Document X3T9.2/92-199 Revision 2

SCSI 3 Serial Bus Protocol A Logical I/O Model for SCSI on the 1394 Serial Bus

November 6, 1992

Gerald Marazas
Scott Smyers
Ed Gardner

Point of Contact:
Gerald A. Marazas
IBM Corporation
P.O. Box 1328
Boca Raton, Florida 33429
Mail Stop 5432

Phone: (407) 982-4423

Internet: marazas@bcrvmpc2.vnet.ibm.com

Secretariat

Computer and Business Equipment Manufacturers Association

Abstract: This document is a proposed working draft of the SCSI 3 Serial Bus Protocol. The purpose of the protocol is to define how SCSI 3 functions are transported over the IEEE 1394 High Speed Serial Bus.

Contents

185

186

Figures

Acknowledgements

The editors wish to express appreciation to the multiple parties who have provided input and/or critique to this document. In particular, the following individuals have made valuable contribution by virtue of their inputs of text and their comments:

Jon Haswell
IBM Fujisawa, Japan

Andy McNeill
IBM Boca Raton, Florida

Michael Teener
Apple Computer Cupertino, California

187

Goals and Objectives

This document describes a command and status delivery protocol for controlling the operation of devices attached to an IEEE 1394 Serial Bus physical interface. This protocol is based on a mapping of parallel SCSI commands and messages. Trade offs have been made in function placement so as to provide improvements in performance and subsystem functionality, while allowing Target and Initiator designers to build upon existing parallel SCSI hardware and software.

A major goal motivating this protocol definition was to define a model acceptable to device vendors, looking for an evolution from parallel SCSI, and systems designers looking for opportunities to more fully exploit the capabilities inherent to the IEEE 1394 I/O bus.

The protocol described in this document attempts to solve some of the problems associated with parallel SCSI protocol through provision of additional functional capability such as:

1. Ability to queue multiple commands for a Target without concern for arbitrary "queue full" conditions within the Target.

2. Target does not need to interrupt processing of a prior command in order to queue a subsequent command.

3. Ability to design Target devices that have varying levels of support for command queueing or overlapped command processing without affecting the design of the Initiator. Conversely, the Initiator design can be structured so as to support multiple device and/or command queueing capability.

4. Distribute the system DMA context handling (Disconnect, Reconnect, Save Data Pointer, Restore Pointers, etc.) required by parallel SCSI so as to reduce device and adapter overhead for higher subsystem performance. The Target device is given an important role in maintenance of DMA context.

This protocol also attempts to ease the transition to a Serial SCSI environment by consideration of significant aspects of the Parallel SCSI environment:

1. Integrates standard parallel SCSI Command Descriptor Block (CDB) within Serial SCSI Command Block.
2. Closely follows standard parallel SCSI command queuing/error recovery procedures and concepts.
3. Requires less packet overhead because of command/status list management being performed locally in the Initiator.

In striking a balance between support for existing Parallel SCSI functions, and providing a growth path to exploitation of new features of Serial SCSI environment, a new fixed length command block has been defined. This command block provides support for advanced features of todays SCSI such as automatic sense data reporting and scatter/gather host data buffer handling. Support is provided for interesting new features of the IEEE 1394 High speed Serial Bus such as Isochronous data transfer. The present approach does not require "locked" memory operations for list pointer update/management.

Other aspects of this protocol address the broader elements of operation currently under discussion in the SCSI 3 community.

1. Taking into account the command chaining capability of Serial SCSI as well as the SCSI 3 Queueing Model so as to achieve comparable order of command fetching and command execution between Serial SCSI and Parallel SCSI.

2. It is desired to maintain some level of commonality for similar functions between SCSI as supported by by the IEEE 1394 transport mechanism and SCSI as supported by Fibre Channel transport.

189

1.0 Overview

The SCSI 3 Serial Bus Protocol has the following features:

- Fixed length command blocks (See note 1 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

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. One accommodation to vendors of low-end Target devices is the organization of the command block into a baseline section which must be fetched under all circumstances and an extended portion for which the option is granted that it may be fetched only when a check condition occurs during execution of the command. It would be anticipated that high-end Target devices would always fetch both the baseline as well as the extended portions of the command block.

190

2.0 Referenced Standards

This Serial Bus Protocol document describes a Logical I/O Model for the physical Transport means provided by the IEEE 1394 High Speed Serial Bus.

For compatibility purposes reference is made to SCSI 2 (X3.131-1992)

3.0 Glossary and Conventions

This section is to be provided in a later revision level.

192

4.0 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.

4.1 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. Individual commands within the chain do not need to have any particular connection to one another except that they originate from the same Initiator. One reason for forming commands into a chain is that processing efficiency is enhanced within the Initiator. An additional reason for forming commands into a chain is that there is some relation among the commands as viewed by the Initiator.

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, the entire collection of Linked commands are considered as a single I/O Process. The elements within a Linked I/O process must be completed before work may be 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 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. Link commands may be used in a chain as can any other SCSI command.

In the SCSI 3 Serial Bus Protocol, command blocks must be queued by the Initiator and the Initiator must be capable of providing the queued command blocks one at a time to the Target whenever the Target is ready to begin processing a new command. This 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.

4.1.1 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 the Initiator through use 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 first command in the chain. 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.

4.1.2 Focus of a Command Chain

In conformity to the SCSI 3 queueing model, each chain of command blocks is directed to a specific Logical Unit Number (LUN) of a specific Target. An Initiator is allowed to have multiple command block chains directed to the same Target/LUN combination. An Initiator is also allowed (in fact required) to have a different command block chain for each and every Target/LUN which is to receive commands from that Initiator. Observe that a given Target/LUN combination may also have multiple command block chains directed to it. Each command block chain within an Initiator is distinguished by having a different starting address within that Initiators address space.

4.1.3 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)

A chain of command blocks is restricted to contain one and only one type I/O process drawn from the list above. A TYPE_ID parameter is connected to each chain for the purpose of identifying the nature of commands contained within any given command block chain.

If an Initiator has a sequence of Simple I/O commands which are to be completed before a sequence of one or more Ordered commands,then the Initiator must construct one chain consisting only of the Simple commands and a second chain consisting only of the Ordered commands. The Initiator must first inform the Target of the chain of Simple commands, and second, inform the Target of the chain of Ordered commands. These restriction on contents of a chain and this procedure for informing the Target about chains facilitate processing by the Target which conforms to the completion sequence rules defined in the SCSI 3 Queueing Model.

4.1.4 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 read and then discarded in order to which command is best to execute next so as to achieve best performance. The command selected for execution is then refetched.

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.

4.1.5 Shoulder 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. Two 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. 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 set of these two FIFOs shall be provided per Target. This single set is to be shared by all Logical Units connected to the given Target.

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.

After the Target has received this "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, the address for the next command is contained as a field within the present command block.

4.2 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.

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 may not be written 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.

4.2.1 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 have many notions in common. A key element common to both the Status FIFO in the Initiator and the Command FIFO 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 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 and thereby be able (at the hardware level) to also function as a Target.

4.2.2 Usage of the M_Flag

When a Target reads a command 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.

4.2.3 Management of Target Resources

It is important to recognize there are two important and potential 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. It is important to the successful operation of the Serial Bus Protocol that the Target be able to accept a stated minimum number of the shoulder taps at any point during processing of some command or at any point in any other activity undertaken by the Target. In order to fulfill these obligations, it is quite possible that specialized hardware would be used to implement the Normal FIFO. The specific obligation of the Target relative to a Shoulder Tap is to place the contents of the 12-Byte Tap Packet (SCSI Command Initiation Packet) into storage. 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.

4.2.3.1 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 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. Such early release of Tap Slots may prove useful against factors which tend to increase the number of chains to be managed. One factor increasing the need for command chains is the RAID environment in which a large number of Logical Units are present. Another factor increasing the number of chains is the environment in which a given Target/ LUN must be given commands of different types and the requirement that only one type of command may be placed in any 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.

4.2.3.2 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 preprocessing is underway prior to entry into execution phase.

4.2.3.3 Schedule Policy for Fetching Among Multiple Command Chains

Means are provided within the Tap Packet (SCSI Command Initiation packet) to identify the type of command exclusively contained within each and every command chain presented to a Target. An implied time stamp exists for each of these chains. That implied time stamp is the relative order of arrival of each Tap packet at the given Target. 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. 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 processed is fetched. Once fetched, the 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.

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". For this Allowed Set, it is required that Round Robin scheduling be employed for the command block fetch policy when multiple command chains are involved. This requirement is 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.

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 Round Robin policy for fetching commands from multiple chains of Simple 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 only requirement is for that single command to be determined by a Round Robin fetch policy. 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.

128

4.2.3.4 Option to Defer Fetch of Command Block Extended Portion

Another aspect of the storage management problem is influenced by the organization of the command block into a baseline section and into an additional component called the extended portion. This division of the command block is intended to offer an option to low-end Target devices in which the storage resource for command blocks may be very limited. The baseline portion is composed of those elements which must be fetched in all cases by all levels of implementation for a Target. The baseline portion contains those fields needed to pursue completion of the associated I/O process under normal circumstances in which no check condition occurs and for those commands which do not require return of sense data even when completion with good status occurs.

The extended portion of the command block contains those fields providing information as to process steps when a check condition occurs and/or return of sense data is needed. The option is extended to low-end Target devices that they may defer fetch of this portion of the command block until it is determined that a check condition has occurred and/or it is necessary to return sense data to the Initiator. The option is intended to provide the benefit of reducing the amount of storage needed in the Target to hold the command block. For high-end devices, the expected situation is that both the baseline and the extended portion of the command block would be fetched at the same time and as a single operation.

4.2.3.5 Repeated Access to Same Command Block

The Target is allowed the option to 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. One expectation is allowed as to 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.

4.2.3.6 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 means for the Initiator to inform the Target on a command by command basis whether autosense is enabled or disabled.

5.0 Command Transfer Protocol

5.1 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 14. For information describing the format of these various types of packets refer to Appendix A, "Packet Formats" on page 35. For the detailed contents of

200

the Command Block refer to 8.1, "Command Blocks" on page 18. For details of the contents of the SCSI Command Initiation Packet refer to 9.1, "Payload of Initiator to Target Packet - "Tap Packet"" on page 25.

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 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 end of processing being reached for some earlier 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 = 0)

**********************************     IEEE 1394 Transaction

Tap on shoulder    ------> Receive Tap Packet
                   <-----  Tap Acknowledgement

**********************************     IEEE 1394 Transaction

                  <------  Read Request  (For a  Command Block)
Command 3         ------> Read Response (Supply Command Block)
(M_Flag = 1)
```

Figure 2. Conceptual Initiator-Target Conversation, Example 2

5.2 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 concurrent chains 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 should be able to process concurrently.

Note: When implementing the system level software, a choice can be made in terms of how the finite number of Tap slots in the Target may be handled. The first choice is to include code to handle the rejection of a "Tap" request and to retry at a later time. Clearly, provision must be made so that system software will appropriately release Tap slots from each Initiator. Another alternative (and probably good practice) is to check the number of Tap slots each Target in the system may support at power on time or configuration/re-configuration time. The host software may then either choose to limit the number of "Taps" it has outstanding at any time or declare a configuration error if the number of Taps available is less than it requires.

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.

6.0 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.

7.0 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.

7.1 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.

There are a number of pieces of information which an Initiator must send to a Target to help it manage the processing of commands. Some of these items of information are at the specific request of the Target. In such cases of information being sent by invitation and therefor being expected, the Target has ensured that necessary room exists within Target memory space. In other instances, a Target must be prepared to accept information from a Initiator without prior warning or expectation that such information is to be sent. A particularly significant part of this architectural definition is the description of capabilities required at a minimum to accommodate such unexpected inputs.

The following sections describe the several data structures needed to respond to unsolicited data packets send
$ to a Target supporting the SCSI 3 Serial Bus Protocol. These unsolicited transmissions are divided into two
$ categories, named, Urgent and Normal. 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 FIFO address alone is sufficient to determine the nature of the packet being received and the degree of immediacy required for the processing expected per such a packet. In all instances, there are multiple different types of packets which may be sent. to both the FIFO servicing Urgent items as well as the FIFO servicing Normal items. 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.

7.1.1 Normal FIFO

$ This data structure shall have its register space address specified by an address in configuration ROM.

$ This FIFO is intended for use receiving shoulder taps associated with command chains containing Simple,
$ Ordered, Head of Queue, and Autocontingent Allegiance (ACA) command types.

$ 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 be returned back from the
$ Target to the Initiator.

$ 7.1.2 Urgent FIFO

$ This data structure shall have its register space address specified by an address in configuration ROM.

$ Items of control information which may be sent to the Urgent FIFO are analogous to parallel SCSI mes-
$ sages 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.

7.2 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.

7.2.1 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.

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.

8.0 Command and Status Information

8.1 Command Blocks

As described below, the SCSI 3 Serial Bus Protocol uses a 60-Byte command block which embeds a standard parallel data SCSI Command Descriptor Block (CDB) of up to 12 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 34.

In support of the need by low-end Target devices, the command block is dived into a baseline portion and an extended portion as follows. The baseline portion consists of the first 48-Bytes of the command block. This baseline portion must fetched in all instances and by SCSI devices at all levels of implementation. The option is granted to devices that they may choose to fetch the final 12-Bytes (Extended Portion) of the command block only when a check condition occurs during execution of the SCSI command specified by the baseline portion. It is expected that only low-end SCSI devices might wish to take advantage of this option.

$ 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, given that each command block is a 60-Byte quantity, there would need to be a
$ 4-Byte pad after the first command block and between each of the following command blocks so that each
$ command block starts on an 8-Byte boundary when these blocks are located immediately following one
$ another within a chain. Clearly, the Target does not need to fetch field since there is no start boundary
$ requirement in Target memory for for the copy of the command block fetched by the Target.

	Byte	Function			
$	0	Flags		Initiator_ID	
$	4	CDB 0	CDB 1	CDB 2	CDB 3
$	8	CDB 4	CDB 5	CDB 6	CDB 7
$	12	CDB 8	CDB 9	CDB 10	CDB 11
$	16	Reserved	Reserved	Reserved	Reserved
$	20	Transfer Length			
$	24	Data Buffer Address (MSQ)			
$	28	Data Buffer Address (LSQ)			
$	32	Next Command Address (MSQ)			
$	36	Next Command Address (LSQ)			
$	40	Status FIFO Address (MSQ)			
$	44	Status FIFO Address (LSQ)			
$	48	Sense Data Buffer Address (MSQ)			
$	52	Sense Data Buffer Address (LSQ)			
$	56	Reserved	Reserved	Reserved	Sense Length

$ Figure 3. Command Block

$ **Note:** The terms MSQ and LSQ mean Most Significant Quadlet and Least Significant Quadlet respectively.

208

$ **Field** **Function**

$ **Flags** The flags field currently has the following bits defined. All bits not specified are reserved.

Bit(s)	Name	Function
15 (MS) to 9	Reserved	These bits are reserved for possible future expansion of the Flags field.
8	A_Flag	When set equal to value one, the Initiator wishes this command to be aborted if possible. 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 to13.1, "Payload of Abort Packet" on page 31.
7	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.
6	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.
5	I_Flag	When set equal to value one, this bit indicates that the data transfer for this command will be via an Isochronous channel. Value equal one for this bit implies ordered data transfer and the O_Flag will be ignored by the Target. Value equal zero for this bit indicates Nonisochronous (Asynchronous) data transfer.
4	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.
3	S_Flag	When set equal to value one, this bit indicates the optional Initiator memory scatter / gather function is being invoked for the present command.
2 to 0 (LS)	Queue Type	Value 001 means Tag Type is Simple Tag. Value 010 means Tag Type is Ordered Tag. Value 011 means Tag Type is Head of Queue. Value 100 means Tag Type Autocontingent Allegiance (ACA) All other values for this field are reserved.

$ Figure 4. Control Flags

$ **Initiator_ID**
$ These 16-bits provide "in-band" means to identify the Initiator who originated the present
$ command block.

$ **Editorial Note:** This field serves to aid in the compatible support for Serial SCSI via the IEEE
$ 1394 serial bus and Serial SCSI via the Fibre Channel. In the situation in which various
$ bus bridges intervene between source Initiator and destination Target, the IEEE 1394
$ packet header does not supply the correct information to identify the source Initiator.
$ The subject field does provide the correct identification information.

CDB This field carries a standard SCSI CDB.

$ **Reserved** These four field, each at 8_bits in length are reserved for possible future growth in function
$ assigned to the command block.

Transfer Length
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 an optional 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.

Data Buffer Address
This field can have two meanings depending on the setting of the I_Flag.

I_Flag = 0
This indicates data transfer using standard asynchronous packets. In this case 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 optional feature of Initiator memory scatter/gather is supported , and 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.

I_Flag = 1
In this case Isochronous transfers have been requested. Refer to Appendix B for information describing Isochronous data transfer. When the Isochronous transfers have been requested, the four Bytes within the Data Buffer Address field (LSQ) are reassigned as follows.

Byte	Name	Function
0		Reserved
1		Reserved
2-3	Length	This field gives the length of each Isochronous packet for the transfer of data.

For commands which do not require any data the value in this field is undefined.

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. In all circumstances, the least significant two 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 11.

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. A given Target may use more than one Status FIFO address for a given Initiator.

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 Target can elect to accumulate Status Blocks within a given Status FIFO and then process them at leisure.

Implementation 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. This field and all following fields of this command block are all elements of the extended portion of the command block. The option is granted to a Target relative to the Extended Portion that it only must be fetched should a check condition occur or if sense data is to be returned.

Reserved This 8-bit field is reserved.

Reserved This 8-bit field is reserved.

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.

8.2 Status Block

The status Byte for a command as carried in the status phase under parallel SCSI is embedded in a short packet which is used to end a command under the SCSI 3 Serial Bus Protocol. For good completion, only this one Byte field needs to be set in the status block. For bad completion the remainder of the fields need to be also be filled by the Target.

Byte	Function			
0	Command Address (MSQ)			
4	Command Address (LSQ)			
8	Status	Sense Key	ASC	ASCQ

Figure 5. 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 simplify the hardware design required to correlate the status with the command.

Status This is the status byte normally returned during a parallel SCSI status phase. It will be zero for good completion.

Sense key This is the "sense key" from the SCSI sense data. If an error occurs on the current command then this field is set by the Target, otherwise it is undefined.

ASC This field holds the "Additional Sense Code" from the SCSI sense data if an error occurs, otherwise it is undefined.

ASCQ This field holds the "Additional Sense Code Qualifier" from the SCSI sense data if an error occurs, otherwise it is undefined.

$ 8.3 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 on 12-Bytes within the command block.

$ **Editorial Note:** The format below consisting of 16-Byte units has not received careful review. The moti-
$ vation for use of a 4-Byte pad field is so that a sequence of Data Buffer addresses can be made to line
$ up on 8-Byte storage address boundaries in the Initiator. Such consistent line up upon address
$ boundaries could not be achieved if each scatter/gather entry was only 12-Bytes in length.

$
$
$
$
$

Byte	Function
0	Data Buffer Address (MSQ)
4	Data Buffer Address (LSQ)
8	Transfer Length
12	Pad

$ Figure 6. 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.

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

$ **Pad** This is a 32-bit field having no function other than to ensure that the multiple Data Buffer
$ Address entries within this list are each separated from one another by some multiple of 8-Bytes.

9.0 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.

9.1 Payload of Initiator to Target Packet - "Tap Packet"

9.1.1 Request Payload

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

Next Command Address (MSQ)		
Next Command Address (LSQ)		
TYPE_ID	Reserved	LUN

Figure 7. Payload of Initiator "Tap" packet

Field **Function**

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

Type_ID This field states which type of command exclusively contained in the associated command chain.

Value	Type Command
01 Hex	Simple Commands
02 Hex	Ordered Commands
03 Hex	Head of Queue Commands
04 Hex	ACA Commands

Reserved This 8-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 filed needs to be larger than 8 bits in order to support future RAID applications.

9.1.2 Response Payload

There will always be a response to a SCSI Command Initiation Packet. Normally, an acknowledge code of "complete" will be returned immediately and a split transaction will not be required. If a split transaction is required, a standard Write Response packet will be returned. If the Target cannot accept the "Tap" packet due to an overrun of the Command FIFO, (because the Target has no storage available to hold the next command address) it will return a 'resp_conflict'.

9.2 Command Read Request

9.2.1 Request Payload

A Target requests a new 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. The data transfer amount referenced by this request packet shall be 60-Bytes if both the baseline and extended portions of the command block are being requested. If the Target elects in favor of using the option to request only the baseline portion of the command block, then the data transfer amount shall be 48-Bytes.

9.2.2 Response Payload

The response payload consist of the 60-Byte combination of Baseline plus Extended portion of the command block, or the 48-Byte Baseline portion, as originally requested by the Target.

10.0 Data Transfer Protocol

10.1 Asynchronous Transfer

The transfer of all data required by commands is accomplished by standard format (per IEEE 1394) read request and write request packets sent by the Target to the Initiator. The combination of a request packet and its associated response packet form a Transaction per IEEE 1394. For various commands the number of bytes required to be transferred will not be an even multiple of 4. In this case the transfer will be padded to 4*n bytes. The padding bytes may be of any value.

The starting address the packets are 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.

10.1.1 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 address being referenced within the Initiator buffer. Such updating accounts for portions of data from the buffer previously transferred.

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. 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.

10.1.2 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.

11.0 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.

In reply to the status packet, a split transaction is allowed if needed; otherwise, the Initiator should return a "complete" acknowledge code.

12.0 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.

12.1 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.

```
     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 command -------------------
                               <-----   Block Read Request Packet
                                        Address=First Command Address
     Ack (e.g. pending)         ------>
     Block Read Response        ------>
     Data=Command Block
     M_Flag=0
     -------------- Target returns data read ------------------
                               <-----   Block Write Packet
                                        Address=Data Address
                                        Data =Requested data
     Ack (complete, pending,    ------>
             or, busy_X)
     Block Write Response       - - ->
 (optional if initiator ack=pending)

     -------------- Target returns status ---------------------
                               <------  Block Write Packet
                                        Address=Status FIFO
                                        Data=Status Block
     Ack (complete, pending,     ------>
             or, busy_X)
     Block Write Response        - - ->
 (optional if initiator ack=pending)
```

Figure 8. Target Read Command

12.2 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 required to complete the data transfer for each command.

218

```
      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
                                          Address=Status Address #1
                                          Data=Status Block
       Ack (complete, pending,   ------>
               or, busy_X)
       Block Write Response        - - ->
  (optional if initiator ack=pending)
       ----------- Target returns data read for cmd #2 ----------
                                <------   Block Write Packet
                                          Address=Data Address #2
                                          Data =Requested data
       Ack (complete, pending,   ------>
               or, busy_X)
       Block Write Response        - - ->
  (optional if initiator ack=pending)
       ------------ Target returns status for cmd #2 -------------
                                <------   Block Write Packet
                                          Address=Status Address #2
                                          Data=Status Block
       Ack (complete, pending,   ------>
               or, busy_X)
       Block Write Response        - - ->
  (optional if initiator ack=pending)
```

Figure 9. Target Multiple Read Commands

13.0 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. 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 write quadlet packets. The response to these if a split transaction occurs would be a standard Write Response packet.

The defined message packets are

TAP	This establishes that an Initiator has commands for a Target
ABORT	This aborts all commands from this initiator
ABORT TAG	This aborts a particular tagged command
RESET	This resets the Target
CLEAR QUE	This aborts all commands from all Initiators which are already queued at the Target.
CONTINUE	This clears any contingent allegiance condition and has no side effects

13.1 Payload of Abort Packet

The function of this message is identical to the parallel SCSI "Abort" message. In order to secure proper correlation, there may be a need to expand upon the cross reference information indicated here.

Command Address (MSQ)		
Command Address (LSQ)		
Message Id ABORT Code = 11 Hex	Reserved	LUN

Figure 10. Payload of Abort Packet

13.2 Payload of Abort Tag Packet

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.

220

Command Address (MSQ)		
Command Address (LSQ)		
$ $ $ Message Id ABORT TAG Code = 12 Hex	Tag (See Note)	Lun

Figure 11. Payload of Abort Tag Packet

$ **Editorial Note:** The intention for SCSI 3 is to use the 64-bit command address as a tag or anywhere else a
$ correlation ID is required. The above 8-bit Tag is used as a place holder should discussion be found
$ necessary as to compatibility with SCSI 2 regarding the Abort Tag.

13.3 Payload of Reset Packet

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

Command Address (MSQ)
Command Address (LSQ)

Message Id RESET Code = 13 Hex	Reserved	Reserved	Reserved

Figure 12. Payload of Reset Packet

13.4 Payload of Clear Queue Packet

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

Command Address (MSQ)		
Command Address (LSQ)		
Message Id CLEAR_QUE Code = 14 Hex	Reserved	Lun

Figure 13. Payload of Clear Queue Packet

221

14.0 Contingent Allegiance

14.1 Aims

It is desired to attempt to follow the model presented by parallel SCSI to as great a degree as possible.

Additional detail is to be supplied in a later version of this document.

15.0 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 on both the target and the initiator being unaware 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.

15.1 Increased Initiator problems

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 jog 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 register 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.

15.2 Asynchronous Event Notification

15.2.1 Aims

It is desired to attempt to follow the model presented by parallel SCSI to as great a degree as possible.

Additional detail is to be supplied in a later version of this document.

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.

A.1 Write Packets

Destination ID		Tl	Rt	TCode	Res	P
Source ID		Destination Offset				
Destination Offset						
Byte 0	Byte 1	Byte 2		Byte 3		
Header CRC						

Figure 14. Quadlet Write Request Packet

Destination ID	Tl	Rt	TCode	Res	P
Source ID	Destination Offset				
Destination Offset					
Block Length	Tr Data 0000				
Header CRC					
Data block					
Data CRC					

Figure 15. Block Write Request Packet

Destination ID	Tl		Rt	TCode	Res	P
Source ID	RCode					
Reserved						
Header CRC						

Figure 16. Write Response Packet

229

A.2 Read Packets

Destination ID	Tl	Rt	TCode	Res	P
Source ID	Destination Offset				
Destination Offset					
Header CRC					

Figure 17. Read Request Packet

Destination ID	Tl	Rt	TCode	Res	P
Source ID	RCode				
Read Data					
Header CRC					

Figure 18. Read Response Packet

Destination ID	Tl	Rt	TCode	Res	P
Source ID	Destination Offset				
Destination Offset					
Block length					
Header CRC					

Figure 19. Block Read Request Packet

Destination ID	Tl	Rt	TCode	Res	P
Source ID	RCode				
Data Length	Tr Data				
Header CRC					
Data Block					
Data CRC					

Figure 20. Block Read Response Packet

Data Length	Channel	TCode	Sy	Rs
Header CRC				
Data				
Isochronous CRC				

Figure 21. Isochronous 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.)

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.

226

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 I_Flag is set in the command packet this indicates a request to transfer the data via Isochronous channels rather than by Asynchronous packets. In this case the Data Address field of the command packet contains the Isochronous channel and packet size to be used for the transfer.

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

Data Length	Channel	TCode	Sy	Rs
Header CRC				
Data				
Isochronous CRC				

Figure 22. 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.

228