WORKING DRAFT X3T9.2/92-199r9

Revision 9
March 14, 1993

Information Technology
SCSI-3 Serial Bus Protocol

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

ASC X3T9.2 Technical Editors:

Edward A. Gardner	Gerald A. Marazas	Scott Smyers
CXO 1-2 / N26	IBM Corporation	MS 69G
Digital Equipment Corp.	P.O. Box 1328	Apple Computer
301 Rockrimmon Blvd.	Boca Raton, FL 33429	3535 Monroe Street
Colorado Springs, CO 80919	Mail Stop 5432	Santa Clara, CA 95051
Voice:		
(719) 548-2247	(407) 982-4423	(408) 974-7057
Fax:		
(719) 548-3364		(408) 974-2898
Email:		
gardner@ssag.enet.dec.com	marazas@bcrvmpc2.ibm.com	smyers.s@applelink.apple.com

Reference number
ISO/IEC ***** : 199x
ANSI ***** : 199x
Printed August 11, 2000

Other Points of Contact:

X3T9.2 Chair	X3T9.2 Vice-Chair
John B. Lohmeyer	I. Dal Allan
NCR Corporation	ENDL
1635 Aeroplaza Drive	14426 Black Walnut Court
Colorado Springs, CO 80916	Saratoga, CA 95070
Voice:	
(719) 596-5795 x362	(408) 867-6630
Fax:	
(719) 574-0424	(408) 867-2115
Email:	
john.lohmeyer@ftcollinsco.ncr.com	2501752@mcimail.com

Revision History

Changes from X3T9.2/92-199r8 to X3T9.2/92-199r8

- More editting towards ISO style compliance.

Changes from X3T9.2/92-199r7 to X3T9.2/92-199r8

- Editting towards compliance with ISO style guidelines.
- Revised definitions.

Changes at Revision 7 from the base of Revision 6:

- Made Changes in the Command block so as to gain better commonality with the Fibre Channel command block supporting SCSI.

Contents

Revision History ii

Foreward vi

Introduction vii

1. Scope 1

2. Normative References 1

3. Definitions and Conventions 1
3.1. Definitions 1

4. Overview 3

5. Model of Serial Bus Protocol 5
5.1. Model of Serial SCSI Initiator 5
5.1.1. Shouldler Tap Protocol for Command Delivery 6
5.1.1.1. Tap Slots Dedicated to an Initiator 7
5.1.1.2. Grant of an Initiator ID at Log-In Time 7
5.1.1.3. Request for Grant of Dedicated Tap Slots 7
5.1.1.4. Grant of a Tap Slot From a General Pool of Tap Slots 8
5.1.2. Setting of the M_Flag 8
5.1.3. Setting of the ESC_Flag 8
5.1.4. Setting of the L_Flag 9
5.1.5. Focus of a Command Chain 9
5.1.6. Composition of a Command Chain 9
5.1.6.1. Recommendations for Ordered Commands 10
5.1.6.2. Recommendations for Head of Queue Commands 11
5.1.6.3. Requirements for Processing Linked Commands 11
5.1.7. Retention of a Command Block by the Initiator 12
5.2. Model of Serial SCSI Target 12
5.2.1. Usage of the Status FIFO 13
5.2.2. Usage of the L_Flag 13
5.2.3. Usage of the M_Flag 14
5.2.4.Usage of the ESC_Flag 14
5.2.5. Management of Target Resources 14
5.2.5.1. Management of the Tap Slot Resource 15
5.2.5.2. Management of the Command Block Storage Resource 17
5.2.5.3. Schedule Policy for Fetching Among Multiple Command Chains 17
5.2.5.4. Fetch of Command Blocks From a Chain Referenced to the Urgent FIFO 19
5.2.5.5. Repeated Access to Same Command Block 19
5.2.5.6. Support of Autosense 19

6. Command Transfer Protocol 21
6.1. Conceptual Initiator - Target Connection 21
6.2. Multiple Initiator Environment 23

7. Packet Types 25

8. SCSI-3 Serial Bus Protocol Support Elements 27
8.1. Target "Register" Definitions 27
8.1.1. Command FIFOs 28
8.1.1.1. Normal FIFO 28
8.1.1.2. Urgent FIFO 29
8.1.1.3. ACA FIFO 30
8.1.2. First Failure "Register" (optional) 30
8.1.3. First Failure Control "Register" (optional) 31
8.2. Initiator "Register" Definitions 31
8.2.1. Status FIFO 31
8.2.2. Asynchronous Event Reporting 31

9. Command and Status Information 33
9.1. Command Blocks 33
9.2. Status Block 38
9.3. Initiator Scatter/Gather List 39

10. Payload Specification For Command Transfer Packets 41
10.1. Payload of SCSI Command Initiation Packet - "Tap Packet" 41
10.1.1. Response Reactions to Tap Packet 41
10.1.1.1. Response Reactions When Tap Packet is Accepted 42
10.1.1.2. Response Reactions When Tap Packet is Not Accepted 42
10.2. Command Read Request 43
10.2.1. Request Payload 43
10.2.2. Response Payload 43

11. Data Transfer Protocol 45
11.1. Asynchronous Transfer 45
11.1.1. Data Read From Device Medium 45
11.1.2. Data Written To Device Medium 45

12. Status Transfer Protocol 47
12.1. Target Reaction to Initiator Failure to Accept a Status Block 47
12.2. Target Reaction to Abort Tag Request From an Initiator 48
12.3. Target Reaction When a Cmmand Block Cannot Be Fetched 48
12.4. Target Reaction to a Unit Attention Condition 49

13. SBP Control Protocols 51
13.1. Log-in Protocol 51
13.2. Request/Release of a Tap Slot 52
13.3. Request/Release of Notification for Asynchronous Events 55

14. Examples 57
14.1. Target Read Command 57
14.2. Target Multiple Read Commands 58

15. Messages 61
15.1. Abort Tag Payload 62
15.2. Target Reset Payload 62
15.3. Payload of Clear Queue Packet 62
15.4. Priority Tap Message Payload 63

15.5. Log-In_Request Message Payload 63
15.6. FF_Control_Request Message Payload 63
15.7. Request/Release of Tap Slots Message Payload 63
15.8. Request/Release of Asynchronous Notification Message Payload 64

16. Compatibility to Parallel SCSI 65
16.1. Relation of a Target to Multiple Initiators 65

Appendix A. Packet Formats 67
16.2. Write Packets 67

Foreward

tbs

Introduction

tbs

Information Technology
SCSI-3 Serial Bus Protocol

1. Scope

tbs

2. Normative References

tbs

3. Definitions and Conventions

3.1. Definitions

3.1.1. Auto-Contingent Allegiance FIFO: This is a data structure within a target which is intended to hold the contents of a Tap packet associated with a command block chain intended by the initiator to respond to an Autocontingent Allegiance Condition (ACA) existing at the target.

3.1.2. Command Block Chain: A sequence of command blocks created by an initiator and stored within the address space of that initiator.

3.1.3. Command Block: A data structure that an initiator creates to describe a command to be performed by a target.

3.1.4. Fetch: The action of a target obtaining a copy of a command block and entering the command into a task set.

3.1.5. Initiator Identifier: An 8-bit identifier assigned uniquely to a given initiator by a target as a result of completion of a log-in procedure by the initiator at the given target.

3.1.6. Log-in: This is a procedure by which an initiator sends a Log-In Request message to a target for the purpose of obtaining a short form (8-bit) identifier to be used in all subsequent interactions with that target.

3.1.7. Normal FIFO: A data structure within a target which holds the contents of a Tap packet.

3.1.8. Sign-in: This is a procedure by which an initiator requests that a target provide to the initiator notification of asynchronous events which may occur at that target.

3.1.9. Status Block: A data structure sent by the target to convey completion status.

3.1.10. Status FIFO: A data structure within an initiator which holds the contents of a Status Block sent by a target to the initiator.

3.1.11. Tap Packet: This packet conveys a payload sent by an initiator to inform a target of a chain of command blocks in the initiator memory space. The target is requested to begin fetching these command blocks (one at a time) starting with the address of the first command block as conveyed by the Tap packet.

3.1.12. Tap Slot: A resource within a target used to hold a tap packet.

3.1.13. Urgent FIFO: A data structure within a target which holds the contents of a Tap packet.

4. Overview

The SCSI 3 Serial Bus Protocol has the following features:

- Fixed length command blocks (See note below)
- Fixed length status blocks
- Fair support for multiple initiators
- Separate command queues for each initiator/target pair
- Ability to support multiple command queues per initiator
- Command queue depth determined by initiator
- Number of overlapped commands in progress limited by target
- Supports Parallel SCSI error recovery procedures
- Supports Isochronous operation
- Supports a protocol for requesting and dedicating critical target resource associated with reception of notification from an initiator that commands are waiting for the target to work upon.
- Supports a protocol for dynamic allocation of the critical target resource associated with reception of notification of commands waiting in the initiator.
- Supports a protocol whereby initiators signify to a target that they request notification of asynchronous events occurring at that target.

Implementation Note:

target vendors (particularly at the low end) have expressed strong preference for a solution in which every effort is made to reduce the size of the command block. Opportunities should be sought to obtain a smaller command block if this can be done without making unacceptable compromises in functional capability.

The organization of the command block is intended to facilitate the situation in which a target can fetch portions of it in order to determine important aspects of the command which is carried and thereby to decided whether optimization opportunities exist. One instance of an optimization opportunity is selection of the sequence of command execution so as to minimize seek time.

5. Model of Serial Bus Protocol

This section describes the architectural model for the SCSI initiator and for the SCSI target as they interact within the SCSI 3 Serial Bus Protocol.

5.1. Model of Serial SCSI Initiator

The initiator is responsible for building the command blocks, queueing multiple command blocks, allocating status block buffers, and handling the correlation between command blocks and status blocks. It is anticipated that the initiator may find it beneficial to associate command blocks together into a structure commonly called a command block chain, or more simply a chain. A major reason for forming commands into a chain is that processing efficiency is enhanced within the initiator. Individual commands within the chain do not need to have any particular connection to one another. However, in many instances there is some relation among the commands as viewed by the initiator. In certain applications, a high level logical work item results in many physical level I/O commands which originate over such a small interval of time that the multiple commands constitute a block of work to be done by the target device. It becomes a natural as well as efficient solution to process these commands by forming them into a command block chain.

EDITORIAL NOTE: Care should be taken to distinguish between the new facility of commands formed into a chain and the existing facility in SCSI known as the Linked command. With regard to the SCSI 3 Queueing Model, an entire collection of Linked commands are considered as a single I/O Process. Once work by the target has been started on the first element within the set of of Linked commands, the target must fetch and complete each of the remaining elements within the Linked I/O process before any work may de done on behalf of any other I/O process. In important contrast to this situation for elements of a Linked I/O process, within the Serial SCSI command chain, each command is considered a separate and distinct I/O Process. If there are multiple chains, then the target may interleave its work activity among the chains. Linked commands may be used in a chain subject to the restriction that the collection of Linked commands must only be placed as the last element of a given chain.

In the SCSI 3 Serial Bus Protocol, command blocks are queued by an initiator within its memory space. An initiator must be capable of providing a given, queued command block at any time upon request from the target. This mechanism for command delivery is paced by the target. Such a mechanism controlled by the target helps to avoid busy conditions where the target cannot accept any further command blocks. As a supported and encouraged option within the Serial Bus Protocol, target devices may fetch multiple commands (one at a time) from any initiator. In this manner (via pre-fetch), the target has the optimization option of overlapping certain process steps for a later command while the current command is in execution.

Provision is made for the initiator to specify the manner in which the target is to fetch commands blocks when multiple chains are present. This control is exercised through use of the concept of sub-chains within a chain. The notion is that once a target has begun fetching command blocks from some subchain, the target is required to fetch command blocks, in sequential order, from start to finish of that sub-chain. Such a fetch policy is also known as "fetch until the queue is empty", or more briefly as a "QE" fetch policy. In contrast to the QE fetch policy, fetching between sub-chains is done on a Round Robin (RR) basis. A flag called the End Sub-Chain, or ESC_Flag, may be used to mark the end of a given sub-chain. Any command blocks following in the chain would thus be considered as being within a new sub-chain.

At one extreme, the ESC_Flag might never be used within any command block in a given collection of command block chains. In this circumstance the QE fetch policy is used among the chains so that every command within one chain is fetched before any command is fetched from some other chain. At the other end of the spectrum, the ESC_Flag is set to value one in each and every command block within every chain. In this circumstance a Round Robin fetch policy is used among all command blocks such that commands are fetched, one command from one chain, and then another command from some other chain. Should a

Linked command be encountered, then all commands from the Linked set must be fetched and completed before the target is allowed to process non-Linked commands from any chain.

5.1.1. Shouldler Tap Protocol for Command Delivery

As there are potentially multiple initiators in a system, there has to be a method for an initiator to indicate to a target that it has some work for that target. A conceptual "tap on the shoulder" is performed via a SCSI Command Initiation Packet sent to a set of Command FIFOs maintained by the target. Three members are defined for this set of Command FIFOs. The first member is the Normal FIFO and this is intended for use by command chains for which it is not necessary to fetch the associated command blocks on a special or a priority basis. The second member is the Urgent FIFO and this is intended for use by SCSI messages sent to the target by the initiator for various control purposes. One instance of use of the Urgent FIFO is for an initiator to call attention to a specific command (such as a Head of Queue command) which is to receive special or expedited processing by the target. The third member of of the Command FIFO set is the ACA FIFO and this is intended for use in support of sending ACA commands to a target for which an ACA condition is in effect.

IMPLEMENTOR NOTE: It is highly important that proper and appropriate use be made of Tap packets sent to the Urgent FIFO versus Tap packets sent to the Normal FIFO. The Urgent FIFO should be used to reference command blocks which are to receive priority fetching, which is to say, fetching in front of fetching of command blocks referenced to the Normal FIFO. Expected and appropriate allocation of target resource is such that many more Tap slots are to be provided for use with the Normal FIFO than are provided for use with the Urgent FIFO. This allocation of Tap Slots between the Normal and the Urgent FIFO is in expectation that the largest number of command chains and the largest number of command blocks are to be referenced to the Normal FIFO. Performance problems may be encountered if Tap packets which should have been sent to the Normal FIFO are instead sent to the Urgent FIFO.

One set of these three FIFOs shall be provided per target. This single set is to be shared by all Logical Units connected to the given target. In particular, the single Normal FIFO represents a conceptual common entry point for all SCSI Command Initiation packets (Tap packets) being sent to the given target. The functional use of this single common entry point is to ensure that relative order of arrival is correctly maintained among all Tap packets. This relative order of arrival defines an implied time stamp which is to be used for the command block chain associated with the tap packet. All of the allowed policies for fetching command blocks from a chain are based on this implied time stamp.

Many implementation choices are available as to hardware and/or software means to support the Normal FIFO and the Urgent FIFO data structures. A comparable structure, the Status FIFO, is defined for use within the initiator in order to receive status information from the target regarding command completion. Observe, at the conceptual level, the Status FIFO and the Normal FIFO and the Urgent FIFO have many notions in common. A key element common to both the Status FIFO in the initiator and the set of two FIFOs in the target is that the same size packet (12 Bytes of payload) is sent to all of them. This is not to say that any requirement exists to provide the same hardware implementation for the Normal FIFO, the Urgent FIFO, and the Status FIFO.

There are always limits as to the number of "Shoulder Taps" which can be accommodated at any one time by a target. The initiator must be prepared to accept a response from a target which states the present Tap cannot be accepted. Various software means may be employed by the initiator to deal with this situation that a target is temporarily not able to accept a Tap. Various elements of the Command Delivery architecture of the Serial Bus Protocol operate so as to make it an unusual and infrequent circumstance that a given Tap cannot be accepted. Some of these features are described in the following sections.

After the target has received a "Tap", it will read command blocks, one at a time, using a Read Transaction as defined by the Transaction Layer architecture of the IEEE 1394 standard. The Request portion of the Read Transaction makes use of IEEE 1394 style address of the command block. For the first command, the needed command block address is supplied via the SCSI Command Initiation Packet. For second and

following commands within a chain, if present, the address for the next command is contained as a field within the present command block.

5.1.1.1. Tap Slots Dedicated to an Initiator

As a key element of the way in which it can become a rare event for a Tap packet to be rejected by a target, there is the facility provided at the target that the target reserve a given number of slots (Tap Slots) for acceptance of a Tap packet from a given initiator. For its part, each initiator knows the minimum number of Tap slots at each target which are exclusively available to it. The initiator can then choose to assume the responsibility to manage its transmission of Tap packets so that the number of Tap Slots which it has in use at any point in time are less than or equal to its allocated number of Tap Slots.

So as to make efficient usage of the Tap Slot resource at the target, the initiator is allowed to attempt to secure usage of a larger number of Tap Slots than the number dedicated to that initiator by the given target. The key point is that the target is "allowed" to reject a Tap packet from an initiator if that Tap packet would consume a Tap Slot above and beyond the number of Tap Slots dedicated at the target for use by the given initiator. The initiator would need to provide software to manage its reaction to the rejection of a Tap packet when such rejection is "allowed". In the alternative, the initiator could choose to provide software which manages the number of Tap packets consistent with the number of Tap Slots dedicated for its use.

There is the related notion that a target is "not allowed" to reject a Tap packet from an initiator if the current number of "in-use" Tap Slots is less than the number of Tap Slots declared as dedicated to the given initiator. In this context, the term "not allowed" means that any rejection by the target of a Tap packet is a failure by the target to administer correctly the Shoulder Tap protocol. Such failure by the target, if occurring, shall cause appropriate error recovery procedures by the target. The intent, is to minimize, and hopefully reduce to zero, the impact on the initiator when a rejection of a Tap packet occurs under the "allowed" circumstance. To contrast for emphasis, it is the responsibility of the initiator to invoke initiator-side recovery/reaction procedures when rejection of a Tap packet occurs in the "allowed" situation. It is the responsibility of the target to invoke target-side recovery/reaction procedures when rejection of a Tap packet occurs in the "not allowed" situation.

5.1.1.2. Grant of an Initiator ID at Log-In Time

An initiator Identifier is used to provide convenient means for the target to monitor Tap packets received from each initiator so as to determine if the Tap packet would require use of a Tap Slot beyond the number of Tap Slots dedicated to that initiator. The grant of an initiator Identifier occurs as a result of successful completion of a log-in procedure by the initiator at the given target. Refer to the section entitled, "Log-In Protocol" for details.

5.1.1.3. Request for Grant of Dedicated Tap Slots

In order to receive an allocation of Tap Slots granted for its exclusive use, the initiator is required to make use of its initiator Identifier. Refer to the section entitled, "Request/Release of a Tap Slot" for details. As a result of the successful completion of this protocol the initiator is provided a statement from the target of the number of Tap Slots reserved at that target for exclusive use by the initiator. Once a grant of exclusive use Tap Slot is made by the target, the given Tap Slot or set of Tap Slots remains dedicated to the initiator until explicit release of some or all of these Tap Slots is made by the initiator.

Thus, the normal situation is that a dedicated Tap Slot will be reused, possibly indefinitely, by multiple Tap packets coming from the given initiator. Making the same statement in different words, a dedicated Tap Slot, in the normal situation, will be reused by multiple command block chains originating from the same initiator.

5.1.1.4. Grant of a Tap Slot From a General Pool of Tap Slots

In order to make efficient use of target resources, the target shall maintain a general pool of all its Tap Slots which have not been allocated and therefore dedicated to a specific initiator. Each initiator is permitted to

obtain the temporary use of a Tap Slot from this general pool. The meaning of temporary in this context is that the grant of the Tap Slot is limited for the duration of a single command block chain. Once, all command blocks within the chain have been completed by the target (and completion status returned for each of these command blocks), then the target shall return that Tap Slot to the general pool.

An initiator gains temporary use of a Tap Slot when it gains acceptance by the target of a Tap packet in the situation that all of the Tap Slots reserved for that initiator are presently in use. As a special emphasis point, since all of the Tap Slots dedicated to this initiator are in use, the target is "allowed" to reject the new Tap packet.

5.1.2. Setting of the M_Flag

The mechanism for forming commands into a chain makes use of the field in the command block which is the address of the next command. Thus, a chain consists of those commands connected to one another by an initiator (or collection of initiators) through the use of a forward pointer consisting of the next command address field found in each command block. Each command block also contains a flag, called the More_Flag, or more briefly the M_FLAG. When the M_FLAG has value equal to one, it means there is at least one command block in the chain occurring after this present command block. The very last command block in the chain is indicated by having value equal to zero for the M_FLAG. The initiator preserves for later use the address of the next command block which is to be fetched from the given chain.

When a chain has been made known to the target, and before the first command block has been fetched from the chain, the address stored by the target is address of the head of the command block chain in initiator memory space. It is convenient to refer to the starting address of a command block chain as being the address of the first command block in that chain.

5.1.3. Setting of the ESC_Flag

The mechanism for control by the initiator of the command block fetch policy used by the target is by means of the ESC_Flag. Once a target fetches a command block from a given chain, the default fetch policy is for the target to continue fetching additional command blocks from the same chain until: (a) the end of the given chain has been reached, or (b) the target encounters a command block in which the ESC_Flag has been set to value one. When the ESC_Flag has value equal one, the initiator is providing explicit statement that the target is to fetch the next command block from some other chain if there are other chains presently identified to that target. If there are no other chains known to the target, then the next command block would be fetched from the present chain if there are still unfetched command blocks within the subject command block chain.

It is useful to consider an alternate explanation and interpretation for the significance of setting the ESC_Flag to value one within a given command block. The implications on fetch policy employed by the target are the same. The alternative explanation provides useful perspective on the nature of the implied time stamp associated with each command block chain.

Each command block chain has an implied time stamp based on the relative order of arrival of Tap packets at the target. Unless use is made of the ESC_Flag, each and every command block within a given chain has the same time stamp value; namely, the implied time stamp given to the chain via the arrival time of the Tap packet. Additionally, chains are to have command blocks fetched from them in the order specified by the implied stamp. Thus, the chain with lowest implied time stamp has command blocks fetched from it and before command blocks are fetched from chains with higher value of implied time stamp.

As a consequence of setting the ESC_Flag to value one within a command block, the implied time stamp is advanced to the present time for any and all command blocks remaining within the given chain. Thus, the command blocks are to be viewed as existing within a new chain, a sub-chain, and this sub-chain now has an implied time stamp later than the implied time stamp of all other chains known to the target. These remaining command blocks each have the same time stamp just assigned to the sub-chain. Since this time stamp is the latest value of all time stamps, then the new sub-chain will have its command blocks fetched

after the command blocks from every other chain known to the target. It should be observed, there is no restriction on the number of command blocks which may have their ESC_Flag set to one. Should other command blocks have their ESC_Flag set to value one, then remaining command blocks from those chains would have implied time stamp advanced to a value later than the subject sub-chain.

5.1.4. Setting of the L_Flag

A redundant indication for a Linked command is given within the command block by means of the Link_Flag, or more briefly the L_Flag. In compatibility with SCSI 2, the presence of a Linked command must be indicated by appropriate setting of the Linked bit within the SCSI 2 Command Descriptor Block (CDB). However, if the CDB were providing the only means to identify the presence of a Linked command, then every CDB would have to be interpreted in sufficient detail to determine if a Linked command were present before the target could decide upon the required course of action relative to order of completion among the possibly multiple commands which may have been fetched by the target. Thus, as a matter of convenience, the target has means to identify a Linked command based on examination of the L_Flag within the Serial Bus Protocol command block.

5.1.5. Focus of a Command Chain

Per the Serial Bus Protocol, a wide range of choices is given to initiator as to the relationship between commands in a chain, and the various Logical Units (LUNs) supported by a given target. In one permitted alternative, the same chain may contain command blocks intended for different LUNs supported by the same target. In another permitted alternative, the initiator may organize a chain so that it contains commands intended for only one LUN. As a still additional permitted alternative, the initiator may direct multiple different chains to the same LUN. In all alternatives, every command within the same chain must be directed to the same target.

A target may choose to implement the facility in which multiple command blocks have been fetched, and these command blocks may have come from different chains and even from different initiators. In conformity to the notion of Task Set within the SCSI 3 Queueing Model, the target which has fetched multiple command blocks must be able to develop different lists drawn from the present population of commands presently maintained in target working storage. It is necessary for the target to identify all fetched commands associated with a given LUN. It is also necessary for the target to identify all fetched commands associated with a given initiator. The most simple case from the perspective of the target is that it fetches and maintains in working storage one and only one command block at any point in time.

5.1.6. Composition of a Command Chain

The following four types of I/O process are defined in the SCSI 3 queueing model:

1. Simple
2. Ordered
3. Head of Queue
4. Autocontingent Allegiance (ACA)

Additionally, subject to the restriction described below, a collection of SCSI Linked commands may be included within a command block chain. Per the SCSI 3 queueing model, a given collection of linked commands is considered as a single I/O process such that each of the linked elements must be completed by the target in the order specified by the initiator, from start to finish and without interruption by other I/O processes.

With the obvious exception of the ACA I/O process, any of the above types of I/O process drawn from the list above may be included within the same command block chain. In the following three subsections:

(a) Recommendations for Ordered commands,

(b) Recommendations for Head of Queue commands, and

(c) Requirements for Processing Linked commands, alternatives and implications are considered regarding processing of these types of commands.

The Urgent FIFO is a supported alternative to send a message to the target in order to provide the initiator with a processing alternative for command blocks which need to be fetched in a special or in a priority manner relative to the fetching of command blocks associated with a chain referenced to the Normal FIFO.

If the initiator can accept a modest level of loss of control over the time sequence of processing for Ordered commands and for Head of Queue type commands, these two types of commands may be placed by the initiator at any location within one of its command block chains referenced to a Tap packet sent to the Normal FIFO. Linked type commands be must placed only as the last element of a command block chain. A significant reason for this restriction on placement of Linked commands in a command block chain follows from the difficulty faced by the target in making acceptable recovery from execution time failure of a Linked command in a sequence of Linked commands.

5.1.6.1. Recommendations for Ordered Commands

If there exists a sequence of Simple I/O commands which are to be completed before a sequence of one or more Ordered commands,then at least two alternatives are available to the initiator or initiators interested in the situation. In the discussion of these two alternatives, it is assumed that no initiator issues a Head of Queue command. It is further assumed that no check condition occurs so that no Contingent Allegiance Condition is declared.

The first alternative is probably best suited to the case in which only one initiator is relevant to the situation. The initiator creates a chain in which the given Ordered command is placed in the correct position within the chain relative to Simple commands which are to be completed prior to the Ordered command. Once the chain has been entered via a fetch of the first command block, all subsequent fetches by the target are required to be from the same chain as long as each of the fetched command blocks has value zero for the ESC_Flag. This is to say, the initiator constructing the given chain has it within its own ability to control fetch policy by the target so that the given chain is processed in the manner considered appropriate by that initiator.

Alternative two is applicable to the case in which multiple command block chains are present and the completion of the given Ordered command has significance relative to commands blocks from two or more of these chains. In alternative two the interested initiator or initiators construct multiple chains such that Simple commands are exclusively placed in one set of chains and the given Ordered command is placed in a chain of its own. Tap packets for these several chains are sent in an appropriate time sequence such that the desired relative order of arrival (and therefore implied stamps in the desired order) is achieved at the given target.

5.1.6.2. Recommendations for Head of Queue Commands

In concept, the same two alternatives for dealing with Ordered commands also exist for dealing with Head of Queue commands. However, the notion of time stamp does not really have application to processing by the target of the Head of Queue command. The obligation of the target is to place the subject command at the head of the queue maintained by the target for those commands which have been fetched but not yet completed. There is the additional complexity that: (a) targets may differ with regard to how many command blocks are prefetched beyond the command block presently in execution, and (b) the number of command blocks within the prefetch set which are decoded with regard to queue type. Thus, when a Head of Queue command is placed in a general command chain along with other command blocks, there are many aspects of uncertainty as to when the Head of Queue command is actually advanced to the head of the queue. These uncertainties include the point in time when the command is fetched and the point in time when the command is decoded and thereby recognized.

Thus, from the perspective of the initiator, it is most important to have more control over the point in time that the target makes the determination that a Head of Queue command has been sent by some initiator. This objective of the initiator is best accomplished by an additional alternate, namely alternative 3. In alternative 3 a Tap packet message is sent to the target's Urgent FIFO. In principle, any type of I/O process can be included within the command block chain referenced by the Tap packet message sent to the target's Urgent FIFO. It is strongly recommended that the facilities of a Tap packet sent to the Urgent FIFO are best used when the referenced chain contains only a Head of Queue command. To a much lesser extent a case can be made that it is acceptable to included Ordered commands within a chain referenced to the Urgent FIFO. In selected instances it may also be appropriate to include certain Isochronous commands in the command block chain referenced to the Urgent FIFO. Inclusion of all other types of command in the chain referenced to the Urgent FIFO should be considered inappropriate and an abuse of this facility.

The alternative in which a Tap packet message is sent to the Urgent FIFO informs the target of a priority matter needing immediate treatment relative to the procedure for fetching commands blocks. As a special emphasis point, the priority processing associated with the Urgent FIFO is focused on fetching command blocks and not upon executing them. The same order of execution rules apply to command blocks whether these commands are referenced to the Urgent FIFO or to the Normal FIFO. When a target receives the Tap packet in the Urgent FIFO, the target has the obligation to begin fetching commands from the referenced command block chain as soon as possible. The target must fetch command blocks from this chain before it fetches any command from any chain referenced to he Normal FIFO. Once command blocks are fetched from the chain referenced to the Urgent FIFO, normal processing rules apply as determined by the queue type associated to each of the fetched commands. In the case of the Head of Queue command, the target is required to place this command at the head of its queue. The target is allowed to apply its own optimization rules as to when it should complete execution of a command with queue type equal to Simple independent of whether that command was fetched from a command chain referenced to the Urgent FIFO. Implied time stamp dictates the order of execution for a command with queue type equal to Ordered even though that command is referenced to the Urgent FIFO.

The facilities of the Urgent FIFO are well adapted for fetching and then completing execution of a command block carrying a Head of Queue type command. The Tap packet message sent to the Urgent FIFO reveals to the target the existence of a Head of Queue command in a direct fashion compared to the alternative in which a command block is fetched and no knowledge of the type command is gained until that command block is decoded.

5.1.6.3. Requirements for Processing Linked Commands

A collection of Linked commands may be included within a command block chain if the subject Linked I/O process is the very last I/O process within the chain. Additionally, the L_Flag must be set to value one for every Linked command except the final Linked command in the collection. For this final Linked command, the appropriate bit in the CDB must indicate this to be the ending command in the Linked set. Additionally, both the L_Flag and the M_Flag in the command block must be set each to value zero.

5.1.7. Retention of a Command Block by the Initiator

The initiator must be aware that the target may have legitimate need to refetch the same command block at any arbitrary time after the time of original fetch up to the time of return of completion status and/or sense data. One reason for a refetch is that a target may elect to take advantage of the option in which ending portions of the command block are read only when a check condition occurs. A second instance of refetching a command is the optimization policy in which many command blocks are read and then discarded in order to determine which command is best to execute next so as to achieve best performance. Some target devices may even elect to read less than the full portion of a command block in order to determine whether the given command is a suitable candidate for execution via this optimization scheme. The command selected for execution is then refetched. At the time when a command is in fact to be put into execution by the target/LUN, at a minimum, the contents of the baseline portion of the command block must be fetched by the target/LUN.

Thus, the initiator must maintain the command block in memory at the same location as originally stated to the target. The initiator must assume full consequences for any change in contents of the command block from the time of first fetch of that block until the time of any later fetch. Aborting a command is an example of a situation in which the initiator may feel it has a proper and good reason for altering some part of a command block after it has been fetched and before return of completion status information.

5.2. Model of Serial SCSI Target

The target is responsible for fetching command blocks, one at a time from the various initiators having indicated to it there is work to be performed on their behalf. Upon completion of any given command, the target returns completion status information back to the appropriate initiator. The addressing scheme of IEEE 1394 is sufficiently flexible that specification can be made for the target to return Status information to a different initiator than originated the command block.

The general notion for a target is that it follows orders as provided by the initiator. In this context, flags, such as the L_FLag and the ESC_Flag, are provided to the initiator so that appropriate control may be exercised over the fetch policy of command blocks by the target. Additionally, there is a log-in and a sign-in process in which initiators identify themselves as well as their special needs which require service by the target. Thus, initiators advise the target if notification is to be given them regarding unit attention conditions arising at the target. As the key part of the log-in process there is assignment of a special identifier to initiators by the target so that the target may conveniently make distinctions among the initiators. Maintenance of a guaranteed minimum number of Tap slots for a given initiator is a very important case of making distinctions among the various initiators.

Since a given initiator may have multiple commands fetched by a target and not yet completed, the target must provide correlation information between command blocks and status blocks. The address of the associated command block shall be placed in the status block in order to provide this needed correlation.

Conceptually, there is a one to one relation between Sense Data blocks and Command blocks. Also, there is also a one to one conceptual relation between a Status block and its associated Command block. Sense Data blocks are most often written by the target to general purpose memory within the initiator address space. Status blocks may be written by the target to special purpose hardware provided in support of the Status FIFO. While a Sense Data block is not always sent for each and every command completing with "good status", there is the notion of allocating a Sense Data block area for each and every command.

Editorial Note: Based upon discussion within the SCSI Committee, it was determined to be necessary to maintain full generality in the specification of starting address for the Sense Data buffer within the initiator. As a result, the Sense Data buffer start location is specified by means of a full 64-bit address. The notion of a Sense Data Buffer Offset has been removed from this document.

5.2.1. Usage of the Status FIFO

For purposes of this Serial Bus Protocol, status information is sent by the target to a Status FIFO within initiator address space. Many implementation choices are available as to hardware and/or software means to support the Status FIFO data structure. Observe, at the conceptual level, the Status FIFO and the Command FIFO set (Normal FIFO and Urgent FIFO) have many notions in common. A key element common to both the Status FIFO in the initiator and the Command FIFO set in the target is that the same size packet (12 Bytes of payload) is sent to both. This is not to say that any requirement exists to provide the same hardware implementation for the Command FIFO set and the Status FIFO. However, if specialized hardware support is provided for each FIFO it can involve the same hardware design. One useful result of the same hardware treatment for each FIFO is that it becomes more convenient for an initiator with its Status FIFO to use the same equipment as a Command FIFO set and thereby be able (at the hardware level) to also function as a target.

5.2.2. Usage of the L_Flag

As processing begins for a fetched command block, the target must examine the L_Flag to determine if the present command is a SCSI Linked command. Should the L_Flag have value to one, then the given command is a Linked command and there is no relevant meaning to either the M_Flag or to the ESC_Flag. Special processing and fetch policy rules exclusively apply to the collection of Linked commands.

It is mandatory within the Serial bus Protocol for the L_Flag to be supported by the target device. As per SCSI 2, it is optional for the target to support Linked commands. If Linked commands are not supported by the target, then is returned to the initiator as described by SCSI-2.

If SCSI Linked commands are supported by the target, then processing per the Serial Bus Protocol should not preclude emulation of the same functional capabilities for Linked commands as described within SCSI 2. Each SCSI command within the Linked set is represented by its own Serial Bus Protocol command block. With the exception of the last command block in the Linked set, the Next Command Address field within command block N points to the start address of command block (N+1). For the final command block in the Linked set, the L_Flag has value zero.

As per the processing rules within SCSI 2 for Linked commands, no attempt at overlap processing is permitted between command block (N+1) and command Block N given that each of these are Linked commands. Before any attempt is made by the target to fetch command block (N+1), it is necessary for both command block N to be completed as well as return made to the initiator of the Status block for command block N. SCSI 2 permits the initiator to select dynamically the next Linked command for execution based on the result of completion for Linked command N. Should the initiator choose to take advantage of the dynamic capability, it would need to modify the contents of command block (N+1) prior to return of it in reaction to a IEEE 1394 Request Request from the target.

Implementor Note: The initiator may choose to take advantage of the IEEE 1394 facility of the Split Transaction as an aid to modifying contents of command block (N+1) based upon completion status for command block N. With the Split Transaction, the initiator has up to 100 milliseconds before it must make a Read Response containing the contents of command block (N+1) in reaction to the Read Request from the target for this command block.

Upon successful completion of the last Linked command in a Linked set, the target is to consider the given command block chain to be concluded. The Next Command Address field is not to be considered valid for this final Linked command. Neither is it necessary for the target to check the value of the M_Flag or the ESC_Flag for this final Linked command.

5.2.3. Usage of the M_Flag

When a target reads a command block and determines it does not contain a Linked command, then it must examine the M_Flag. If the M_FLAG is set to value equal zero, then the initiator has no further commands for the target. If the flag has value equal one, then further commands within the given chain are waiting for the target. In this case the Next Command Address field will contain the address from which the target should read the next command. Note that this quantity is a full 64-bit address per the format specified within the IEEE 1394 standard. Consequently, the command could be physically located in any device attached to the IEEE 1394 bus.

5.2.4.Usage of the ESC_Flag

The ESC_Flag has meaning only once it is determined that that the present command is not a Linked command (L_Flag equals zero) and there are more commands still to be fetched from this chain (M_Flag equals one). If per the above rules it is appropriate to check the ESC_Flag, then the following reactions are required of the target. If the ESC_Flag has value zero, then the default fetch policy applies and the target is to obtain the next command block from the same chain based on the value of the Next Command address field.

If the ESC_Flag has value one, then the present command block marks the end of one sub-chain and the next command block (within this chain) marks the start of a new sub-chain. The target is to advance the implied time stamp of the new sub-chain to be the present time. In consequence of this advance of implied time stamp, the new sub-chain is to be placed at the end of the list maintained by the target to describe the order in which command blocks are to be fetched from command block chains. This is to say, command blocks are to be fetched (in time sequence order) from all other chains before any command block is to be fetched from the new sub-chain.

5.2.5. Management of Target Resources

It is important to recognize there are two important and potentially very different resources for the target to manage. One resource is associated with processing a "Shoulder Tap" sent by the initiator to the target's Command FIFO. The Shoulder Tap may be sent at any arbitrary time with no prior warning given to the target. Potentially, a sequence of Shoulder Taps may occur, one immediately after another, in a "rapid fire" scenario. In such a rapid fire case, there may not be time for the target to do anything with the 12-Byte Tap packets other than store them. It is crucial that whenever a Tap packet is acknowledged as having been accepted by the target, the contents of that Tap packet are safely stored and available for later use by that target. Hardware and software support must be provided as necessary to ensure that the target be able to accept a stated minimum number of shoulder taps at any point during processing of some command or at any point in any other activity undertaken by the target. It is convenient to use the term "Tap Slot" when dealing with the obligations associated with accepting a Shoulder Tap. The requirement placed upon the target is to support an architected minimum number of Shoulder Taps which must be supported by an target compliant to the Serial Bus Protocol.

The second resource which must be managed by the target is the storage necessary to hold a command block. In managing this second resource it is most significant that command blocks do not come at unexpected points in time. Each command block is specifically requested by the target. Thus the target can ensure that sufficient storage is available. The target can also make sure the request for a new command block is only made at a favorable time in the processing activity currently underway.

Management of these two resources is likely to proceed by substantially different means as discussed below. One technical issue with management of the Tap Slot resource is the minimum number of shoulder taps which can be guaranteed to exist at any target. A key point regarding a Tap Slot is the time duration it is kept in a committed state dedicated to one given chain. With regard to storage of command blocks, some questions are: how many can be prefetched, how much of the command block must be fetched, and the policy for fetching blocks from different chains.

5.2.5.1. Management of the Tap Slot Resource

The Command Delivery protocol does not require any notice to be given by the target to the initiator concerning completion of processing for the chain. Nor does this protocol require the target to give any notice when it can make free a Tap Slot. Should a Tap Slot be a resource limiting the capability of the target and/or initiators using that target, then it can be argued that the target should make this item available for reuse at the earliest possible time. On the other hand, if active Tap Slots become too numerous, it may mean the target has to manage an excessive number of commands chains, thereby providing a severe challenge to the management of the resource for storing command blocks fetched by that target.

The very earliest point in time that a Tap slot can be released for use by another chain is immediately after the contents of the Tap Packet have been copied to the general storage pool available to the target. Observe, this earliest time is prior to actual fetch of the first command block from the initiator to the target. A somewhat later point in time to release the Tap Slot is immediately after the fetch of the very first command block in the chain. At this point (first command block stored by the target), all necessary information is available to the target in order to proceed down the chain. The very latest point in time for the target to release a Tap Slot is when the last command block has been completed and all necessary status information has been returned back to the initiator. While still later release times for a Tap Slot are possible,

they would seem to add undue delay in making free a valuable resource. Clearly, the target can choose to release a Tap Slot at any time between the above earliest time and the above latest time.

The advantage in making an early release of a Tap Slot is that it makes the hold time for the Tap Slot resource less than the hold time for the command block storage resource. In effect, the early release policy for Tap Slots serves to increase the effective number of Tap slots relative to the number of command chains which can be serviced by the target.

Editorial Note: Consideration is being given to provision of an optional facility whereby a target does advise the initiator as to the number of Tap Slots available to the initiator relative to the number of Tap Slots dedicated to the given initiator at that target. The vehicle for advising the initiator is the Status block sent upon completion of each command within a command block chain. The SBP Status Byte within this block is to carry the Tap Slot information. As a special emphasis point, the target is permitted to maintain a Tap Slot in the "in-use" state until there is both completion and return of completion status for each and every command block within the chain. In a permitted alternative, the target may utilize an implementation in which the Tap Slot associated with a command block chain is put into the "available for use" state at some point in time earlier than completion and return of completion status for all command blocks within the chain. As an additional emphasis point, there is no requirement to that a target inform the initiator when a Tap Slot has become available should it become available prior to completion of the command block chain.

The Serial Bus Protocol does not place any requirement upon the target to conform to any specific policy for release time management of Tap Slots. Such release time policy is left to the target as an implementation consideration. The default policy for release of a Tap Slot is that the Tap Slot does not enter the "available for use" state until completion and return of completion status of each and every command block within the associated command block chain. If no specific advisement is made to the contrary, an initiator is to understand that the above default policy has been applied by a target relative to when a Tap Slot enters the "available for use" state.

Tap Slots Dedicated for Exclusive Use by an Initiator: An initiator is given the facility to request the target to grant to it some number of Tap Sots which shall be reserved by the target for the exclusive use of that initiator. It is required:

(a) that the initiator has completed a log-in procedure in which an initiator Identifier is assigned to the initiator by the target, and

(b) the initiator make use of that initiator Identifier for the purpose of securing or releasing some number of Tap Slots dedicated to its exclusive use.

Once the initiator has received both an initiator Identifier and an allocation of Tap Slots, that initiator may send Tap packets to the given target.

As the target receives Tap packets from the initiator, the target is required to accept the Tap packets up to the point that the target is making use of all of its dedicated Tap Slots. Each Tap packet consumes one and only one Tap Slot. A Tap Slot must be returned for use to the initiator at a point in time no later the time in which all commands in the associated chain have been fetched, completed, and have had completion status returned for them. Optionally, the target may choose to release for reuse a Tap Slot at a point in time prior to completion of all required processing and status return for all of the associated command blocks.

As a convenience to the target, the initiator Identifier is required to be used by the initiator on all Tap packets sent to the target. The target is to use this Identifier so as to monitor the requests for Tap Slots by the initiator. Repeated for emphasis, each Tap packet accepted by the target consumes a Tap Slot at that target. Once the initiator has its number of dedicated Tap Slots placed into the "in-use" state, then that initiator must wait to send a new Tap packet until it has received completion status for every command block in at least one of its chains if that initiator expects the new Tap packet to be accepted without question by the target.

Use of Tap Slots from the General Pool of Tap Slots: It is considered desirable to make efficient use of all Tap Slots available at a target. At one extreme, if only one initiator is involved with a target, then all of the Tap Slots should be made available to that single initiator. It is also desirable to share among all interested initiators, all of the Tap Slots provided at the target. In order to secure these objectives, the total set of Tap Slots at a target are dived into two groups:

(a) the collection of all Tap Slots dedicated to initiators into the set of Dedicated Tap Slots, and

(b) the collection of all other Tap Slots into the General Pool of Tap Slots.

As Tap packets are received from initiators, these Tap packets are first assigned to Tap Slots dedicated to the individual initiators. This assignment to the dedicated Tap Slots continues up to and including the point at which all available Dedicated Tap Slots are assigned. Should a Tap packet be received from an initiator such that all of its dedicated Tap Slots are "in-use", then the target attempts to make use of a Tap Slot from the General Pool of Tap Slots. If such an assignment beyond the dedicated number of Tap Slots is made, then those extra Tap Slots are assigned only for the duration of use for a single command block chain. When all command blocks from a chain have been fetched, executed, and return made of completion status, then that temporary Tap Slot is returned to the General Pool of Tap Slots.

It may occur that a new initiator makes a request for assignment of dedicated Tap Slots when there are in fact Tap slots existing in the General Pool of Tap Slots. In principle, the requesting initiator should have all or part of its request for Tap Slots satisfied, up to the point at which no Tap Slots remain in the General Pool of Tap Slots. In a specific instance in which a request for Tap Slots is made, it may occur that the Tap Slots in the General Pool of Tap Slots have all been given temporary assignment to other initiators. Thus, in this specific instance, while the request for Tap Slots can ultimately be satisfied, it may not be satisfied for some period of time until which the needed Tap slots are made free from their temporary assignment to some other initiator. Since command block chains may be of any length, it may occur that the ability of the target to honor the request for Tap Slots is delayed for some indefinite period of time; perhaps, a period of time longer than the relevant time-out interval for the requesting initiator.

Editorial Note: Concern has been expressed as to adverse impacts on the initiator in the situation in which a request for Tap Slots which should be satisfied is in fact delayed for an indefinite period of time. The notion is that initiators have expectations as to reaction which should be made by a target based upon published specification of functional capabilities at that target. The concern is that when these Initialization time requests, such as request for Tap Slots, cannot be satisfied in a predictable and understandable manner, then unacceptable problems occur from the perspective of the initiator. comments on this point are requested.

One alternative which has been suggested is the fixed allocation of Tap Slots into one pool of Tap slots available only for dedicated use, and into another pool of Tap Slots available for temporary and nondedicated use. While this alternative has the opportunity to make predictable the satisfaction of Tap Slot allocation requests from initiators, there is the problem that some number of Tap Slots in the dedicated Pool may go unused if there fewer than the expected number of initiators making requests for Tap Slot allocation.

5.2.5.2. Management of the Command Block Storage Resource

The Command Delivery protocol does not require a target to fetch multiple command blocks for purpose of storing them so as to attempt overlap of processing or performance optimization. A logically correct and internally consistent implementation is to fetch and process one command block up to the point of return of completion status before any effort is made to fetch another command block. Thus, even when the architected minimum number of 32 Tap slots are made active by the various initiators, the target is allowed to process one and only one command block at a time, and process it to completion before fetching some other command block. A more interesting case, is for the target to fetch multiple command blocks so that

there is overlap in the processing of a command in execution phase with other commands for which pre-processing is underway prior to entry into execution phase.

5.2.5.3. Schedule Policy for Fetching Among Multiple Command Chains

The Tap Packet (SCSI Command Initiation packet) is given an implied time stamp by the target. This implied time stamp is the relative order of arrival of each Tap packet at the given target. The Tap packet is associated with one and only one command block chain. Each command block in that chain is initially given the same value of implied time stamp as was given to the Tap packet. Thus, information exists for the target to comply with the order of execution rules established by the SCSI 3 Queueing model for treatment of Simple, Ordered, Head of Queue, and ACA I/O processes.

In the situation in which an Ordered chain is present along with chains of Simple I/O processes the following command fetch and command completion policy is required. All Simple commands with lower value of implied time stamp must be fetched and completed prior to start of an Ordered I/O process with higher (later) value of implied time stamp. No fetch or completion of Simple I/O processes is undertaken for those command blocks within a chain having an implied time stamp is higher (later) than the time stamp of an Ordered chain. It is convenient to give the name "deferred chains" to these chains having a higher (later) implied stamp than some Ordered command chain. Once completion is achieved for each command in the subject Ordered chain, it is no longer necessary to defer fetching and completion for commands in the previously named "deferred chains".

Within any chain, all of the included I/O processes have the same implied time stamp as dictated by the arrival at the target of the associated Tap packet. Thus, for all chains of Simple commands with lower time stamp than some ordered command, every command in such a Simple chain will be fetched and subsequently completed prior to any attempt at completion for the subject ordered command.

Before further discussion on fetch policy and completion policy for chains of Simple Commands, it is necessary to consider the case of Head of Queue chains, and also the case of ACA chains. The ACA chain comes into existence only upon an ACA condition having been established by a target/LUN. As such, the ACA chain becomes the only chain from which commands can be fetched and completed for the associated target/LUN. The ACA condition is ended upon completion of the ACA command chain. In a somewhat similar fashion, when a Head of Queue chain is accepted by a target/LUN (subject to no ACA condition existing), that the Head of Queue chain becomes the very next chain from which an I/O process is fetched. Once fetched from a Head of Queue command chain, the present Head of Queue command becomes the very next command to be completed relative to all other non-ACA commands fetched but not yet completed by the target/LUN. Within a Head of queue chain, the last command within the chain has precedence over all earlier commands in the Head of queue chain. The first command within a Head of queue chain has lowest precedence relative to all (later) commands within the Head of Queue chain.

Now it is appropriate to return back to the case of fetching I/O processes from chains of Simple commands. It is required that fetching is allowed from these chains of Simple commands as per the above policy of dealing with Ordered commands, Head of Queue commands, and ACA commands. For convenience, refer to this set of Chains of Simple commands for which fetching is allowed as being the "Allowed Set".

From this Allowed Set, the target proceeds from one command block chain to the next as specifically directed by the initiator. The default fetch policy is called "Queue Exhaustion" or more briefly "QE". In this default policy, the target proceeds from start of a chain to the end of the chain. The first chain selected is the chain having the lowest implied time stamp. Fetches (one at a time) of command blocks are made only from that chain until the last command block has been fetched. The first chain is then "exhausted" and fetches may begin from the chain having the next highest value of implied time stamp. In this default policy, fetches are made via the QE approach from the various chains; starting from the chain with the lowest value of implied time stamp and ending with the chain having the highest value of implied time stamp.

The initiator may cause the the target to fetch command blocks from a new chain before all command blocks are fetched from a given chain. The mechanism causing a change to a new chain is via the ESC_Flag. If the initiator sets the ESC_Flag equal to value one in some command block, then the target is required to fetch the next command block from a new chain (if an additional chain exists). The new chain is that chain having the next highest value of implied time stamp. Should only one chain be known to the target, then the next fetch would proceed from the same chain.

In the limiting case, each and every command block might have the ESC_Flag set equal to value one. In this situation, the fetch policy followed by the target is equivalent to a Round Robin fetch policy. The Round Robin fetch policy might be intended to ensure fairness in the treatment given to each chain when multiple chains of Simple commands are presented to a target.

The sequence of fetching one command block within a given chain relative to all other command blocks within that same chain is dictated by the initiator though the mechanism of the Next Command Address field. When a change is made so as to fetch command blocks from a new chain, the first command block from a new chain is fetched from the location recorded as being the current head of that chain. If no command blocks have been fetched from a chain, the head of the chain is the address of the first command block in the chain. If the given chain has had some of its command blocks previously fetched, the head of chain is the address found in the Next Address field in the command block within that chain having the ESC_Flag set equal to one.

Next, refer to the collection of Simple commands blocks which have in fact been fetched by the target/LUN as the "Fetched Set". Observe, entry into the "Fetched Set" is exclusively on the basis of the policy for fetching commands from multiple chains of commands. Once a command is within the Fetched Set, it is left to the target to determine the appropriate order of execution. No requirement is placed by the Serial Bus Protocol as to preferred order of execution for commands within the "Fetched Set" of Simple commands. Thus, the target is free to execute the commands in the same order as fetched or in some other order if advantage is felt to be gained.

As a specific emphasis point, it is allowed for a low-end target to fetch and complete commands on a one at a time basis. Should the "Fetched Set" be restricted by target implementation to consist of only one command, the requirement continues for that single command to be determined by application of the fetch policy commanded by the initiator through the means of the ESC_Flag. It is encouraged for targets to pursue a more advanced implementation such that the "Fetched Set" is allowed to be larger (perhaps much larger) than one command.

5.2.5.4. Fetch of Command Blocks From a Chain Referenced to the Urgent FIFO

Provision is made for an initiator to send a Tap packet message to the Urgent FIFO at the target. When this Tap packet is received at the target, then the target is required to begin fetching commands blocks from this chain as soon as possible. The target is allowed to complete a command currently in execution. Commands from the chain referenced by the Urgent FIFO must be fetched before command blocks of any chain referenced to the Normal FIFO. If there are multiple active chains each of which is referenced to the Urgent FIFO, then the fetch policy is that the subject chains must be served in the order of their arrival at the target. Thus, the first arriving chain referenced to the Urgent FIFO must have all of its command blocks fetched before the effort is made to fetch command blocks from a later arriving chain referenced to the Urgent FIFO.

Once command blocks have been fetched by the target, the order of execution for these command blocks is determined by:

(a) the queue type of the given command, and

(b) the optimization rules employed by the target.

Thus, if a given command has the queue type equal to Head of Queue, there is no choice allowed the target. The Head of Queue command must be placed at the head of the target queue. At the other end of the spectrum, if the fetched command block has the queue type equal Simple, then the target is free to apply its own optimization rules for order of execution for Simple commands independent of whether they are referenced to the Urgent FIFO or to the Normal FIFO.

5.2.5.5. Repeated Access to Same Command Block

The target is allowed the option to fetch the same command block more than one time prior to completion of the command and return of status information and/or sense data. Should the target elect to fetch a given command block more than one time, the requirement exists that the same contents would obtained by the target upon each of the fetch operations toward the same command block. At least one exception is allowed to the general notion that a target may refetch a given command block at any time, and expect to see the same contents, field by field, being read each time. This exception is when the initiator elects to abort the command prior to having received notification of completion for that command. In this exception circumstance, the target is allowed to change such field as needed in order to satisfactorily mark the command block as having been aborted.

5.2.5.6. Support of Autosense

The Serial Bus Protocol supports the model of a target device in which automatic return of sense data to the target is enabled. A flag is provided in the command block provides a means for the initiator to inform the target on a command by command basis whether autosense is enabled or disabled.

6. Command Transfer Protocol

6.1. Conceptual Initiator - Target Connection

Example 1 below provides a sequence of IEEE 1394 packets which illustrate the operation of the Command Delivery protocol used by the SCSI 3 Serial Bus Protocol. The example depicts the situation within a single initiator such that a chain has been formed consisting of three commands. The commands are referred to as Command 1 (first command), Command 2, and Command 3 (last command). Only Command 3 has the M_FLAG set equal to value zero since it is the last command in the chain.

---

initiator	target
Tap on shoulder	------> Receive Tap Packet
	<----- Tap Acknowledgement
**********	IEEE 1394 Transaction
	<------ Read Request (For a Command Block)
Command 1	------> Read Response (Supply Command Block)
(M_Flag = 1)	
**********	IEEE 1394 Transaction
	<------ Read Request (For a Command Block)
Command 2	------> Read Response (Supply Command Block)
(M_Flag = 1)	
**********	IEEE 1394 Transaction
	<------ Read Request (For a Command Block)
Command 3	------> Read Response (Supply Command Block)
(M_Flag = 0)	

---Figure 1. Conceptual Initiator-Target Conversation, Example 1

The initiator sends the target a SCSI Command Initiation Packet, more informally known as a "Tap" packet. This "Tap" packet is the mechanism by which the target is informed as to the existence of a chain consisting of one or more commands and representing "work" to be done by the target on behalf of the initiator. The Tap packet is called the "Tap on the Shoulder" in the figure and contains the address of Command 1 within the initiator address space. The target sends an acknowledgement to the initiator that the Tap correctly received (CRC tests passed) and that the payload has been stored in an appropriate manner in a available Tap slot. In an alternative scenario, the target may not have a free Tap slot and would therefor have to reply

that the the payload of the Tap packet could not be accepted. The combination of Tap on Shoulder and associated Acknowledgement constitute one IEEE 1394 Write Transaction.

As shown in the figure, the target sends to the initiator a Read Request packet containing the address of Command 1 as provided within the "Tap" packet. The initiator sends a Read Response Packet to target with payload consisting of the command block associated with Command 1. The combination of Read Request and Read Response constitute an IEEE 1394 Read Transaction. For a list of the various IEEE 1394 packet types refer to 6.0, "Packet Types" on page 32. For information describing the format of these various types of packets refer to Appendix A, "Packet Formats" on page 84. For the detailed contents of the Command Block refer to 8.1, "Command Blocks" on page 41. For details of the contents of the SCSI Command Initiation Packet refer to 9.1, "Payload of SCSI Command Initiation Packet - "Tap Packet"" on page 54.

Before continuing with Example 1, it should be mentioned that in order to simplify the drawing, no information is provided regarding either data delivery or status delivery. The exclusive focus is upon providing an explanation of the command delivery mechanism. In this spirit, the explanation continues as to command delivery. The target is able to determine from the value of the M_FLAG within command block 1 that at least one and possibly more commands follow the present command (Command 1) within the given command chain. Each command has a field containing the address of the next command in the chain. At a time determined by the target to be appropriate, the target sends to the initiator another Read Request Packet containing the address of Command 2. The initiator provides another Read Response Packet now containing the command block associated with Command 2. As a major emphasis point, this Command Delivery Protocol leaves it to the target to decide when to fetch the next command. One choice which could be made by a low-end target is to wait until it completes Command 1 before it attempts to fetch command 2. Another choice is to fetch Command 2 prior to completion of Command 1 so that some degree of overlap processing may be achieved among the commands.

Eventually the target fetches a command (Command 3 in this example) which is the last command in the chain. Upon completion of Command 3, the target is to consider all required processing to be completed relative to the subject command chain. The Command Delivery protocol does not require any notice to be given by the target to the initiator concerning completion of processing for the chain. The only notice expected by the initiator from the target is associated with making normal and appropriate indication of end of processing for each command in the chain. In low-end targets, there would be some limit to the number of command chains which can be accommodated at any one time by that target. There would also be a limit on the number of Tap Slots. Upon completion of processing for one chain, the target has the ability to accept a new command chain, assuming that it was at the limit of ability to accept new chains prior to completion of the subject chain. The target makes an independent choice as to when it finished with a given Tap Slot and can return it to a "pool" of available Tap Slots. The target may elect to release a Tap Slot at the same time it releases (completes) the associated command chain. The target also has the privilege of releasing the Tap Slot at an earlier time than it releases the command chain.

Example 2 provides a second sample sequence for the command delivery protocol. The only difference between the situations depicted between the two examples is that example 2 explicitly shows the completion of one chain and the start of a new chain. The second chain is made known to the target by means a second shoulder tap. In the case shown, the first chain has been completed. At time after the end of chain 1, the second chain is created by the initiator and then the associated tap is sent to the target.

While not shown in Example 2, it is also possible for the same initiator to have two or more chains in existence at the same time and in processing by the same target. Thus, it is possible for the new chain to be made known to the target by means of a new tap prior to the end of processing being reached for some earlier chain.

---

initiator target

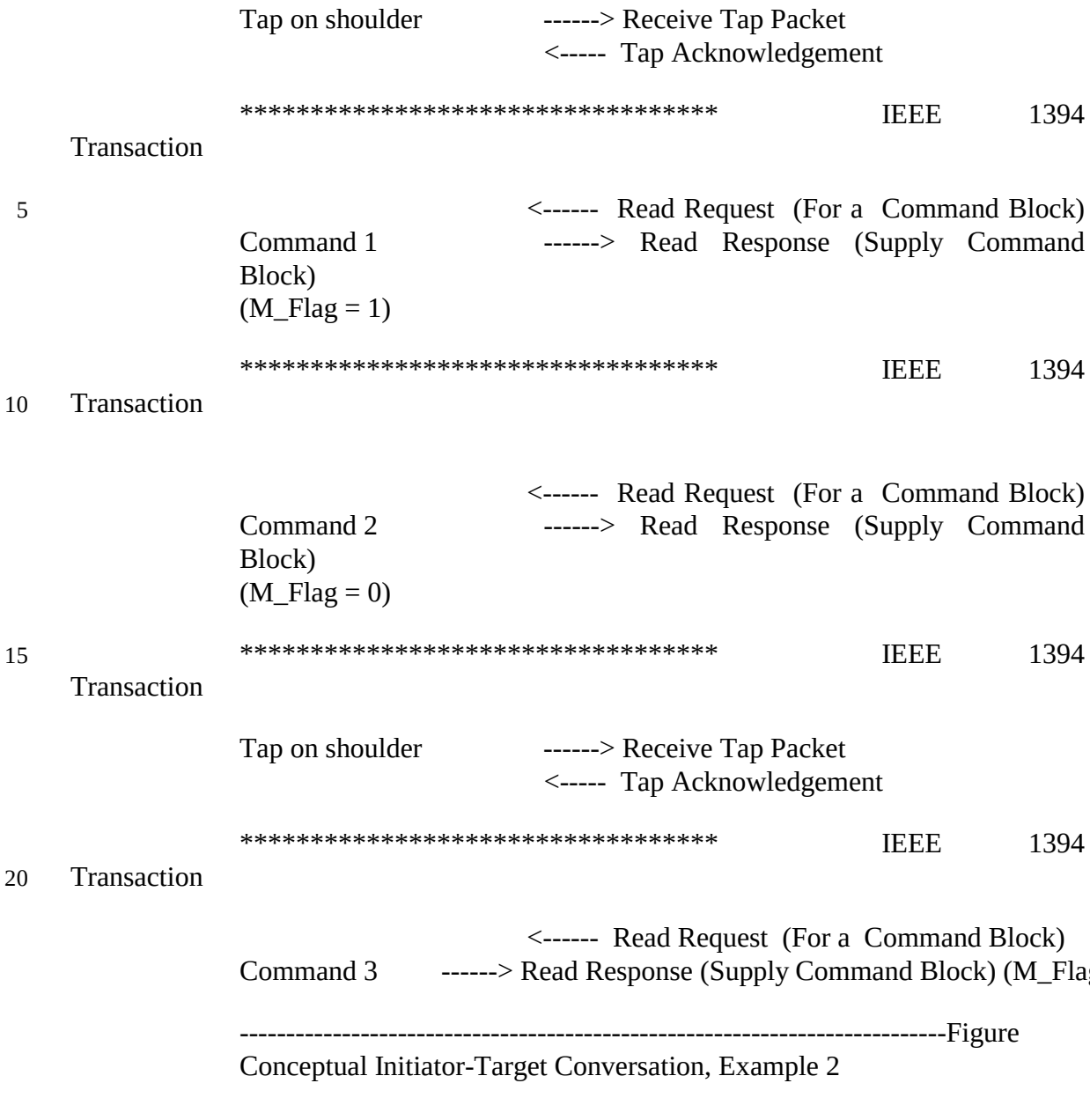


Figure 2. Conceptual Initiator-Target Conversation, Example 2

6.2. Multiple Initiator Environment

At any point in time, a "Tap" packet may be received by a target from any initiator connected to the bus. The target must accept this Tap unless the target already has reached its particular limit on the number of tap slots it supports. When a target receives a Tap, it must store the information passed in the Tap Packet. Provided within the Tap Packet is a 8-Byte 1394 style address of a command block and a 1-Byte Type_ID. Because this amount of information is relatively small, it is expected that even low-cost targets would be

able to support a relatively large number of Tap Slots, with multiple initiators. This document sets 32 as the minimum number of Taps that a target shall be able to process concurrently.

Editorial Note: The section "Conversation Packets" has been removed from the Serial Bus Protocol document during the transition to Revision Level One. The content of this section has been reassigned to other sections, mainly the sections describing support elements for the Serial Bus Protocol.

7. Packet Types

The IEEE 1394 High speed Serial Bus defines a number of packet types, such as read and write. These all have a header which identifies the type of the packet and its source and destination.

To implement the SCSI 3 Serial Bus Protocol, an initiator and a target need to understand and be able to create the following types of packets:

1. Quadlet Write Request
2. Block Write Request
3. Write Response
4. Quadlet Read Request
5. Block Read Request
6. Quadlet Read Response
7. Block Read Response

The SCSI 3 Serial Bus Protocol utilizes packet formats as defined by the IEEE 1394 High Speed Serial Bus standard. In recognition of ownership of packet formats by the named IEEE standard, no change in packet header or packet structure is considered within this document. Packet payload in the form of a command block and a status data block is the object of definition by this SCSI 3 Serial Bus Protocol document.

8. SCSI-3 Serial Bus Protocol Support Elements

Implementation of the SCSI 3 Serial Bus Protocol may proceed by various hardware or software means. The operational phase entails, conceptually, definition of a First In First Out (FIFO) data structure in the target and a counterpart FIFO data structure in the initiator. A key property associated with the concept of a FIFO is that when several data values are sent to the FIFO, each value is preserved by being stored within the structure rather than having the last value replace and write over all previously received values.

In many instances it may be convenient to think of the FIFO as having a hardware register and acting as an access port at the given address. It is beyond the scope of this document to suggest implementations. A particular implementation possibility may be used as an aid to providing a more concrete and therefore a more easily understood explanation of the protocol requirements. Likewise, in the configuration management and in the parameter control phases, additional data structures are defined within this document. Whenever the term register is used, this term is for convenience only and is not meant to suggest or to dictate a preferred implementation.

8.1. Target "Register" Definitions

As stated, the term register is used without prejudice to aid in the understanding of functional elements required within an implementation compliant to present SCSI 3 Serial Bus Protocol. Such functions as needed during operational, control, or configuration phases may be viewed as possessing associated "register" addresses. Such "register" addresses are specified within architected locations of Read Only Memory (ROM) located within the given target.

The address of a needed data structure (or register) may be viewed as a relocatable constant for devices implementing the Serial Bus Protocol. The notion of a relocatable constant means the following. An architected location in the target device configuration ROM contains the address (or offset) of the given structure within the the target address space. Vendors have the choice of which location within target memory address space may be placed in this configuration ROM location.

It is expected that during system initialization any initiators wishing to use the target will read the addresses of the various structures indicated below and store them internally. It would be possible for an initiator to read the address of the registers more frequently but this would reduce system performance. As needed, targets and/or other initiators may read such configuration ROM as provided within the target.

The following sections describe the several data structures needed to respond to unsolicited data packets sent to a target supporting the SCSI 3 Serial Bus Protocol. These unsolicited transmissions are divided into three categories, named, Urgent, Normal, and ACA. The Urgent category contains those packets for which the initiator desires the target undertake a reaction on a high priority basis. These urgent items represent management type activity as directed by some initiator. The Normal category consist of ongoing activity which is to receive a timely reaction rather than a priority reaction. Submission of a Tap packet is a key example of an item in the Normal category. The ACA category supports commands sent to the target for the purpose of directing recovery and other desired actions when the ACA condition exists at the target.

The FIFO address alone is sufficient to determine the class of the packet being received and the degree of immediacy required for the processing expected per such a packet. There are multiple different types of packets which may be sent to the FIFO servicing Urgent items. In future expansions of the Serial Bus Protocol there may be different type items sent to the Normal FIFO. Under these circumstances, the Type_ID field within the packet payload must be examined in order to determine what type of processing is to be specifically undertaken.

The case of solicited data packets occurs as a Response Packet in relation to some Read Request sent by the target. These Response packets are expected so that the target can ensure sufficient memory to hold them.

The Response packets are sent to the general node address of the target rather than to some specific register address within the target. Examples of such expected response information are the SCSI command block and the data to be placed onto target Media in relation to some SCSI Write command. Correlation information within the header of the IEEE 1394 packet header is used to distinguish among the several different type information sent to the general node address of the target.

Implementation Note: While is is true that multiple "register" addresses are defined below, there is no requirement to implement each of these registers as a unique item of register hardware. It is fully possible to consider some of these register addresses as defining a "virtual register" supported by a much smaller collection of actual hardware. The key element of the "virtual register" is its address. This address is intended to define the content of the payload portion of any packet sent to that address. Also defined from the address of the "virtual register" is the nature of a special processing needed.

8.1.1. Command FIFOs

The command FIFO base address is specified in configuration ROM. Figure 3 defined the structure of the command FIFO address.

8.1.1.1. Normal FIFO

The Normal FIFO is the Commmand FIFO address with a Reg_Type value of 00 (binary). This FIFO is used for receiving Serial SCSI Command Initiation Packets.A very crucial element of capability for a target is the number of "command Taps" which may be accepted from a set of connected initiators. A well configured system should match the number of initiators and work process queues to known capabilities of connected targets so that the number of instances is very small wherein a target's capability to accept command Taps will be exceeded. In such instances where target capabilities are exceeded, then resource conflict type of busy condition must eventually be returned back from the target to the initiator.

The format below is intended for the 64-bit address of the Normal FIFO and the Urgent FIFO. The intent is to simplify management by the target of Tap slot resources reserved for the exclusive use of some initiator. A key element of this format is the introduction of a densely packed 8-bit field to be used as an initiator Identifier to distinguish among the various initiators who wish to send Tap packets to the given target. One result of the log-in process is the assignment of the subject initiator Identifier.

As a convenience to the process by which Tap slot management occurs, it is useful to consider the address of the Command FIFO as conveying information about the initiator who is responsible for the Tap packet. Due to the need to maintain the relative order of arrival for Tap packets, it is important that the interpretation be made of their being a single Normal FIFO used by all initiators rather than a collection of multiple Normal FIFOs with one member each assigned to a given initiators. Multiple software and hardware implementations are possible consistent with the maintenance of ability of the target to maintain relative order of arrival of arrival of Tap packets.

63 (MS) to 48	IEEE 1394 Node Address	This 16-bit field follows the structure of an IEEE 1394 address and conveys the the address of the given node on the IEEE 1394 bus. This field consist of the most significant 10-bit sub-field with is the identification number of the IEEE 1394 bus (bus_id) and the least significant 6-bit sub-field which is the unique number of the node on that bus (offset_id).
47 to 16	SBP target	This 32-bit field constitutes the most significant 32-bits of the 48-bit complete address of the Command FIFO within the address space within the given target. The value of these high order 32-bits is found at an architected location within the configuration ROM supporting the given target. The value of the low order 16-bits is established according to the use which is to be made of the Command FIFO. It is observed that it is possible for multiple target devices to be implemented at a single IEEE 1394 node. In such case, the high order 32-bits help to distinguish among these several targets.
15 to 8	initiator ID	The value of this 8-bit field is used as a short form identifier to distinguish among initiators sending tap packets to this command FIFO. It is intended that value zero in this field is used by an initiator which has not yet completed log-in and therefore does not yet have assigned to it an initiator identifier. Non-zero values for this field are valid instances of an assigned initiator identifier.
7 and 6	Reg_Type	This 2-bit field is used to identify the specific type of Command FIFO. Values for this field are defined as follows when the initiator ID field contains zero: • 00 means the log-in request FIFO. • 01 means the ACA FIFO (shared by all initiators). Values for this field are defined as follows when the initiator ID field contains a non-zero value: • 00 means the Normal FIFO. • 01 means the Urgent FIFO. All other values for this field are reserved.
5 to 0 (LS)	Reserved	This 6-bit field is required to be set to zero and is intended to simplify initiator interfaces. One possible instance of simplification is to provide desired address boundary placement for Tap messages.

Figure 3. Structure of Command FIFO Address

8.1.1.2. Urgent FIFO

The Urgent FIFO is the Command FIFO address with a Reg_Type value of 01 (binary). This FIFO is used for receiving urgent taps for messages and tap packets.

Items of control information which may be sent to the Urgent FIFO are analogous to Parallel SCSI messages and state changes. The payload of these control packets is to be placed into the same format as the 12-byte payload used various command Tap packets. A Tap packet message is one specific instance of a message which can be sent to the Urgent FIFO. This Tap packet message makes reference to a command chain from which command blocks are to be fetched by the target on a priority basis.

8.1.1.3. ACA FIFO

The ACA FIFO is the Command FIFO address with a Reg_Type value of 01 (binary) and a zero initiator ID. This FIFO is used for receiving Serial SCSI Command Initiation Packets pointing to command block chains with queue type of ACA.

It is required that the only packet sent to this FIFO is a Tap packet message making reference to a command block chain containing only ACA commands. It is noted, that ACA commands may only be accepted by a target which has the ACA condition being True. The payload of these control packets is to be placed into the same format as the 12-byte payload used various command Tap packets.

8.1.2. First Failure "Register" (optional)

The possibility exist that a request by a target to an initiator may fail. In spite of best and repeated efforts by the target, the request may continue to fail. In a favorable circumstance, the failure occurs in relationship to execution of a command block such that both a Status FIFO address and a Sense Data buffer address are known and available for use in reporting the failure. In an unfavorable circumstance, the failure occurs in the fetch of a command block so that the target does not have either an address for the Status FIFO or the address of the Sense data buffer. In another unfavorable circumstance, the access failure occurs relative to attempted use of either the Status FIFO or the Sense Data Buffer address.

Editorial Note: The text as now written indicates the First Failure register to be a required support element for a Serial Bus Protocol target. Comments are requested if it is more appropriate to make this register as well as the associate First Failure Control register to be optional. One argument in favor of making these two registers optional is that they are an instance of having the Serial device provide a better level of support than is the situation in a comparable case within Parallel SCSI. Provision of such improved support would perhaps represent a needless increase in "expense" for a low end target supporting Serial SCSI. Should these First Failure registers not be provided, the final recourse of the initiator is to rely upon a time-out mechanism in relation to an expected reaction or response from the target which does not occur in a timely manner.

The target experiencing a request failure such that a Status FIFO or a Sense Data buffer address cannot be used is required to record information regarding this failure in a First Failure Register. The contents of this register are placed in a "locked" state by the target after the given failure information has been recorded. Release of the lock upon the First Failure register is made by an initiator through use of the associated First Failure Control Register. If one or more additional failures occur while the First Failure register is in the locked state, then no information need be preserved by the target regarding these failures which can not be recorded in the First Failure register. Until the First Failure register is returned to the unlocked state, no additional failure information can be recorded by the target in the First Failure register.

The First Failure register is defined to be 96-bits in storage capacity. Information placed into this register shall consist of:

(a) the 64-bit address at which this failure occurred,

(b) the identification of the IEEE 1394 transaction that failed, and

(c) the type of Serial Bus Protocol activity (command block access, status block store, etc.).

The contents of this register can be written only by the target. The contents of this register can be read by any interested node on the IEEE 1394 bus.

8.1.3. First Failure Control "Register" (optional)

This "register" is a 32-bit entity maintained within a target for purposes of "locking" and "unlocking" the associated First Failure register at that same target. The target is the only IEEE 1394 node which can lock the associated First Failure register. The lock operation is accomplished by target by a write operation to this register consisting of a preestablished 32-bit pattern. Such lock operation is required whenever the

target writes information into the associated First Failure register. The unlock operation occurs when any IEEE 1394 node other than the given target writes a preestablished 32-bit reset pattern into this control register.

Editorial Note: Comments are requested as to the proposition that if a First Failure Control Register is provided by a target, then it is useful for the target to send a Status indication of a successful unlock operation whenever such operation occurs upon this control register. Such notification can be considered a form of Asynchronous Event Notification. As such, the notification of an unlock operation on this control register would only be sent to those initiators who have "signed-in" with that target as to wanting notification of various asynchronous events at that target.

8.2. Initiator "Register" Definitions

In many important cases the public knowledge concerning initiator registers is different than the public knowledge concerning target registers. In particular, the Status FIFO address is an example of a quantity which is not put into configuration ROM and made publicly available to any interested party. The Status FIFO address is included in the command block and thereby communicated to only the target having need for this information.

The observation is made that implementation by the initiator for the Command FIFO set of registers is needed only when that initiator desires to be able to function as a SCSI target device.

8.2.1. Status FIFO

The obligation of the target is to present status information back to the initiator regarding completion of a command. A critical issue is the manner of control of interrupts to be presented to the initiator when the target delivers operation complete status. One desired option is that the initiator have the ability to specify a general interrupt to be presented on completion of a command. An alternative option also desired is to have status information stored in an area which might be examined by the initiator at a convenient time rather than on an interrupt driven time basis. In either option it is necessary for the target to provide back to the initiator some form of correlation information between command blocks and status blocks. This Serial Bus Protocol requires the very suitable form of correlation information which is the 64-bit address of the associated command block. Placement of completion status information is to be made to the Status FIFO maintained within initiator memory address space.

8.2.2. Asynchronous Event Reporting

An initiator may wish to sign-in with a target so that the initiator might receive reports of Asynchronous Events defined per SCSI 2 and occurring at the given target. An initiator may sign-in for at most one Asynchronous Event from any any given target. The initiator may decide not to sign-in for reporting of Asynchronous Event reporting so that no buffer area need be reserved for this purpose.

In order to receive such Asynchronous Event information, the initiator must reserve two buffer areas for this purpose. One buffer area is for receiving a Status Block and the other buffer area is for receiving Sense Data. The starting address of each buffer must be made known to the target. A separate and distinct buffer set must be maintained to receive inputs from each target from which Asynchronous Event reporting is desired.

Whenever data is sent to either of the two buffers, the target shall consider both buffers as no longer available. A new sign-in process is required by the initiator in order to identify a new set of two buffers for use by a target wishing to report on other Asynchronous Events.

9. Command and Status Information

9.1. Command Blocks

As described below, the SCSI 3 Serial Bus Protocol uses a 64 Byte command block which embeds a standard parallel data SCSI Command Descriptor Block (CDB) of up to 16 Bytes in length. Additional standard or vendor unique CDBs may be defined for target types or functions which were not defined for parallel SCSI, see 15.0, "Compatibility to Parallel SCSI" on page 83.

Editorial Note: The area within a Serial bus Protocol command block reserved for holding a SCSI CDB has grown from 12-Bytes to the present 16-Bytes. No expansion has been made in command block length in order to accommodate the longer CDB. Instead, a previously reserved area of 4-bytes immediately following the CDB has now been formally declared as dedicated to holding a new CDB which is expected to grow to 16-bytes in length. While no 16-Byte CDBs are formally defined at present, there is sufficient information indicating that such a longer CDB can reasonably be expected during the lifetime of the Serial Bus Protocol.

In support of the need by low-end target devices, the command block can be viewed a a hierarchy of sections such that only a portion of the command block need be fetched for some specific purpose. One instance of a specific purpose, is the operation to examine the CDBs carried by several command blocks within a chain in order to determine which command is best suited for execution according to optimization rules employed by the target. Once a command block has been fetched, even in part, the target is fully responsible for noting and taking appropriate process steps for the queue type indicated by that command. Thus, if a Head of Queue command is detected during a series of fetch operations in which only a portion of the command block has been fetched, the target must take required and appropriate action based upon determining that a Head of Queue command is present. At time of placing a given command into execution, the target must ensure that is has access and required knowledge of the information contents conveyed by the entire command block.

In support of initiator units in which memory cache is employed, the requirement is set that the command block is to start upon an 8-Byte boundary. Additionally, each address in the command block is to start as well upon an 8_Byte boundary. In conformity to the IEEE 1394 standard, each of these addresses is an 8-Byte quantity. Observe, each command block is a 64-Byte quantity. Thus when command blocks are placed back to back in initiator memory space, and the very first command block starts on an 8-Byte boundary, then each of the following command blocks in a chain also starts on the desired 8-Byte boundary.

Contents of the first two quadlets of the Serial Bus Protocol Command Block have been established so as to enhance compatibility with Serial SCSI as supported by the Fibre Channel. Specifically, the placement of the queue type field is intended to agree with the bit positions adopted within the Fibre Channel SCSI packet for indicating Simple, Ordered, and Head of Queue Types. Observe closely that the Serial Bus Protocol treats this subject as a 3-bit coded field in which only one queue type is valid at any one time. In contrast, the Fibre Channel SCSI packet treats this field as independent Flag bits, such that more that one queue type indication is possible. Resolution of this difference is needed.

Another aspect of the Serial Bus Protocol command block is taken from the Fibre Channel SCSI packet and may need interpretation based on differences between the Fibre Channel and IEEE 1394. This aspect is the notion of indicating whether the direction of flow for data transfer is the same as the direction of flow for Command Transfer. In the case of the Serial Bus Protocol, the direction of command transfer is exclusively from the initiator node to the target node.

Finally, the placement of the Transfer length field is selected to agree with the placement of the similar field within the Fibre Channel SCSI packet.

Byte	0	1	2	3
0	Next Command Address (MSQ)			
4	Next Command Address (LSQ)			
8	Reserved		LUN	
12	Codes		Flags	
16	CDB (MSQ)			
20	CDB			
24	CDB			
28	CDB (LSQ)			
32	Transfer Length			
36	Control		Reserved	Sense Length
40	Data Buffer Address (MSQ)			
44	Data Buffer Address (LSQ)			
48	Status FIFO Address (MSQ)			
52	Status FIFO Address (LSQ)			
56	Sense Buffer Address (MSQ)			
60	Sense Buffer Address (LSQ)			

Figure 4. Asynchronous Command Block

Field **Function**

Next Command Address

This field contains a 1394 format 64-bit address which is used by the target when it fetches the following command from the initiator. The least significant three bits of this field must be equal to value zero. A full description of the mechanism by which the target fetches commands is presented in 5.0, "Command Transfer Protocol" on page 28.

Reserved This 16-bit field is reserved.

LUN This 16-bit field carries the Logical Unit Number (LUN) as specified in the SCSI Identify message. Unused bits are reserved. It is observed that SCSI 2 uses only 3-bits and these are the right justified bits in this field. SCSI 3 is anticipated to expand this field to at least 5-bits, and some argue this field needs to be larger than 8-bits in order to support future RAID applications.

Codes This 16-bit field is a coded field which identifies format, function code, and queue type for this command block.

Bit(s)	Name	Function
15 (MS) to 14	Format Identifier	These two bits identify the format of the command block. Values for this field are defined as follows: • Value 01 confirms this to be a command block as specified by the Serial Bus Protocol. All other values for this field are reserved.

13 to 8	SBP Function Code	These 6-bits provide information as to whether the command carried by this command block is a general purpose asynchronous command, is an Isochronous command or supports some control function of the Serial Bus Protocol (SBP). Values for this field are defined as follows: • Value 000000: this is a general asynchronous command (Carries a CDB). • Value 000001: this is an Isochronous Port_Setup command. • Value 000010: this is an Isochronous Port_Append command (Carries a CDB). • Value 000011: this is an Isochronous Port_Teardown command. • Value 000100: this is a SBP Request for an initiator Identifier. • Value 000101: this is a SBP Request/Release for Tap slot(s). • Value 000110: this is a SBP Request/Release notification of asynchronous events. All other values for this field are reserved. **Note:** (1) Refer to Appendix C for a description of the Isochronous commands. **Note:** (2) Refer to the section entitled, "SBP Control Protocols" for a description of the SBP Request/Release commands.
7 to 3	Reserved	These 5-bits are reserved.
2 to 0 (LS)	Queue Type	• Value 100 means Tag Type Autocontingent Allegiance (ACA). • Value 000 means Tag Type is Simple Tag. • Value 010 means Tag Type is Ordered Tag. • Value 001 means Tag Type is Head of Queue. All other values for this field are reserved.

Figure 5. Codes Field

Flags

The 16-bit flags field currently has the following bits defined. All bits not specified are reserved.

Bit(s)	Name	Function
15 (MS) to 13	Reserved	These 3-bits are reserved.

12	Read Data	When set equal to value one, this bit indicates the direction of I/O data movement is from the target to the initiator. This is opposite to the direction of command movement. **Note:** If both the Read Data Flag and the Write Data Flag are set to value zero, then no data transfer is to occur.
11	Write Data	When set equal to value one, this bit indicates the direction of I/O data movement is from the initiator to the target. This is the same as the direction of command movement. **Note:** It is not valid for both the Read Data Flag and the Write Data Flag to be set to value one at the same time.
10	L_Flag	When set equal to value one, the CDB field is carrying a SCSI Linked command. This flag must have value zero for the last command in a collection of Linked commands.
9	M_Flag	When set equal to value one, the target must fetch the next command when it is ready. If this bit has value zero, the present command is the last in the chain from the initiator.
8	ESC_Flag	This is the End of Sub_chain Flag. When set equal to value one, the present command block marks the end of one sub-chain and the Next Command Address field marks the start of a new sub-chain. The next fetch of a command block will not be from the new sub-chain unless it is the only chain known to the target. When set equal to value zero, the next command block is to be fetched from the address indicated by the Next Command Address field and this next command block is considered to be a member of the same sub-chain as is the present command block. This flag has meaning only when the M_Flag has value one.
7	A_Flag	When set equal to value one, the initiator wishes this command to be aborted. If this bit has value zero then no expression of intent has been made to abort this command. For information regarding the SCSI Abort message please refer to *****.
6 to 3	Reserved	These 4-bits are reserved.
2	S_Flag	When set equal to value one, this bit indicates the initiator memory scatter / gather function is being invoked for the present command.

1	O_Flag	When set equal to value one, this bit indicates the target must transfer data in sequential order to/from the initiator. If this bit has value zero, the target may transfer data out of order.
0 (LS)	C_Flag	When set equal to value one, this bit indicates an automatic clearing of any contingent allegiance conditions which may occur as a result of this command. When this bit is reset to value zero, contingent allegiance conditions are handled as in SCSI 2 and SCSI 3. The target waits for action by the initiator.

Figure 6. Contrl Flags

CDB This field carries a standard SCSI CDB.

Transfer Length The following definition of meaning for this field applies to the case of the SBP Command Operations Code equal to value (000 binary), which means the command block carries a general asynchronous command. Refer to the description of Isochronous commands and to the description of SBP Request/Release commands for the meaning of this field in either of these contexts.

For general asynchronous commands this field supports one of two functions, depending on the value of the S_Flag within the Flag field of this command block. When the S_Flag has value equal to one, this field supports the initiator memory scatter/gather function, which is a required support capability provided by the target. The length value indicates the number of Bytes of scatter/gather entries which are to be found in the list found at the 1394 style address specified by the Data Buffer Address entry of this command block. When the S_Flag has value equal zero, the present command does not involve scatter/gather, and the Transfer Length field contains the number of bytes which are to be transferred as a result of successful completion.

Control Reserved

This 16-bit field is reserved with the intention to possibly relocate control function specified elsewhere in the command block to this location.

Editorial Note: Consideration is being given to relocate data transfer control flags such as the S_Flag and the O_Flag from the flags field to this field. In addition, consideration also is being given to place in this field selected data transfer control parameters associated with the IEEE 1394 High Speed Serial bus in this field. Examples of IEEE 1394 parameters would include a statement of the data transfer rate and maximum data packet size supported by the IEEE 1394 bus.

Reserved This 8-bit field is reserved.

Sense Length This 8-bit field represents an unsigned number which specifies the number of 16-Byte units of length for the Sense Data buffer.

Data Buffer Address

The following definition of meaning for this field applies to the case of the SBP Command Operations Code equal to value (000 binary), which means the command block carries a general asynchronous command. Refer to the description of Isochronous commands in Appendix C for the meaning of this field in the Isochronous command context.

For general asynchronous commands the Data Buffer Address field contains an IEEE 1394 format 64-bit address that the data should be written to or read from. It is observed that this data address may or may not be associated with the same node (initiator) having sent the given command. If the required feature of initiator memory scatter/gather has been indicated by means of the S_Flag having value equal one, then the Data Buffer Address location contains the scatter/gather list. As an additional observation, the Data Buffer Address has no specific alignment restriction so that it can point to a byte boundary.

Status FIFO Address

This field contains the IEEE 1394 style, 64-bit address of the Status FIFO associated with the given initiator responsible for the associated command.

Implementation Note 1: The Status FIFO address can be used by the initiator to control whether or not interrupts are to be signalled to the initiator upon receipt of a status block. Thus, the initiator can elect to accumulate Status Blocks within a given Status FIFO and then process them at leisure.

Implementaton Note 2: Attention is called to the fact that alignment restrictions apply to certain types of objects pointed to by an IEEE 1394 style address. In particular, the Status FIFO (also the Command FIFO) must be aligned on a 4 Byte boundary. Thus, low order two bits of the Status FIFO address must be zero with the consequence that the least significant bit can used as a Validity Indicator for that address. Should the Status FIFO address be indicated as not valid, (least significant bit of the address has value equal to one), this means can be used by the initiator to inform the target that status block information should not be returned to the initiator by means of the Status FIFO.

Sense Data Buffer Address

This field contains the IEEE 1394 style, 64-bit address of the Sense Data Buffer associated with the given initiator responsible for the associated command.

9.2. Status Block

The Status Byte for a Serial SCSI command is similar to the Status Byte in Parallel SCSI. The Serial SCSI Status Byte is embedded in a packet with 12-Byte payload (the Status Block) which is used to mark the end of a command under the SCSI 3 Serial Bus Protocol. The Serial SCSI Status Byte is intended to be upward compatible to the Status Byte in SCSI 2. For conditions under which the command completes with good status, this single Status Byte is the only completion status returned back to the initiator. Other fields in the Status Block serve to: (a) identify the payload as being a Status Block, and (b) to facilitate correlation of the Status Byte to the associated command. Depending on the command and on the SCSI device, sense data may be available even when the command completes with good status.

Byte	0	1	2	3
0	Command Address (MSQ)			
4	Command Address (LSQ)			
8	Type_ID Status Block Code = 80 hex	Reserved	Status Byte per SBP	Status Byte per SAM

Figure 7. Status Block

Command Address

This is the address from which the command block was read. This value is returned to the initiator in the status packet to correlate the status with the command.

Type_ID This field confirms that the present data block is a Status Block.

Reserved This field is reserved.

Status Byte per SBP

This is the status Byte as defined for use within the Serial Bus Protocol architecture to describe completion of events or other information uniquely defined within the Serial Bus Protocol.

Editorial Note: Consideration is being given to define an encoding of information which would assist the initiator in knowing when Tap Slots have been made available for use by the given initiator. Current thinking is that providing such information is optional to the target. Should the target decide to supply such information, then in the current thinking, the information would be in the form of the number of Tap Slots dedicated for use exclusively to the given initiator which are presently available as of the time at which completion status is returned for this command.

Status Byte per SAM

This is the Status Byte as defined in the SCSI Architecture Model (SAM) document. It is intended that this status definition be upwardly compatible to status definition existing within SCSI 2.

9.3. Initiator Scatter/Gather List

The scatter/gather list shall consist of one or more 16-Byte units in the format depicted below. While there is no restriction against creation of a scatter/gather list consisting of only one 16-Byte unit, such usage would be wasteful of initiator memory. If there is only one element to the scatter/gather list, that single element could be accommodated much more efficiently using only 12-Bytes within the command block.

Byte	0	1	2	3
0	Data Buffer Address (MSQ)			
4	Data Buffer Address (LSQ)			
8	Reserved			
12	Transfer Length			

Figure 8. Scatter/Gather List Format -- Single 16-Byte Unit

Data Buffer Address

This field contains an IEEE 1394 format 64-bit address that data should be written to or read from. It is observed that this data address may or may not be associated with the same node (initiator) having sent the given command. It is a requirement per the IEEE 1394 standard that at least the low order two-bits be zero as a data buffer address is specified as being a four-Byte aligned address.

Reserved This is a 32-bit field is reserved. The presence of this field ensures that the multiple Data Buffer Address entries within this list are each separated from one another by some multiple of 8-Bytes.

Transfer Length This 32-bit field contains the number of bytes which are to be transferred as a result of successful completion.

10. Payload Specification For Command Transfer Packets

This section details the contents of the payload for those packets used by the Command Transfer Protocol. Refer to Appendix A for a summary of the format of the IEEE 1394 packets used to transfer these payloads.

10.1. Payload of SCSI Command Initiation Packet - "Tap Packet"

The packet carrying this payload is to be sent to the Destination Offset address within the target representing the Normal FIFO.

Byte	0	1	2	3
0	First Command Address (MSQ)			
4	First Command Address (LSQ)			
8	Type_ID Status Block Code = 01 hex	Reserved		

Figure 9. Payload of SCSI Command Initiation Packet - "Tap" Packet

Field Function

First Command Address
This 64-bit field carries the Address of the first command in a command chain.

Type_ID The value 01/Hex in this field confirms the present packet payload to be the payload of a SCSI Command Initiation packet.

Reserved This 24-bit field is reserved.

10.1.1. Response Reactions to Tap Packet

There will always be a reaction to a SCSI Command Initiation Packet (Tap Packet) and this reaction will contain one element or two elements depending on:

(a) whether the target has a Tap slot available to store the Tap packet, and

(b) whether the target does accomplish every element of its mandated reactions in the time period before the IEEE 1394 bus must be released.

The term "Unified Reaction" is used to refer to the case in which the target is able to accomplish all of its mandated reactions in the time interval before the IEEE 1394 bus must be released and thereby made available for arbitration by some other node. The mandated reaction will always consist of sending an IEEE 1394 defined acknowledgement back to the initiator. Depending on the situation, the mandated reaction may also consist of sending an IEEE 1394 defined Write Response Packet back to the initiator. The term "Unified" is based on IEEE 1394 terminology for Transaction types. Unified, in this context, means that if a Write Response Packet is required as an element of the reaction, then the Response Packet is placed onto the IEEE 1394 bus shortly after the Acknowledgement reaction is sent, and also in the time interval before the IEEE 1394 bus must be released by the target.

The Term "Split Reaction" is used to refer to the case in which the target must send a Write Response Packet as an element of its reaction back to the initiator, but cannot send this packet in the time interval before the IEEE 1394 bus must be released. Observe closely, the present discussion applies only in those instances in which a required part of the reaction by the target involves sending a Write Response Packet back to the initiator. In all instances, it is required that the target provide back to the initiator an acknowledgement reaction before the bus is released. The term "Split Reaction" is based on IEEE 1394 terminology for Transaction types. Split, in this context means that within a IEEE 1394 defined time interval, the target must begin a new transaction (the split transaction), for the purpose of sending the Write

Response Packet back to the initiator. Per requirements of IEEE 1394, if a Split Transaction is involved as a reaction to an earlier transaction, then the acknowledgement of "pending" must be made as the end to that earlier transaction.

Editorial Note: It has been expressed by many parties that it is highly desirable to make it almost impossible for a target to reject a Tap Packet. Toward this objective of ensuring that Tap Packets are not rejected by targets, it appears that enhancements to the Tap Protocol need to be defined. Current exploration is centering on providing means to confirm to an initiator that the target has reserved a some number of Tap Slots for exclusive use by that initiator. It is desired to minimize the degree of coordination undertaken by targets relative to allocation of Tap slots among the initiators. In this exploration, the initiators would collaborate with each other so as to determine the division of Tap Slots within a given target. The manner of collaboration among initiators does not need to be stated by the Serial Bus Protocol. Current exploration thus proceeds from the assumption that initiators have agreed to some allocation of Tap Slots. The Serial Bus Protocol would then specify: (a) the means whereby a given initiator issues a request for tap slots to a given target, (b) the means whereby a target confirms an allocation back to the requesting initiator, and (c) the requirement that target devices manage Tap Packet receptions consistent with allocation commitments made to the various targets.

Current thinking on this subject is in favor of making it optional for targets to support the above scheme for management of Tap Packet slots. It would be mandatory for a target to specify within a configuration ROM (or by some other means) whether that target does, or does not support the above scheme for management of Tap Slots.

10.1.1.1. Response Reactions When Tap Packet is Accepted

When the target is able to accept a SCSI Command Initiation Packet (Tap Packet), there is only one element required as part of the target reaction. This required reaction is to send the IEEE 1394 acknowledgement code of "complete". The initiator then is to have the understanding that the Tap Packet has been accepted by the target and no further activity is required of the initiator other than to be prepared to receive Read Requests Packets sent by the target in order to fetch command blocks.

Implementation Note: Per the IEEE 1394 standard, an alternative reaction is possible by the target in the circumstance in which that target can accept a Tap Packet. By action of the Serial Bus Protocol, this alternative reaction is prohibited. The alternative reaction consists of the Split Transaction scenario. Due to slow processing within the target, it may occur that certain "housekeeping" operations are not completed in time to justify the acknowledgement code of "complete". The IEEE 1394 standard permits an acknowledgement code of "pending" and a later submission of a Response Packet carrying a response code of "resp_normal". Per requirements of the Serial Bus Protocol, a compliant target must provide processing capability sufficient to ensure an acknowledgement code of "complete" in all circumstances in which a Tap Packet has been accepted by that target.

10.1.1.2. Response Reactions When Tap Packet is Not Accepted

In spite of best efforts by configuration management, it is possible that circumstances will arise such that a Tap Packet cannot be accepted by the target. Under these circumstances it is necessary to use suitable means to inform the relevant software within the initiator that some form of retry activity is necessary in order to reach the condition in which the target can accept the Tap packet.

If the target cannot accept a Tap Packet, by requirement of the Serial Bus Protocol, the reaction of the target must consist of two elements and these are forced to be satisfied by means of the IEEE 1394 Split transaction scenario. The first element of the target reaction consist of sending the acknowledgement code of "pending" back to the initiator. By operation of the IEEE 1394 standard, the code of "pending" forces the target to send an explicit Write Response Packet. Additionally, this Write Response Packet is sent in a new IEEE 1394 bus arbitration interval thereby making this a Split Transaction. The Write Response Packet has

no payload nor does it need a payload in order to convey a reaction to the circumstance that the target cannot accept a Tap Packet. This Response packet does have a Header containing among other fields, a 4-bit R_Code field. The R_Code field contains response code and this code must contain the "resp_conflict" value when the target cannot accept a Tap Packet. It is by means of the R_Code that initiator software is made aware that a Tap Packet has not been accepted by the target. Initiator software may then decide the best manner to repeat the sending of the given Tap Packet to the target.

Implementation Note: Per the IEEE 1394 standard, alternative reaction is possible by the target in the circumstance in which that target cannot accept a Tap Packet. By action of the Serial Bus Protocol, this alternative reaction is prohibited. The alternative reaction consists of sending an IEEE 1394 acknowledgement code of busy_X. Still other alternatives are sending IEEE 1394 acknowledgement codes of busy_A or busy_B. In all three of these acknowledgement codes, the appropriate initiator reaction consists of an retry transmission of the Tap Packet with the retry decision made automatically by the IEEE 1394 hardware and no involvement by the initiator software. Due to various steps taken to make it a remote circumstance for a Tap Packet to be rejected, it is more appropriate to have the initiator reaction explicitly determined by initiator software.

10.2. Command Read Request

10.2.1. Request Payload

A target requests a command by sending a Read Block request packet to the address indicated in the "Tap" packet or to the address specified in the previous command. Normally, this request will be handled as a split transaction by the initiator.

10.2.2. Response Payload

The response payload consist of the 64 Byte combination of Baseline plus Extended portion of the command block, or the 48-Byte Baseline portion, as originally requested by the target.

11. Data Transfer Protocol

11.1. Asynchronous Transfer

The transfer of all data associated with the execution of a command is accomplished by standard format (per IEEE 1394) block read request and block write request packets sent by the target to the initiator.

The starting address the block data is read/written from/to is the address passed in the Data Buffer Address field in the command block. For each packet read/written from/to a buffer, the address is specified by the target and placed in an appropriate type request packet. As necessary, the target must increment the referenced data address by the packet size for subsequent packets to the same buffer unless the out-of-order flag bit has enabled the target to transfer data in any order it desires. In that case the target must insure the starting address of each packet is set properly.

11.1.1. Data Read From Device Medium

This is the case where the device is producing data to be transferred to the initiator. This is done using a series of Write Block request packets. The target is responsible for updating the request packet destination offset field to reflect progress made in moving through the initiator buffer.

The size of the data packets sent by the target to the initiator is of a maximum defined in the device ROM or dictated by the speed of the particular IEEE 1394 Serial Bus. In the case of Write Block packets sent from a target to an initiator, it is desired that after receiving a data packet from a target with the header and data CRC correct, the initiator should return an acknowledge code indicating "complete". If an initiator is unable to receive a packet because of temporary buffering limitations or receives a packet with incorrect CRC, the initiator should return an appropriate acknowledge code as dictated by the IEEE 1394 standard.

11.1.2. Data Written To Device Medium

For the case where the target is receiving data from the initiator, it requests this data via Read Block request packets. It sends these requests as it requires the data. Hence the target automatically paces the transfer.

In the case of Read Block request packets sent from a target to a initiator, the target may send multiple outstanding requests to the initiator provided that each outstanding request (split-transaction) has a unique transaction label (number) in the Read Block request packet header so that the multiple response packets being returned to the target may be correlated to the original requests.

12. Status Transfer Protocol

At the end of a command the target must return status information to indicate success or failure. This is done using a Write Block request packet. The packet is sent to the address specified in the Status Buffer Address field passed in the command block. Payload of the packet is the 12-byte Status block. In reply to the status packet, a split transaction is allowed if needed; otherwise, the initiator should return a "complete" acknowledge code.

In the most frequent case, the completion of an I/O process is with "good status" and no specific action is required by the initiator. The target sends to the initiator a 12-Byte Status Block directed to the Status FIFO address specified by the initiator in the command block.

Should the completion status be "bad", then both a Status Block and a Sense Data Block would be made available to the initiator. If the initiator has specified automatic presentation of Sense Data for this command, then the target first sends to the initiator a Write Request packet which contains the Sense Data as payload. This Sense Data is sent to the Sense Data Buffer address specified by the initiator in the associated command block. As a second operation, the target sends to the initiator a Write Request packet with the 12-byte Status Block as payload. This Status Block is sent to the Status FIFO specified by the initiator in the associated command block.

If the initiator does not want automatic presentation of Sense Data, then the target holds this information for request by the initiator as provided by SCSI 2. The target still must send the Status Block to the initiator in this situation of completion with "bad" status.

12.1. Target Reaction to Initiator Failure to Accept a Status Block

It can be argued that a similar condition exists at the initiator relative to acceptance of status blocks as exists at the target relative to acceptance of Tap Packets. In both instances, there exists the possibility that the intended destination cannot accept the given 12-Byte packet payload. While this may be true in theoretical terms, there is a substantial difference in the nature of the two situations. For a variety of reasons, there are considerably more memory storage resources available to the initiator as are available to the target. Additionally, the initiator is presumed to be expecting the return of one status block for every command block which was allocated. Consequently, there is the notion of a much greater level degree of obligation and commitment by the initiator to be in a position to accept a status block whenever it might be sent by the target.

Notwithstanding the commitment of the initiator to accept status blocks, it is still possible for momentary circumstances to exist which lead to the initiator rejecting a given status block. Repeating for emphasis, it should be the objective that initiators never reject a status block. In the rare circumstance that an initiator does reject a status block, it should involve a situation in which the initiator shall very quickly be able to accept that previously rejected status block. Additionally, the target may not have the ability to undertake a higher level software protocol for reacting to a rejected status block. Thus, the mechanism for dealing with a rejected status block should be based on an automatic retry wherein IEEE 1394 hardware at the target attempts a retransmission of the status block at an early opportunity.

Based upon the above discussion, should an initiator need to reject a status block (for reasons of not having sufficient storage to accept it), the initiator reaction is to respond with an IEEE 1394 acknowledgement code of busy_X to the target. The reaction of the target to this acknowledgement code is to make a hardware initiated attempt to resend the status block.

Should all retry efforts by the target fail with regard to acceptance by the initiator of the Status block, then the target must record the appropriate information in its First Failure register.

12.2. Target Reaction to Abort Tag Request From an Initiator

The Serial Bus Protocol with its facility for command chains is a substantially different environment than the environment in Parallel SCSI. A key aspect of this difference is the circumstance that a target may have fetched certain of the commands from a given chain while other commands remain within initiator memory. The general existence of additional commands yet to be fetched is made known to the target by means of the M_Flag. No detailed knowledge concerning any of these unfetched commands is available to the target until the associated command block is actually fetched.

Thus, relative to an ABORT TAG message from the initiator, one of three circumstances may exist:

(1) the command has been fetched and is completed,

(2) the command has been fetched but no attempt at execution has been made by the target, or

(3) the command has not yet been fetched by the target.

In conformity to SCSI 2, no reaction is required by the target relative to a command which has been completed and is now requested to be Aborted by the target. For such a completed command, the target returns the same completion status as would have been the case had the ABORT TAG message not been received by the target. In passing, it is observed that no mechanism exist within SCSI to send two different and possibly conflicting sets of completion status information about the same command.

For the case in which the command has been fetched, but not yet completed, the ABORT TAG message provides the target with all needed information to make the required reaction. The target is to Abort the associated I/O process and make return of status information consistent with the attempt to Abort the subject command block. It would appear that two new STATUS Codes are needed for this Abort situation. One of the new STATUS Code values would indicate that the Abort was successfully accomplished. This new STATUS Code value shall be 44/Hex. The other new STATUS Code value shall be 46/Hex and this applies to when the Abort operation could not be completed in a successful manner.

For the case in which the command has not yet been fetched, it is desired to use the A_Flag within the command block rather than save the ABORT TAG message as defining a future work item for the target. In this circumstance in which the given command is unknown to the target because the command block has not yet been fetched, the target is to ignore the ABORT TAG message. Instead, the target is to fetch each command block in the normal manner. Upon receipt of a given command block, the target is to check the value of the A_Flag. If the A_Flag has value equal to one, then this command is to be Aborted. As previously discussed, the target makes return of a Status block in which the STATUS Byte has value 44/Hex or value 46/Hex depending upon whether the attempt to abort the command was successful or not successful. No response from the target to the initiator is required in order to indicate the situation that the ABORT Tag message was discarded and the abort operation proceeded on the basis of examining the A_Flag.

12.3. Target Reaction When a Cmmand Block Cannot Be Fetched

Another new circumstance arises in Serial SCSI, and this is the case in which all attempts fail regarding the fetch of a command block from a chain. The situation faced by a target is that the information known to the target is the starting address of the given command block. Since the command block could not be fetched (for whatever reason), the target does not have the address of the Status FIFO which is to be used for purposes of reporting on attempts to execute the given command. Neither does the target known if this is the last command block in the chain or if there are other command blocks in the same chain following the subject command block. Based on the format of the command block address, the target does know the the 16-bit IEEE 1394 bus address of the initiator having responsibility to provide the subject command block.

It would appear the target is faced with one of two choices. First, the target might ignore the situation and consider the given command chain to be terminated and thereby requiring no further attention. A possible

outcome from this reaction by the target is that the initiator will detect that all action has ceased relative to command blocks in the subject chain. Should the initiator be able to detect this state of affairs, then the initiator could instigate appropriate recovery action. The second alternative is that the target sends an unsolicited status block to some prearranged Status FIFO previously established for receiving unsolicited packets at an initiator.

The decision reached in the Serial Bus Protocol is in favor of having the target:

(a) abandon the given command chain and

(b) record appropriate information about the failure in the First Failure register maintained by that target.

No further reaction on the part of the target is required to inform the initiator that action has been terminated relative to the given chain. Information in the First Failure register is to be maintained in a "locked" state by the target until a release of the lock is made through use of the associated First Failure control register by some other node.

As further aspects of the reaction by the target when a command block cannot be fetched, the target shall not declare an ACA condition relative to the given initiator. No sense data need be retained as the target has included within the contents of the 12-Byte payload of the unsolicited packet all available information concerning the situation. Normal processing shall continue regarding any other command block chains associated with the given initiator and the given target/LUN.

12.4. Target Reaction to a Unit Attention Condition

A Serial Bus Protocol SCSI 3 target is required to declare a Unit Attention Condition for the same reasons and under the same circumstances as would be the case of a SCSI 2 target device. Further elements of the target reaction to a declared Unit Attention Condition are based on the notion that initiators sign-in with a target if they desire to receive notification of an Asynchronous Event from that target. Registration with a target means that the initiator has supplied the address of a status Block buffer and the address of a Sense Data buffer for purposed of receiving a report of an Asynchronous Event.

If an Asynchronous Event does occur, the target shall send an appropriate status block and appropriate sense data to the buffer address of each initiator having signed-in to receive notification of an Asynchronous Event. After transmission of the notification information, the target shall consider all initiators as having been taken out of the sign-in condition for purposes of receiving any further notification of an Asynchronous Event. Explicit and new sign-in action is required by each initiator wishing to receive notification regarding any further asynchronous Event.

13. SBP Control Protocols

Additional protocols are required within the Serial SCSI environment to support the ability of the initiator to interact with targets regarding management of:

- Tap Slot resource
- Sign-in for notification of asynchronous events
- Information contained in the First Failure register

The assignment of a short form initiator identification number is the initial as well as the most critical element within these SBP control protocols. targets are not allowed to respond to SBP control protocol requests from an initiator unless that initiator has previously obtained a short form initiator identification number from that target. As a further consequence of not having obtained a short form identification number, an initiator will not be able to secure a Tap Slot from a target and will thus not be able to secure acceptance by the target of a Tap packet sent from that initiator.

Thus, the key to securing interaction between targets and initiators for purposes of the command transfer protocol is an initialization phase in which an initiator executes a "log-in" procedure with a given target. As the outcome of a successful log-in operation, an initiator is assigned a short form (8-bit) identification number. As a required second phase of the initialization process, a "logged-in" initiator uses its short form identification number to request some number of Tap slots reserved for exclusive use by that initiator at the target.

As an optional third phase of the Initialization process or an optional procedure at any other time, a "logged-in" initiator may sign-in with the target for the purpose of receiving notification of asynchronous events at that target. An additional interaction between a "logged-in" initiator and a target is the purpose of a write operation to the First Failure Control Register, should that optional register be implemented at the given target. Any initiator, logged-in or not, may read the First Failure register if this register exists at the target.

13.1. Log-in Protocol

The log-in protocol is the first phase of an initialization procedure between a given initiator and a given target. Log-in is started by an initiator sending the Log-In_Request message to the urgent FIFO at the subject target. The log-in message contains the address at the initiator of the command block which contains the details of the log-in request. Refer to the section entitled, "Log-In_Request Message Payload" for details concerning the format of the log-in request message.

These results of log-in processing by the target are sent in a short (4_Byte) data packet to the Data Buffer address specified in the command block. If the log-in is successful, then the most significant item of returned information is the short form (8-bit) identifier assigned by the target for the use of the initiator. An additional item of returned information is a Status Block containing an SBP Status Byte carrying a code confirming the log-in to be successful. If the log-in was not successful, the returned code has an appropriate value indicating the failure.

Byte	0	1	2	3
0	Reserved			
4	Reserved			
8	Reserved		LUN	
12	Codes		Flags	
16	Reserved			
20	Reserved			
24	Reserved			
28	Reserved			
32	Transfer Length (04h)			
36	Control Reserved		Reserved	Sense Length
40	Data Buffer Address (MSQ)			
44	Data Buffer Address (LSQ)			
48	Status FIFO Address (MSQ)			
52	Status FIFO Address (LSQ)			
56	Sense Buffer Address (MSQ)			
60	Sense Buffer Address (LSQ)			

Note: (1) The Transfer Length field contains the fixed length of 4 Bytes of information to be returned to the initiator as the reaction of the target to the request from the initiator for the log-in operation.

Note: (2) The SBP Function Code must have the value "000100" binary which indicates this command block carries the request for an initiator Identifier.

Note: (3) The Queue Type Code is recommended to be value "001" binary as indication of this command block to be a Head of Queue command.

Note: (4) It is required that the Flags field indicate the present command block to be the only command block in the present chain and further that this is not a Linked command.

Note: (5) The Next Command Address Field is reserved in this command block.

Note: (6) This command block does not carry a SCSI CDB.

Figure 10. Format Command Block Used for Log-In Request

The table below indicates the specific format of the quadlet of data returned to the Data Buffer address as a result of successful completion of the log-in operation:

Byte	0	1	2	3
0	Reserved			initiator Identifier

Figure 11. Format of Returned Log-In Data

13.2. Request/Release of a Tap Slot

After the initiator has successfully completed a log-in operation with a target, the initiator may request the target assign it some number of Tap slots. In the alternative, the initiator may wish to release some or all of the Tap Slots which may have been previously allocated to it. In either situation, initiator must proceed via a two-step operation as described below.

In step one, the initiator sends a message to the target which advises the target that the initiator has a command block which contains the details of a request or a release of Tap Slots. The message contains the address in initiator memory space of this command block. In step two, the initiator supplies the target with the subject command block when requested to do so by the target. The command block approach is adopted so as to provide a mechanism for:

(a) providing the target with more details concerning the request than are feasible for provision within a message, and

(b) providing the target with means to give the initiator with more extensive results of the reaction at the target in relation to the request.

Refer to the Messages section entitled "Request/Release of Tap Slots Message Payload" for details concerning the message issued by the initiator as step one of the subject protocol.

The table below indicates the specific format of the command block provided by the target as step two of the subject protocol:

Byte	0	1	2	3
0	Reserved			
4	Reserved			
8	Reserved		LUN	
12	Codes		Flags	
16	Reserved			
20	Reserved			
24	Reserved			
28	Reserved			Number of Tap Slots
32	Transfer Length (08h)			
36	Control Reserved		Reserved	Sense Length
40	Data Buffer Address (MSQ)			
44	Data Buffer Address (LSQ)			
48	Status FIFO Address (MSQ)			
52	Status FIFO Address (LSQ)			
56	Sense Buffer Address (MSQ)			
60	Sense Buffer Address (LSQ)			

Note: (1) The value in the Number of Tap Slots field is an unsigned 8-bit number which indicates the number of Tap slots which the initiator wishes to have allocated to it after completion of this command. If this is a Request operation, then the Number of Tap Slots field carries a value which is larger than the present allocation (if any). If this is a Release operation, then the Number of Tap Slots field carries a value which is smaller than the present allocation.

Note: (2) The Transfer Length field contains the fixed length of 8 Bytes of information to be returned to the initiator as the reaction of the target to this command.

Note: (3) The SBP Function Code must have the value "000101" binary which indicates this command block carries the command for Request or Release of Tap Slots.

Note: (4) The Queue Type Code is recommended to be value "001" binary as indication of this command block to be a Head of Queue command.

Note: (5) It is required that the Flags field indicate the present command block to be the only command block in the present chain and further that this is not a Linked command.

Note: (6) The Next Command Address Field is reserved in this command block.

Note: (7) This command block does not carry a SCSI CDB.

Figure 12. Format Command Block Used for Request/Release of Tap Slots

The table below indicates the specific format of the two quadlets of data returned to the Data Buffer address as a result of successful completion of the Request/Release of Tap Slots command.

Byte	0	1	2	3
0	Prior Number of Allocated Tap Slots			
4	New Number of Allocated Tap Slots			

Figure 13. Format of Returned Data by Request/Release of Tap Slots Command

13.3. Request/Release of Notification for Asynchronous Events

After the initiator has successfully completed a log-in operation with a target, the initiator may make a sign-in request in which it requests the target to provide it with notification of asynchronous events at the target. In the alternative, the initiator may wish to gain release from a prior sign-in request. In either situation, the initiator must proceed via a two-step operation as described below.

In step one, the initiator sends a message to the target which advises the target that the initiator has a command block which contains the details of a request or a release regarding sign-in for notification of asynchronous events. The message contains the address in initiator memory space of this command block. As a special emphasis point, the sign-in for notification applies to one and only one asynchronous event at a time. When a target sends information concerning a given asynchronous event to those targets which have "signed-in", the target immediately and automatically takes all initiators out of the sign-in state. Thus, if an initiator wishes to continue receiving notification of some future asynchronous event, that initiator must complete another sign-in process. The new, sign-in process functions as if the initiator is undertaking its very first sign-in operation.

In step two of the sign-in process, the initiator supplies the target with the subject command block when requested to do so by the target. The command block approach is adopted so as to provide a mechanism for:

(a) providing the target with more details concerning the request than are feasible for provision within a message, and

(b) providing the target with means to give the initiator more extensive results of the reaction at the target in relation to the request.

Refer to the Messages section entitled "Request/Release of Notification for Asynchronous Events Message Payload", for details concerning the message issued by the initiator as step one of the subject protocol.

As a special emphasis point, an initiator may send to the target a second request for notification of asynchronous events even though a prior request for notification is still valid and in effect. The new Request is to "overwrite" the previous request in the sense that the new Status FIFO and Sense Data Buffer addresses replace the previous values for these two quantities. Sense data for this command shall consist of the two new addresses.

The table below indicates the specific format of the command block provided by the target as step two of the subject protocol:

Byte	0	1	2	3
16	Reserved			
20	Reserved			
24	Reserved	Reserved	LUN	
28	Codes	Reserved	Flags	AEN Indicator
32	Transfer Length (00h)			
36	Control Reserved		Reserved	Sense Length

40	Reserved
44	Reserved
48	Status FIFO Address (MSQ)
52	Status FIFO Address (LSQ)
56	Sense Buffer Address (MSQ)
60	Sense Buffer Address (LSQ)

Note: (1) The AEN Indicator is an unsigned 8-bit number. A positive value (normally value equal to one), indicates this command block carries a Request to receive notification of asynchronous events at the target. If this is a Release operation, then this field carries value equal to zero.

Note: (2) The Transfer Length field contains the fixed length of 0 Bytes of information to be returned to the initiator as the reaction of the target to this command.

Note: (3) The SBP Function Code must have the value "000110" binary which indicates this command block carries the command for Request or Release of notification for asynchronous events.

Note: (4) The Queue Type Code is recommended to be value "001" binary as indication of this command block to be a Head of Queue command.

Note: (5) It is required that the Flags field indicate the present command block to be the only command block in the present chain and further that this is not a Linked command.

Note: (6) The Next Command Address Field is reserved in this command block.

Note: (7) The Data Buffer Address Field is reserved in this command block.

Note: (8) This command block does not carry a SCSI CDB.

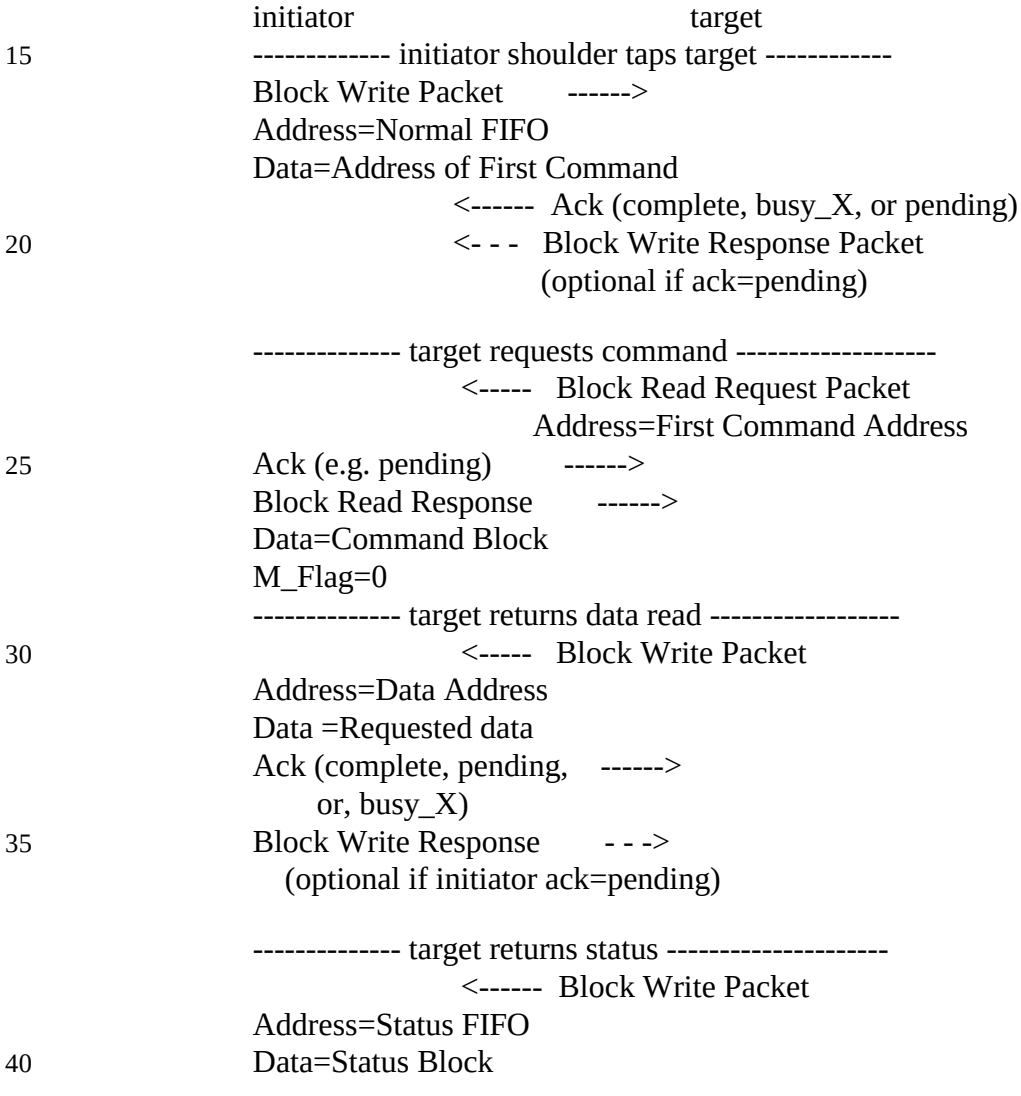
Figure 14. Format Command Block Used for Request/Release of AEN

14. Examples

This section details a few typical commands being processed by a single initiator and target. Note, not all packet acknowledge (acks) are shown in the diagrams to make them more readable and optional split-transactions should be avoided as much as possible to optimize performance.

14.1. Target Read Command

This example shows a read command to a simple low-cost target which can process a single command at a time with no queueing in the target. In this example the amount of data requested for transfer from the target to the initiator is sufficiently small that it can be accommodated in a single data packet. Should a larger amount of data be involved, then the target would send multiple data packets, such that one packet at a time is sent at each appropriate IEEE 1394 bus arbitration interval won by that target. The data transfer process ends when all of the requested data has been sent. The target is responsible for making appropriate changes in the destination offset address to reflect the progress made in filling the data buffer in initiator memory.



```
Ack (complete, pending,     ------>
      or, busy_X)
Block Write Response        - - ->
   (optional if initiator ack=pending)
```

Figure 15. Target Read Command

14.2. Target Multiple Read Commands

This example shows two read commands being issued to a target which can internally queue commands. Note that multiple data packets may be requiredto complete the data transfer for each command.

```
initiator            target
            ------------ initiator shoulder taps target ------------
            Block Write Packet       ------>
            Address=Normal FIFO
            Data=Address of First Command
                                 <------ Ack (complete, busy_X, or pending)
                                 <- - - -  Block Write Response Packet
             (optional if ack=pending)
            -------------- target requests 1st command ---------------
                                 <----- Block Read Request Packet
            Address=First Command Address
            Ack (e.g. pending)       ------>
            Block Read Response      ------>
            Data=Command Block #1, M_FLag=1
            -------------- target requests 2nd command ---------------
                                 <----- Block Read Request Packet
            Address=Next Command Address
            Ack (e.g. pending)                    ------>
            Block Read Response                   ------>
            Data=Command Block #2, M_Flag=0
            ----------- target returns data read for cmd #1 ----------
                                 <------          Block Write Packet
            Address=Data Address #1
            Data =Requested data
            Ack (complete, pending,               ------>
                  or, busy_X)
            Block Write Response                   - - ->
               (optional if initiator ack=pending)
            ------------ target returns status for cmd #1 -------------
                                 <------          Block Write Packet
```

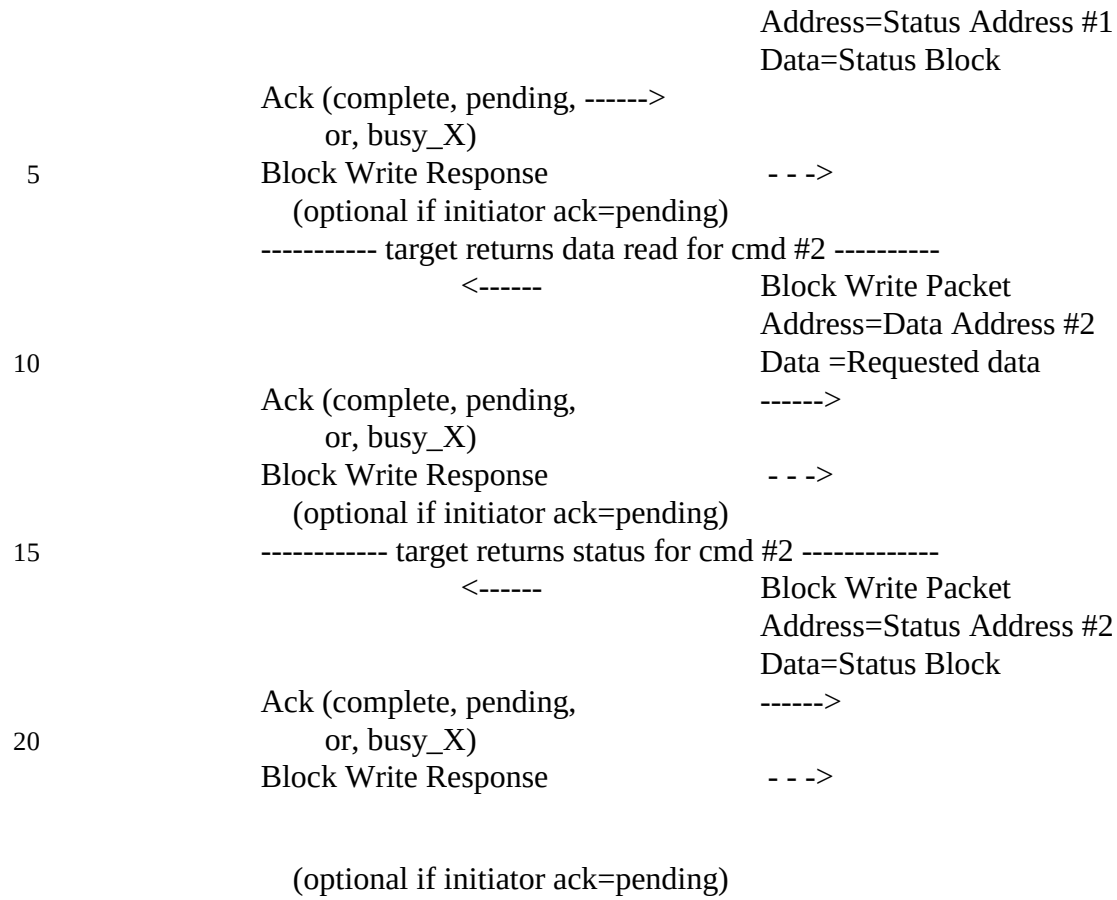


Figure 16. Target Multiple Read Commands

15. Messages

There are a number of operations, such as aborts and resets, which the initiator may wish the target to perform which are not covered by the standard command protocol. There are management aspects of the Serial Bus Protocol, such as a log-in request and a request or release of Tap Slot which are either not addressed or not completely handled by the command protocol. To support these situations, a number of "message packets" are defined. These would be sent by the initiator to the target Urgent FIFO, in a similar fashion as the "Tap" is sent to the Normal FIFO when a new command chain is ready. Message Packets are standard Block Write Request packets. The SCSI message content is conveyed in the 12-Byte payload. The response to these if a split transaction occurs would be a standard Write Response packet.

The defined message packets are:

ABORT TAG: This aborts a particular tagged command address from this Inititor.

Implementation Note: The intention is that the ABORT TAG message would be used by the initiator to emulate the Abort function in which it is desired to abort all commands from an initiator directed to some given LUN. The initiator shall determine which command blocks are to be aborted and then send to the target an appropriate ABORT TAG message for each of these command blocks.

RESET: This resets the target.

CLEAR QUE: This aborts all commands in all command chains from all initiators as are directed to a specific LUN.

Editorial Note: The Editors believe that the CLEAR QUE message is undesirable for inclusion in the Serial Bus Protocol. The only reason for its present inclusion is for concern in maintaining compatibility with the SCSI Architectural Model (SAM).

PRIORITY TAP: This acts as a Tap packet for a command block chain which is to receive priority treatment with regard to fetching of its associated command blocks.

Log-In_Request: This is a special form of Tap packet which informs the target there is the need to fetch a command block from a given initiator which contains the request for assignment by the target of a short form (8-bit) initiator Identifier which will be the principal means by which the target manages subsequent SBP control Protocol requests and selected other requests which may originate from the initiator.

FF_Control_Request: This is a formal request made by an initiator to write control information into the First Failure Control register for purpose of releasing the Associated First Failure register from a locked state in which it might presently been placed by the target.

Request/Release of Tap Slots: This is a special form of Tap packet which informs the target there is the need to fetch a command block from a given initiator which contains either (a) a request to assign or (b) a request to release some number of Tap slots maintained at that target and for the exclusive use of that initiator.

Request/Release of Asynchronous Event Notification: This is a special form of Tap packet which informs the target there is the need to fetch a command block from a given initiator which contains a request to "sign-in" for notification of asynchronous events, or (b) a request to release/end a previous "sign-in" for notification of asynchronous events.

The intent is that a target shall always be able to accept a SCSI Message Packet sent to the Urgent FIFO. A similar intent exists relative to Status block Packets sent to the to the Status FIFO at the initiator. It is expected that the frequency of SCSI Messages will be relatively low so that no major difficulty exists at the

target as far as accepting and storing the 12-Byte payload carried by the SCSI Message packet. Nonetheless, it may occur that a sufficiently large number of SCSI Message packets are sent to the target that a given SCSI Message Packet must be rejected for a brief interval of time. The target shall take design steps to ensure that a Message packet is only rejected for a short time period should rejection be necessary.

The reaction of the target when a SCSI Message packet is rejected shall be the same as the reaction of the initiator when it must reject a Status block. This reaction is to consist of one element. The target sends an IEEE 1394 acknowledgement code of "Busy_X" back to the given initiator. This acknowledgement code forces the initiator IEEE 1394 hardware to make a retransmission of the given Message packet. The intention is that the target will be able to accept the SCSI Message packet on this new transmission attempt by the initiator.

15.1. Abort Tag Payload

The function of this message is identical to the parallel SCSI "Abort Tag" message. There may be a need to consider the Reserve and Release situations if the Serial Bus Protocol decides to deal with these SCSI concepts.

Byte	0	1	2	3
0	Command Address (MSQ)			
4	Command Address (LSQ)			
8	Type_ID ABORT TAG Code = 12 Hex	Reserved		

Figure 17. Payload of Abort Tag Message

15.2. Target Reset Payload

The function of this message is identical to the parallel SCSI "Bus Device Reset" message.

Byte	0	1	2	3
0	Command Address (MSQ)			
4	Command Address (LSQ)			
8	Type_ID RESET Code = 13 Hex	Reserved		

Figure 18. Payload of Reset Message

15.3. Payload of Clear Queue Packet

The function of this message is identical to the parallel SCSI "Clear Queue" message.

Byte	0	1	2	3
0	Command Address (MSQ)			
4	Command Address (LSQ)			
8	Type_ID CLEAR_QUE Code = 14 Hex	Reserved	LUN	

Figure 19. Payload of Clear Queue Message

15.4. Priority Tap Message Payload

The function of this message is to act as a Tap packet advising the target of the existence of a command block chain containing one or more command blocks which are to receive fetching on a priority basis by the target.

Byte	0	1	2	3
0	Command Address (MSQ)			
4	Command Address (LSQ)			
8	Type_ID Priority Tap Code = 50 Hex	Reserved		

Figure 20. Payload of Priority Tap Message

15.5. Log-In_Request Message Payload

The function of this message is to inform the target that an initiator wishes to receive a short form (8-bit) identifier. The target is to fetch the referenced command block on a priority basis.

Byte	0	1	2	3
0	Command Address (MSQ)			
4	Command Address (LSQ)			
8	Type_ID Log-In Code = 60 Hex	Reserved		

Note: The Command Address is the location in initiator memory address space of the command block which carries the details of the log-in request.

Figure 21. Payload of Log-In_Request Message

15.6. FF_Control_Request Message Payload

The function of this message is to request a write operation be performed by the target to the First Failure Control register which may be implemented at that target. The reason for the write operation is to secure an "unlock" of the associated First Failure register possibly located at that target. If the unlock operation is successful, then the target is able to record new error information into the unlocked First Failure register.

Byte	0	1	2	3
0	Command Address (MSQ)			
4	Command Address (LSQ)			
8	Type_ID FF_Write Code = 61 Hex	Unlock Value Code = FC Hex	Reserved	

Figure 22. Payload of FF_Control_Request Message

15.7. Request/Release of Tap Slots Message Payload

The function of this message is to inform the target of the location in initiator memory address space of a command block which carries either:

(a) a request for allocation of Tap Slots, or

(b) a release of some number of Tap Slots previously allocated to that initiator.

The command block shall provide the target with statement if this a request or a release; and, information as to where results, status, and sense data (if needed) are to be sent the initiator.

Byte	0	1	2	3
0	Command Address (MSQ)			
4	Command Address (LSQ)			
8	Type_ID Tap_Slot Code = 70 Hex	Reserved		

Note: The Command Address is the location in initiator memory address space of the command block which carries the details of the Tap Slot request.

Figure 23. Payload of Request/Release Tap Slots Message

15.8. Request/Release of Asynchronous Notification Message Payload

The function of this message is to inform the target of the location in initiator memory address space of a command block which carries either:

(a) a request for allocation to sign-in for purposes of receiving notification of asynchronous notification at that target, or

(b) a release/end of a previous sign-in granted to that initiator.

The command block shall provide the target with statement if this a request or a release/end; and, information as to where results, status, and sense data (if needed) are to be sent the initiator.

Byte	0	1	2	3
0	Command Address (MSQ)			
4	Command Address (LSQ)			
8	Type_ID AEN Notice Code = 71 Hex	Reserved		

Note: The Command Address is the location in initiator memory address space of the command block which carries the details of the sign-in request or release.

Figure 24. Payload of Request/Release of Asynchronous Notification Message

16. Compatibility to Parallel SCSI

It has been the intention in drawing up this document to maintain as close a compatibility with parallel SCSI, as defined by the SCSI-2 specification as possible. There are the obvious changes in delivery of packets as described above but other than that it is the intention that any SCSI CDB could be delivered and any data could be sent and received with higher level microcode without the target and the initiator being aware of the change from a parallel interface to the serial.

Listed below are a number of additions that are required to the SCSI standard to support the serial interface. The rule that has been followed in drawing up this list is ALL EXISTING SCSI COMMANDS MUST WORK AS TODAY.

16.1. Relation of a Target to Multiple Initiators

In parallel SCSI various parameters are stored on a per initiator basis. For example check conditions and certain read and write parameters.On IEEE 1394 it is possible that a target may have to deal, at various times, with 65534 initiators. It clearly becomes impractical to store any parameters on a per initiator basis.

This affects the following

Power on Reset: After a power on or reset each target usually holds a check condition for each initiator. In this model it is considered a job of the IEEE 1394 management layer to notify initiators about reconfigurations in the network and hence this function is delegated to this layer. After a power on or reset a target will accept and action the first command received.

Microcode code changed/media changes/parameters changed by another initiator: There is no inherent check condition generated for each initiator under this model. If an initiator wishes to be notified of such an event then it may sign-in to receive a callback from the target to be notified of this or any other asynchronous event.

Per initiator mode sense/select pages: These are no longer supported. All parameters are now global across all initiators. In a multi initiator system it is expected that the initiators can agree on a common set of parameters.

Appendix A. Packet Formats

For purposes of convenient reference, the section describes the packet formats specified in the IEEE 1394 standard and utilized in support of the SCSI 3 Serial Bus Protocol. Ownership of these packet is acknowledged as being with the stated IEEE committee. No effort shall be made in this SCSI 3 Serial Bus Protocol document to alter these formats.

16.2. Write Packets

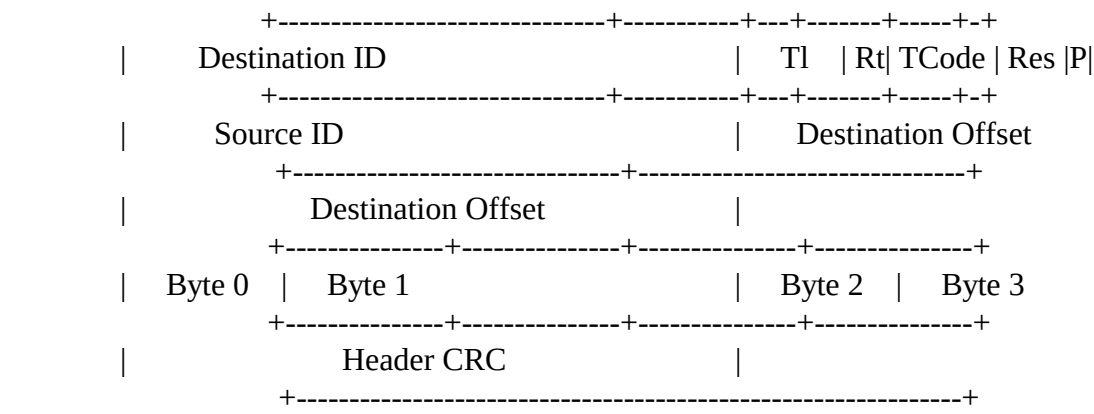
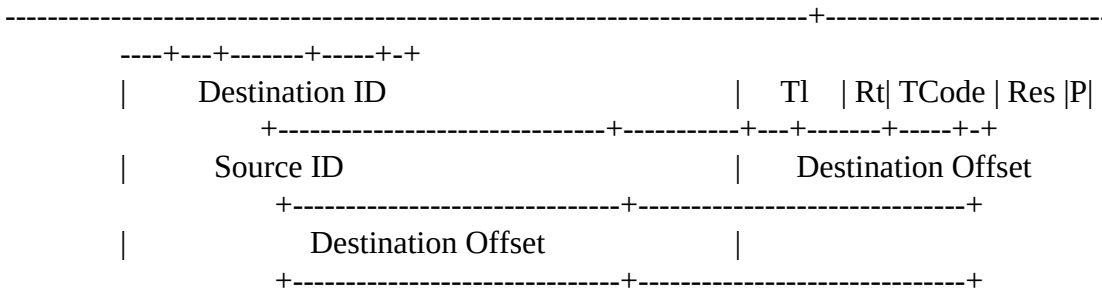


Figure 25. Quadlet Write Request Packet

NOTE: Bytes 0,1,2,3 are the subject quadlet of data.



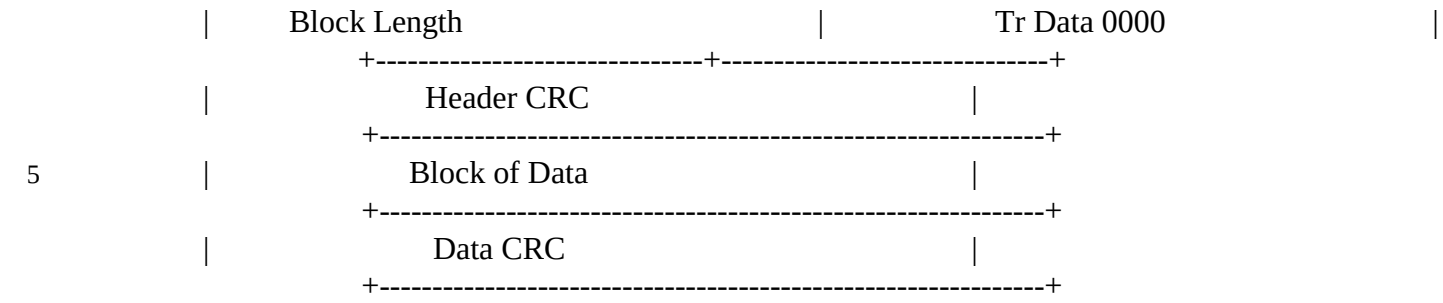


Figure 26. Block Write Request Packet

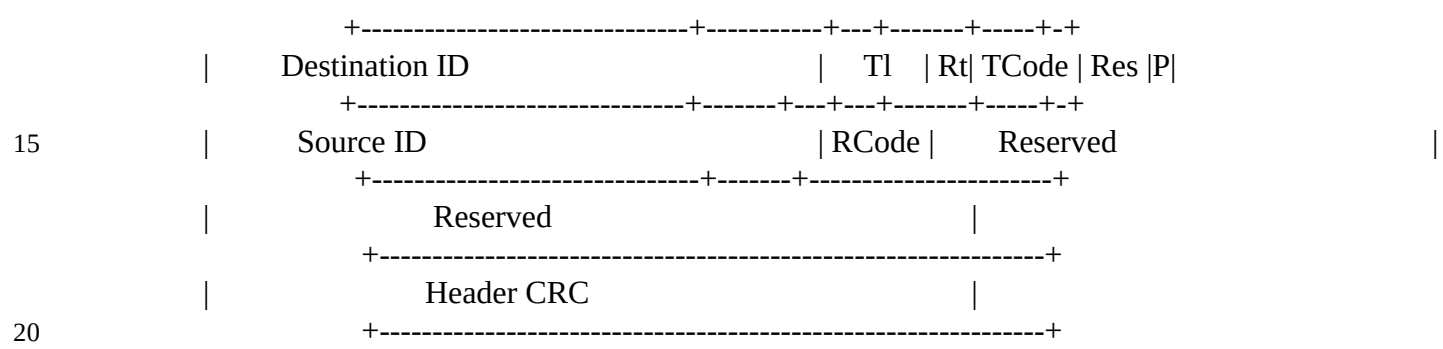
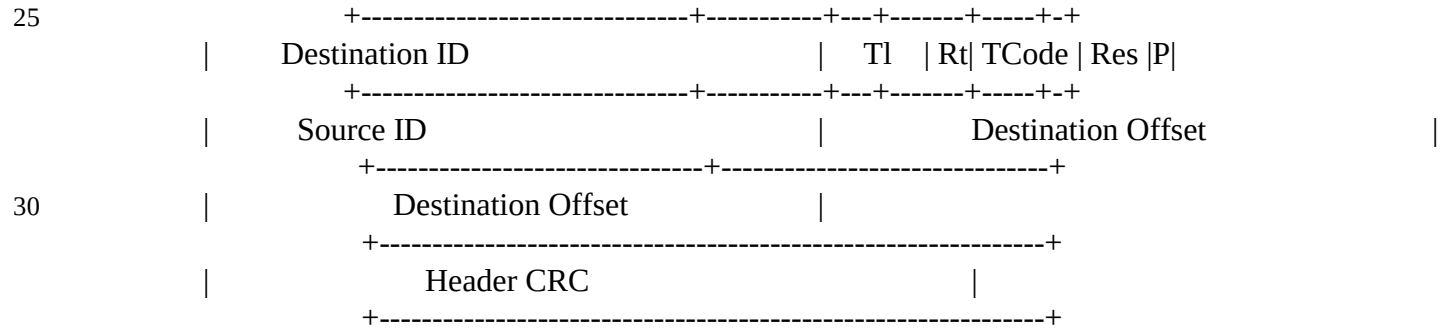


Figure 27. Write Response Packet

A.2 READ PACKETS



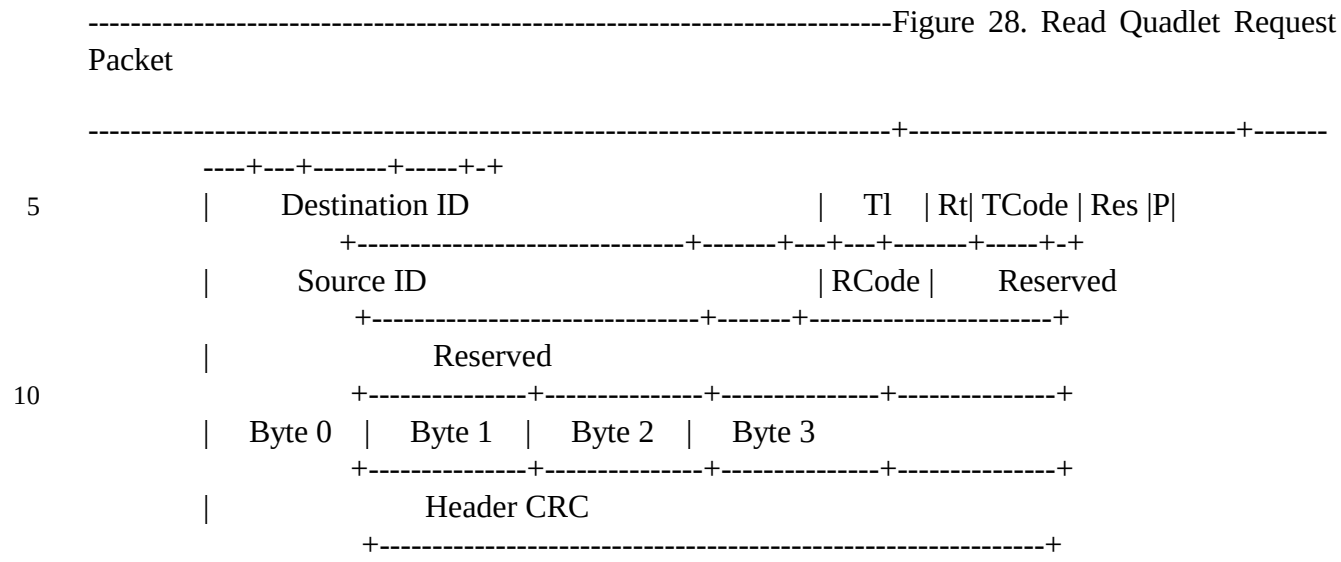


Figure 28. Read Quadlet Request Packet

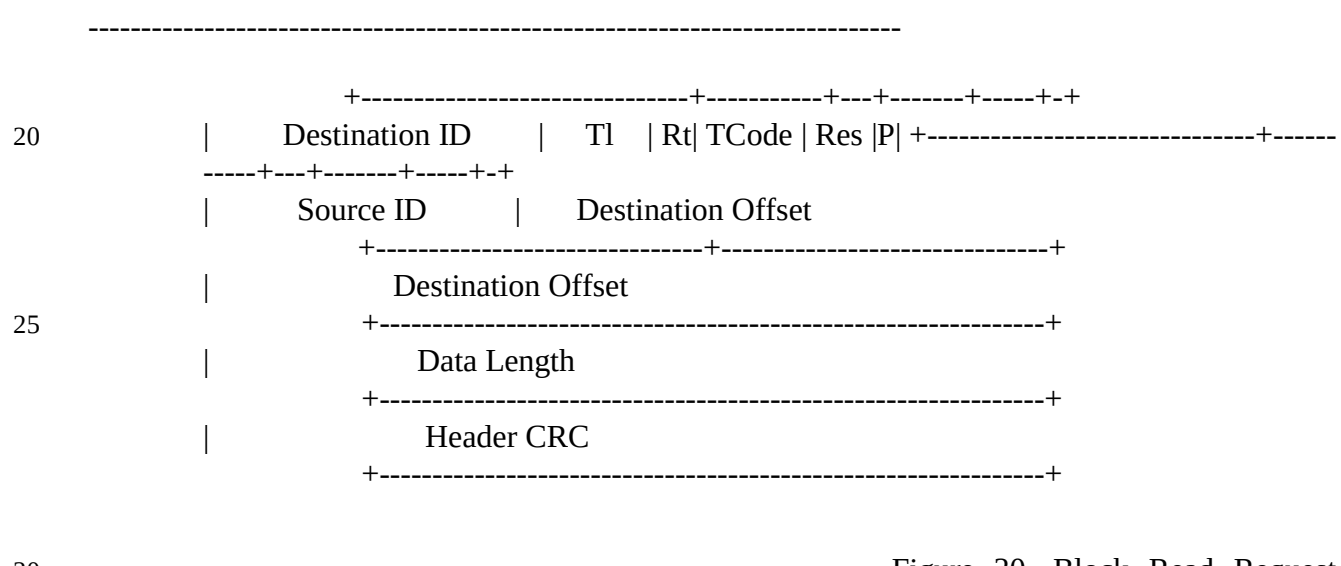


Figure 29. Quadlet Read Response Packet

NOTE: Bytes 0,1,2,3 are the subject quadlet of data.

Figure 30. Block Read Request Packet

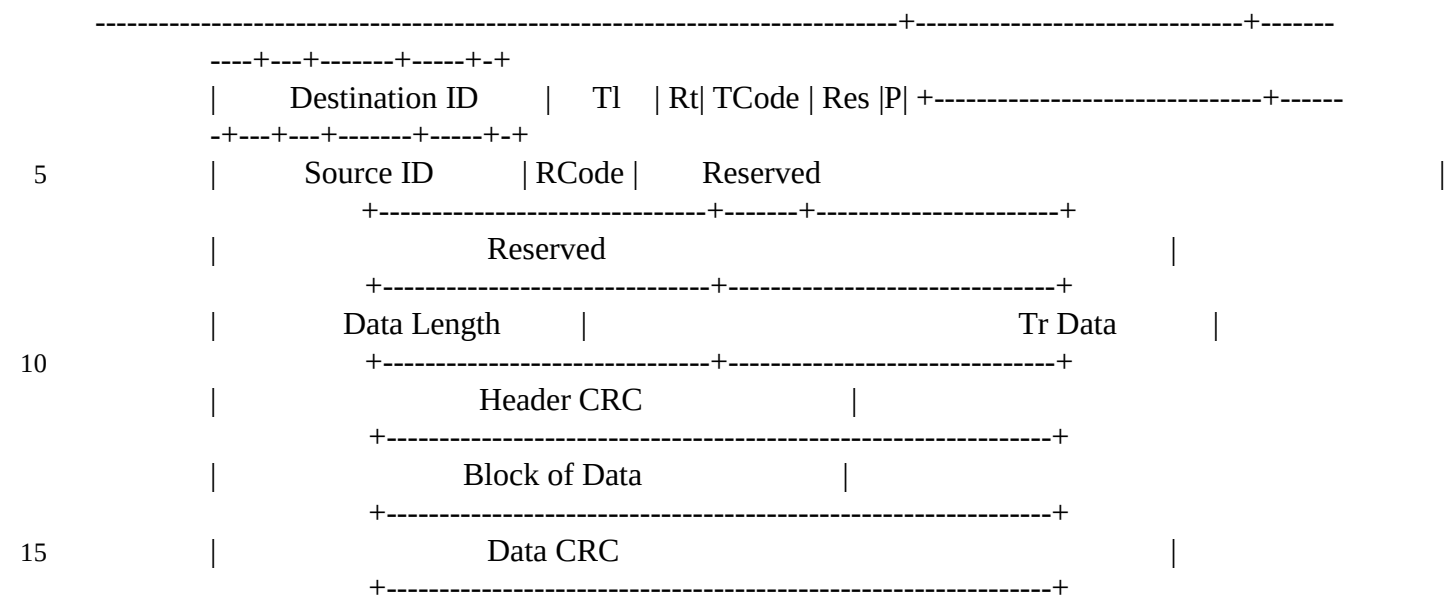


Figure 31. Block Read Response Packet

DESTINATION ID

This is the bus and node ID of the unit receiving the packet.

DESTINATION OFFSET

This is the address within the receiving node at which the data should be read from or stored into.

SOURCE ID This is the bus and node ID of the unit sending the packet.

TL This field carries the transaction label.

RT This field carries the retry code.

RCODE This field carries the response code. This code defines the type of response being returned (e.g. normal, abnormal, bad CRC, etc.)

TCODE This field carries the transaction code. This code defines the type of packet (e.g. Quadlet Write Request, Block Read Response, etc.)

R This bit field is reserved.

P This bit field is used as a priority indicator.

DATA LENGTH

This is the length of valid data that follows the header CRC field.

TR DATA (RESERVED)
A reserved field - all zeros.

HEADER CRC
The CRC code for the header information only.

DATA This field, of variable size holds the data to be transported. For a quadlet operation it is 4 bytes wide. For a block operation it is n*4 bytes in size. The Data Length field indicates the exact number of bytes of valid data in this field.

DATA CRC This CRC field protects the data within a block operation.

APPENDIX B. ISOCHRONOUS TRANSFER

Note well, it is anticipated that Apple Computer (and other parties) intend to make additional and substantial input to the description of the Isochronous data transfer process. Thus, the material below is presented as a "place holder" subject to the understanding that it is subject to significant change.

When the Isochronous Command Operations Code (bits 15 to 13 in the Flags field of the command block) indicates the command block to be carrying an Iscochronous command, then the detailed format of the command block is specified as per Appendix C following. This appendix describes the format of data packets supporting Isochronous data transfer.

All Isochronous data transfers, either read or write, use a common packet type.

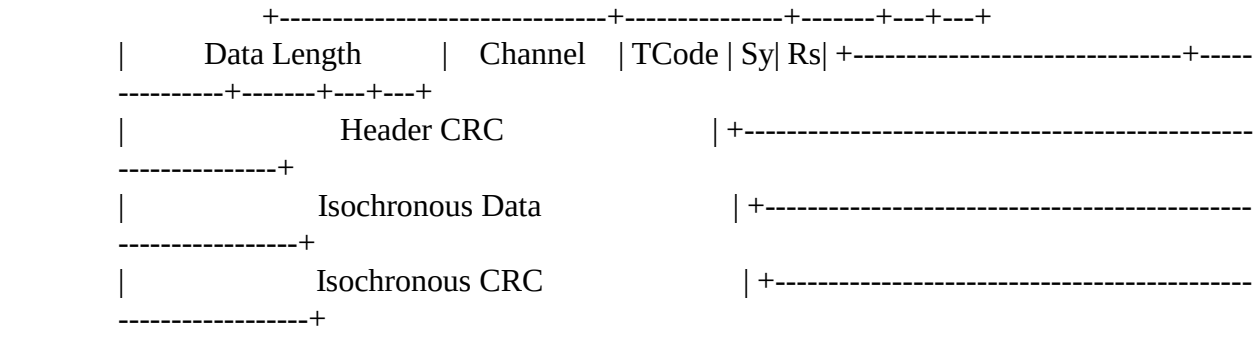


Figure 32. Isochronous packet

Each target will support a number of Isochronous channels simultaneously, varying from 0 to N. Proper and appropriate management of the Isochronous transfer facilities ensures there will be a resource available to handle the Isochronous packet.

B.1 READ FROM MEDIA - ISOCHRONOUS

This is the case where the target device is generating data and transmitting it to a remote device. Under these circumstances the target is responsible for scheduling the data reads

from the media and sending the data in the appropriate channel. For this type of transfer all the normal error checks performed for an asynchronous read must be carried out but in addition a check must be made for any underrun conditions, i.e. if the target device

fails to supply data in the appropriate time slot.

B.2 WRITE TO MEDIA - ISOCHRONOUS

In this case the data will be transmitted to the target by another device.

The target has no means to pace the data and must accept every Isochronous packet received

on its channel.

APPENDIX C. ISOCHRONOUS COMMAND FORMATS

Please note that the primary point of contact for the material contained in this appendix is:

Scott Smyers
Apple Computer
3535 Monroe Street, MS 69G
Santa Clara, California 95051
Voice: (408) 974-7057
FAX: (408) 974-2898
email: smyers.s@applelink.apple.com

C.1 SCOPE

This appendix defines the format of Isochronous commands and Isochronous control registers for the Serial Bus Protocol. An initiator uses these commands to control a device which is sourcing or sinking an Isochronous stream of data through use of the Isochrononos data transfer facility of the IEEE 1394 High Speed Serial Bus.

The following standards documents contain additional information which is useful for understanding the subject environment:

IEEE P1394 Standards Document

SCSI-3 Serial Bus Protocol

C.2 DEFINITIONS

Below are definitions of some terms which are used freely throughout this appendix:

ISOCHRONOUS DATA STREAM

Data which is carried in Isochronous data packets which all have the same Isochronous channel number, occur on consecutive Isochronous cycles and have a length of zero or more bytes.

LISTENER — An IEEE P1394 target which is receiving, or sinking, one or more Isochronous data streams.

TALKER — An IEEE P1394 target which is transmitting, or sourcing, one or more Isochronous data streams.

ISOCHRONOUS CONTROL REGISTER (ICR)
A register which must be implemented in a target capable of Isochronous data transfer. This register controls the starting and stopping of Isochronous data to or from the target.

C.3 OPEN ISSUES

The information presented in this appendix is not yet complete. There remain significant open issues. While the behavior that is intended for Isochronous disk drives is mostly understood, the details necessary to coax that behavior out of the commands and registers defined here has not been completely resolved. This section discusses the identified open issues. Work continues regarding resolution of these topics. The editors encourage those readers with opinions to come forward and offer any constructive input they may have.

One open issue has to do with the type of data that is envisioned to be carried Isochronously. Specifically, the Isochronous data transport mechanism is intended to transport Isochronous "objects" continuously from a single talker to one or more listeners. An Isochronous object is a fixed size, and it may be one sample from each channel of a stereo sound source, or one frame of a video, plus associated digitized sound.

Only rarely and by complete coincidence will an Isochronous object fall on a logical block boundary of a disk drive. However, it is extremely important, if not essential, that a controlling initiator be able to force a disk drive to source an Isochronous stream data beginning at an object boundary. In order to do this, the Port_Append command must have addressing granularity of one Byte. The Port_Append command uses a SCSI CDB to address media data, and a SCSI CDB can not address media data on a Byte boundary. In addition, the Isochronous Control Register described below will need to pause on a Byte boundary. While this capability is present in the control register, there remains some details to be worked out.

C.4 OVERVIEW

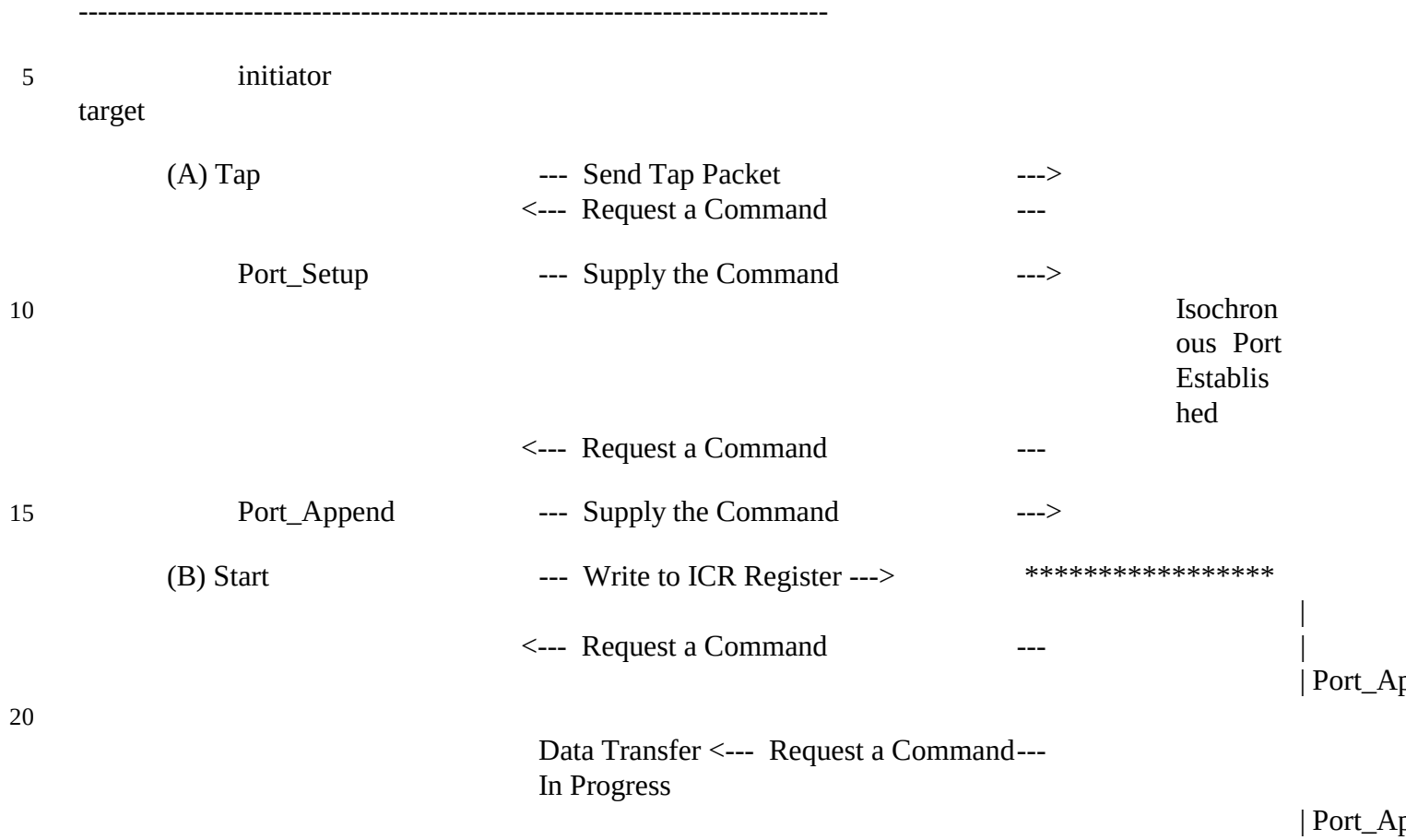
This section contains a brief overview of Isochronous data transfer and describes the control flow that the initiator and target participate in during the Isochronous setup, data transfer and teardown operations. Note there is a more complete overview in Appendix D, entitled "Isochronous Streams for the IEEE Serial Bus devices".

| The figure below illustrates the situation of a single initiator and a single
| target which is a disk drive. The figure depicts two different kinds of flow
| between the given initiator and the given target. One of these flows is the
| control flow which is shown in high-level form,and in an explicit fashion.
| The other flow is the flow of Isochronous data which is shown only in concep-
| tual form in the right-hand margin of the figure. No attempt is made in the
| conceptual flow to show any of the Isochronous packets involved. For sim-
| plicity reasons, the flow of Isochronous data is shown conceptually as a time
| period during which prerequisite control information has been established at
| the target by means of a appropriate write operation to the Isochronous
| Control Register located at that target.

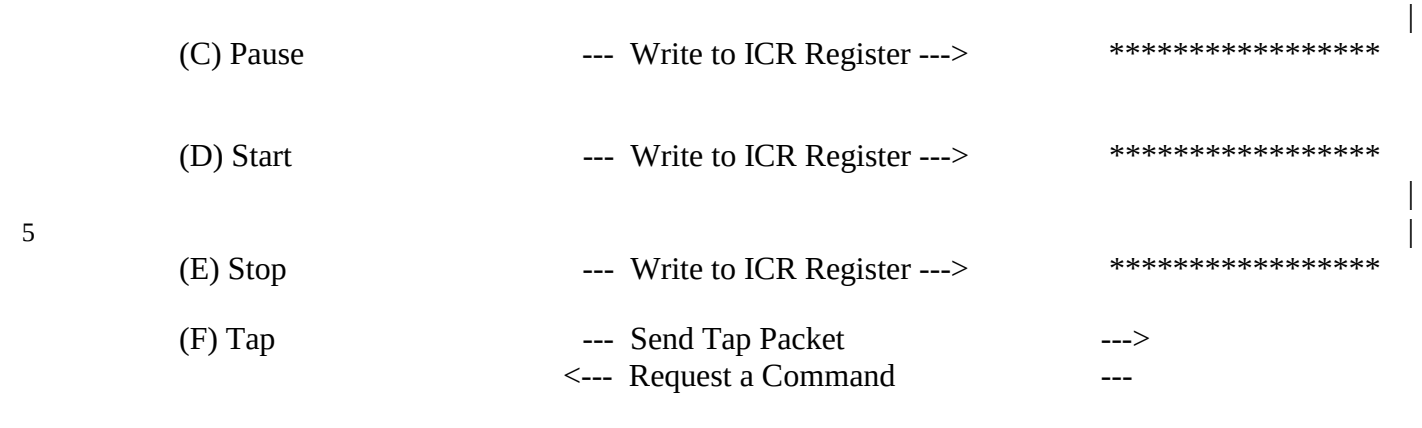
| Emphasis in the diagram is upon the control flow. The Isochronous Port_Setup | command supplies the target with certain needed configuration type informa| tion. At the end of the flow, the Isochronous Port_TearDown command deini| tializes the given port. During the main part of the flow, the initiator

| uses the Port_Append command to maintain an Isochronous stream of data to or | from discontiguous areas of target address space. Each Port_Append command | describes a contiguous

area of the device media. The Port_Append command | does not instruct the target to begin or halt transferring Isochronous data. | This function is provided by the Isochronous Control Register (ICR) which is | described in a following section.



Write Packets



----------Figure 33. Isochronous Control and Data Flow

The letters in the figure correlate with the text below:

(A) The initiator taps the target to inform it that there are some commands available for it to fetch. In response, the target requests the first command, which is a Port_Setup command. The target configures the selected port to transfer data according to the parameters contained in the Port_Setup command. The Port_Setup command exists in a chain with three Port_Append Commands. In anticipation of transferring

Isochronous data, the target requests the next command in the chain, which happens to be a Port_Append command.

(B) At this step, the initiator tells the target to start sourcing or

sinking Isochronous data. It does so by writing to the Isochronous Control Register. In response, the target begins transferring Isochronous data. As it proceeds, the target fetches additional Port_Append commands, as needed to maintain the stream of Isochronous data.

(C) The target, after transmitting Isochronous data for a while, continues fetching Port_Append commands as needed. Then, the initiator issues a Pause request. It does this by writing to the Isochronous Control Register. The target responds by pausing the transfer of Isochronous data.

(D) The initiator again issues a Start request to the target. The target then resumes transferring Isochronous data. In the situation illustrated, the given Isochronous data comes from the same area of media at the target as was the case data transfer prior to the pause. Again, in the situation illustrated, the given area of target media contains all the data needed for the duration of the Isochronous transfer. Consequently, no additional requests for a Port_Append command are needed.

(E) The initiator issues a Stop request to the target by writing a value to the Isochronous Control Register. The target responds by halting all transfer of Isochronous data on the designated port.

(F) The initiator sends a Tap packet to the target indicating that a command is available. The target fetches the command, discovers that it is a Port_TearDown and responds by de-initializing the designated port.

The above discussion covers the entire "life" of an Isochronous port. The remainder of this appendix describes the three Isochronous commands (Port_Setup, Port_Append and Port_TearDown) and the Isochronous Control Register.

C.5 OBJECTS AND OBJECT BOUNDARIES

When a node is configured to send or receive Isochronous data, it is given an object size. The object size defines a periodicity to the Isochronous data. For example, consider a talker configured to send 4 bytes per Isochronous cycle with an object size of 6 bytes. Periodically, the boundary of a 6 byte object will fall on an Isochronous cycle boundary, as depicted in the figure below:

Figure 34. Isochronous Boundaries Example

Cycle N begins on an object boundary, cycle N+1 begins at some offset within an object and cycle N+2 happens to end on an object boundary. Not shown in this figure is cycle number N+3, which begins on an object boundary.

The Isochronous talker marks each Isochronous cycle which begins on an object boundary by setting a bit in the header of the Isochronous data packet transmitted on that Isochronous cycle. In the example above, cycles N and N+3 would be so marked, because those cycles begin on object boundaries.

The object which begins on an Isochronous cycle boundary and which is flagged with a bit in the Isochronous header is called a marked object. The Isochronous listener uses marked objects to resynchronize in the event that data was missed, either due to a faulty transmission or an internal error in the listener or talker. Both the Isochronous talker and listener use their awareness of object boundaries, marked or not, to pause Isochronous data transmission on object boundaries.

C.6 ISOCHRONOUS COMMANDS

This section defines all Isochronous commands. An initiator supplies these commands to a target per the request of the target in conformity to the Serial Bus Protocol for command delivery.

EDITORIAL NOTE: No reasons have been identified which prevent Isochronous command blocks from being mixed in the same chain with asynchronous command blocks. The two types of command block are in the same general format. The value of the SBP Function Code provides means to distinguish the Isochronous command block from the general asynchronous command block.

Isochronous commands reserve resources in the target and steer Isochronous data to or from the target's address space. The one Isochronous command which transfers data carries a SCSI CDB which addresses media data. For Isochronous commands which do not result in data transfer,the command block does not carry a SCSI CDB.

The following table shows the general format of an Isochronous command. An Isochronous command is identified by selected values within the SBP Function Code field.

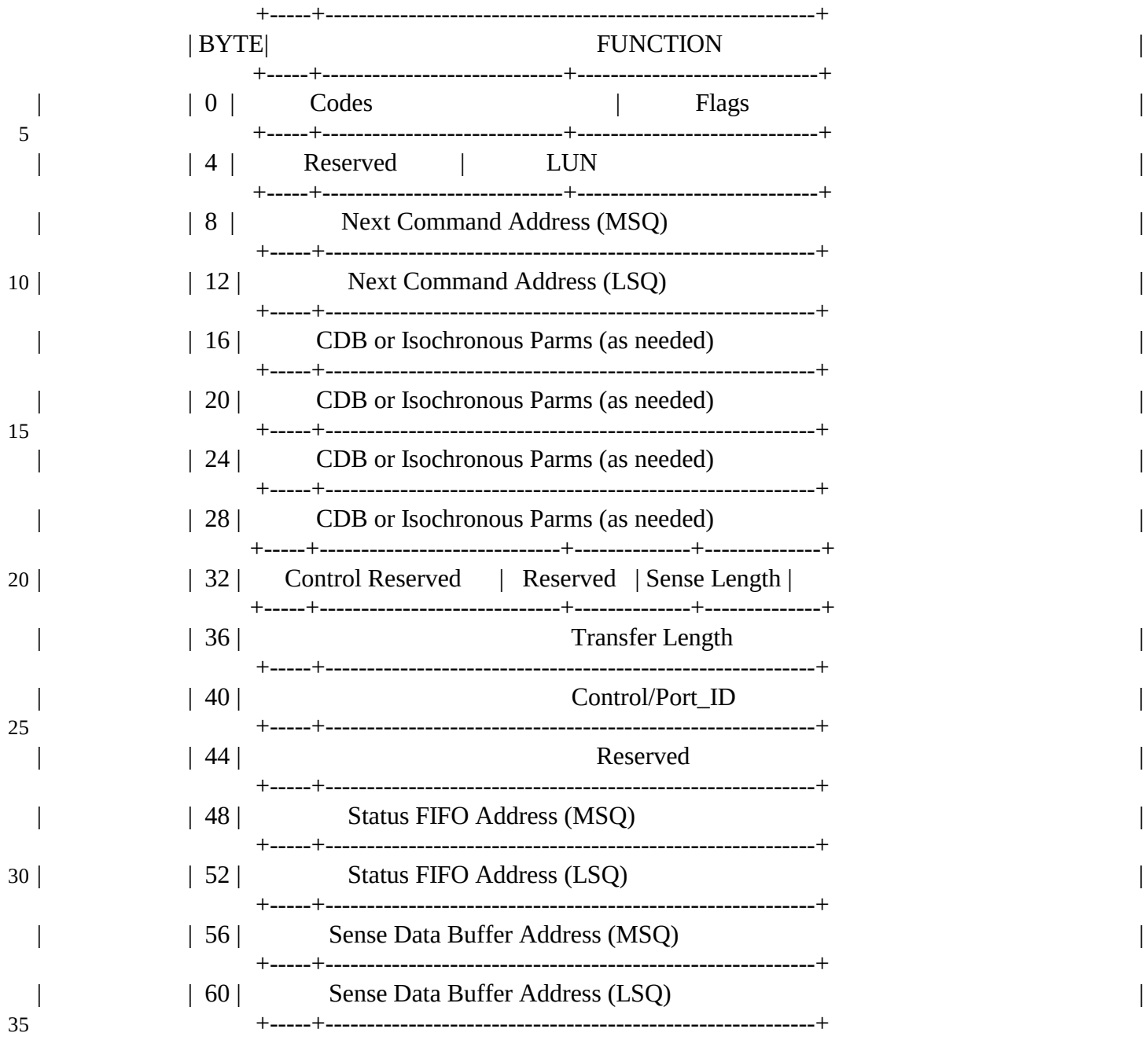


Figure 35. General Format of Command Block for Isochronous Commands

C.6.1 PORT_SETUP COMMAND

The Port_Setup command reserves resources in the target in anticipation of transferring data Isochronously. The Port_Setup command establishes an association between these reserved

resources and the 16 bit port identifier contained in the Port_ID field. The Port_Setup command creates a port and the Port_TearDown command destroys a port.

The table below defines the format of this command.

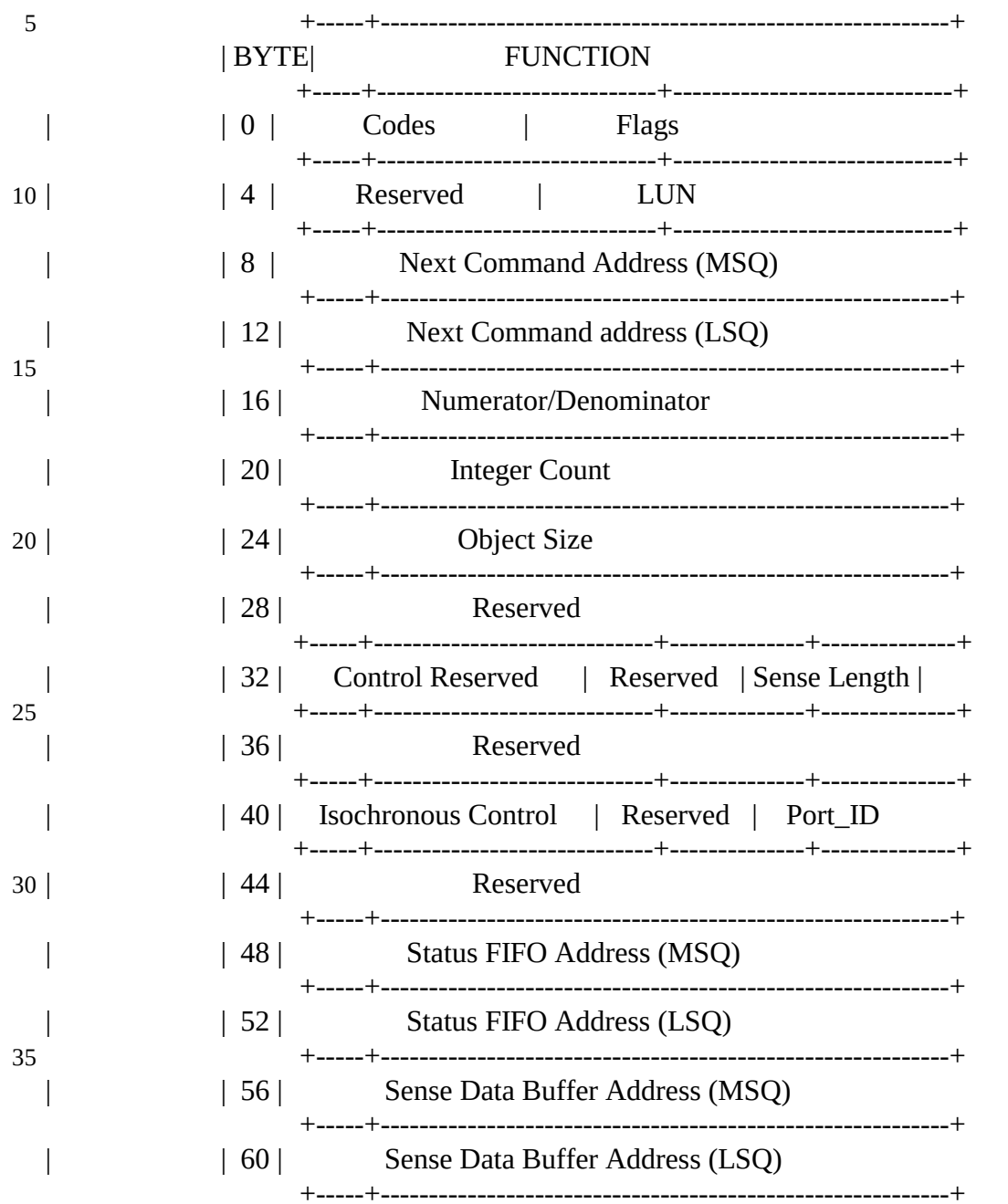


Figure 36. Command Block Format for the Isochronous Port_Setup Command

The following description of terms is given only for those items items which are unique to the given Isochronous command.

All other items are the same as previously described for the command block associated with an asynchronous command.

FIELD FUNCTION

NUMERATOR/DENOMINATOR

The
defin
Isoch

form

BIT(S)	NAME	FUNCTION
31 (MS) to 16	Denominator	These 16-bits are the Denominator term (D) in the alogrithm described in the following section, entitled "Discussion of Isochronous Data Transfer Algorithm".
15 to 0 (LS)	Numerator	These 16-bits are the Numerator term (N) in the alogrithm described in the following section, entitled "Discussion of

Figure 37. Numerator/Denominator Field in the Isochronous Port_Setup Command

INTEGER COUNT

The 32-bit Integer Count field along with the the Numerator/Denominator field define the rate at which the target is to transmit or receive Isochronous data on this port.

OBJECT SIZE

The 32-bit Object Size field contains the size, in bytes, of the objects that the talker is to transfer Isochronously. The talker uses this size to stop on an object boundary and to set the marked object bit in the Isochronous header. The listener uses the object size to maintain synchronization with the talker and to stop on object boundaries.

ISOCHRONOUS CONTROL

This 16-bit field contains bits which establish the speed, direction and error handling modes for this port. The Isochronous Control field currently has the following bits defined. All bits not specified are reserved.

BIT(S)	NAME	
15 (MS)	Talker/ Listener	source(talk) or sink(listen) Isochronous data. Value 0 means the target is a listener. Value 1 means the target is a talker.
14		nition by this standard. The initiator shall always set this bit to zero and the target shall ignore the setting of this bit.
13 to 12	Port Speed	This field determines the bus speed at which the Isochronous data is to be transmitted according to the following: Value 00 means 100 Megabits per second. Value 01 means 200 Megabits per second. Value 10 means 400 Megabits per second. Value 11 is reserved. NOTE: This is the signalling rate at which the target is to transmit Isochronous packets on this port. The aggregate rate

			of data transfer in terms of sustained number of Bytes per unit time is controlled with other parameters in the Port_Setup command.
	Reporting		11 to 10 | Error strategy for Isochronous data transfer on this port. These error reporting strategies are defined elsewhere in this appendix. Value 00 means stop on any error. Value 01 means log error and continue. Value 10 means ignore all errors. Value 11 is reserved.

9 to 8	Continue Mode		Under the condition that the target is programmed to not stop on any error, this field determines how missing cycles of Isochronous data are to be treated.
7 to 0 (LS)	Reserved		These bits are reserved.

Figure 38. Port_Setup Isochronous Control Field

RESERVED This 8-bit field is reserved.

PORT_ID This field contains the identifier for the Isochronous port that the target is to create as a result of executing this command. Following the successful completion of the Port_Setup command, all commands having the same value in the Port_ID field affect this port. This port exists until the successful completion of a Port_TearDown command having the same value in the Port_ID field.

C.6.1.1 Discussion of Isochronous Data Transfer Algorithm

An Isochronous talker transmits Isochronous packets at a rate of 8,000 packets per second for a single channel. Each of these packets carries an integer number of Bytes. In order to manufacture isochronous rates which are not multiples of 8,000 bytes per second, a talker must send Isochronous packets of more than one size.

The parameters above describe to the Isochronous talker the rate (R) which is desired in units of bytes per Isochronous cycle. This rate is calculated from the parameters in the above fields according to the following equation:

$$R \text{ (Bytes per Isochronous cycle)} = I + N/D$$

Where I, N and D are all non-negative integers and N is less than D. If N is zero, D is ignored. The number of Bytes per second that this translates into is calculated by multiplying R by 8,000.

As an example, let's assume that an Isochronous talker is being configured to carry 1 channel of digital audio Isochronously. Assume that this channel requires 1 Byte at 44.1 KHz. The necessary Isochronous rate, then, is:

| (44,100 samples/sec) * (1 Bytes/sample) = 44,100 Bytes/second

| which is described as:

| (5 + 41/80) Bytes per Isochronous cycle
| The Isochronous talker would fabricate this rate by sending Isochronous
| packets of length 5 Bytes alternating with Isochronous packets of length 6
| bytes. Every 80 Isochronous cycles, the talker would transmit an extra
| Isochronous packet of length 6 bytes. With this algorithm, the number of
| Bytes transmitted over the course of 80 Isochronous cycles is exactly what is
| required to carry the data. A listener to this stream of data would not have
| to buffer more than two Bytes of data at any given time.

C.6.2 PORT_TEARDOWN COMMAND

The Port_TearDown command instructs the target to destroy a port and release all resources in the target which were reserved for that port. The port must first have been created with the Port_Setup command described previously.

The table below defines the format of this command.

BYTE	FUNCTION		
0	Codes	Flags	
4	Reserved	LUN	
8	Next Command Address (MSQ)		
12	Next Command address (LSQ)		

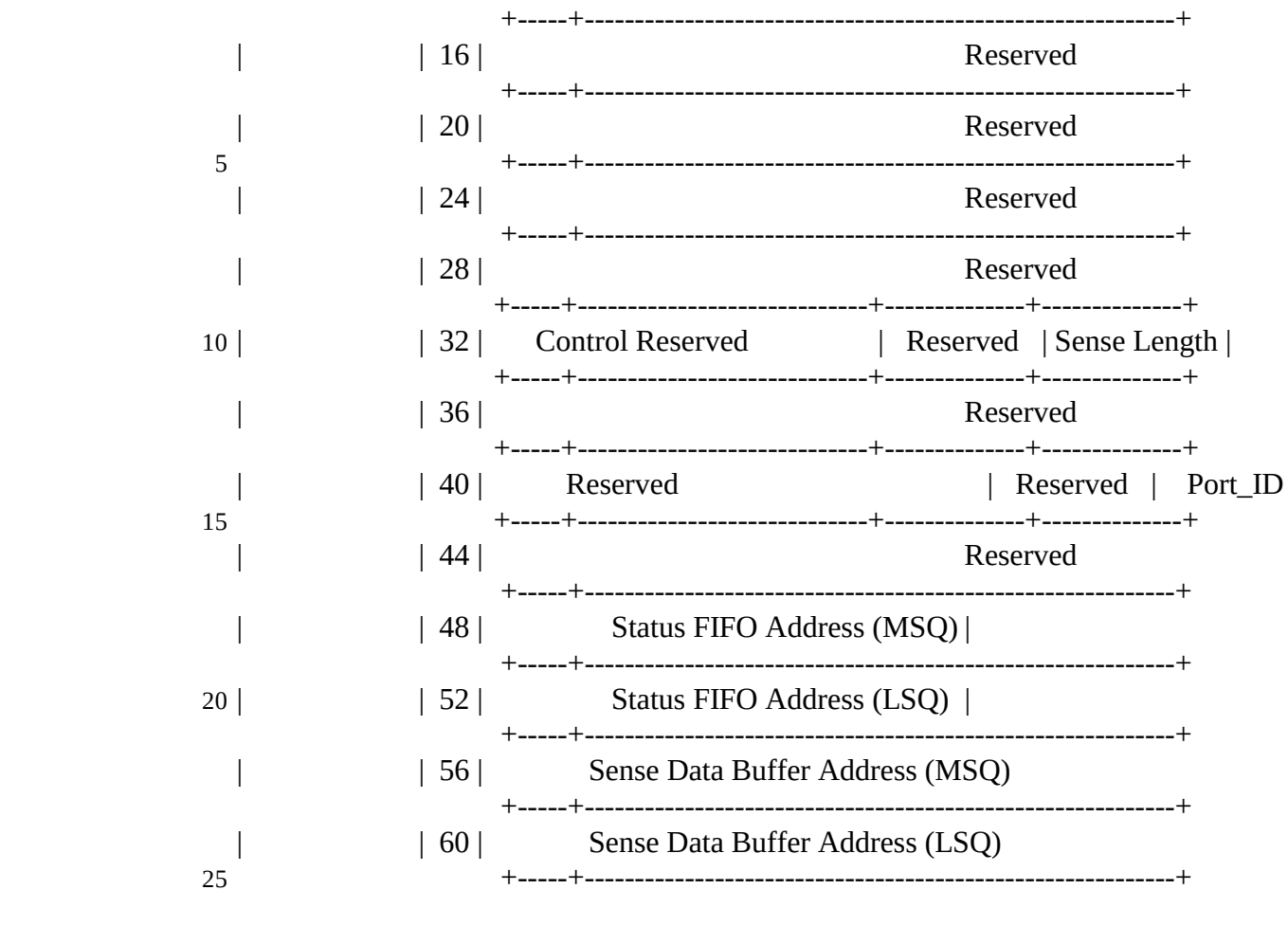


Figure 39. Command Block Format for the Isochronous Port_TearDown Command

The following description of terms is given only for those items items which are unique to the given Isochronous command. All other items are the same as previously described for the command block associated with an asynchronous command.

FIELD FUNCTION

PORT_ID This 8-bit quantity is the identifier of the Isochronous port that the target is to destroy as a result of executing this command. In order to successfully execute the Port_TearDown command, the port with the same Port_ID must first have been created using the previously described Port_Setup command.

C.6.3 PORT_APPEND COMMAND

The Port_Append command provides the target with SCSI CDB's which the target uses to steer the Isochronous data to or from locations on the media. The initiator uses the Port_Append command to maintain an Isochronous stream of data to or from discontiguous areas of target address space. Each Port_Append command describes a contiguous area of the device media.

The following table lists the information which this command carries from a controlling initiator to a target.

BYTE	FUNCTION		
0	Codes	Flags	
4	Reserved	LUN	
8	Next Command Address (MSQ)		
12	Next Command address (LSQ)		
16	CDB Bytes 0 to 3		
20	CDB Bytes 4 to 7		
24	CDB Bytes 8 to 11		
28	CDB Bytes 12 to 15		
32	Control Reserved	Reserved	Sense Length
36	Transfer Length		
40	Reserved	Reserved	Port_ID
44	Reserved		
48	Status FIFO Address (MSQ)		
52	Status FIFO Address (LSQ)		
56	Sense Data Buffer Address (MSQ)		

		60	Sense Data Buffer Address (LSQ)

Figure 40. Command Block Format for the Isochronous Port_Append Command

The following description of terms is given only for those items items which are either unique to the given Isochronous command, or require special dis-

cussion in the context of the given Isochronous command. All other items are the same as previously described for the command block associated with an asynchronous command.

FIELD FUNCTION

CDB (SCSI COMMAND DESCRIPTOR BLOCK)
These four quadlets contain a SCSI read or write CDB which carries the starting Logical Block Address and the Logical Block Count for this command.

NOTE: When the target has transferred this data Isochronously (either sourced it or sinked it), there must be an additional Port_Append Isochronous command available to the target for this same Port_ID in order for the target to avoid gaps in the data.

TRANSFER LENGTH
The 32-bit Transfer Length field contains the number of bytes that will be transferred as a result of executing this command. Note that this is the same definition that this field has for normal asynchronous commands.

RESERVED QUADLET
The quadlet is reserved.

RESERVED DOUBLET
The doublet is reserved.

PORT_ID This 8-bit quantity is the identifier of the Isochronous port that the target is to use for data transfer when executing this command.

NOTE: This port must have already been created with a Port_Setup command and data transfer must have already be started via the Isochronous Control Register (ICR) before a Port_Append command can be executed.

C.7 ISOCHRONOUS CONTROL REGISTER

The Isochronous Control Register (ICR) is an 8 byte register contained in the target's CSR address space. The ICR is a read/write register that shall only be accessed using an 8 byte IEEE P1394 read or write bus transaction. Note that these bus transactions are indivisible with respect to all other nodes on the same IEEE 1394 High Speed Serial Bus.

All targets supporting Isochronous data transfer must implement the ICR. The address of the ICR is contained in the target's ROM data structure.

The ICR controls the flow of Isochronous data that the target sources or sinks. A single write to the ICR causes the target to begin or halt the sourcing or sinking of Isochronous data, according to the Port_Setup and one or more Port_Append commands which must have already been delivered to the target. The value written to the ICR determines the requested action (start, stop, or pause). The requested action can be effective immediately, or on a particular cycle number or other stream event.

When the target is a talker and is told to pause, the target waits for the programmed event, then transmits zero length Isochronous data packets. When the target is a listener, a pause instructs the target to temporarily stop recording a stream of Isochronous data.

The start function tells the target to either begin or continue talking or listening at the place where it was paused. Neither the start nor the pause functions affects the state of the stream of data. Upon resuming, the target continues where it left off.

The figure below shows the layout of the ICR:

+-----------+----------------+---+ | BIT(S) | NAME
+-----------+----------------+---+ | 63 (MS) | Action
| to 60 | | which informs the target the action that it |

			is to perform, according to the following:
			VALUE ACTION DEFINITION
			0 Stop sourcing/sinking Isochronous data and enter the stopped state. If the target is sourcing data, then the target sends an end of stream packet on the Isochronous cycle on which this action takes affect. The target does not send out zero length Isochronous packets while in the stopped state.
			1 Pause sourcing/sinking Isochronous data and enter the paused state. If the target is sourcing data, then the target is to continuously send zero length Isochronous packets while in the paused state.
			2 Start sourcing/sinking Isochronous data on the Port_ID using the Isochronous channel number contained in the Channel Number field.
			3 TO 15 Reserved

			59 to 58	Stream Event
			the requested action is to take place on a certain event, according to the following encoding:

		VALUE	STREAM EVENT DEFINITION
		0	As soon as possible, i.e., immediately.
		1	Perform the requested action on a specific Isochronous cycle number. The Cycle Number field contains the number of the Isochronous cycle on which the action is to take affect.
		2	Perform the action on receipt of either a start of stream packet (for the start action) or an end of stream packet (for the stop or pause actions).
		3	Reserved

			57 to 56	Reserved
			55 to 40	Byte_Offset
		and pause actions. This field contains the number of Bytes that the target is to source or sink on the Isochronous cycle on which the stop or pause actions take affect.	
			39 to 32	Port_ID
		port that the target is to perform the requested action on.	
			31 to 12	Cycle_Number
		the Isochronous cycle on which the requested action is to be performed, if the designated event is "Cycle Number".	
			11 to 8	Reserved
			7 to 0	Channel_Number (LS)
This 8-bit field is only used for the start		the Isochronous cycle that the target is to use while sourcing or sinking Isochronous data.	

Figure 41. Structure of the Isochronous Control Register

C.8 ISOCHRONOUS STATUS REPORTING

This section describes status reporting for the Port_Append command only. This command is the only Isochronous command which results in data being transferred to or from the target.

There are three status reporting modes for Isochronous data data transfer commands. The Port_Setup command declares the status reporting mode for a port. The mode remains in effect for the entire life of the port.

The three modes differ in the way that the target handles error conditions during Isochronous data transfer. The three modes are:

IGNORE AND CONTINUE target ignores all errors and continues Isochronous data transfer.

REPORT AND CONTINUE target logs errors as they occur and continues Isochronous data transfer.

REPORT AND STOP target reports the first error that occurs and stops Isochronous data transfer.

The following sections describe these operational modes.

C.8.1 MODE "IGNORE AND CONTINUE"

The target generates no status during operation. Upon encountering any error while sourcing or sinking Isochronous data, the target ignores the error and continues. There is no status block associated with this mode of operation.

C.8.2 MODE "REPORT AND CONTINUE"

In the "report and continue" mode of operation, the target generates a running log of errors which occur during execution of an Isochronous data transfer command. The target writes this log to the sense buffer whose start address and length are contained command block. Upon command completion, the target reports either success, to indicate that no entries were generated in the log, or an appropriate error code to indicate that the target generated one or more entries in the error log during command execution.

If the target completely fills the allocated sense buffer, the target stops writing error log entries, but continues executing the command to completion. Upon completing the command, the target reports a status code indicating that the sense buffer is filled with error log entries and some entries were discarded due to overflow.

Upon encountering an error, the target's operation is affected by the setting of the "Ok to Continue" bit in the flags field of the command entry. The

target behaves as follows:

- If the "Ok to Continue" bit is set to zero and the target encounters errors during execution of a command, the target logs all errors as described above. When the command completes, the target stops processing commands in the Isochronous chain until told to continue by way of the ICR.

- If the "Ok to Continue" bit in the flag field is set to one and the target encounters errors during execution of a command, the target logs all errors as described above and continues processing commands in the Isochronous list.

| EDITORIAL NOTE: Definition is not yet complete with regard to the "OK to Continue" bit, nor has a location been established for it. Present intention is to support this bit as a Flag in the Isochronous Control field appearing in the Isochronous Port_Append command.

Each entry in the error log reports an event. There are two defined events: (1) beginning of a stream gap, and (2) end of a stream gap. A stream gap is defined as one or more consecutive Isochronous cycles during which no data was transmitted or received on a given channel number. The start of a gap is signified by one Isochronous cycle during which an Isochronous data packet for a given channel is not generated (due to a talker problem) or is not received (due to an Isochronous header or Isochronous data field CRC error, or missing packet). The end of a gap is signified by the first Isochronous cycle following the start of a gap in which a valid Isochronous data packet is transmitted or received for a given channel number. Notice that a stream gap may appear at one device and not another.

As an example, if an Isochronous stream is interrupted for multiple consecutive Isochronous cycles and then resumed, the target will log two entries to the error log regardless of the duration of the stream gap: One error log entry to report the start of the start of the stream gap, and another error log entry to report the end of the stream gap.

Each entry in the error log is one quadlet long. The format of the error log

entry is:

BIT(S)	NAME	FUNCTION
31 (MS) to 12	Cycle Number	Isochronous cycle at which the event occurred.
11 to 0 (LS)	Status Code	which occurred on the reported Isochronous cycle number according to the following table: VALUE STATUS CODE 0 Cycle number of the first cycle in the stream gap. 1 Pause sourcing/sinking Isochronous data and enter the paused state. If the target is sourcing data, then the target is to continuously send zero length Isochronous packets while in the paused state. Cycle number of the first valid cycle of data following the stream gap. ALL OTHER CODES Reserved

Figure 42. Format of the Error Log Entry for an Isochronous Activity

The target's error log contains one entry for each start of gap detected and one entry for each end of gap detected (i.e., two error log entries per stream gap). In the case where the target is the

talker, the target reports errors to the error log if it is unable to generate Isochronous packets for whatever reason. If the target is a listener, the target reports error to the error log if it is unable to receive a packet, due to a missing packet, or a data or header CRC error.

C.8.3 MODE "REPORT AND STOP"

In this operational mode, upon encountering an error, the target generates a status report and immediately stops execution of the current command. The setting of the "Ok to Continue" bit in the flag field of the command entry affects the target's next action as follows:

- If the "Ok to Continue" bit is set to zero, the target stops processing commands for that Isochronous port until told to resume by way of the ICR.
- If the "Ok to Continue" bit is set to one, the target continues processing subsequent commands in the Isochronous chain.

APPENDIX D. ISOCHRONOUS DATA STREAMS FOR IEEE 1394 SERIAL BUS DEVICES

Please note that the primary point of contact for the material contained in this appendix is:

Rob Lash
Apple Computer
Cupertino, California
Voice: (408) 974-3889
FAX: (408) 446-9154

D.1 INTRODUCTION

This document describes an Isochronous data stream on Serial Bus and functional requirements of node and unit implementations on the bus. It addresses issues relevant to the transaction layer and below. An attempt is made not to stray above the transaction layer boundary. The concepts developed within are Isochronous data stream specific and are independent of any command/status delivery mechanism or I/O process. Please contact the author at (408) 974-3889 if you have any questions or comments concerning this document.

This document will develop several key concepts useful in the description of the isochronous Serial Bus model. Terminology is defined at the first mention and any mnemonics are given in parenthesis.

This document assumes the reader has a strong understanding of the IEEE P1394 Serial Bus standard. Important concepts include the Isochronous cycle start packet and Isochronous cycle number, Isochronous data packet and its construction, and the bus access arbitration method. If the reader does not feel they have a good grasp of the IEEE 1394 Serial Bus standard, that standard should be read first.

D.1.1 TALKERS AND LISTENERS

The talker is the source of stream data. It transmits an Isochronous data packet every 125 microsecond for the duration of the stream. The listener is a sink of stream data. It receives Isochronous data addressed to a specific

channel number. For any stream there will be one and only one talker at a time and one or more listeners.

Appendix D. Isochronous Data Streams for IEEE 1394 Serial Bus Devices 113

D.1.2 ISOCHRONOUS DATA STREAMS

A stream is a contiguous series of Isochronous packets (one per cycle) on Serial Bus. Each packet in a stream shares a common channel number. The channel number is in effect the destination address of each packet in a stream. Each stream has a start of stream (SOS) and an end of stream (EOS). The SOS and EOS are simply Isochronous data packets which use two synchronization bits (SY) in the packet header as illustrated below:

SY bits

00 - normal data packet

01 - start of stream (SOS) data packet

10 - end of stream (EOS) data packet

11 - invalid (do not use)

Refer to appendix B of this document for a summary of the format of the Isochronous packet.

D.1.3 GAPS IN DATA STREAMS

A stream may contain zero length data packets but this is not a gap. A gap is an error condition which indicates a missing packet. This missing packet could be the SOS or EOS packet as well as any normal data packet within the stream.

The purpose of the SOS and EOS packets are to delimit the stream for the listener. Talkers will set the SOS bit in the header of the first packet of a stream and will set the EOS bit in the last packet of a stream. If either the SOS or EOS packets are not detected, there were one or more lost packets at the corresponding start or end of the stream. If a packet does not arrive for any Isochronous cycle during a stream (delimited by SOS and EOS) there was a lost packet within that stream. Single packet streams (i.e., SOS = EOS) are not allowed. They complicate the state machines and are of questionable utility.

D.1.4 FIXED RATE VS. VARIABLE RATE

There are two types of data transfer rates for streams. They are fixed rate and variable rate data transfer. A variable rate stream will vary its bandwidth utilization depending upon certain requirements specific to that stream. This means that variable rate data transfer will be specified as a maximum or peak transfer rate. If this peak transfer rate is exceeded, data loss and maybe the complete collapse of the Isochronous transport should be expected.

A fixed rate stream will sustain an average bandwidth utilization in a deterministic manner. This means that a fixed rate data transfer will be specified as a set of integers which characterize the method used to sustain this average transfer rate. A simple method known as the A/B count is used for this purpose. The fixed rate transfer algorithm is comprised of two phases, A and B. The A count and B count parameters specify the duration of each phase as a number of Isochronous cycles. The A length and B length parameters specify the number of bytes to send each cycle of the associated phase. In this way, non-harmonic data transfer rates can be specified.

For example, an A/B count could specify that 1 byte is sent for 7 cycles (A), then 2 bytes are sent for 1 cycle (B). This would define a fixed rate transfer greater than the 8 Khz cycle start rate.

D.1.5 ERROR HANDLING BEHAVIOR

There are three defined behavioral modes or talkers and listeners upon detection of an error during a stream. They are ignore and continue (IGNORE), report and continue (CONTINUE) and report and stop (STOP). There is no defined ignore and stop behavior.

During transmission of stream there are several error conditions which might occur (e.g., missing packet, data error). Errors are detected and reported on transition boundaries (e.g., Isochronous cycle or the detection event) and not on state. For example, when several consecutive data packets are lost, only one event (missing packet) is reported. Each subsequent missing packet is not reported.

Consider the case where a talker or listener (a unit) is operating in the CONTINUE mode. A stream gap is detected and reported. On a subsequent cycle the stream has recovered and this non-error event is also detected. The unit must now issue a second report indicating that the stream has been recovered. These two reports delimit the error (gap) and allow higher level functions the option to repair the stream via post-stream processing. These reports do not need to be issued immediately. They may be logged by the unit for use after the stream has completed. The second report message is not issued when a unit is operating in the STOP mode. Neither report is issued when a unit
is operating in the IGNORE mode.

D.1.6 WHAT DOES CONTINUE MEAN?

There are three ways in which a unit can continue. The modes are skip, fill and concatenate. In the concatenate mode, when a unit detects a gap, it does not try to manage buffer pointers but simply appends (concatenates) subsequent data to its existing buffer, effectively eliminating the stream gap.

In the skip mode, when a unit detects a gap in a stream, it determines the extent of the missing data. The unit then advances its buffer pointer (or performs other appropriate behavior such as transmitting zero length data packets if it is a talker) to leave a known gap in the receiving buffer. This gap in the receiving buffer can then be filled by the talker after the stream has completed.

Appendix D. Isochronous Data Streams for IEEE 1394 Serial Bus Devices 115

In the fill mode, the units fills any gaps in a stream with a predefined data pattern. The fill mode is dependent upon the devices involved and the nature of the stream. Definition of the fill mode is beyond the scope of this document.

D.1.7 NODES, UNITS AND PORTS

A node is an IEEE 1212 addressable entity on the Serial Bus. A unit is any process within a node that can be identified by parsing the node's IEEE 1212 ROM space. A node may implement one or more units with either shared or dedicated local resources. A collection of local resources enabling talker or listener functionality are referred to as a port. A unit requires a port before it can become a talker or listener. Consider the simple case of a expansion memory unit. The port

could be simply a DMA engine (address/count/direction) operating on a memory buffer. The unit process would be a simple module which initializes the port (local resources) in anticipation of a data stream. Using this example, it is possible to have multiple ports dynamically associated with a single unit (process) or even shared among multiple units.

In the preceding example an intermediate buffer was used to store and source stream data. Other node configurations are possible such as a port dedicated to a hardware unit process (e.g., an audio speaker). In this case, there would be no need to dynamically associate the port with the unit.

It is not necessarily apparent when parsing the node's ROM space whether it is implemented with shared or dedicated ports (local resources). These local resources are not visible on the Serial Bus. It is one of the functions of a unit to acquire the necessary local resources which comprise a port. This function may be invoked by the unit's drive software.

D.2 ISOCHRONOUS COMMAND SET

The Isochronous command set is used to control the programmable configuration and behavior of talkers and listeners (units). There are two classes of Isochronous commands. The first class of commands provides the ability to control a unit's modes and local resources. The second class of commands provides the ability to control a unit's state. Both classes of commands share a common command block structure.

An Isochronous command block is a structure containing parameters and operation codes. In general, there are two major areas within an Isochronous command block, stream specific and unit specific. Details of the unit specific area of the Isochronous command block are defined by agreement between the unit and its driver software and are beyond the scope of this document. This document discusses the stream specific aspects of an Isochronous command.