as shown in Figure 2.4.

The recommended driver/receiver is SN75176 or equivalent. See EIA's standard proposal #SP1488.

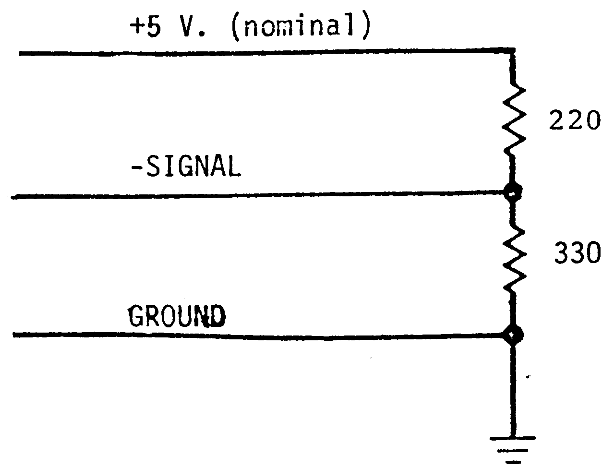


Figure 2.3 TERMINATION FOR SINGLE-ENDED OPTION

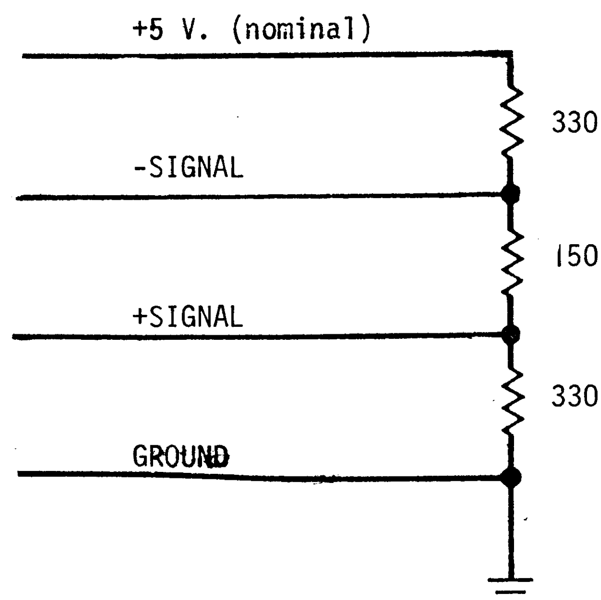
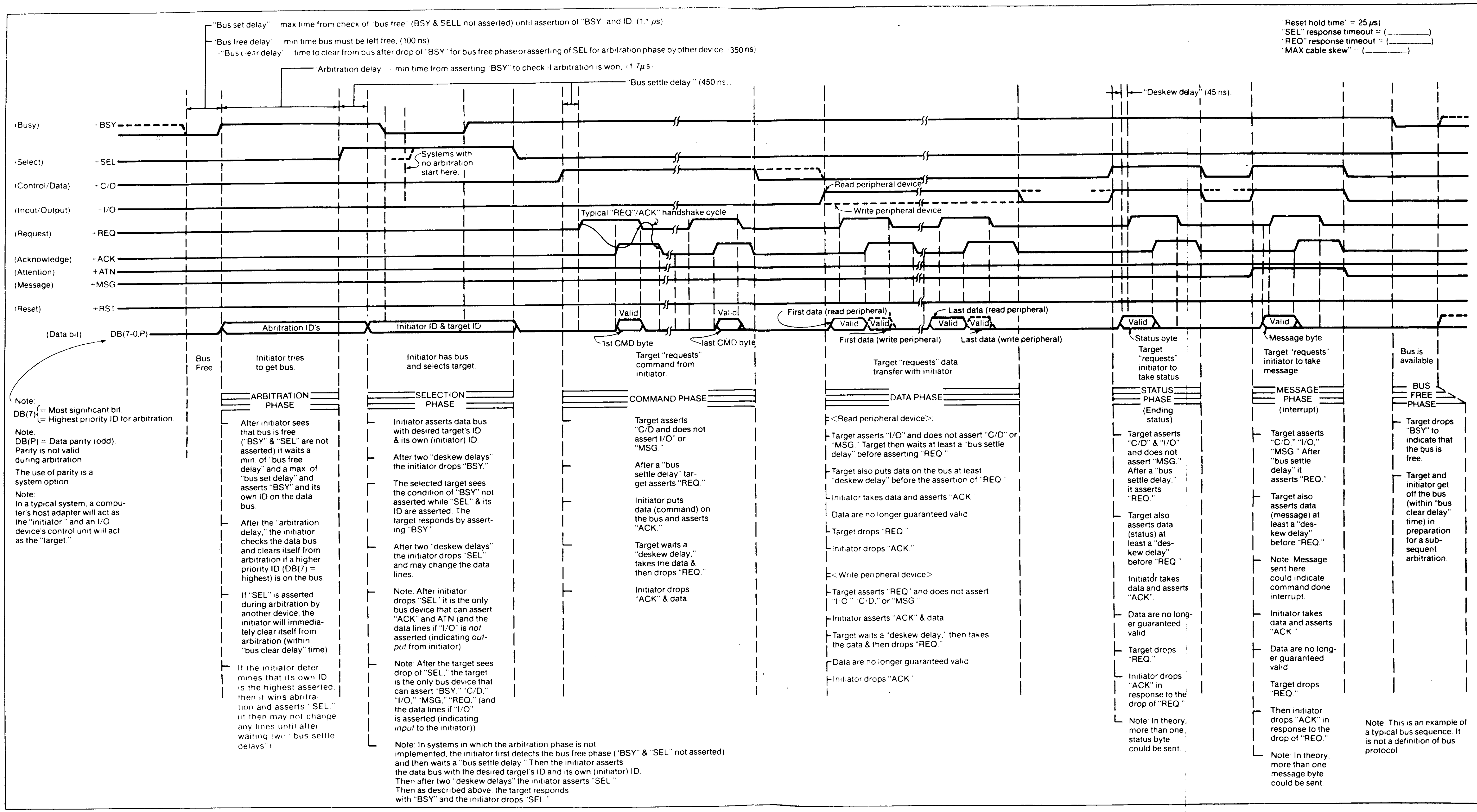


Figure 2.4 TERMINATION FOR DIFFERENTIAL OPTION

PART B

FUNCTIONAL SPECIFICATION



SASI TIMING CHART

Appendix - A. Timing Sequence Example
(Command without Disconnect/Reselect)

1.0 SCOPE

This is the functional definition of the Shugart Associates System Interface, which includes the software command set, as well as specific status information related to the various commands. The document defines the logical structure of the **I/O** subsystem, the functions available from the **I/O** subsystem, and how to request these functions.

By defining a fixed block structure using a simple logical address scheme, the **I/O** interface can support device independence. In addition, by including the address as a component of the command structure, physical requirements such as **SEEK** can be imbedded within the basic **READ** and **WRITE** requests.

Although the interface has been kept quite simple, it has been designed to provide high performance in a multiple host multi-task environment. Powerful functions have been included to enhance random access applications. Multiple block transfers may be accomplished with a simple command.

By keeping to a minimum the functions essential to communicate via this protocol, a wide range of devices of varying capability can operate in the same environment. The objective of low cost may be satisfied without precluding that of high performance.

1.1 SASI FUNCTIONAL CONTROL PHILOSOPHY

This subsection describes the process of obtaining commands and storing results. Command execution itself (the actual implementation of a command) is **SASI** bus device specific and is therefore not discussed. Because many subsets of the full architecture may be implemented, optional functions will be noted.

1.1.1 Single Command

A typical operation on the **SASI** interface is likely to include a single **READ** to the I/O subsystem. This operation will be described in detail starting with a request from the system to the **INITIATOR** path control logic.

The **INITIATOR** path control logic has an active state and a set of stored states representing active disconnected devices (**INITIATOR** path control logic without disconnect capability does not require stored states). Upon receipt of an "Establish Path" request from the system, the **INITIATOR** path control logic sets up the active state for the operation, arbitrates for the bus, and selects the **LUN**. Once this process is completed, the **TARGET** function control logic (a functional component of the **LUN**) assumes control of the operation.

The **TARGET** obtains the command from the **INITIATOR** (in this case a **READ** command) via the "Get Command" request to the **TARGET** path control logic. The **TARGET** then reads the data from the peripheral device and sends it to the **INITIATOR** using the "Send Data" function of the **TARGET** path control logic. At the completion of the **READ** command, the **TARGET** stores the command's completion status in the **INITIATOR** using the "Send Status" path control function. To end the operation, and to acknowledge completion, the **TARGET** path control logic is given the "End Path" function.

1.1.2 Disconnect

In the above **READ** example, the length of time necessary to obtain the data may have been considerable (e.g., requiring a time-consuming physical seek). In order to improve system throughput, the **TARGET** may disconnect from the **INITIATOR**, freeing the bus to allow other requests to be sent to other **LUN**'s. To do this, the **INITIATOR** path control logic must be reselectable and capable of restoring the device state upon reconnection. The **TARGET** path control logic must be capable of arbitrating for the data bus and reselecting the **INITIATOR**.

After the **TARGET** has received the **READ** command (and determined that there will be a delay), it will request, via path control logic, that the **INITIATOR** save the present state; this state differs from the state when the **TARGET** was initially selected because the command has already been transferred. The **TARGET** then requests that the **TARGET** path control logic break the path (i.e., disconnect).

When data is ready to be transferred, the **TARGET** will request that path control logic reconnect the **INITIATOR**. As a a result of this reconnection, the **INITIATOR** path control logic will restore the "saved" state (last saved by the reconnecting **LUN**); the **TARGET** will continue (as in the single command example) to finish the operation. At "Path End," the **INITIATOR** path control logic recognizes that the operation is complete. In addition to signalling this to the functional component in the system, it frees the location that had been used to save the state for that **LUN**.

On those occasions when a **TARGET** could disconnect without first saving the latest state (as might occur if an error was detected while transferring data to the **INITIATOR**), the operation may be repeated by either restoring the previous state or by disconnecting without saving the present state. When reconnection is completed, the previous state will be restored.

1.1.3 Link

The "Link" function defines a relationship between commands which allows previous operations to modify subsequent commands. "Link" makes high performance I/O functions possible by providing a relative addressing capability and allowing multiple command execution without invoking the functional component of the **INITIATOR** and without requiring reselection.

If the desired data address (in the previously described **READ** example) is unknown, but a search key defined as some particular bytes of the field is known, then by linking the **READ** to a **SEARCH EQUAL** command this data can be quickly and effectively transferred to the **INITIATOR**.

One additional function must be completed prior to requesting the next command. This function, "End of Link," is sent from the **TARGET** to the **INITIATOR** to acknowledge command completion. The **INITIATOR** then updates the stored state so that subsequent requests from the **TARGET** will reference the next command of the chain. Other than the "End of Link" function and the address modification of linked commands, command processing of linked and single commands is identical.

2.0 COMMAND AND STATUS STRUCTURE

2.0.1 Command Description Block (CDB)

An **I/O** request to a device is performed by passing a Command Description Block (CDB) to the target. The first byte of the CDB is the command class and operation code. The remaining bytes specify such things as the Logical Unit Numbers (LUN), block starting address, control byte, and the number of blocks to transfer.

Commands are categorized into eight groups as follows:

Group 0 - 6-Byte commands including **CONTROL, DATA TRANSFER** and **STATUS** commands.

Group 1 - 10-Byte commands.

Group 2 - Undefined.

Group 3 - Undefined.

Group 4 - Undefined.

Group 5 - 12-Byte commands.

Group 6 - **CONTROLLER SPECIFIC** commands.

Group 7 - **CONTROLLER SPECIFIC DIAGNOSTIC** commands.

Figure 2.0 shows the command descriptor block formats.

Note: The reserved or **RESERV** used within this document indicates its state may be undefined or that it is reserved for future use and should be set to zero but not checked.

Group 0 Commands
6 Byte Commands

BYTE \ BIT	7	6	5	4	3	2	1	0
00	Group Code			Opcode				
01	**Logical Unit Number**			**MSB Logical Block Address**				
02	**Logical Block Address**							
03	**LSB Logical Block Address**							
04	Number of Blocks							
05	VU	VU	Reserv	Reserv	Reserv	Reserv	Flag Req	Link

Group 1 Commands
10-Byte Commands (Extended Block Address)

BYTE \ BIT	7	6	5	4	3	2	1	0
00	Group Code			Opcode				
01	**Logical Unit Number**			Reserved				
02	MSB **Logical Block Address**							
03	**Logical Block Address**							
04	**Logical Block Address**							
05	**LSB Logical Block Address**							
06	Reserved							
07	Number of Blocks							
08	Number of Blocks							
09*	VU	VU	Reserv	Reserv	Reserv	Reserv	Flag Req	Link

*Control Byte

Figure 2.0 COMMAND DESCRIPTOR BLOCK FORMATS

Group 5 Commands
12-Byte Commands

BYTE \ BIT	7	6	5	4	3	2	1	0
00	Group Code			Opcode				
01	**Logical Unit Number**			Reserved			Read INH	Write INH
02	MSB **Logical Block Address**							
03	**Logical Block Address**							
04	**Logical Block Address**							
05	**LSB Logical Block Address**							
06	2nd **Logical Block Address** MSB							
07	2nd **Logical Block Address**							
08	2nd **Logical Block Address**							
09	2nd **Logical Block Address** LSB							
10	Reserved							
11	VU	VU	Reserv	Reserv	Reserv	Reserv	Flag Req	Link

Figure 2.0 COMMAND DESCRIPTOR BLOCK FORMATS (continued)

2.0.1 Group Code

The group code can be 0 to 9.

2.0.2 Operation Code

The operation code for each class allows 32 commands (0 to 31).

2.0.3 Logical Unit Number

The **LOGICAL UNIT NUMBERS** designate the source unit for all classes.

Logical unit number allows 8 devices per **TARGET** (0 to 7). This method of device addressing is designed for low-end systems that do not implement separate paths for command and address functions. It is not recommended for more sophisticated implementations.

2.0.4 Logical Block Address

Six byte commands contain 21 bit starting block addresses. Eight and twelve-byte **EXTENDED ADDRESS COMMANDS** contain 32 bit starting block addresses.

The concept of "block" implies that the **INITIATOR** and **TARGET** have previously agreed to the number of bytes of data to be transferred.

2.0.5 Number of Blocks

The number of blocks the command is to transfer allows 1to 256 blocks (1 to 255, and 0 = 256 blocks).

2.0.6 Control Byte (Last byte in all commands)

Bit 7-6 Are vendor unique.

Bit 5-2 Reserved.

Bit 1 This bit is only meaningful when Bit 0 is set and means status is requested for this command in a group of linked commands. Ending status must be sent if the link bit is not on, or if the **TARGET** encounters an abnormal condition.

Bit 0 The use of this bit is optional and means an automatic link to the next command upon completion of the current command for this **INITIATOR**. Status may be sent for each command executed.

2.1 COMMAND DESCRIPTIONS (GROUP 00)

Note: Commands marked standard (S) must be implemented in order to meet the minimum requirement of this specification. Commands marked optional (O) if used must be implemented as defined in this specification.

Command operation codes in hex and names are listed in a table associated with each section (i.e. Section 2.1, Table 2.1). Command codes in these tables marked VENDOR UNIQUE are in use by some controllers currently on the market.

TEST UNIT READY (00)
(Optional)

Returns zero status if addressed unit is powered on and ready. This is not a request for unit self test. A fast response is expected.

REZERO UNIT (01)
(Optional)

Sets the unit to a specific known state. (Example: Rewind - Recalibrate)

REQUEST SENSE (03)
(Standard)

Returns unit sense. The sense data will be valid for the check condition (status) just presented to the **INITIATOR.** This sense data must be preserved in the **TARGET** for the **INITIATOR.** Sense data will be cleared on the reception of any subsequent command to the unit in error from the **INITIATOR** receiving the check condition. (See Section 2.10 for sense data format.) Byte 4 in this command will specify the number of bytes that the host has allocated for returned **SENSE.** (0 = 4 bytes. To be compatible with existing initiators. Future initiators which use the extended sense command will specify in byte 04 the number of bytes allocated for sense data (e.g., 1 to 255 bytes).

FORMAT UNIT (04)
(Standard)

Formats the entire media. Bit 4 of Byte 1 indicates that the following data is available. (See Figure 2.0 for command format.)

Format Data:

- 2 Bytes - Block Size
- 2 Bytes - Reserved
- 2 Bytes - Length of defect list (number of bytes)
- X Bytes - Defect List (List of defective block addresses) (4 bytes each defect) Block addresses in the defect list must be in ascending order.

Format Command

BYTE \ BIT	7	6	5	4	3	2	1	0
00	0	0	0	0	0	1	0	0
01	Logical Unit Number			Data	Cmpl Lst	Reserv		
02	Reserved							
03	Interleave							
04	Interleave							
05	VU	VU	Reserv	Reserv	Reserv	Reserv	Flag Req	Link

The block size in the defect list is identified using the existing block size for previously formatted units. For units that are unformatted, the block size is specified in the first two bytes of format data.

READ (08)
(Optional)

Transfers to the **INITIATOR** the specified number of blocks starting at the specified logical starting block address. See the **SEARCH** commands for explanation of linked modification of this command.

WRITE (0A)
(Optional)

Transfers to the **TARGET** the specified number of blocks starting at the specified logical starting block address. See the **SEARCH** commands for explanation of linked modification of this command.

SEEK (0B)
(Optional)

Requests that the unit ready itself to transfer data from the specified address in a minimum time.

RESERVE UNIT (16)
(Optional)

Reserves this unit for use by the requesting **INITIATOR** until a **RELEASE UNIT COMMAND** is received. No other **INITIATOR** can perform any function on this unit.

RELEASE UNIT (17)
(Optional)

Releases this unit from the requesting **INITIATOR.** This **INITIATOR** must have issued the previous reserve unit command.

READ DIAGNOSTIC (1C)
(Optional)

Sends analysis data to **INITIATOR** after completion of a **WRITE DIAGNOSTIC** command. The analysis data is vendor unique with bytes 3 and 4 of the CDB specifying the length of the data.

Read Diagnostic Command

BYTE \ BIT	7	6	5	4	3	2	1	0
00	0	0	1	0	1	0	1	0
01	**Logical Unit Number**			Reserved				
02	Reserved							
03	Variable Length Indicator							
04	Variable Length Indicator							
05	VU	VU	Reserv	Reserv	Reserv	Reserv	Flag Req	Link

WRITE DIAGNOSTIC (1D)
(Optional)

Sends data to the **TARGET** to specify diagnostic tests for **TARGET** and peripheral units. The analysis data is vendor unique with bytes 3 and 4 of the CDB specifying the length of the data.

INQUIRY (12)
(Standard)

This command sends to the **INITIATOR** information regarding unit and target parameters.

INQUIRY Data:
1 Byte Code 0 - Direct access device
1 - Sequential access device
2 - Output only device
1 Byte - Length of additional bytes
X Bytes - Additional unit parameters

Byte 4 byte in this command will specify the number of bytes that the host has allocated for returned Unit Parameters (0 indicates 256 bytes).

Group 00 Commands

OP CODE		
00	O	TEST UNIT READY
01	O	REZERO UNIT
02		VENDOR UNIQUE
03	S	REQUEST SENSE
04	S	FORMAT UNIT
05		VENDOR UNIQUE
06		VENDOR UNIQUE
07		VENDOR UNIQUE
08	O	READ
09		VENDOR UNIQUE
0A	O	WRITE
0B	O	SEEK
0C		VENDOR UNIQUE
0D		VENDOR UNIQUE
0E		VENDOR UNIQUE
0F	O	VENDOR UNIQUE

Figure 2.1 COMMAND CODES

Group 00 Commands

OP CODE		
10		
11		
12	O	INQUIRY
13		
14		
15		
16	S	RESERVE UNIT
17	O	RELEASE UNIT
18		
19		
1A		
1B		
1C	O	READ DIAGNOSTIC
1D	O	WRITE DIAGNOSTIC
1E		
1F		

Figure 2.1 COMMAND CODES (continued)

2.2 COMMAND DESCRIPTIONS (Group 01)

READ CAPACITY (05)
(Standard)

If byte 4 is zero, the command will return the address of the last block on the unit. If byte 4 is one, this command will return the address of the last block after the specified address to the point at which a substantial delay in data transfer will be encountered (e.g., a cylinder boundary).

Read Capacity Data:
4 Bytes - Block Address
2 Bytes - Block Size

READ (08)
(Standard)

Transfers to the **INITIATOR** the specified number of blocks starting at the specified logical starting block address. See the **SEARCH** commands for explanation of linked modification of this command.

WRITE (0A)
(Standard)

Transfers to the **TARGET** the specified number of blocks starting at the specified logical starting block address. See the **SEARCH** commands for explanation of linked modification of this command.

WRITE AND VERIFY (0E)
(Optional)

Writes and verifies the data for the specified number of blocks. See the **SEARCH** commands for explanation of linked modification of this command.

VERIFY (0F)
(Optional)

Verifies the data for the specified number of blocks. No data is transferred with this command. See the **SEARCH** command for linked modification of this command.

SEARCH DATA EQUAL (11)
(Optional)

This command specifies a block address and a number of blocks to search. The data transferred to the target defines the area of the block to search and includes the search argument. If a command is linked to the search command and the search is successful, then the next command is fetched and executed. In this case, the address portion of the command is used as a displacement from the address at which the **SEARCH** was satisfied. If the search was not satisfied, the link is broken and end status is presented. If the address of a block that has satisfied the **SEARCH** is desired to be known, a **SENSE** command must be issued. First two bytes of additional sense define the record offset in units of records. That is, records do not span blocks.

The **SEARCH** function contains the concept of sub-fields within a data block, to allow multiple areas within a block to be searched as logical blocks. Any search satisfied within this addressable block will reference the entire block for the purpose of linked commands. A **SENSE**, however, will indicate the number of the sub-field within the block which satisfied the **SEARCH.**

Format of the search data:

Byte 0-1	Length of record.
Byte 2-3	Number of records to skip.
Byte 4-5	Length of following search argument.

Search argument repeated n times:

- 2 byte displacement of field to compare (within record)
- 2 byte length of field to compare (within record)
- m byte data to compare

SEARCH DATA HIGH (10) (Optional)

This command performs the same function as the Search Data Equal command, but is satisfied by a compare of high or equal. A write command may not be linked to this command.

SEARCH DATA LOW (12) (Optional)

This command performs the same function as the Search Data Equal command, but is satisfied by a compare of low or equal. A Write command may not be linked to this command.

Group 01 Commands

OP CODE		
00		
01		
02		
03		
04		
05		
06		
07		
08	S	EXTENDED ADDRESS READ
09		
0A	S	EXTENDED ADDRESS WRITE
0B		
0C		
0D		
0E	O	WRITE AND VERIFY
0F	O	VERIFY

Figure 2.2 COMMAND CODES

Group 01 Commands

OP CODE	
10	O SEARCH DATA HIGH
11	O SEARCH DATA EQUAL
12	O SEARCH DATA LOW
13	
14	
15	
16	
17	
18	
19	
1A	
1B	
1C	
1D	
1E	
1F	

Figure 2.2 COMMAND CODES (continued)

Group 05 Commands

OP CODE		
00		VENDOR UNIQUE
01		VENDOR UNIQUE
02		VENDOR UNIQUE
03		
04		
05	S	READ CAPACITY
06		
07		
08		
09	O	SET BLOCK LIMITS
0A		
0B		
0C		
0D		
0E		
0F		

Figure 2.3 COMMAND CODES

2.3 COMMAND DESCRIPTIONS (Group 05)

SET BLOCK LIMITS (09) (Optional)

This command defines the addresses outside of which any following linked commands may not operate. A second set block limits command may not be linked to a chain of commands in which a set block limits command has already been issued. The two low order bits of byte 01 define the legal operations within the limits of the specified addresses. Bit 0 indicates write inhibit, and Bit 1 indicates read inhibit.

2.4 COMMAND DESCRIPTIONS (Group 02)

No commands currently defined for this group.

2.5 COMMAND DESCRIPTIONS (Group 03)

No commands currently defined for this group.

2.6 COMMAND DESCRIPTIONS (Group 04)

No commands currently defined for this group

2.7 COMMAND DESCRIPTIONS (Group 06)

Commands in this group are controller specific commands. The commands in this group will be described only in controller specifications.

2.8 COMMAND DESCRIPTIONS (Group 07)

Commands in this group are **CONTROLLER** specific diagnostic commands. The commands in this group will be described only in **CONTROLLER** specifications.

2.9 COMPLETION STATUS BYTE

Status must always be stored at the end of a command or set of linked commands. Intermediate status may be stored at the completion of a linked command. Any abnormal conditions encountered during command execution will cause command termination and ending status.

Byte 00

Bit 0 Reserved. Currently used in some targets for bus parity error.

Bit 1 Check condition. Sense is available. (See Section 2.1 for sense command)

Bit 2 Equal. Will be set when search equal is satisfied.

Bit 3 Busy. Device is busy or reserved. Busy status will be stored whenever a target is unable to accept a command from an **INITIATOR**. The most common instance of this condition arises when an **INITIATOR** that does not allow reconnection requests an operation from a reserved or busy device.

Bit 4 Intermediate status stored. This bit is set and status is stored as a result of a request made via the interrupt request bit in the command control byte. This bit will not be set regardless of the interrupt request bit in any ending status.

Bit 5 - 6 Reserved. Some current controllers place LUN here.

Bit 7 Extended status. This bit set means that the second status byte is valid.

Byte 01

Bit 0 Host adapter detected error. Reserved for the host adapter H.A. to flag the host for errors not detected by the **TARGET.**

Bit 7 Extended status. This bit, when set, means that the third status byte is valid.

BYTE / BIT	7	6	5	4	3	2	1	0
00	Ext Stat	VU		Int Sta	Busy	VU	Check	VU
01	Ext Stat							H.A. Err

2.10 SENSE BYTES

BIT	7	6	5	4	3	2	1	0
00	Ad Valid	Error Class			Error Code (See Figure 2.4)			
01	Reserved			MSB Logical Block Address				
02	Logical Block Address							
03	LSB Logical Block Address							

2.10.1 EXTENDED SENSE BYTES

BYTE \ BIT	7	6	5	4	3	2	1	0
00	Ad Valid	Class 07			Code 00			
01	Reserved							
02	File Mk	Bot/Eot	Reserved		Sense Key			
03	MSB Logical Block Address							
04	Logical Block Address							
05	Logical Block Address							
06	LSB Logical Block Address							
07	Additional Sense Length							
08 - NN	Additional Sense Bytes							

Note: Bytes 08 - NN may be predefined vendor unique bytes as defined in Figure 2.4, or any other not yet defined data.

The Sense Key is a device independent code designed to aid the system in quickly resolving the following Sense Data.

SENSE KEYS -
- 00 = No Sense Available
- 01 = Recoverable Error
- 02 = Not Ready
- 03 = Media Error (Non Recoverable)
- 04 = Hardware Error (Non Recoverable)
- 05 = Illegal Request
- 06 = Media Change (Must be reported to all **INITIATORS**)
- 07 = Write Protect
- 08 = Diagnostic Unique
- 09 = Vendor Unique
- 0A = Power Up Failed (Must be reported to all **INITIATORS**)
- 0B = Aborted Command
- 0C = Condition Met

ERROR CODES IN SENSE BYTE

Class 00 Errors
Drive Errors

ERROR CODE	
00	NO SENSE
01	NO INDEX SIGNAL
02	NO SEEK COMPLETE
03	WRITE FAULT
04	DRIVE NOT READY
05	DRIVE NOT SELECTED
06	NO TRACK 0
07	MULTIPLE DRIVES SELECTED
08	NO ADDRESS ACKNOWLEDGE
09	MEDIA NOT LOADED
0A	INSUFFICIENT CAPACITY
0B	VENDOR UNIQUE
0C	VENDOR UNIQUE
0D	VENDOR UNIQUE
0E	
0F	

Figure 2.4 ERROR CODES

ERROR CODES IN SENSE BYTE

Class 01 Errors
Target Errors

ERROR CODE

00	I.D. CRC ERROR
01	UNCORRECTABLE DATA ERROR
02	I.D. ADDRESS MARK NOT FOUND
03	DATA ADDRESS MARK NOT FOUND
04	RECORD NOT FOUND
05	SEEK ERROR
06	DMA TIMEOUT ERROR
07	WRITE PROTECTED
08	CORRECTABLE DATA CHECK
09	BAD BLOCK FOUND
0A	INTERLEAVE ERROR
0B	DATA TRANSFER INCOMPLETE
0C	VENDOR UNIQUE
0D	VENDOR UNIQUE
0E	VENDOR UNIQUE
0F	

Figure 2.4 ERROR CODES (continued)

ERROR CODES IN SENSE BYTE

Class 02 Errors
System Related Errors

ERROR CODE	
00	INVALID COMMAND
01	ILLEGAL BLOCK ADDRESS
02	VENDOR UNIQUE
03	VENDOR UNIQUE
04	VENDOR UNIQUE
05	
06	
07	
08	
09	
0A	
0B	
0C	
0D	
0E	
0F	

Figure 2.4 ERROR CODES (continued)

ERROR CODES IN SENSE BYTE

Class 07 Errors
Extended Sense

ERROR CODE	
00	EXTENDED SENSE
01	
02	
03	
04	
05	
06	
07	
08	
09	
0A	
0B	
0C	
0D	
0E	
0F	VENDOR UNIQUE SENSE

Figure 2.4 ERROR CODES (continued)

VENDOR UNIQUE COMMANDS FOR SASI

Class 0 Non-Data Commands

02	Request Syndrome
05	Check Track Format
06	Format One Track
07	Format Bad Track/Block(s)
09	Write-Protect the Block(s)
0C	Request Drive Type
0D	Verify
0E	Assign Alternate Track
0F	Start/Stop Drive

Class 1 Device to Device Commands

00	Copy Block(s)

Class 5 Device-to-Device Tape Commands

00	Copy Block(s)
01	Restore
02	Backup

Class 6 Device Assignment Commands

00	Specify Floppy Track Format
01	Assign Drive Type

Class 7 Device Diagnostic Commands

00	RAM Diagnostic
01	Write ECC
02	Read ID
03	Drive Diagnostic
04	Read Entire Disk (No data transfer)
05	Read Blocks (No data transfer)