PROPOSED WORKING DRAFT

X3T9.2
93-0181
Revision 00
November 12th, 1993

Information Systems - Serial Storage Architecture - SSA-SSP (SCSI-3 Mapping)

ABSTRACT: This document describes the SCSI-3 mapping protocol to be used on the Serial Storage Architecture Transport Layer (SSA-PH). The goal is to minimize the microcode impact in converting between parallel SCSI-3 and SSA-SSP, while gaining the advantages of the higher performance SSA interface. A separate document (SSA-SCSI) handles the SCSI-2 mapping of SSA. This proposal is the SCSI-2 mapping and will need to undergo changes (both technical and editorial) to make it map to SCSI-3.

DRAFT: This document is preliminary, and the contents are subject to change without notice.

COMMENTS: Written comments are solicited from readers. Comments received by members will be considered for inclusion in future revisions of this document. Please forward your comments to the working group editor (John Scheible) at the address shown on the back of this page.

COPIES: of this document, when approved, may be purchased from:

Global Engineering
15 Inverness Way East
Englewood, CO 80112-5704
(800) 854-7179 or (303) 792-2181
Facsimile: (303) 792-2192

ISO LOGO **IEC LOGO**

Reference number
ISO/IEC xxxxx: 199x
ANSI X3.xxx: 199x

OTHER DOCUMENTATION: The current document should be read in conjunction with:

- 'Serial Storage Architecture, SSA-PH (Physical Layer)', Document number 0989-D revision 00, dated November 12th, 1993.
- 'Serial Storage Architecture, SSA-SCSI (SCSI-2 mapping)', Document number SSA-UIG / 93-036 revision 00, dated October 11th, 1993.
- 'Small Computer System Interface - 2 (SCSI-2)', X3.131.199X, Revision 10i. (ANSI X3T9.2 committee.)

DRAFT: This document is preliminary and the contents are subject to change without notice.

COMMENTS: Enquiries, and suggestions may be directed to:

SSA-UIG Chairman:	Editor:
Ken Hallam	John Scheible
ENDL Associates	IBM, ADSTAR
29112 Country Hills Road	Dept G46, Bldg 028
San Juan Capistrano, CA 92675	5600 Cottle Road
	San Jose, CA 95193
Voice: 714-364-9626	Voice: 408-256-7275
FAX: 714-364-3271	FAX: 408-256-2254
E-Mail: khallam@endlas.com	E-Mail: scheible@vnet.ibm.com

IBM may use any information that you supply without incurring any obligation.

Revision History...

SSA-UIG/93-036 revision 0 (October 12th, 1993)

When the SSA-UIG voted to accept SSA-UIG/93-002r4 as baseline, the document number was changes to "SSA-UIG/93-036r0", the revision bars removed, the update history removed, and the IBM copyright notice removed. Please refer to this document (SSA-UIG/93-036) for future revisions (SSA-UIG/93-002 will no longer be maintained).

This same document has been submitted to X3T9.2 as a proposed working draft for SSA-SSP. This document is written as a SCSI-2 mapping and will need to undergo changes (both technical and editorial) to make it map to SCSI-3.

Contents

Tables

1.0 Introduction

This document describes an upper-level protocol of Serial Storage Architecture. SSA-SCSI is a mapping of the existing SCSI-2 protocol, as defined by the ANSI X3T9.2 committee for a parallel bus, with extensions to map SCSI-2 to a serial bus.

SSA defines a serial interface for use within future storage sub-systems. It is divided into a common transport layer and two alternative upper-level protocols. These are described in separate documents as follows.

SSA-PH	The physical transport layer
SSA-SCSI	SSA-SCSI is a mapping of the existing SCSI-2 protocol, as defined by the ANSI X3T9.2 committee for a parallel bus, with extensions to map SCSI-2 to the SSA serial interface (this document).
SSA-SSP	SSA-SSP will be a mapping of the SCSI-3 protocol as being defined by the ANSI X3T9.2 committee. The document has yet to be written, since it is waiting for an approved SSA-SCSI document from which it will be derived.

1.1 Objectives

It is intended that SSA-SCSI should conform as closely as possible to the existing SCSI-2 logical model. This minimizes the programming changes that may be required to convert existing systems and devices from the parallel bus to a serial interface. Therefore the following functions of SSA-SCSI are identical to SCSI:

- Tagged queuing.
- Command descriptor blocks.
- Status byte.
- Architected Sense bytes.

Except where necessary for clarity the above functions are not described in this document. (Please refer to the ANSI SCSI-2 specification for information.) Rather SSA-SCSI concentrates on mapping the following aspects of parallel SCSI-2:

- Bus functions.
- Addressing.
- Messages.

SSA-SCSI supports open-ended networks containing strings, loops and switches. The concepts of Initiator, Target and Logical Unit are retained although SSA-SCSI supports larger configurations than SCSI-2. Initiators and Targets can be freely mixed through-out the network. Each node can have from 1 to 127 physical ports.

SSA-SCSI offers the following benefits compared to the parallel SCSI bus:

- Open-ended networks with alternate paths for availability.
- Full-duplex communication with spatial reuse on strings and loops.
- Frame multiplexing.
- No overheads for arbitration, disconnection & reselection
- Integrated spindle synchronization for array applications.
- Fewer Initiator-Target exchanges.

- Concurrent I/O processes on the same device or different devices.
- Out-of-order data transfers.

Restrictions:

- Untagged queuing can be simply emulated with Tagged Queueing

  In SSA, the Target effectively disconnects after each frame because of frame multiplexing. Hence each command must have a tag for identification. Untagged command queueing can be simply emulated by having the Initiator only have one outstanding command, re-use the same tag, and use the simple queue type.

- SSA-SCSI uses Auto Contingent Allegiance.

  The SCSI-2 concept of Contingent Allegiance does not work for SSA since the next command resets the Contingent Allegiance condition, and the next command may be in the SSA pipeline prior to the status being received by the Initiator. Therefore SSA-SCSI will adopt the SCSI-3 concept of Auto Contingent Allegiance where the Auto Contingent Allegiance condition must be explicitly reset by a Clear_ACA_condition message.

1.2 Conventions

- Bits in a byte a numbered 7 to 0 from left to right. Bit 7 is the most-significant bit. The most-significant byte of a number is first.
- Bit values are represented as, eg., 1b.
- Hexadecimal values are represented as, eg., A2h.

1.3 Frames

As described in SSA-PH, the frame address field generally contains two components:

1. A **Path** address of the destination node. (Initiator or Target.)
2. A **Channel** address within the destination node.

The channel addressed by the single byte 00h is predefined to receive message frames. All other Channels are used to receive data frames.

Data Channels are allocated dynamically by the destination node either in the command message (SCSI_command) or by subsequent data transfer messages (Data_ready, Data_Request) between the Target and the Initiator. The messages allocate a Channel and specify the starting location and length of the transfer.

2.0 Messages

Messages are used to communicate control information between the Target and the Initiator. They are typically used for commands, status and initiating data transfers. Some SSA-SCSI messages are similar in function to SCSI messages but there is not a one-to-one mapping and they should not be confused.

The address field in a message contains the Path address to the destination node followed by the byte 00h to select the message Channel. The length of the data field depends on the particular message. Each message must be fully contained in a separate frame.

The function of a message is specified by a Message_code in the first byte of the data field. The Message_code is allocated according to the message type as follows:

00h - 0Fh	Reserved for use by the SSA-PH transport layer.
10h - BFh	Allocated to standard SSA-SCSI messages.
C0h - FFh	Reserved for vendor-unique messages.

All messages from the Initiator to the Target contain a 4-byte Return_path parameter. This contains the Path component for the address field of all message and data frames to be returned to the Initiator. Since the Path component may be less than 4 bytes it is left-aligned in Return_path. Bit 7 is set to 0b in the last valid byte and to 1b in the preceding bytes, if any. Thus the Target can determine the significant bytes.

Some messages contain a 2-byte Channel parameter. This contains the Channel component for the address field of data frames. Since the Channel component may be only a single byte it is left-aligned. Bit 7 is set to 0b in the last valid byte and to 1b in the preceding byte, if any.

All messages contain a 2-byte Tag that is used to relate replies to the original request. The Tag is assigned by the Initiator and it must be unique among all of the Tags that are currently active from that Initiator on any Target or Logical Unit throughout the network. Effectively the Tag identifies the Target, LUN and Queue-tag components of the SCSI-2 nexus. (The Initiator is identified by a Path address in the message.)

A Target always replies to a given message using the same port from which it received the message.

The following sections define the messages that are used by SSA-SCSI.

2.1 SCSI_command

This message is sent from the Initiator to a Target to initiate an I/O process or to send the next command in a linked list. The Target will respond by initiating a data transfer and/or by sending a SCSI_Status message.

The contents of the data field in a SCSI_command message are as follows:

Table 1. SCSI_command message

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Message_code = 10h							
1	LUNTAR	LUNTRN						
2 3	Tag							
4 5 6 7	Return_path							
8 9	Vendor_unique							
10	DDRM	Split	Reserved = 0000b				Queue_ctl	
11	Reserved = 00h							
12 13	Channel							
14 15	Reserved							
16 m	Command_descriptor_block							

Message_code	This byte identifies the message as 'SCSI_command'.
LUNTAR	If LUNTAR = 0, then LUNTRN refers to a Logical Unit. If LUNTAR = 1, then LUNTRN refers to a Target routine.
LUNTRN	This field specifies either a Logical Unit or a Target Routine, as indicated by the value of LUNTAR. Note that SSA-SCSI provides 7 bits for LUNTRN although SCSI-2 has only 3 bits. All SCSI_command messages in a linked list must specify the same values for the LUNTAR and LUNTRN parameters.
Tag	This is a 2-byte field that is assigned by the Initiator and used to relate subsequent messages to this I/O process. The Tag becomes available for re-assignment when the Initiator receives a SCSI_Status message for the command. If the Initiator is directly mapping the functions of parallel SCSI-2 then the Tag parameter of the SCSI-2 Queue Tag message can be included as one byte of this field. The other byte can contain the SCSI-2 Target address, LUNTAR and LUNTRN. All SCSI_command messages in a linked list must specify the same value for the Tag parameter.

Return_path This 4-byte field specifies the Path component of the address field for the associated frames that the Target returns to the Initiator (Data Ready, Data_Request, read data, SCSI_status).

All SCSI_command messages in a linked list must specify the same value for the Return_path parameter.

Vendor_unique These 2 bytes are available for the definition of vendor_unique functions.

DDRM Disable Data_Ready Messages. When set to 1b, bit 7 disables Data_ready messages for data transfers from the Target to the Initiator. In this case the Initiator must specify the Channel to be used in the Channel field. If the Split flag is also set then the Initiator must be able to identify each logical block from a header within the block. DDRM is ignored for any commands that do not transfer data to the initiator.

Split When bit 6 is set to 1b the Target or Logical Unit is permitted to transfer **read/write** data out of order to/from the Initiator. This enables a device to get the maximum advantage from a split read, or split write. It is also useful for array controllers, eg. RAID-5.

When Split is reset to 0b then read/write data must be transferred to/from the Initiator sequentially from the beginning. A disk drive must also write data sequentially but internally it may still read data out of order.

For control commands the Split parameter is ignored and data is always transferred in order from the beginning.

Queue_ctl Bits 1:0 are coded to control how the I/O process is to be queued at the specified Logical Unit or Target routine:

0 0 **Auto Contingent Allegiance.** The Target will execute only I/O processes with the Auto Contingent Allegiance attribute while an Auto Contingent Allegiance condition exists for the same nexus. (See 3.1.2, "Auto Contingent Allegiance Condition" on page 22.) For example, a Request Sense command specifying a Queue_ctl of Auto Contingent Allegiance may be used to retrieve sense information after a command terminates with Check Condition status.

0 1 **Head.** The Target will place an I/O process with this attribute first in the queue, ie. it will be executed next when the currently active I/O process finishes. As in SCSI-2, a Head of Queue tag may or may not execute prior to a previously issued head of queue.

1 0 **Ordered.** The Target will execute a sequence of I/O processes with the Ordered attribute in the strict order that they were issued.

1 1 **Simple.** The Target is permitted to reorder a sequence of I/O processes with this attribute. This allows a device to optimize performance, eg. by using an elevator seeking algorithm.

All previously issued Ordered I/O processes will be executed prior to executing any of the Simple I/O processes. Any subsequently issued Ordered I/O process will be executed after all of the Simple I/O processes have completed.

For a linked list of commands the Queue_ctl field is ignored in all SCSI_command messages except the first.

Editor's note: The Queue_Ctl field is expected to grow in SCSI-3. Therefore, bit 2 and maybe bit 3 in byte 10 should be considered as reserved for Queue_Ctl expansion.

Channel This 2-byte field is valid only if DDRM = 1b. It specifies the Channel component of the address field for data frames that the Target sends to the Initiator (the channel to be used within the destination). Channel is ignored for any commands that do not transfer data to the initiator.

CDB

This field is the Command Descriptor Block, as defined by SCSI-2 except that the Logical Unit Number in CDB byte 1, bits 7:5 is not used.

For SCSI-2 commands, the CDB is 6, 10 or 12 bytes long. *Editors note: SCSI-3 has also defined 16 byte CDBs (the maximum size).* For vendor-specific commands, the CDB may be up to 16 bytes long since the the maximum length of an SSA message frame is 32 bytes.

Typically the CDB contains a starting Logical Block Address and a Transfer Length. The last byte of the CDB is always a Control byte containing the **Flag** and **Link** bits.

The Flag bit is returned to the Initiator in the SCSI_status message corresponding to the command. When set it typically causes the Initiator to interrupt the application to indicate that a linked command has completed.

When set to 1b, the Link bit indicates that the I/O process is to be continued with a further command. In this case the Initiator will send a further SCSI_command message in response to the SCSI_Status message for the current command, provided that the status indicates Good Completion.

In the case of any of the following errors, a Response message will replace the SCSI_status message that normally completes the SCSI_command message.

- Unknown return_path field
- Unknown Vendor_unique field
- Invalid (non-zero) reserved fields (outside the CDB).

The Return_code in the Response message will be one of the following:

03h	Unknown Return_path.
FFh	Invalid parameter.

2.2 SCSI_status

This message is sent from a Target to the Initiator to indicate that a command and any associated data transfer have been terminated or completed. The message is returned using the path specified in *Return_path* in the associated SCSI_command message.

This message is returned for each SCSI_command message unless the command is cleared by any of the following:

- Abort_tag message
- Abort message
- Clear_queue message
- Device_reset message
- Total/Absolute Reset control frame
- Internal Target reset

The contents of the data field in a SCSI_status message are as follows:

Table 2. SCSI_status message

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Message_code = 11h							
1	Reserved = 000000b						Flag	Link
2 3	Tag							
4	Status							

Message_code	This byte identifies the message as 'SCSI_status'.
Flag	Bit 1 is a copy of the Flag parameter in the corresponding SCSI_command message.
Link	Bit 0 is a copy of the Link parameter in the corresponding SCSI_command message.
Tag	This 2-byte field is a copy of the Tag field in the corresponding SCSI_command message. It allows the Initiator to associate the status with the correct I/O process.
Status	This byte contains status as defined by SCSI-2 with the addition of Auto Contingent Allegiance Active status (30h).

2.3 Data_ready

This message is sent by the Target to the Initiator during the execution of a command that will transfer data to the Initiator. The message is returned using the path specified in *Return_path* in the associated SCSI_command message. The Initiator returns one or more Data_reply messages.

The Target may use more than one Data_ready message to transfer all of the data for a particular command. For example, the Target may perform a split read.

The contents of the data field in a Data_ready message are as follows:

Table 3. Data_ready message

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Message_code = 20h							
1	Reserved = 00h							
2 3	Tag							
4 5 6 7	Byte_offset							
8 9 10 11	Byte_count							

Message_code	This byte identifies the message as Data_ready.
Tag	This 2-byte field contains a copy of the Tag field in the corresponding SCSI_command message. It allows the Initiator to associate this message with the correct I/O process.
Byte_offset	This 4-byte unsigned integer indicates the byte offset of the first byte to be transferred, relative to first byte requested by the command. For read/write commands the offset must be a multiple of the block size. For control commands the offset must be a multiple of 16.
Byte_count	This 4-byte unsigned integer is a count of the number of significant bytes to be transferred for the current message. For read/write commands the count must be a multiple of the block size. For control commands the count must be a multiple of 16, unless the data to be transferred by this message includes the last byte requested in the command.

The protocol for the Data_ready and Data_reply messages is as follows (DDRM = 0):

1. When the Target is ready to transfer data it sends a Data_ready message to Initiator. This specifies how many bytes it currently has available and the starting offset within the data requested by the command.

   If the count or the offset does not conform to the rules above then the Initiator may issue an Abort_tag message to terminate the command.

2. Otherwise the Initiator allocates a Channel to receive the data.
3. The Initiator responds with a Data_reply message indicating the Channel address and how many bytes it can currently accept.

4. The Target sends data frames addressed to the specified Channel.
5. If all of the data offered by the previous Data_ready has not been transferred then the Initiator returns to step 3 on page 8.

   To improve performance the Initiator is permitted to send the next Data_reply message before it has received all data frames for the previous Data_reply.
6. If all of the data requested by the command has not been transferred then the Target returns to step 1 on page 8.

2.4 Data_reply

This message is sent by the Initiator to the Target in response to a Data_ready message. The Target replies by sending the requested data frames.

The Initiator may send more than one Data_reply message in response to a single Data_ready message. For example, the Initiator may have limited buffer space.

The contents of the data field in a Data_reply message are as follows:

Table 4. Data_reply message

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Message_code = 21h							
1	Reserved = 00h							
2 3	Tag							
4 5 6 7	Return_path							
8 9 10 11	Byte_count							
12 13	Channel							

Message_code	This byte identifies the message as 'Data_reply'.
Tag	This 2-byte field contains a copy of the Tag field in the corresponding SCSI_command message.
Return_path	This 4-byte field specifies the Path component of the address field for data frames. It is used by the Target together with the Tag field to identify the correct I/O process for this message.
Byte_count	This 4-byte unsigned integer is a count of the number of bytes that the Initiator can currently accept. If it is less than the number of bytes offered in the corresponding Data_ready message then the Initiator will send another Data_reply message when it is ready to receive the remainder.

For read/write commands the count must be a multiple of the block size. For control commands the count must be a multiple of 16, unless the data to be transferred by this message includes the last byte offered by the Data_ready message.

If the Initiator violates these rules then the Target may reject the message with a Response message (Response_code = Protocol Error).

Implementer's note: Since data can be retransmitted, the Initiator shall not compare the sum of the Byte_count fields in all the associated Data_reply messages with the expected total byte count to check for "lost" data.

Channel	This 2-byte field specifies the Channel component of the address field for the data frames (the channel to be used within the initiator for read data). The Path component is obtained from the Return_path field.

In the case of an invalid or unexpected Data_reply message (no Data_ready message sent from the Target with that Tag value, an unknown Return_path, or an invalid parameter), then a Response message will be sent with a Return_code of:

03h	Unknown Return_path.
10h	Protocol Error.
FFh	Invalid parameter.

The associated I/O processes identified by the Tag value (if any) is not terminated, instead the Data_reply message is simply rejected with a Response message.

2.5 Data_request

This message is sent from a Target to an Initiator during the execution of a command that will transfer data to the Target. The message is returned using the path specified in *Return_path* in the associated SCSI_command message. The Initiator responds by sending the requested data addressed to the specified Channel.

The Target may use more than one Data_request message to transfer all of the data for a particular command. For example, the Target may have limited buffer space.

The contents of the data field in a Data_request message are as follows:

Table 5. Data_request message

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Message_code = 22h							
1	Reserved = 00h							
2 3	Tag							
4 5 6 7	Byte_offset							
8 9 10 11	Byte_count							
12 13	Channel							

Message_code	This byte identifies the message as Data_request.
Tag	This 2-byte field contains a copy of the Tag field in the corresponding SCSI_command message. It allows the Initiator to associate this message with the correct I/O process.
Byte_offset	This 4-byte unsigned integer indicates the offset of the first byte to be transferred, relative to first byte requested by the SCSI command. For read/write commands the offset must be a multiple of the block size. For control commands the offset must be a multiple of 16 bytes.
Byte_count	This 4-byte unsigned integer is a count of the number of significant bytes that the Target is currently requesting. For read/write commands the count must be a multiple of the block size. For control commands the count must be a multiple of 16, unless the data to be transferred by this message includes the last byte requested in the command. : *Implementer's note: Since data can be retransmitted, the Initiator shall not compare the sum of the Byte_count fields in all the associated Data_request messages with the expected total byte count to check for "lost" data.*
Channel	This 2-byte field specifies the Channel component of the address field to be inserted in the data frames (the channel to be used within the target for data from the initiator).

The protocol for the Data_request message is as follows:

1. When the Target is ready to receive data it allocates a data Channel and sends a Data_request message to the Initiator. This specifies the Channel address, how many bytes the Target is currently requesting and the starting offset within the data specified by the command.

   If the count or the offset does not conform to the rules above then the Initiator may issue an Abort_tag message to terminate the command.

2. The Initiator responds by sending data frames addressed to the specified Channel. Although write data does not need to follow the same path to the Target used by the SCSI_command message, it must arrive at the same port.

3. If all the data specified by the command has not been transferred then the Target returns to step 1.

2.6 Abort_tag

This message is sent from the Initiator to the Target to abort a particular I/O process. Other I/O processes are not affected. If the Initiator is using multiple paths to the Target, the aborted I/O process may have been initiated over a different path (See 3.3.2.1, "Using alternate paths within an I/O process" on page 25 for a warning about using multiple paths).

Before issuing the Abort_tag message the Initiator should terminate any related outbound data transfer to ensure that data is not sent to a non-existent nexus. The Target terminates execution of the I/O process if it has already begun or the Target removes the I/O process from a queue if execution has not begun. The Target does not return a SCSI_status message unless the I/O process has already completed. However the Target always replies to the Abort_tag with a Response message.

The contents of the data field in the Abort_tag message are as follows:

Table 6. Abort_tag message

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Message_code = 30h							
1	Reserved = 00h							
2 3	Tag							
4 5 6 7	Return_path							
8 9	Tag_2							

Message_code	This byte identifies the message as 'Abort_tag'.
Tag	This is a 2-byte field that is used to relate the Response message to the Abort_tag message.
Return_path	This 4-byte field specifies the Path component in the address field of the Response message sent by the Target. The Return_path is also used by the Target to identify the Initiator that issued the I/O process. The Target should access the Initiator table using the Return_path field to determine the Unique_ID of the Initiator.
Tag_2	This is a 2-byte field that contains the Tag of the I/O process to be aborted. It is not an error if Tag_2 is unknown to the Target since the execution of the I/O process may have already completed.

The Return_code in the Response message will be one of the following:

00h	I/O process successfully aborted.
01h	I/O process not found.
FFh	Invalid parameter.

If the I/O process has already completed then the SCSI_status message will have been sent prior to the Response.

2.7 Abort

This message is sent from an Initiator to a Target to abort all I/O processes from that Initiator for a selected Logical Unit or Target routine. I/O processes from other Initiators are not affected. If the Initiator is using multiple paths to the Target, the aborted I/O processes may have been initiated over a different path (See 3.3.2.1, "Using alternate paths within an I/O process" on page 25 for a warning about using multiple paths).

Before issuing the Abort message the Initiator should terminate any related outbound data transfers to ensure that data is not sent to a non-existent nexus. The Target does not generate a SCSI_status message for an aborted I/O process unless that process has already completed. However the Target always replies to the Abort message with a Response message.

The contents of the data field in the Abort message are as follows:

Table 7. Abort message

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Message_code = 31h							
1	LUNTAR	LUNTRN						
2 3	Tag							
4 5 6 7	Return_path							

Message_code	This byte identifies the message as 'Abort'.
LUNTAR	If bit 7 is reset to 0b then LUNTRN refers to a Logical Unit. If it is set to 1b then LUNTRN refers to a Target routine.
LUNTRN	This field specifies either a Logical Unit or a Target Routine, as indicated by the value of LUNTAR.
Tag	This is a 2-byte field that is used to relate the Response message to the Abort message.
Return_path	This 4-byte field contains the Path component for the address field of the Response message from the Target. The Return_path is also used by the Target to identify the Initiator and select the I/O processes to be aborted. The Target should access the Initiator table using the Return_path field to determine the Unique_ID of the Initiator.

The Return_code in the Response message will be one of the following:

00h	One or more I/O processes was successfully aborted.
01h	No I/O process was found.
FFh	Invalid parameter.

If an I/O process has already completed then the SCSI_status message will have been sent prior to the Response.

2.8 Clear_queue

This message is sent from an Initiator to a Target to abort all I/O processes from all Initiators for a selected Logical Unit or Target routine.

Before issuing the Clear_queue message the Initiator should terminate any related outbound data transfers to ensure that data is not sent to a non-existent nexus. If the Target has issued any Data_request messages to other Initiators on behalf of the selected Logical Unit or Target routine then the Target must receive and flush all of the requested data before aborting the corresponding I/O processes.

The Target does not generate a SCSI_status message for an aborted I/O process unless that process has already completed. However the Target sets Unit Attention for all Initiators for which an I/O process was aborted. The sense code indicates 'Command cleared by another Initiator'. The Target always replies to the Clear_queue message with a Response message.

The contents of the data field in the Clear_queue message are as follows:

Table 8. Clear_queue message

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Message_code = 32h							
1	LUNTAR	LUNTRN						
2 3	Tag							
4 5 6 7	Return_path							

Message_code	This byte identifies the message as 'Clear_queue'.
LUNTAR	If this bit is reset to 0b then LUNTRN refers to a Logical Unit. If it is set to 1b then LUNTRN refers to a Target routine.
LUNTRN	This field specifies either a Logical Unit or a Target Routine, as indicated by the value of LUNTAR.
Tag	This is a 2-byte field that is used to relate the Response message to the Clear_queue message.
Return_path	This 4-byte field contains the Path component for the address field of the Response message from the Target.

The Return_code in the Response message will be one of the following:

00h	One or more I/O processes was successfully aborted.
01h	No I/O process was found.
FFh	Invalid parameter.

If an I/O process has already completed then the SCSI_status message will have been sent prior to the Response.

2.9 Device_reset

This message is sent from an Initiator to a Target to clear all I/O processes for all Initiators on all Logical Units and all Target routines. The Target resets all Logical Units.

Before issuing the Device_Reset message the Initiator should terminate any related outbound data transfers to ensure that data is not sent to a non-existent nexus. If the Target has issued any Data_request messages to other Initiators then the Target must receive and flush all of the requested data before aborting the corresponding I/O processes.

The Target does not generate a SCSI_status message for an aborted I/O process unless that process has already completed. However the Target sets Unit Attention for **all** Initiators. The sense code indicates 'power-on or reset occurred'. The Target always replies to the Device_Reset message with a Response.

The contents of the data field in a Device_Reset message are as follows:

Table 9. Device_reset message

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Message_code = 33h							
1	Reserved = 00h							
2 3	Tag							
4 5 6 7	Return_path							

Message_code	This byte identifies the message as 'Reset'.
Tag	This is a 2-byte field that is used to relate the Response message to the Device_reset message.
Return_path	This 4-byte field specifies the Path component for the address field of the Response message from the Target.

The Return_code in the Response message will be one of the following:

00h	One or more I/O processes was successfully aborted.
01h	No I/O process was found.
FFh	Invalid parameter.

If an I/O process has already completed then the SCSI_status message will have been sent prior to the Response.

2.10 Clear_ACA_condition

This message is sent from an Initiator to a Target to clear an Auto Contingent Allegiance condition for that Initiator and a selected Logical Unit or Target routine. After the Auto Contingent Allegiance condition is cleared, any suspended queued command for that Initiator may become an active I/O process subject to the ordering rules.

Auto Contingent Allegiance conditions for other Initiators, Logical Units or Target Routines are not affected.

The Target always replies to the Clear_ACA_condition message with a Response message.

The contents of the data field in the Clear_ACA_condition message are as follows:

Table 10. Clear_ACA_condition message

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Message_code = 34h							
1	LUNTAR	LUNTRN						
2 3	Tag							
4 5 6 7	Return_path							

Message_code	This byte identifies the message as 'Clear_ACA_condition'.
LUNTAR	If bit 7 is reset to 0b then LUNTRN refers to a Logical Unit. If it is set to 1b then LUNTRN refers to a Target routine.
LUNTRN	This field specifies either a Logical Unit or a Target Routine, as indicated by the value of LUNTAR.
Tag	This is a 2-byte field that is used to relate the Response message to the Clear_ACA_condition message.
Return_path	This 4-byte field contains the Path component for the address field of the Response message from the Target. The Return_path is also used by the Target to identify the Initiator and select the Auto Contingent Allegiance condition to be cleared. The Target should search its Initiator table using the Return_path field to determine the Unique_ID of the Initiator.

The Return_code in the Response message will be one of the following:

00h	Auto Contingent Allegiance condition was successfully cleared.
20h	No Auto Contingent Allegiance condition existed for the addressed Logical Unit or Target routine.
FFh	Invalid parameter.

2.11 Response

Response is used to acknowledge the following SSA-SCSI messages:

- Abort_tag
- Abort
- Clear_queue
- Device_reset
- Clear_ACA_condition

The Response message is also used by the SSA-PH transport layer.

The contents of the data field in a Response message are as follows:

Table 11. Response message

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Message_code = 03h							
1	Return_code							
2 3	Tag							

Message_code This byte identifies the message as 'Response'.

Return_code This byte is coded to indicate the result of the original message. The following definitions are common to all messages:

00h	Requested function was completed successfully.
01h	No I/O Process found.
02h	Unique_ID not found (SSA-PH only)
03h	Unknown Return_path
04h-0Fh	Reserved (controlled by SSA-PH)
10h	Protocol error
11h-1Fh	Reserved (controlled by SSA-SCSI)
20h	No Auto Contingent Allegiance condition existed for the addressed Logical Unit or Target Routine.
21h-EFh	Reserved (controlled by SSA-SCSI)
F0h-FDh	Reserved (controlled by SSA-PH)
FEh	Requested function failed (SSA-PH only)
FFh	Invalid parameter. This indicates that the message was parsed successfully but one or more parameters was invalid. (See 2.13, "Invalid messages" on page 21 for the action taken when a message cannot be parsed.)

Tag This 2-byte field is copied from the original message. It identifies the message that is being acknowledged.

The Path component of the address field in Response is obtained from the Return_path field in the message being acknowledged.

2.12 Summary of Messages

The following is a summary of all SSA messages including the privileged messages defined in the SSA-PH document:

Table 12. Summary of SSA messages

Message_code	Message
00h	Query_node
01h	Query_node_reply
02h	Configure_port
03h	Response
04h	Async_alert
05h	Master_alert
06h	Quiesce
10h	SCSI_command
11h	SCSI_status
20h	Data_ready
21h	Data_reply
22h	Data_request
30h	Abort_tag
31h	Abort
32h	Clear_queue
33h	Device_reset
34h	Clear_ACA_condition

2.13 Invalid messages

If a node receives an invalid message then it proceeds as follows:

1. If the node can associate a Return_path with the message then it returns a Response message with a Return_code of FFh when a parameter is invalid.
2. If the node does not know where the message came from (eg. Message_code is invalid or the length of the message is incorrect), the Target sends an Async_alert message to the Master Initiator. The Async_alert indicates 'message reject' in the *Type* field and it specifies the node (via the *Tag* field), the port that received the message and the first 24 bytes of the message. The Master forwards this information to all other Initiators.

   Please consult the SSA-PH specification for details.
3. If a protocol error is detected by the Target (eg. a Data_reply message without a previous Data_ready message or a Data_reply message for an unknown tag), a Response message with a Return_code = 10h (Protocol error) is sent.
4. If Return_path in a message is not in the Initiator table of the Target, a Response message with a Return_code = 03h (Unknown Return_path) is sent.
5. If fields in the SCSI CDB are invalid or duplicate tags are used, a SCSI_status message with Check Condition status is sent.

If a SCSI_command message is invalid and a Response message or Async_alert message is sent, the I/O process is discarded and no SCSI_status message is sent. If a Data_reply message is invalid, the I/O process identified by the tag is not terminated by the invalid message and the Target will continue to wait for a valid Data_reply message if it has sent a Data_ready message.

3.0 Some differences between SSA-SCSI and parallel SCSI-2

This chapter will explore some of the differences between SSA-SCSI and parallel SCSI-2.

3.1 Non-interlocked

Since SSA-SCSI is frame multiplexed rather than interlocked, it cannot depend on the bus phases to indicate acceptance or completion of a message. The ACK response in SSA-SCSI only indicates that the down-stream node has correctly received the frame. This implies that some form of response is needed for each message to indicate completion (See 2.11, "Response" on page 19 for more details).

3.1.1 Untagged queueing is emulated

Since SSA-SCSI is frame multiplexed, a tag or channel must be associated with each frame to indicate the proper context. For this reason, untagged commands are not supported by SSA-SCSI. Untagged command queueing can be simply emulated by the Initiator having only one outstanding command per LUN, re-use the same tag, and use the simple queue type. This is enforced by not allowing the untagged queue type in Queue_ctl in the SCSI_command message of SSA-SCSI.

3.1.2 Auto Contingent Allegiance Condition

In SSA-SCSI the SCSI-2 Contingent Allegiance Condition is replaced by the SCSI-3 Auto Contingent Allegiance Condition. This is necessary because an SSA Initiator can send further SCSI_command messages prior to receiving the Check Condition status for a previous command. This would result in the Contingent Allegiance Condition being reset inadvertently.

When a Target sends Check Condition status to an Initiator it creates an Auto Contingent Allegiance condition for that Initiator and Logical Unit or Target Routine. The Auto Contingent Allegiance condition causes the Target to either suspend or purge queued I/O processes for that nexus, depending on the value of the QErr bit in Mode Sense/Select parameter page 0Ah.

The Auto Contingent Allegiance condition also causes subsequent I/O processes received for the same nexus to be rejected with Auto Contingent Allegiance Active status (30h), unless the Queue_ctl field specifies Auto Contingent Allegiance. Such an I/O process will be referred to as an ACA I/O process. ACA I/O processes are executed normally during an Auto Contingent Allegiance condition.

Sense data is cleared after completion of the first command of an ACA I/O process for the nexus that has the Auto Contingent Allegiance condition. Typically an Initiator will issue an ACA Request Sense command first to retrieve the sense data.

Only one ACA I/O process can be active for a given nexus at a time. If a Target receives an ACA I/O process while it has another active ACA I/O process for the same nexus then it rejects the second I/O process with Auto Contingent Allegiance Active status.

If a Target receives an ACA I/O process while it does not have an Auto Contingent Allegiance condition for the same nexus, then it returns Check Condition status. The sense data contains a sense key of 'Illegal Request' and an additional sense code of 'Invalid Message Error'.

The Auto Contingent Allegiance condition is reset by any of the following:

- Clear_ACA_condition message
- Device_reset message

- SCSI Hard Reset conditions (including Total/Absolute reset)
- Internal Target reset

After the Auto Contingent Allegiance condition is reset by a Clear_ACA_condition message, the Target resumes processing suspended I/O processes for that nexus.

The Auto Contingent Allegiance condition is associated with a particular Initiator and Logical Unit or Target routine. It does not affect other I/O processes. Multiple Auto Contingent Allegiance conditions can exist for different Initiators, Logical Units and Target routines at the same time.

3.1.3 Message Reject not applicable

Since SSA-SCSI is not an interlocked protocol, the Message Reject condition cannot exist as it does today in the parallel environment. SSA-SCSI has already indicated acceptance of the message by the downstream node by the receipt of the ACK. Even if the downstream node is the destination, the ACK indicates the successful acceptance of the frame, not it's successful parsing.

However, SSA-SCSI does respond to some of the message reject cases with a Response message (with Return_code = Protocol Error). Refer to section 2.13, "Invalid messages" on page 21 for the proper handling of invalid messages such as unknown Initiator, or unexpected Data_reply message.

3.1.4 All messages give responses

As an additional error detection condition, SSA-SCSI requires all messages to generate a response condition (either SCSI_status message or the Response message). This includes such messages as Device_reset, Abort, Abort_tag, and Clear_queue which would not generate a response in parallel SCSI-2.

3.1.5 Bus phases do not exist

Since SSA-SCSI is frame multiplexed, parallel SCSI-2 bus phases do not exist. Each frame could be considered to do an arbitration, selection/reselection, and disconnect phase (without the parallel SCSI-2 bus overhead). A data frame (channel > 0) or SCSI_command frame is considered to have done data transfer phase.

3.2 Performance enhancements

3.2.1 Less overhead for arbitration, selection, disconnection, reselection

Since SSA is frame multiplexed, each frame does implicit arbitration, selection/reselection, and disconnection all with only 8-12 bytes of overhead per frame (400 ns to 600 ns). This is considerably faster than SCSI-2 parallel can run for reasonable transfer lengths.

3.2.2 Out of order data transfers

Out of order data transfers (zero latency reads or writes) reduce latency by allowing the Target to begin transfer from at the current rotational latency position. For a full track read, this can save an average of 1/2 a revolution or 33% over a Target that does not support out of order transfers. Without out of order, 1/2 rotation average is used to find the beginning, and one revolution to read or write the track. With out of order transfers, only one revolution is used (R/W last x%, then R/W first 100-x%).

In parallel SCSI, out of order data transfers were optional by using the Modify Data Pointers message. SSA-SCSI makes the support mandatory for the Target, and selectable by the Initiator. The Split field in the SCSI_command message is used along with the Data_Ready, Data_Reply, and Data_Request messages to activate out of order data transfers.

3.2.3 Concurrent data transfers within a network

Since SSA-SCSI has a packet multiplexed interface with a fixed packet overhead and multiple channel capability, there is no penalty for transferring multiple data transfers simultaneously. This can be an advantage in a RAID application using a single SSA network, because the data from each of the RAID components can be processed (ie reconstruction) as the data comes in. In other words, a parallel bus would tend to transfer all the data from one Target and then disconnect, which would allow the next Target to transfer a chunk of data, etc, before the data could be verified. But with SSA, all RAID components could be transferring simultaneously in an interleaved fashion, allowing the data to be verified as it arrives, reducing latency.

3.2.4 Concurrent data transfers from a single target

In addition to a network transferring multiple data transfers concurrently, a single device could be transferring multiple data transfers. For example, a media read might be transferring over one SSA port, while a cached read (or write prefetch) is transferring over the same or different ports.

3.2.4.1 Error handling of concurrent transfers from a single target.

When a target generates an Auto Contingent Allegiance condition, any I/O processes within the Auto Contingent Allegiance domain must be shut down. When shutting down an I/O process that involves a concurrent data transfer, the data transfer burst can complete, but the target shall not initiate a new data burst (Data_ready or Data_request), nor complete the I/O process (SCSI_status) until the Auto Contingent Allegiance has been cleared.

3.2.5 Minimal Initiator/Target exchanges

Initiator/Target exchanges can be minimized by grouping the parallel SCSI-2 messages of Identify, Queue Tag, Command, and data transfer (for the CDB bytes) into the SSA-SCSI SCSI_command message. The SCSI_command message has addition function of being able to define options not available in SCSI-2 such as DDRM, Split, and the vendor unique bytes on a command by command basis.

3.3 Availability enhancements

By nature of it's rich topologies and dual port nature, SSA-SCSI allows for increased availability.

3.3.1 Dual port

A typical SSA implementation consists of dual ports.

No ordering can be assumed between ports. For example, assume a node receives three ordered commands. Assume Port 1 receives CMD1, port 2 receives CMD2, and port 1 finally receives CMD3. It is indeterminate as to whether the queue will contain CMD1,CMD2,CMD3 or CMD2,CMD1,CMD3 or CMD1, CMD3, CMD2. However, ordering is maintained within a port (CMD3 is always after CMD1).

Unit attentions are based on Unique ID rather than return path. Therefore, an Initiator will only receive one Unit Attention for a given condition, even if it uses multiple paths to a node.

For an Initiator, the route to a Target consists of an Initiator port ID and a path. This means that the same path value could be used on different ports to address different Targets. Likewise for a Target, the return route consists of a Target port ID and a return path. In both the Initiator and Target case, all frames associated with an I/O process must use the same Initiator and Target ports. For example, when a SCSI_Command message arrives for a read command, the Target must use that port ID for the Data_Ready messages (if any), and the SCSI_Status message. The exception to this is that the Abort, Abort_tag, and Quiesce messages may terminate I/O processes initiated on a different path/port.

Dynamic pathing is **not** supported on SSA-SCSI. Dynamic pathing would allow a Target to dynamically use different paths to receive/transfer commands, data, and status.

3.3.2 Alternate paths

The dual port and switch functions of SSA-PH allow for alternate paths between the Initiator and node. An abort or clear queue function can occur on a different path than that used by the associated SCSI_command message. This complicates matters as the system must know that the termination of an I/O process (Abort, Abort_tag, Clear_queue, Device_Reset) on a second path may be received by the destination prior to the preceding SCSI_command message frames on the original path. This complicates Tag management.

The SCSI Architectural model may restrict the simultaneous use of multiple paths between an Initiator and a Target, due to the lack of ordering between paths when using two different paths. Likewise, the SCSI architecture model may add additional restrictions of taking the failing path offline prior to issuing Abort, or Abort_tag messages down a different path than that used by the SCSI_command.

3.3.2.1 Using alternate paths within an I/O process

The Target will always send all frames associated with an I/O process using the Return_path field specified in the appropriate message (SCSI_command, Abort, etc).

The Initiator is required to send all frames associated with an I/O process to the same port on the Target (the write data frames associated with a SCSI_command must arrive on the same port that the SCSI_command message arrived on). The same port is required, but not the same path.

However, SCSI requires ordering to be maintained. Therefore, only interlocked sequences can use different paths. This allows several scenarios to use different paths/same port.

1. Data_reply messages

Since the Data_reply message is interlocked with the receipt of the Data_ready message and the read data, the Data_reply message itself can use a different path/same port that the SCSI_command message.

2. Write data frames

   Since the Write data frames are interlocked with the Data_request and SCSI_status messages, the write data frames can use a different path/same port as the SCSI_Command message.

Also, the Abort, Abort_tag, Clear_queue, Device_reset, and Clear_ACA_condition message frames can be sent down a different path than the I/O processes they are resetting. Indeterminate results can occur if a different path was used, or the associated Response message uses a different Return_path than the reset I/O process's SCSI_status return path. In the case where a SCSI_command is issued prior to an associated Abort_tag message, the following possibilities can occur:

1. The SCSI_command is executed prior to the Abort_tag arriving at the Target. The SCSI_status may arrive either before the Abort_tag Response message, or after (if a different path was used). In either case, the Response_code is "No I/O process was found", but the SCSI_status may arrive before or after the Response message.
2. The SCSI_command is held in its path and has not yet arrived at the Target. The Abort_tag message uses a different path and arrives and generates a Response message. The SCSI_command arrives at the Target following the Response message, executes, and generates a SCSI_status message. The SCSI_status may arrive either before the Abort_tag Response message (if a different path was used), or after. In either case, the Response_code is "No I/O process was found", the SCSI_status message status byte indicates normal execution, and the SCSI_status may arrive before or after the Response message.
3. The SCSI_command is executing during the Abort_tag arriving at the Target. A Response_code is "I/O process was successfully aborted" is returned. This case is not a problem.

A problem exists in cases 1 and 2 when the SCSI_status has not yet arrived and the Abort_tag Response message indicated a "No I/O process was found" Response_code. The Initiator does not not know if the if the Tag can be re-used, nor if the associated SCSI_command will execute in the future or not.

These problems can be partially alleviated by sending the Abort_tag using the same path and Return_path of the associated SCSI_Command. If the path has a hard failure, then the Initiator can issue the Abort_tag down another path/Return_path, but should "throw away" any unexpected SCSI_status or Response messages on that port.

If an Abort message is used to abort a set of commands that were sent over different paths, then an Abort should be issued over all paths used, and should not be considered complete until all Response messages associated with the Abort messages are received. This insures that all paths have been "cleared". Any SCSI_status messages received prior to that path's Abort Response message, should be considered to have executed. If the path has a hard failure, then the Initiator should "throw away" any unexpected SCSI_status or Response messages on that port.

3.4 Addressing

SSA-SCSI greatly enhances the maximum number of addresses on an SSA network. This mandates some changes to parallel SCSI-2.

3.4.1 Additional addressing

SCSI-2 allows addressing of 8 (8 bit SCSI) or 16 (16 bit SCSI) on a SCSI bus. SSA allows approximately ¼ million devices on an SSA path. This is limited by the four byte path in SSA-SCSI. SSA-SCSI can have a 126 port switch connected to an Initiator and the other 125 ports each connected to a 126 port switch, each of which is connected to a 126 port switch, each of which is connected to a 128 port string (125*126*126*128)+1 = 254,016,001 addresses.

3.4.2 Larger LUN field

SSA-SCSI defines a 128 Logical Units (LUN) and 128 Target Routines within a Target, as opposed to parallel SCSI-2, which allows 8 LUNs and 8 Target routines.

3.4.3 Third party commands

SSA-SCSI needs more addressing information to process a third party command than SCSI-2, namely:

- Path to the source (4 bytes)
- Return_Path to the source (4 bytes)
- LUN of the source (1 byte)
- Port to use to access the source (1 to 7 bits)
- Path to the destination (4 bytes)
- Return_Path to the destination (4 bytes)
- LUN of the destination (1 byte)
- Port to use to access the destination (1 to 7 bits)

Since SCSI-3 is currently actively working on third party commands, SSA-SCSI will not attempt define a potentially conflicting method of doing third party commands. Until SSA-PH adopts the SCSI-3 solution (if any) or defines it's own method of doing third party commands, the following SCSI commands are not supported.

- Copy command (18h)
- Copy and Verify command (3Ah)
- Release command (17h) (Third party only)
- Reserve command (16h) (Third Party only)

3.4.4 Unique ID and Unit Attentions

An SSA network may contain many Initiators. During configuration every node receives a Query_node message from each Initiator. If an Initiator intends to use alternate paths to the same Target then it issues a Query_node over each path. A Target node uses the information in Query_node to build an entry in its Initiator table. The entry contains the Unique_ID and Return_path for that Initiator. (Please consult the SSA-PH specification for details.)

When a Target generates a Unit Attention it sets a flag in each unique Initiator in the Initiator table. (If the Unit Attention was generated by a message from an Initiator then the Target does not set the flag(s) for that particular Initiator.)

When a Target receives a SCSI_command message it locates the entry for that Initiator by searching the table with the Return_path. If the flag is set then the Target presents Check Condition status and generates sense. It then resets all of the flags for the corresponding Unique_ID. Thus the Unit Attention is eventually presented once to each Initiator.

It is recommended that Targets hash the Return_path in a SCSI_command message in order to speed up the search of the Initiator table. The search can be bypassed completely when the Target has no Unit Attentions out-standing.

3.4.5 Return_path vs. Unique_ID

During the configuration process, the Target created an Initiator table that associated an Initiator's Unique_ID with a set of Return_paths.

The Initiator is identified with a Unique_ID regardless of the number of paths used. Therefore, an I_T_L nexus involves the group of paths that connect a Target/LUN to an Initiator.

However, some messages will use a Return_path to identify an Initiator. The following messages supply a Return_path which the Target is to use to operate on *all* paths to the associated Initiator.

- **Abort message** uses Return_path to identify an Initiator, and causes all commands issued on all paths associated with that Initiator to be aborted.
- **Clear_ACA_condition message** will clear the ACA condition for the affected Initiator, even if the message arrives on a different path than the Check Condition Status was returned on.

A Unit Attention condition is associated with a Initiator. The Initiator will only see the Unit Attention condition on the first path to the Target that it uses following the Unit Attention condition.

3.5 Other functions

3.5.1 Spindle synchronization

SSA-SCSI defines the 8B/10B character K28.0 as a SYNC character as spindle sync, which can be interleaved within frames on the link subject to the rules for User-defined characters in SSA-PH.

One node in the network should be nominated to originate the SYNC characters at the nominal rotation rate of the disk drives. For example, this node could be an array controller or a disk drive. The other disk drives decode the SYNC characters and use the decode to synchronize their spindle servos. (Thus the decode replaces the Master_sync pulse that was provided on a separate cable for the SCSI bus.) In the event that the originating node fails a backup node can be nominated to replace it.

The propagation of User-defined characters is controlled by the EUDC parameter of the Configure_port message. (See the SSA-PH specification for details.) Dual-port and switch nodes should normally be programmed to propagate User-defined characters received on one port to the other port(s).

The origination and receipt of SYNC characters is controlled by the RPL parameter in page 4h of the SCSI Mode Select parameters.

3.5.2 Effects of reset conditions

SSA-PH defines five levels of reset: Link Reset, Local Reset, Total Reset, Absolute Reset, and Power On Reset.

Link Reset has no affect on the SCSI model.

Local Reset, Total Reset, Absolute Reset, and Power On Reset will generate the Hard Reset Condition as defined by SCSI-2. In addition to the normal clearing of all I/O processes, reservations, and operating conditions to their last saved positions; the Initiator Table (Target) and Configuration table (Initiator) will be cleared.

Currently neither SSA-PH nor SSA-SCSI provide a global reset equivalent to the RST signal in the parallel SCSI bus.

3.6 Optional SCSI-2 features not supported in SSA-SCSI

The following optional SCSI-2 features are not supported in SSA-SCSI. Future proposals may add these features.

3.6.1 Third party commands

SCSI-2 third party commands will not work in SSA-SCSI, and SCSI-3 has not yet approved a new third party command definition. To avoid defining a conflicting method, SSA-SCSI does not support third party commands at this time. See 3.4.3, "Third party commands" on page 27 for more information.

3.6.2 Asynchronous Event Notification (AEN)

SSA-SCSI does not support Asynchronous Event Notification (AEN).

3.6.3 Terminate I/O process message

SSA-SCSI does not support the Terminate I/O Process message, due to lack of implementation in today's parallel SCSI products.

4.0 Scenarios

4.1 Read command

The scenario below shows the messages exchanged between the Initiator and the Target for a typical SCSI Read command. It is assumed that the DDRM field in the SCSI_command message is set to 0b.

Table 13. Read scenario with DDRM = 0b

Initiator	Target
SCSI_command message ---->	
	Queue command
	Execute command
	<------ **Data_ready** message
Initialize channel (buffer pointer set)	
Data_reply message ----------->	
	<------ **Data** frames
	"
	"
Free channel (when Byte_count satisfied)	
	<------ **SCSI_status** message

There can be more than 1 Data_ready message, eg. a split read might have 2 Data_ready messages, one for the tail followed by another for the head. Also there can be more than 1 Data_reply message for each Data_ready message.

When DDRM is set to 1b the Data_ready and Data_reply messages are not generated:

Table 14. Read scenario with DDRM = 1b

Initiator	Target
Initialize channel (buffer pointer set)	
SCSI_command message ---->	
	Queue command
	Execute command
	<------ **Data** frames
	"
	"
Free channel (when Byte_count satisfied)	
	<------ **SCSI_status** message

4.2 Write command

The scenario below shows the messages exchanged between the Initiator and the Target for a typical SCSI Write command:

Table 15. Write scenario

Initiator	**Target**
SCSI_command message ---- >	
	Queue command
	Execute command
	Initialize channel (buffer pointer set)
	< ------ **Data_request** message
Data frames --------------------- >	
"	
"	
	Free channel (when Byte_count satisfied)
	< ------ **SCSI_status** message

There can be more than 1 Data_request message, eg. if the Target has limited buffer space.

Appendix A. Parallel to SSA SCSI message conversion

A.1 SCSI Message summary

The table below shows the mapping (if any) between the functions of the SCSI-2 messages and SSA-SCSI:

Table 16. Mapping from SCSI-2 messages to SSA-SCSI

SCSI-2 message	SSA-SCSI message
No Operation	N/A
Simple Queue Tag	SCSI_command(Queue_ctl = 11b, Tag)
Ordered Queue Tag	SCSI_command(Queue_ctl = 10b, Tag)
Head of Queue Tag	SCSI_command(Queue_ctl = 01b, Tag)
ACA Queue Tag (SCSI-3)	SCSI_command(Queue_ctl = 00b, Tag)
Identify (Out)	SCSI_command(LUNTAR, LUNTRN)
Command Complete	SCSI_status
Linked Command Complete	SCSI_status(Link = 1b)
Linked Command Complete with Flag	SCSI_status(Link = 1b, Flag = 1b)

Identify (In)	SCSI_status(Tag)
	Data_ready(Tag)
	Data_request(Tag)
	Response

Modify Data Pointer	Data_request(Byte_offset)
	Data_ready(Byte_offset)

Disconnect	N/A
Save Data Pointer	N/A
Restore Pointers	N/A
Initiate Recovery	N/A
Release Recovery	N/A
Terminate I/O Process	N/A
Abort	Abort
Abort Tag	Abort_tag
Clear Queue	Clear_queue
Clear ACA Condition (SCSI-3)	Clear_ACA_condition
Bus Device Reset	Device_reset
Message Reject	Response Async_alert & Master_alert
Initiator Detected Error	N/A
Message Parity Error	N/A
Synchronous Transfer Request	N/A
Wide Data Transfer Request	N/A
Ignore Wide Residue	N/A
N/A	Data_reply

A.2 Parallel to SSA SCSI Message Conversion Discussion

The following sections give a very brief description of how to map each SCSI-2 message into an SSA-SCSI message.

A.2.1 (00h) Command Complete

Command Complete maps to the SSA Message "SCSI_status" with the status byte residing in byte 4, the Link bit = 0, and the Flag bit = 0.

A.2.2 (01h, 00h) Modify Data Pointer

The Modify Data Pointer message maps to the SSA messages of Data_Ready and Data_Request.

A.2.3 (01h, 01h) Synchronous Data Transfer Request

N/A since SSA is a packet interleaved serial bus that always transfers at a fixed speed.

A.2.4 (01h, 02h) Extended Identify (SCSI-1)

N/A since Extended Identify was a SCSI-1 concept and is not supported in SCSI-2, SCSI-3 or SSA.

A.2.5 (01h, 03h) Wide Data Transfer Request

N/A since SSA is a serial interface, and does not operate in a "wide" mode.

A.2.6 (02h) Save Data Pointer

N/A since a packetized protocol does an implicit disconnect after every packet.

A.2.7 (03h) Restore Data Pointer

N/A since a packetized protocol does an implicit disconnect after every packet.

A.2.8 (04h) Disconnect

N/A since a packetized protocol does an implicit disconnect after every packet.

A.2.9 (05h) Initiator Detected Error

N/A as frame rejects are handled in the physical layer of SSA. The level of reset will depend on the Initiator's error recovery procedure.

A.2.10 (06h) Abort

This messages maps directly to the SSA message "Abort".

A.2.11 (07h) Message Reject

N/A as frame rejects are handled in the physical layer of SSA.

A.2.12 (08h) No Operation

N/A for SSA. When an node does not need to send any information, flag characters as sent over the SSA cable.

A.2.13 (09h) Message Parity Error

N/A as frame CRC checks are handled in the physical layer of SSA.

A.2.14 (0Ah) Linked Command Complete

Linked Command Complete maps into the SSA message "SCSI_Status" with with the status byte residing in byte 4, the Link bit = 1, and the Flag bit = 0.

A.2.15 (0Bh) Linked Command Complete with Flag

Linked Command Complete With Flag maps into the SSA message "SCSI_Status" with with the status byte residing in byte 4, the Link bit = 1, and the Flag bit = 1.

A.2.16 (0Ch) Bus Device Reset

Bus Device Reset maps to the SSA message "Reset".

A.2.17 (0Dh) Abort Tag

Abort Tag maps to the SSA message "Abort_tag".

A.2.18 (0Eh) Clear Queue

Clear Queue maps to the SSA message "Clear_queue".

A.2.19 (0Fh) Initiate Recovery

N/A since SSA does not support Extended Contingent Allegiance.

A.2.20 (10h) Release Recovery

N/A since SSA does not support Extended Contingent Allegiance.

A.2.21 (11h) Terminate I/O Process

N/A since SSA does not support the Terminate I/O process message.

A.2.22 (16h) Clear ACA Condition (SCSI-3)

The Clear ACA Condition SCSI-3 message maps to the SSA message Clear_ACA_condition.

A.2.23 (20h, xxh) Simple Queue Tag

Simple Queue Tag maps to the SSA message "SCSI_command" with a Queue_type field = 11b (Simple) and the queue tag value in byte 3.

A.2.24 (21h, xxh) Head of Queue Tag

Head of Queue Tag maps to the SSA message "SCSI_command" with a Queue_type field = 01b (Head) and the queue tag value in byte 3.

A.2.25 (22h, xxh) Ordered Queue Tag

Ordered Queue Tag maps to the SSA message "SCSI_command" with a Queue_type field = 10b (Ordered) and the queue tag value in byte 3.

A.2.26 (24h, xxh) ACA Queue Tag (SCSI-3)

ACA Queue Tag maps to the SSA message "SCSI_command" with a Queue_type field = 00b (ACA) and the queue tag value in byte 3.

A.2.27 (23h, xxh) Ignore Wide Residue

N/A as a set of data frames can be any number of bytes, and the concept of ignoring data with a grosser granularity of one byte does not apply.

A.2.28 (80h-FFh) Identify

The Identify message maps to the SSA message "SCSI_command" with the LUN value indicated in byte 1. The bit indicating the privilege of disconnection is not applicable in a packetized interface where each frame has an implicit disconnect.

Appendix B. Performance Analysis

B.1 Performance Analysis Summary

Note: This appendix refers to the maximum *throughput* performance of SSA-SCSI, and does not intend to define the minimum *latency* performance of SSA-SCSI. The minimum latency of each node is 5 characters, since the CRC must be updated before the frame can be forwarded.

The following section illustrates the Start I/O performance of a single SSA link. A Start I/O is defined as everything needed to access data on the disk. The calculations show the maximum bandwidth of an SSA link cable segment. Since a single DASD unit cannot maintain the Start I/O rates, it is assumes that enough DASD are on the string to fully load the SSA Link cable segment. Error handling is not considered, but is insignificant based on the projected error rates.

B.1.1 The assumptions

The following assumptions are made.

- 10 byte read or write SCSI CDB's are used (worst case)
- Read Data_Ready and Data_Reply messages are used (worst case)
- 2 byte addressing is used (best case).
- A single 512 byte or 4K record is transferred.
- Spindle Sync is used (worst case)
- Neither the Initiator nor the Target withhold RR to slow the data transfer (best case)

The last item in the list assumes that the Initiator and the Target are fast enough to avoid having to with-hold RR and slow the data transfer. In many applications, dual buffers (A and B) would be sufficient. The first frame will fill the A frame buffer. The second data frame to arrive begins to fill the B frame buffer. The host or Target must be able to unload the A frame buffer, send RR, and have the upstream node process the RR in time to avoid delaying the sending of the third frame. Slower devices may choose to implement triple buffers (A/B/C) if high performance/throughput are critical. This is the best case, and some applications may slow the SSA. However, this performance analysis does not include these delays (if any), since they are application specific.

B.1.2 The results

Table 17. Start I/O rate at various R/W ratios and at 512 and 4K byte record

Read/Write Ratio	512 byte transfer	4,096 byte transfer
1 to 1	60,238	8,728
2 to 1	48.978	6,699
3 to 1	44,196	5,992
4 to 1	41,750	5,635

B.1.3 How to use this appendix

If you want to know the SSA performance figures, they are summarized on this page. If you want to re-create the figures or to understand how they were derived, then the following sections define the how the equations were derived, and what the calculations determined.

B.2 The equations

B.2.1 Read table

Table 18. Read performance (in bytes)

Frame Type	Frames		RR or ACK	
	Toward Target	From Target	Toward Target	From Target
SCSI_Command	OVR + 26	---	---	4
Data_Ready	---	OVR + 12	4	---
Data_Reply	OVR + 14	---	---	4
Data	---	NF*(OVR+128)	NF*4	---
SCSI_Status	---	OVR + 5	4	---

Note:

OVR is the frame overhead (8 to 12 depending on address)
NF is the number of 128 byte frames (for 512 NF=4, for 4K NF=32)
A 10 byte Read SCSI CDB is used (not 6 byte Read).

B.2.2 Read equations

B.2.2.1 Toward Target

The read equation toward the Target is ...

2*OVR + 40 + 4*NF + 8 = **(2*OVR) + (4*NF) + 48**

B.2.2.2 From Target

The read equation from the Target is ...

2*OVR + 17 + NF*(OVR+128) + 8 = **2*OVR + NF*(OVR+128) + 25**

B.2.3 Write table

Table 19. Write performance (in bytes)

Frame Type	Frames		RR or ACK	
	Toward Target	**From Target**	**Toward Target**	**From Target**
SCSI_Command	OVR + 26	---	---	4
Data_Request	---	OVR + 14	4	---
Data	NF*(OVR+128)	---	---	NF*4
SCSI_Status	---	OVR + 5	4	---

Note:

OVR is the frame overhead (8 to 12 depending on address)
NF is the number of 128 byte frames (for 512 NF=4, for 4K NF=32)
A 10 byte Write SCSI CDB is used (not 6 byte Write).

B.2.4 Write equations

B.2.4.1 Toward Target

The write equation toward the Target is ...

OVR + 26 + NF*(OVR+128) + 8 = **NF*(OVR+128) + OVR + 34**

B.2.4.2 From Target

The write equation from the Target is ...

2*(OVR) + 19 +4*NF + 4 = **(2*OVR) + (4*NF) + 23**

B.3 The start I/O calculations

B.3.1 Bytes available

At 20 MB/sec full duplex, 20,000,000 byte times are available every second each direction. However, Spindle Sync and the Fairness algorithm need to be considered.

B.3.1.1 Spindle Sync requirements

A DASD unit spinning at 5400 RPM (90 RPS) will generate 90 Spindle Sync pulses a second.

B.3.1.2 Fairness requirements

If we assume that the fairness quota is set to allow a single 4K start I/O to complete, then the quota will allow the following to pass thru the link **2*OVR + NF(OVR+130) + 25** (See B.2.2, "Read equations" on page 38). With an OVR=8 (2 byte address) and NF=32 (4K record), then the quota will allow **2*8 + 32*(8+130) + 25 = 16 + 32*138 + 25 = 4457** bytes to pass the from Target direction.

If we assume 16 drives (an arbitrary number) on the loop, then the SAT character will pass every 16*4457 characters, and a SAT′ character will also pass. Therefore, every second 20,000,000/(16*4457) or 280 SAT and SAT′ characters will occur every second.

Therefore, allowances should be made for 560 SAT or SAT′ characters.

B.3.1.3 Bytes available

This leaves the following bytes available to the link

20,000,000	Total bytes available
-90	Bytes used by Spindle Sync
-560	Bytes used by the Fairness algorithm
19,999,350	Bytes available for Start I/Os

B.3.2 Start I/Os rates

The following is a definition of the variables.

OVR	The frame overhead (=8 for 2 byte addresses, =12 for 6 bytes addresses (worst case)).
NF	Number of 128 byte data frames per Start I/O (=4 for 512, =32 for 4K).
RSIO	The Read Start I/O rate.
WSIO	The Write Start I/O rate.
RWR	The Read/Write ratio (1:1=1, 2:1=2, 3:1=3, etc.).

Using the following two equations...

1. (RSIO)(Read equation) + (WSIO)(Write Equation) = (Bytes available)
2. RSIO = WSIO * RWR

Which generates... (WSIO*RWR)(Read Equation) + (WSIO)(Write Equation) = (Bytes available)
Which simplifies to... **RSIO = (Bytes available) / (RWR*(Read Equation) + (Write Equation))**

B.3.2.1 Toward the Target

The equation can be simplified as follows...

- WSIO = 19,999,350 / (RWR*((2*OVR)+(4*NF)+48) + (NF*(OVR+128)+OVR+34

For 512 byte records with OVR=8, the result is... **WSIO = 19,999,350 / (RWR*(78) + 586)**
For 4,096 byte records with OVR=8, the result is... **WSIO = 19,999,350 / (RWR*(190) + 4394)**

B.3.2.2 From the Target

The equation can be simplified as follows...

- WSIO = 19,999,350 / (RWR*(2*OVR + NF*(OVR+128) + 25) + (2*OVR)+(4*NF) + 23))

For 512 byte records with OVR=8, the result is... **WSIO = 19,999,350 / (RWR*(585) + 55)**
For 4,096 byte records with OVR=8, the result is... **WSIO = 19,999,350 / (RWR*(4393) + 167)**

B.3.3 Start I/O rate summary

Table 20. Start I/O rates summary.

Read/Write Ratio	512 byte transfer			4,096 byte transfer		
	Write Start I/O Rate		Total Start I/O Rate	Write Start I/O Rate		Total Start I/O Rate
	Toward Target	From Target		Toward Target	From Target	
1 to 1	30,119	31,248	**60,238**	4,362	4,385	**8,724**
2 to 1	26,953	16,326	**48,978**	4,189	2,233	**6,699**
3 to 1	24,389	11,049	**44,196**	4,028	1,498	**5,992**
4 to 1	22,270	8,350	**41,750**	3,880	1,127	**5,635**

Note: OVR=8, NF=4 or 32, RWR=1,2,3,or 4.

Note: Start I/O rate is (RWR+1)*Minimum of the toward and from Target values.

Appendix C. SCSI Status

A SCSI Status Byte is sent to the Initiator in the SCSI_Status message at the termination of each SCSI command unless the command is cleared by any of the following occurrences:

- Abort message
- Abort_tag message
- Clear_queue message
- Device_reset message
- A Hard reset condition (POR, Local Reset, Total Reset, and Absolute Reset).

The SCSI Status Byte is defined in Table 21.

Table 21. SCSI Status Byte

Byte	BIT							
	7	6	5	4	3	2	1	0
Status	Reserved = 0		Status Code					Rsvd

Table 22. Status Code Bit Definitions

	Bits of Status Code							
Status	7	6	5	4	3	2	1	0
Good	R	R	0	0	0	0	0	R
Check Condition	R	R	0	0	0	0	1	R
Condition Met	R	R	0	0	0	1	0	R
Busy	R	R	0	0	1	0	0	R
Intermediate/ Good	R	R	0	1	0	0	0	R
Intermediate/ Condition Met	R	R	0	1	0	1	0	R
Reservation Conflict	R	R	0	1	1	0	0	R
Command Terminated (not used)	R	R	1	0	0	0	1	R
Queue Full	R	R	1	0	1	0	0	R
ACA Active	R	R	1	1	0	0	0	R

Note: All Reserved fields (R) are set to zero.

A description of the status represented by each Status Byte is given below:

00h Good status

This status indicates that the Target has successfully completed the SCSI command. For this status, sense is not valid and the Sense key and the Sense code are set to zero.

02h Check Condition status

This status indicates that an error, exception, or abnormal condition has caused sense data to be set. An Auto Contingent Allegiance condition exists for this Initiator. The Initiator should issue a Request Sense command with an ACA Queue Type to obtain the sense data and determine the cause of the Check Condition status.

04h Condition Met status

This status indicates that the requested operation is satisfied. For this status, sense is not valid and the Sense key and the Sense code are set to zero.

08h Busy status

This status indicates that the Drive is busy performing another operation for a different Initiator and is unable to execute the command received from the currently connected Initiator. The recommended recovery action is to issue the command again at a later time. For this status, sense is not valid and the Sense key and the Sense code are set to zero.

10h Intermediate/Good status

This status indicates that the Target has successfully completed a linked command. This status is returned for every command in a series of linked commands (except the last command) unless an error, exception, or abnormal condition causes a Check Condition, Busy, or Reservation Conflict status to be returned. If this status is not returned, the chain of linked commands is broken. For this status, sense is not valid and the Sense key and the Sense code are set to zero.

14h Intermediate/Condition Met

This status is the combination of Condition Met and Intermediate/Good. For this status, sense is not valid and the Sense key and the Sense code are set to zero.

18h Reservation Conflict status

This status will be returned whenever an Initiator attempts to access a Logical Unit or an extent within that Logical Unit that is reserved with a conflicting reservation type for another SCSI device. The recommended Initiator recovery action is to issue the command again at a later time. For this status, sense is not valid and the Sense key and the Sense code are set to zero.

22h Command Terminated status

This status code is not used in SSA-SCSI.

28h Queue Full status

This status indicates that the Targets command queue is full. This status is returned when a SCSI_Command message is received and there is no room on the command queue for an I/O process from the issuing Initiator. For this status, sense is not valid and the Sense key and the Sense code are set to zero.

30h ACA Active status

This status shall be returned when a Auto Contingent Allegiance condition exists and an Initiator issues a command while one or more of the following conditions are true:

1. There is another I/O process with the ACA queue type; or
2. The Initiator issuing the command did not cause the ACA condition or;
3. The command did not have the ACA Queue type.

The ACA condition is not reset, the command did not execute (or even attempt to execute). Sense data was not generated, nor was it set to zero.