(Proposal)

SCSI-3 Serial Bus Protocol, Rev. 2

This standard describes the frame format and protocal definitions required to transfer commands and data between a SCSI (Small Computer System Interface) Initiator and Target using a serial link interface operating according to the IEEE P1394 protocol requirements.

Contacts:

John Lohmeyer (X3T9.2 Chairman)
NCR Corporation
3718 N. Rock Road
Wichita, KS 67226
E-mail: john.lohmeyer@wichitaks.ncr.com

Scott Smyers (Editor)
Apple Computer, Inc.
20705 Valley Green Drive, m/s 60AR
Cupertino, CA 95014
Tel.: (408) 974-7057
E-mail: smyers.s@applelink.apple.com

1. Scope

This standard, in combination with the IEEE P1394 standard and the SCSI-3 Command Set documents, provides information necessary to implement an IEEE P1394 device which uses the SCSI-3 command Set. Such devices may include disk drives, scanners, optical disks, removable media devices or any other device described in the SCSI-3 standards documents.

For completeness, this document also describes the protocol for issuing commands, receiving status and controlling command execution in the IEEE P1394 serial bus environment. A more general description of command enqueuing to a target device is covered in the IEEE P1212.1 List DMA document. The functionality provided by the mechanisms built into IEEE P1212.1 and described in this document is comparable to that provided by the SCSI parallel protocol with regard to command delivery and list management.

In the interest of maintaining compatibility with related standards, this standard is being developed in coordination with the SCSI-3 FC-4S standards document.

2. Normative References

The following standards documents provide additional information for the implementor which may be helpful.

IEEE 1212

IEEE 1212.1

IEEE P1394

SCSI-3 SAM

SCSI-3 SCC

Other SCSI-3 command set documents

3. Definition of Terms

This section contains definitions of terms which are used freely throughout this document and which have special meaning.

Byte - an 8 bit quantity

Doublet - a 2 byte quantity

Quadlet - a 4 byte quantity

Octlet - an 8 byte quantity

Command Status Register (CSR) - an address in IEEE 1212 64 bit address space which is visible to all devices on the IEEE P1394 serial bus. In general, the act of reading or writing a CSR may have an affect on the operation or hardware state of the IEEE P1394 device whose IEEE 1212 address space contains that CSR.

Sharable List (also referred to as a List) - a linked list of Command Blocks (CB's) which is visible to other devices on the IEEE P1394 serial bus. Each sharable list is owned by a specific target, and each target may own more than one sharable list. Consistent with the rules for target ownership, any device on the IEEE P1394 serial bus may add CB's to any target's sharable list.

CB - Command Block - an element in a sharable list.

Enqueue - The act of adding a CB to a sharable list.

Dequeue - The act of reading a CB from a sharable list and acting on its contents. This operation may include returning a completion status, depending on the contents of the CB.

Initiator - A device on the IEEE P1394 bus which creates CBs and places them in a sharable list for eventual execution by the target for whom the list was established.

Enqueuing Initiator - A device on the IEEE P1394 bus which is in the process of adding a CB to a target's sharable list, or, the owner of an existing CB in a list.

Target - A device on the IEEE P1394 bus which reads the sharable list elements and which is ultimately responsible for executing the commands which each list element describes.

List Head Pointer - A location in IEEE 1212 address space which contains the 64 bit IEEE 1212 address of the first CB in a sharable list.

List Tail Pointer - A location in IEEE 1212 addressable space which contains the 64 bit IEEE 1212 address of the last CB in a sharable list.

Next CB Pointer - A 64 bit field in all CBs in a sharable list which contains the IEEE 1212 address of the next CB in the sharable list. The last CB in the sharable list contains a "Next CB Pointer" field of null.

null - a pointer value which is invalid by definition.

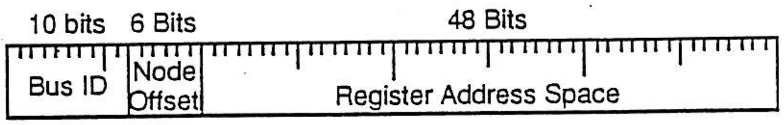
Queue - A data structure which is completely owned and manipulated by a target. The precise format of this data structure is not subject to standardization. However, in a functional sense, a queue is composed of CBs which are either currently executing or have not executed. The target builds its queue from CBs which it reads from its sharable list. The order in which CBs in the queue are executed, and the order in which they may appear in the queue data structure may or may not be the same as the order in which the originating CBs appear in the sharable list.

4. Data Structures and Concepts

This section describes data structures and algorithms that IEEE P1394 devices use to participate in the I/O process.

4.1. IEEE 1212 64 bit address

An IEEE 1212 address is a 64 bit entity which is composed as follows:



Where:

Bus ID - a 10 bit number which identifies one of 1023 possible IEEE P1394 busses in a system. A bus ID of all ones indicates the local bus. Non bus bridging devices respond to addresses with a bus ID of all ones. Bus bridging devices respond to bus IDs which are not all ones.

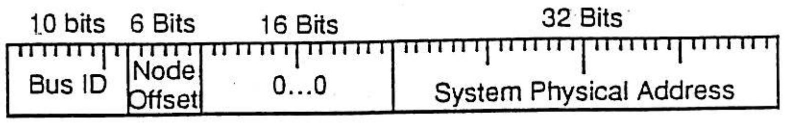
Node Offset - a 6 bit number which identifies one of 63 devices on an IEEE P1394 bus. A node offset of all ones indicates a broadcast.

Register Address Space - a 48 bit number which is available for addressing within a node on the IEEE P1394 bus. A portion of this address space must be reserved for the unit ROM[1]. Other than that, the organization of this address space is left to the implementor.

The device implementor can choose any mechanism for mapping the register address space into that portion of the device which is to be visible to other IEEE P1394 devices on the bus. For a CPU type of device, one simple mapping would be to replace the low 32 bits of the 48 bit register address with the 32 bit system physical address, as shown:

1 The format of the information in the unit ROM is described in appendix B. By definition, the base address of unit ROM is at a fixed location in the register address space. The base address of unit ROM in every IEEE P1394 node is: FFFF F00: 0000h.

236



This provides a simple, straight forward method of mapping the physical address space of a CPU node into the IEEE 1212 64 bit address space.

4.2. Packet Header

During operation, devices arbitrate for the IEEE P1394 bus then transmit a packet. Only one device transmits at a time and all other devices receive. All IEEE P1394 packets contain a header which identifies, among other things, the destination device for that packet and an operation to perform. Other fields in the packet header contain hardware specific information which is used to implement packet retry and other bus integrity functions. Depending on the packet type, the packet header may contain additional information to describe the operation.

The IEEE P1394 packet header is protected by a CRC. For packets which contain a data field, the data field immediately follows the packet header and is protected by an additional CRC. Some IEEE P1394 packets contain a packet header and no data field.

The following sections describe IEEE P1394 bus operations and the role that the packet header plays in those operations. This information is for summary use only. For the complete definition of these facilities, consult the IEEE P1394 specification.

In the following sections, a "requestor" is a IEEE P1394 device which initiates an operation, such as a read request, and a "responder" is a IEEE P1394 device which honors the operation.

4.3. Transaction Layer Facilities

4.3.1. "Quadlet" Read Operation

The requestor issues a quadlet read request packet and the responder returns a quadlet read response packet which contains the requested data. The quadlet read request packet header contains the following information:

Transaction code - this code uniquely identifies this packet as a quadlet read request packet.

Destination address - A 64 bit IEEE 1212 address from which to read a quadlet of information. The upper 16 bits of this address identifies the intended responder.

Source bus ID and node offset - This 16 bit field identifies the requester issuing the read request.

Transaction Label - This field must be unique for all outstanding requests between this requestor and the addressed responder.

The responder receives the read request packet. The transaction code identifies the packet as a quadlet read request packet, the source bus ID and node offset identifies the requestor and the upper 16 bits of the destination address identifies the responder.

The responder retrieves the information from the referenced address and returns it in a quadlet read response packet. The quadlet read response packet header contains the following information:

Transaction code - this code uniquely identifies this packet as a quadlet read response packet.

Destination address - A 64 bit IEEE 1212 pseudo address. The upper 16 bits of this address identify the requestor and the low 48 bits are available for status information regarding this request/response transaction.

Source bus ID and node offset - This 16 bit field identifies the responder who is responding to the quadlet read request.

Transaction Label - This field must be the same as the transaction label which appeared in the original quadlet read request packet header.

Data - A 4 byte field containing data from the address specified in the quadlet read request packet.

The responder reads a quadlet of information from the destination address and constructs a quadlet read response packet. The destination address in the response packet contains the requestor's bus ID and node offset and status information. The source bus ID and node offset contains the responder's bus ID and node offset. The transaction label must be the same transaction label which appeared in the original read request. The data field in the response packet header contains the requested data.

If, for some reason, the responder is unable to honor the quadlet read request, the low order 48 bits of the destination address in the response packet header contains an appropriate status code to indicate the reason. In this case, the data field in the packet header is not valid.

4.3.2. "Quadlet" Write Operation

The requestor issues a quadlet write request packet and the responder returns a quadlet write response packet. The quadlet write request packet header contains the following information:"

Transaction code - This code uniquely identifies this packet as a quadlet write request packet.

Destination address - A 64 bit IEEE 1212 address to which the responder should write the provided data. The upper 16 bits of this address identifies the intended responder.

Source bus ID and node offset - This 16 bit field identifies the requester issuing the quadlet write request.

Transaction Label - This field must be unique for all outstanding requests between this requestor and the addressed responder.

Data - A 4 byte field containing the quadlet of data to write to the destination address.

The responder receives the quadlet write request packet. The transaction code identifies the packet as a quadlet write request packet, the source bus ID and node offset identifies the requestor and the upper 16 bits of the destination address identifies the responder.

The responder writes the provided quadlet to the referenced address and returns status information in the quadlet write response packet. The quadlet write response packet contains the following information:

Transaction code - this code uniquely identifies this packet as a quadlet write response packet.

Destination address - A 64 bit IEEE 1212 pseudo address. The upper 16 bits of this address identify the requestor and the low 48 bits are available for status information regarding this request/response transaction.

Source bus ID and node offset - This 16 bit field identifies the responder who is responding to the quadlet write request.

Transaction Label - This field must be the same as the transaction label which appeared in the original quadlet write request packet header.

The responder performs the requested quadlet write operation and constructs a quadlet write response packet. The destination address in the response packet contains the requestor's bus ID and node offset and status information. The source bus ID and node offset contains the responder's bus ID and node offset. The transaction label must be the same as the transaction label which appeared in the original quadlet write request.

If, for some reason, the responder is unable to honor the quadlet write request, the low order 48 bits of the destination address in the response packet header will contain an appropriate status code to indicate the reason.

4.3.3. "Block" Read Operation

The requestor issues a block read request packet and the responder returns a block read response packet which contains the block read response packet header followed immediately by the data field. The block read request packet contains only the header which contains the following:

Transaction code - this code uniquely identifies this packet as a block read request packet.

Destination address - A 64 bit IEEE 1212 address from which to read the block of data. The upper 16 bits of this address identifies the intended responder.

Source bus ID and node offset - This 16 bit field identifies the requester issuing the block read request.

Transaction Label - This field must be unique for all outstanding requests between this requestor and the addressed responder.

Length - This 16 bit field identifies the number of bytes to read from the destination address. Note that this length must specify an amount of data which can be returned in a single response packet.

The responder receives the block read request packet. The transaction code identifies the packet as a block read request packet, the source bus ID and node offset identifies the requestor and the upper 16 bits of the destination address identifies the responder.

The responder retrieves the information from the destination address and returns it in a block read response packet. The block read response packet contains the following:

Transaction code - this code uniquely identifies this packet as a block read response packet.

Destination address - A 64 bit IEEE 1212 pseudo address. The upper 16 bits of this address identify the requestor and the low 48 bits are available for status information regarding this request/response transaction.

Source bus ID and node offset - This 16 bit field identifies the responder who is responding to the block read request.

Transaction Label - This field must be the same as the transaction label which appeared in the original block read request packet header.

Length - This 16 bit field identifies the number of bytes contained in the data field of this response packet. This length should be the same as the length specified in the corresponding block read request packet header.

Data Field - A field immediately following the packet header which contains <Length> data bytes

The responder reads the requested amount of data from the destination address and constructs a block read response packet. The destination address in the response packet contains the requestor's bus ID and node offset and status information. The source bus ID and node offset field contains the responder's bus ID and node offset. The transaction label must be the same transaction label which appeared in the original read request. Immediately following the block read response packet header CRC is the data field. The data field is protected by a CRC which is separate and independent of the packet header CRC.

If, for some reason, the responder is unable to honor the block read request, the low 48 bits of the destination address field in the response packet header contains an appropriate status code to indicate the reason. In this case, the length field in the response packet header is zero and there is no data field following the packet header.

4.3.4. "Block" Write Operation

The block write request packet header is immediately followed by a data field containing the data to write. The data field is protected by a CRC which is separate and independent of the packet header CRC. The requestor issues a block write request packet and the responder returns a block write response packet. The block write request packet contains the following:

Transaction code - this code uniquely identifies this packet as a block write request packet.

Destination address - A 64 bit IEEE 1212 address to which to write the block of data. The upper 16 bits of this address identifies the intended responder.

Source bus ID and node offset - This 16 bit field identifies the requester issuing the block write request.

Transaction Label - This field must be unique for all outstanding requests between this requestor and the addressed responder.

Length - This 16 bit field identifies the number of bytes to write to the destination address. This is also the number of bytes contained in the data field of the block write request packet.

The responder receives the block write request packet. The transaction code identifies the packet as a block write request packet, the source bus ID and node offset identifies the requestor and the upper 16 bits of the destination address identifies the responder.

The responder writes the data in the block write request packet to the destination address and returns a block write response packet. The block write response packet header contains the following:

Transaction code - this code uniquely identifies this packet as a block write response packet.

Destination address - A 64 bit IEEE 1212 pseudo address. The upper 16 bits of this address identify the requestor and the low 48 bits are available for status information regarding this request/response transaction.

Source bus ID and node offset - This 16 bit field identifies the responder who is responding to the block write request.

Transaction Label - This field must be the same as the transaction label which appeared in the original block write request packet header.

The responder writes the data contained in the block write request packet to the destination address and constructs a block write response packet. The destination address in the response packet contains the requestor's bus ID and node offset and status information. The source bus ID and node offset field contains the responder's bus ID and node offset. The transaction label must be the same transaction label which appeared in the original write request.

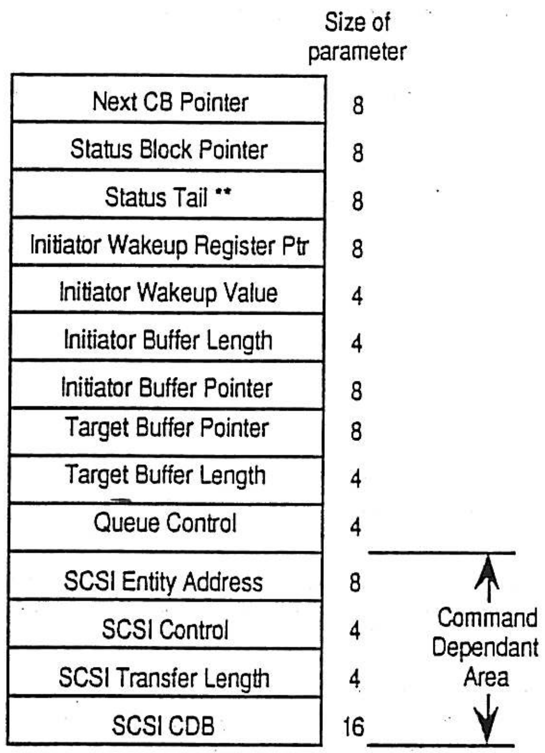
If, for some reason, the responder is unable to honor the block write request, the low 48 bits of the destination address field in the response packet header contains an appropriate status code to indicate the reason.

4.3.5. "Lock" Operation (Swap & Compare/Swap)

4.4. Command Block Packet Data

This section describes the format of a Command Block (CB). The CB contains all the information necessary to describe a data transfer between 2 devices on the IEEE P1394 bus.

The CB data structure is diagrammed below and described in the text which follows. The field in the CB reserved for the SCSI CDB is fixed at 16 bytes. Within this field there can be a SCSI CDB of any length up to 16 bytes.



CB

Where:

Next CB Pointer (8 bytes) - a 64 bit pointer to the next CB in the list. The Next CB Pointer of the last CB in the list shall be set to null.

Status Block Pointer (8 bytes) - a 64 bit pointer to the Command Status Area. This is the location where the target may write status information for this command.

Status Tail Pointer Pointer (8 bytes) - a pointer to the status list tail pointer. The status tail pointer is used to enqueue a status block for this command.

Initiator Wake-up Register Pointer (8 bytes) - a pointer to the initiator's wake-up register. The target optionally writes the initiator wake-up value (described below) to the location that this address points to upon completion of the command.

Initiator Wake-up Value (4 bytes) - the value that the target optionally writes to the initiator wake-up register upon completion of the command.

Initiator Buffer Length (4 bytes) - size, in bytes, of the object pointed to by the Initiator Buffer Pointer.

Initiator Buffer Pointer (8 bytes) - a 64 bit address of an object which describes the initiator's data area for this command. Note that the object pointed to may be either an arbitrarily aligned physically contiguous data buffer or a 16 byte aligned scatter/gather list. The scatter/gather list describes how the initiator's buffer is fragmented. For read commands, this object describes the destination for the data transfer. For write commands, this object describes the source for the data transfer. Bits in the Buffer/Ordering Control field determine the validity of this field and its use (as a pointer to a data buffer or as a pointer to a scatter/gather list).

Target Buffer Pointer (8 bytes) - This is a 64 bit address of an object which describes the target's data area for this command. Note that the object pointed to may be either an arbitrarily aligned physically contiguous data buffer or a 16 byte aligned scatter/gather list. The scatter/gather list describes how the target's buffer is fragmented. For read commands, this object describes the source for the data transfer. For write commands, this object describes the destination for the data transfer. Bits in the Buffer/Ordering Control field determine the validity of this field and its use (as a pointer to a data buffer or as a pointer to a scatter/gather list).

241

Target Buffer Length (4 bytes) - This is the size, in bytes, of the object pointed to by the Target Buffer Pointer.

Queue Control field (4 bytes) - This field contains information which affects the execution of the command. The definition of the fields in this field are as follows:

4	4	16	8
cmd	Q#	Buffer/Ordering Control	Cmd dep size

cmd field - defines the type of CB. The type is either "read", "write", "do", "kill" or "other". The "read" and "write" CBs are described below. The "do" CB is an implementation of the third party copy command, and it is also defined below. The "other" CB is used to implement a standard SCSI CDB, as defined below. The "kill" CB is used to kill other CBs or command groups.

Queue # field - only used for the "kill" CB as described below.

Buffer/Ordering Control field - contains bits which determine the use of the buffer pointers contained in the CB and also defines the re-ordering capabilities of this command relative to other commands in the shared list. The bit definition for this field are as follows:

bit 15 initiator buffer valid - When set to 1, indicates that the initiator buffer pointer field is valid. If set to 0, indicates that the target shall not perform data transfer as a result of this command.

bit 14 initiator buffer indirect - When set to 1, indicates that the initiator buffer pointer points to a scatter gather list. When set to 0, indicates that the initiator buffer pointer field contains the start address of a contiguous buffer which is at least as big as the data transfer size for this command.

bit 13 don't increment initiator buffer pointer - When set to 1, indicates that the target shall not increment the initiator buffer pointer when performing the data transfer. When set to 1, the target shall not transfer data out of order. This bit can only be set to 1 when bit 14 is set to 0 (indicating no scatter/gather function).

bit 12 reserved

bit 11 target buffer valid - When set to 1, indicates that the target buffer pointer field is valid. If set to 0, indicates that the target shall not perform data transfer as a result of this command.

bit 10 target buffer indirect - When set to 1, indicates that the target buffer pointer points to a scatter gather list. When set to 0, indicates that the target buffer pointer field contains the start address of a contiguous buffer which is at least as big as the data transfer size for this command.

bit 9 don't increment target buffer pointer - When set to 1, indicates that the target shall not increment the target buffer pointer when performing the data transfer. When set to 1, the target shall not transfer data out of order. This bit can only be set to 1 when bit 10 is set to 0 (indicating no scatter/gather function).

bit 8 reserved

bit 7 ordered command - When set to 1, indicates that the target shall treat this command as an ordered command as defined in the SCSI-3 queuing model. Note that this bit is not used for SCSI commands. The functionality provided by this bit is implemented in the SCSI Control field in the command dependent area for SCSI commands.

bit 6 sequential data transfer - When set to 1, indicates that the target shall not transfer the data out of order for this command.

bit 5 raw data transfer - When set to 1, indicates that the target shall not perform any data correction on the data.

Command Dependent Size field - contains the size of the command dependent area which immediately follows the CB. The units for the size is 16 bytes - e.g., a value of 2 in this field indicates 16*2 or 32 bytes of additional command dependent data.

SCSI Entity Address (8 bytes) - The first byte (i.e., highest order) in this 8 byte field identifies a target LUN or target routine. The remaining 7 bytes are left for user definition to identify a single logical unit in a target subsystem. The target implementor may choose to use these 7 bytes to define a hierarchical space of units or target routines.

SCSI Control (4 bytes) - This field contains information which controls command execution. The definition of bits in this field are as follows:

Byte 0 (MSB) - reserved

Byte 1 - reserved

Byte 2 -

Byte 3 - bit 7-3 - reserved

bit 2 - reserved

bit 1 - Ordered Queue

bit 0 - Simple Queue

SCSI Transfer Length (4 bytes) - This value contains the number of bytes to be transferred as a result of this command.

SCSI CDB (16 bytes) - This field contains the SCSI Command Data Block (CDB), which is defined in the SCSI–3 Command Set documents.

4.5. Data Packet

This section describes the format of a data packet. This packet is used to transfer data between 2 devices on the IEEE P1394 bus.

The data packet is preceded by an IEEE P1394 header, as defined above. Immediately following the packet header is the data field. The format of the data field is diagrammed below and described in the text which follows:

Data	Size
CRC	4

Data Field

Data (*size* bytes) - This field contains the data being transferred. This data can be anything that cannot be accommodated in the packet header. Any data transfer longer than 4 bytes must be done using a data packet containing a data field. The data packet is used to transfer CBs as well as read and write data. The size of the data field is contained in the length field of the packet header.

The 4 byte CRC field covers only the data field and is checked by the hardware independently of the CRC which covers the header field.

5. IEEE P1394 Command Delivery and List Management

This section describes how to construct commands in the form of a CB, deliver those CBs to the chosen target and manage the target's command list.

5.1. Constructing an IEEE P1394 Command Block (CB) from a SCSI Command Data Block (CDB)

The SCSI specification defines a rich command set for controlling a variety of peripheral device classes. There is an intrinsic advantage in carrying over that command set to the IEEE P1394 environment. This section describes how to take a SCSI-3 CDB and encapsulate it in an IEEE P1394 CB such that it can be acted on by an IEEE P1394 peripheral.

While there is a straight forward mapping of SCSI-3 CDBs onto IEEE P1394, there are two commands which are perhaps the most commonly used commands in SCSI-3 and which have an alternative implementation in the IEEE P1394 environment. These commands are the "read" and "write" commands. The first section below describes how the read and write commands map into IEEE P1394. The section following that describes the general algorithm for mapping any SCSI-3 CDB other than read and write into an IEEE P1394 CB.

5.1.1. Mapping "read" and "write" into an IEEE P1394 CB

The IEEE P1394 CB implements the read, write and third party copy functions. This section describes how to construct a read and write CB from a SCSI CDB. The third party copy function is described in another section of this document.

The format of the IEEE P1394 CB for the read and write function is as follows:

	Size of parameter	
Next CB Pointer	8	
Status Block Pointer	8	
Status Tail **	8	
Initiator Wakeup Register Ptr	8	
Initiator Wakeup Value	4	
Initiator Buffer Length	4	
Initiator Buffer Pointer	8	
Target Buffer Pointer	8	
Target Buffer Length	4	
Queue Control	4	
SCSI Entity Address	8	Command
SCSI Control	4	Dependant
SCSI Transfer Length	4	Area

Read CB

This is the same as the format of the normal CB, except that the SCSI CDB field is removed. All of the information necessary to perform a read or write is contained in this CB. Native IEEE P1394 initiators will most likely generate this CB directly, however, for compatibility reasons, below is a description of how to derive the target information fields for this CB from a SCSI CDB.

The following example describes how to derive the target fields of an IEEE P1394 read CB from a 6 byte SCSI read CDB. The logical block size for the target is 512 bytes.

Following is an example SCSI 6 byte CDB which describes a read command:

244

		Byte #	
Op-Code		0	0000 1000
LUN	LBA	1	000 | 0 0110
LBA (Cont)		2	0001 1001
LBA (Cont)		3	0101 1100
Transfer Length		4	0010 0111
Control		5	0000 0000

SCSI 6 byte CDB example

The SCSI CDB parameters have the following values:

Op-Code - 08 (SCSI 6 byte read)

LUN - 0

Logical Block Address - 06195Ch

Transfer Length - 27h

Control - 0

The following text describes how to transform the fields above into the necessary target fields of the IEEE P1394 CB.

Queue Control Field:

cmd: The value for this field is derived from the SCSI op code in the CDB. The SCSI "read" op-code maps into the IEEE P1394 "read" command, which is 1.

Q#: This field is not used for the read command. It should be filled with zeros.

Buffer/Ordering Control: The target specific bits in this field are 9, 10 and 11. The settings for these bits are derived as follows:

Bit 9: Set to 0 (i.e., the target buffer pointer address shall be incremented during execution of this command).

Bit 10: Set to 0 (i.e., the target buffer pointer is the source address of the data for this read command, as opposed to the address of a scatter/gather list).

Bit 11: Set to 1 (i.e., the target buffer pointer field is valid).

Cmd Dep Size: The command dependent size for the read command is 16 bytes. This translates into a value of 1 for this field (the units for this field are 16 bytes).

Target Buffer Pointer: This field is derived from the value in the LBA field of the SCSI CDB. This field is a byte address, whereas the LBA field is a logical block address. Therefore, the value for this field is equal to the LBA times the logical block size (06195Ch times 512d = C32B800h, or in binary: 0000 0110 0001 1001 0101 1100 * 10 0000 0000 = 1100 0011 0010 1011 1000 0000 0000). Note that this is a left shift by 9.

Target Buffer Length: This field is derived from the value in the Transfer Length field of the SCSI CDB. Note that in the general case, this field defines the size of the Target Buffer or the size of the optional scatter/gather list. In the general case, this may or may not equal the actual number of bytes requested for this command. The requested number of bytes is always found in the SCSI Transfer Length field discussed below.

For the purposes of this discussion, the value for this field is equal to the value in the SCSI Transfer Length field discussed below.

SCSI Entity Address: This field is derived from the SCSI CDB LUN field. For the purposes of this example, assume that a LUN of zero translates into a SCSI Entity Address of zero.

SCSI Control: This field is derived from the optional queue tag message which may precede the SCSI CDB (i.e., ordered queue or simple queue tag).

SCSI Transfer Length: This field is derived from the value in the Transfer Length field in the SCSI CDB. This field is a byte count, however, the SCSI CDB Transfer Length field is in units of logical blocks. Therefore, the value for this field is equal to the Transfer Length field times the logical block size (27h times 512d = 4E00h or in binary: 0010 0111 * 10 0000 0000 = 0100 1110 0000 0000). Note that this is a left shift by 9.

All the information contained in the SCSI CDB has now been translated into the appropriate fields of the IEEE P1394 CB. Before the initiator can give this CB to the target, however, it must first fill out the following additional fields:

Status Block Pointer

Status Tail Pointer Pointer

Initiator Wakeup Register Pointer

Initiator Wakeup Value

Initiator Buffer Length

Initiator Buffer Pointer

The following bits in the Buffer/Ordering Control field of the Queue Control field:

bit 15 - Initiator buffer valid

bit 14 - Initiator buffer indirect

bit 13 - don't increment initiator buffer pointer

bit 6 - sequential data transfer

bit 5 - raw data transfer

These fields contain information that controls how the target delivers data to the host. For details on filling in these fields, see the following section.

5.1.2. Mapping any SCSI-3 CDB other than "read" and "write" into an IEEE P1394 CB

Any SCSI CDB can be transformed into an IEEE P1394 CB independent of target implementation. No assumptions are made about the format or content of the CDB. This discussion only deals with encapsulating CDBs in the IEEE P1394 CB data structure.

The steps necessary to construct a CB are as follows:

1. Construct a normal SCSI CDB.
2. Place the CDB into a CB data structure at the defined offset.
3. Construct the SCSI Control field by setting the appropriate flags for this command. These flags affect how this command is handled in relation to other commands in the list.
4. Fill in the Command Dependent Size field of the CB. This is a fixed number and is equal to the size, in bytes, of the command dependent area.
5. Fill in the Next CB Pointer with null if this is the last command to be enqueued for now. Otherwise, fill it in with the 64 bit address of the next CB in the list.

6. If the No Status bit in the SCSI Control field is set to zero, fill in the Status Block Pointer field with the 64 bit address of the status block where the target is to write the command status upon completion of this command.
7. Fill in the Initiator Buffer Pointer with a 64 bit address of the data buffer for this command.
8. If the No Status bit in the SCSI Control field is set to zero, fill in the Status Tail Pointer Pointer field with a pointer to the status tail pointer that the target is to use to enqueue status upon completion of this command.
9. If the No Wake-up bit in the SCSI Control field is set to zero, fill in the Initiator Wake-up Register Pointer field with the 64 bit address of the initiator's wake-up register. This is the address to which the target should write the Initiator Wake-up Value upon completion of this command.
10. If the No Wake-up bit in the SCSI Control field is set to zero, fill in the Initiator Wake-up Value field with the value that the target should write to the Initiator Wake-up Register address upon completion of this command.

At this point, the Command Block is ready to be enqueued to the target using the algorithm described below.

5.2. Command/Status Queuing/Dequeuing Overview

This section provides an overview of the IEEE P1212.1 list DMA model. For the definitive description of this model, see the IEEE P1212.1 List DMA documentation.

5.2.1. Necessary Bus Specific Facilities

This section describes the IEEE P1394 and IEEE 1212 facilities which are needed to support the enqueuing mechanism.

5.2.1.1. Lock Transaction

A lock transaction is an atomic operation which can be used to implement a test and set or a swap function. The lock facility defines several subcommands, all of which are indivisible operations. The lock subcommands needed for the list model are:

Mask_Swap - Used in the enqueuing mechanism to perform an unconditional swap on the contents of an address in IEEE 1212 address space. This subcommand writes the supplied value to a register and returns the old value in the response packet.

Compare_Swap - Used in the enqueuing model to do a test and set. This subcommand compares the contents of a register in IEEE 1212 address space with an expected value and either replaces the value with a supplied value (if the compare succeeds) or returns the old value in the response packet (if the compare fails). In either case, the response packet indicates which action was taken.

5.2.1.2. IEEE 1212 CSR Registers Necessary for Implementation of a Sharable List

A target which implements a sharable list must implement the following CSR registers[2]:

Command list (C list) registers:

Command List Head Pointer - an IEEE 1212 64 bit address pointer which contains the IEEE 1212 64 bit address of the first CB in the sharable list, or null if the list is empty.

2 There are other registers not listed here which are required in order to completely implement a sharable list according to the IEEE 1212.1 list DMA specification. The function of the CSR registers not listed here include initialization, startup and shutdown of the sharable list. The CSR registers listed here are those used in normal operation, and are therefore sufficient to permit a complete discussion of the operation of sharable list DMA. Refer to the IEEE 1212.1 shared memory DMA specification for a complete description.

247

Command List Tail Pointer - an IEEE 1212 64 bit address pointer which contains the IEEE 1212 64 bit address of the last CB in the sharable list. If the sharable list is empty, the tail pointer contains the IEEE 1212 64 bit address of the command list head pointer.

Command List Target Wake-up Register - an IEEE 1212 CSR register which, when written to, wakes up the target. When an initiator writes to the target wake-up register, it indicates to the target that there are new elements in the command list that may require attention.

Priority list (P list) registers:

Priority List Head Pointer - this register serves an analogous function to the C List Head Pointer CSR. It is used to implement a sharable list which is separate from but related to the C List.

Priority List Tail Pointer - this register serves an analogous function to the C List Tail Pointer CSR. It is used to implement a sharable list which is separate from but related to the C List.

Priority List Wake-up Register - this register serves an analogous function to the C List Target Wake-up CSR. It is used to implement a sharable list which is separate from but related to the C List.

The 6 registers described above implement 2 separate but parallel and related lists (the C list and the P list). The method for adding commands to a list and removing commands from a list and returning status are the same for both the C list and the P list. Initiators wishing to issue commands to a target will enqueue CB's to the target's C list. Initiators requiring execution of priority commands or control functions (such as deleting commands from the C list) issue CB's to the P list.

Adding commands to a list involves performing the atomic bus operations Mask_Swap and Compare_Swap on the target registers List Head Pointer (or any location to which the List Tail Pointer points), List Tail Pointer and a write bus operation to the Wake-up Register.

There shall be no situation under which a command enqueue will fail due to insufficient target resources. This is possible because the command enqueue algorithm (outlined below) does not require any target resources, other than access to the appropriate CSR registers. This eliminates the need for special retry or error recovery software to handle exception conditions.

5.2.2. Adding Command and Status Blocks to a List

In order for an initiator to enqueue a list of CBs to the target's list, the enqueuing initiator performs the following steps. The algorithm for enqueuing a single CB can be derived from the algorithm outlined below.

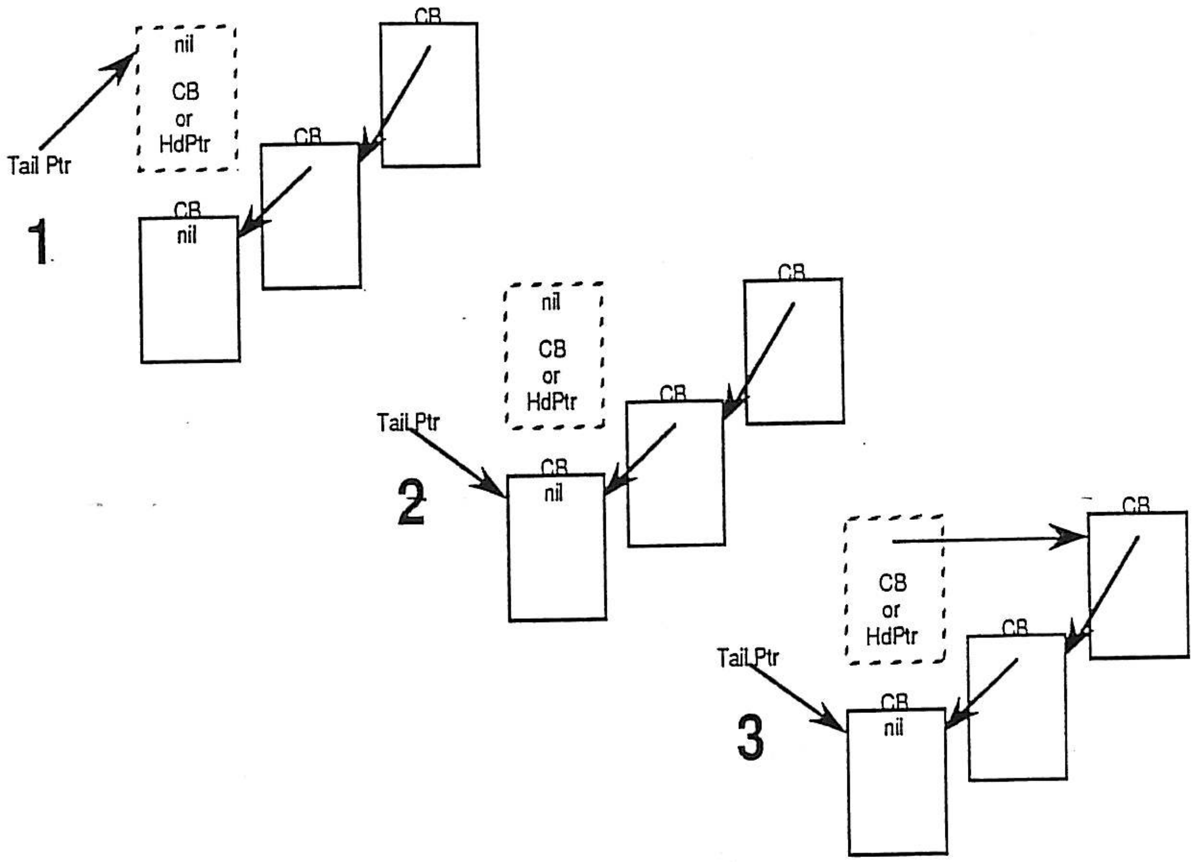
For the sake of this discussion, assume that the initiator has already constructed one or more CBs as described elsewhere in this document. Also assume that if there is more than one CB to be enqueued, the initiator has constructed a linked list of the CBs as described elsewhere in this document.

Step 1 Perform a Mask_Swap on the target's command list tail pointer using the address of the last CB in the list of CBs to be enqueued. Note that the "Next CB Ptr" field of this CB must be null. The Mask_Swap places the address of the proper CB into the target's list tail pointer and returns the old value of the tail pointer to the initiator.

Step 2 Perform a Mask_Swap to the address returned from the last operation using the address of the first CB in the list of CBs to be enqueued. This action places the address of the first CB into the address previously pointed to by the tail pointer.

Step 3 Write the target's wake-up value to the target's wake-up register. This action "wakes up" the target and indicates that the command list has grown.

These three steps are illustrated graphically in the following diagram and further described below:



Label 1 The tail pointer points to either a command block or to the list head pointer. In general, the list tail pointer always points to the location where the enqueuing initiator should write the address of the next CB to add to the list.

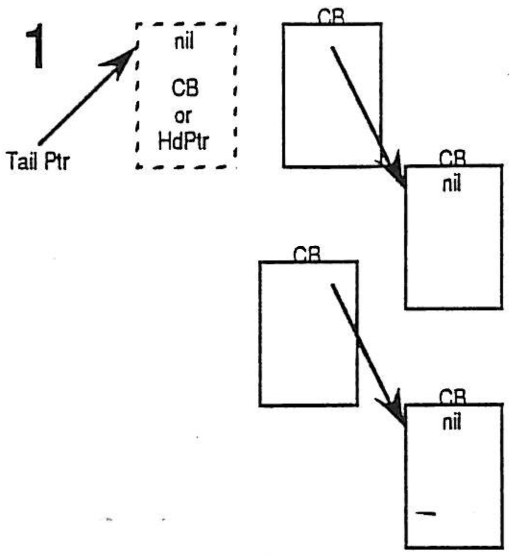
Label 2 The enqueuing initiator has performed the swap on the tail pointer thereby causing the tail pointer to point to the last CB in the list of CBs which are being enqueued.

Label 3 In step 2, the result returned from the swap on the tail pointer was the address of the location to which the tail pointer was previously pointing. The enqueuing initiator writes the address of the first CB in his list thereby adding the new list of commands to the target's list.

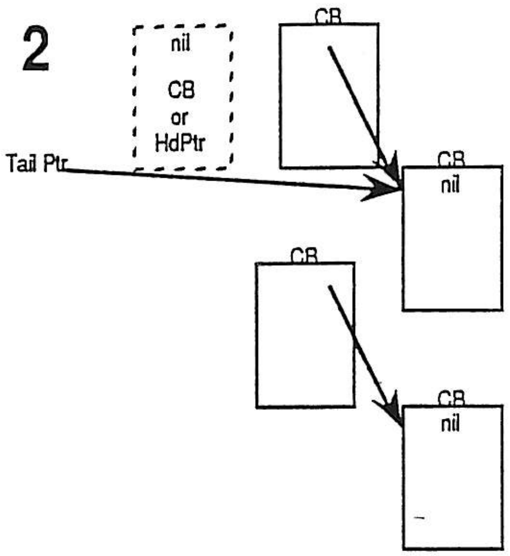
Note that the state of the list prior to the above enqueue operation is not relevant. Also note that the head pointer plays no significant role in the list operation. It merely serves as a place holder for a pointer to the first command in the list in situations where the list is empty. When the list is not empty, the place holder is the "Next CB Ptr" field of the last CB in the list.

Extending this algorithm to the case of 2 initiators attempting to enqueue commands simultaneously to the same target, note the following diagram and the text which follows:

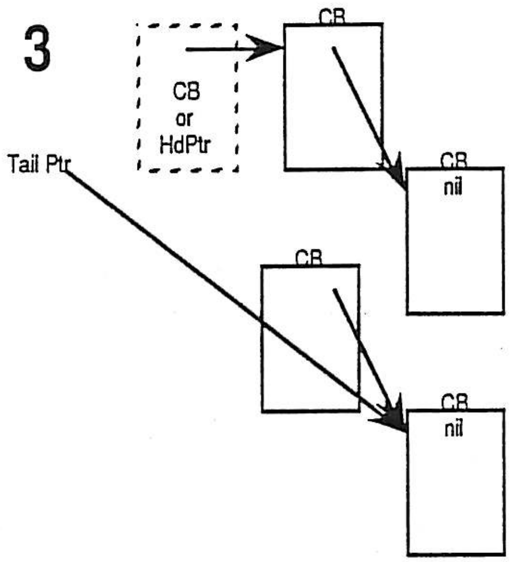
1



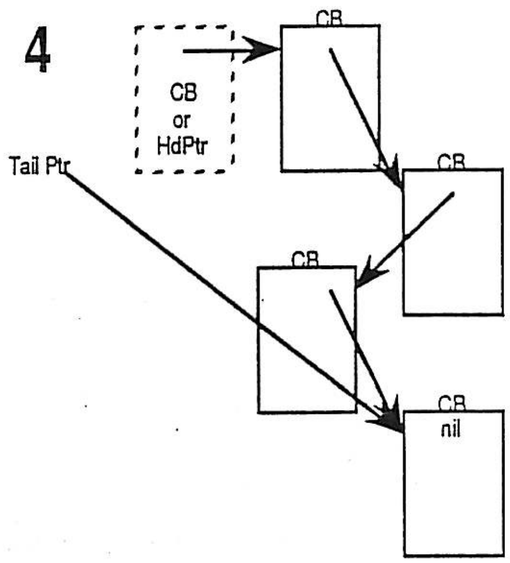
2



3



4



Label 1 In the initial state depicted at label 1, the tail pointer points to either the head pointer or the "Next CB Ptr" in the last CB in the list. The pointer which the tail pointer points to always contains null (i.e., end of list).

Label 2 Initiator 1 performs a swap on the list tail pointer. This forces the target's tail pointer to point to the last CB in initiator 1's list. When initiator 1 performs this swap, it receives in response the old contents of the list tail pointer which, in this case, is the address of either the head pointer or the address of the "Next CB Ptr" field of the last CB in the list.

Label 3 In this step, initiator 2 performs a swap on the contents of the list tail pointer. This forces the target's tail pointer to point to the last CB in initiator 2's CB list. When initiator 2 performs this swap, it receives in response the old contents of the list tail pointer, which, in this case is the address of the "Next CB Ptr" in the last CB in initiator 1's list.

Also in label 3, initiator 1 performs a swap on the address returned from the swap on the tail pointer (i.e., it uses the result of the last swap as an address on which to perform the next swap). This results in the previously null pointer pointing to the first CB in initiator 1's list.

Label 4 Initiator 2 performs a swap on the address it received in response to the swap that it performed in step 3. This results in forcing the "Next CB Ptr" in the last CB of initiator 1's list to point to the first CB in initiator 2's list.

After initiator 1 and 2 have performed the above operations, they write to the target's wake-up register. This will have no effect if the target is already awake and fetching commands.

Note that there is a window of time in the above scenario during which the pointers are not in a consistent state. This occurs in the figure at label 3 where the tail pointer points to the last CB in initiator 2's list, and the previous "head" pointer points to the first CB in initiator 1's list, but the "Next CB Ptr" in the last command of initiator 1's list does not yet point to the first CB in initiator 2's list. The dequeuing mechanism, described in the next section, explains how the target detects this condition, and how it deals with it such that the protocol does not break down.

5.2.3. Dequeuing items from the List

This section covers the steps that a target goes through to read the linked list of CBs and remove items from the list. The command dequeue protocol complements the command enqueue protocol described in the above section. As the initial state, assume that the target has just been awakened by an initiator writing to the target's wake-up register. At this point, the target performs the following steps:

Step 0 Wait for a write to the target wakeup register.

Step 1 Read the command list head pointer. The will be the address of the next CB to process.

Step 2 Read and process CBs until the "Next CB Ptr" field is null. Do not process this command.

Step 3 Perform a swap on the head pointer and replace it with null.

Step 4 Perform a Compare_Swap on the tail pointer. The arguments for this operation are as follows:

Expected value = Address of last CB read (note that this CB has not been processed yet)

Swap value = Address of Command List Head Pointer

Step 5 If the previous operation succeeds (i.e., the compare and swap was successful), process the last CB, then return to step 0 above.

Step 6 If the Compare_Swap done in step 4 failed, write the address of the last CB (which has not yet been processed) into the head pointer, then return to step 0 above.

Note that it is not necessary to write the address of the last CB into the head pointer, as described in step 6. Doing so keeps the algorithm pure (i.e., following any wakeup from a dormant state, the first command to process is the one that the head pointer points to). An alternative to step 6 is the following:

Step 6' Process the last CB, but do not return status for this command yet.

Step 7' Wait for a write to the target wakeup register.

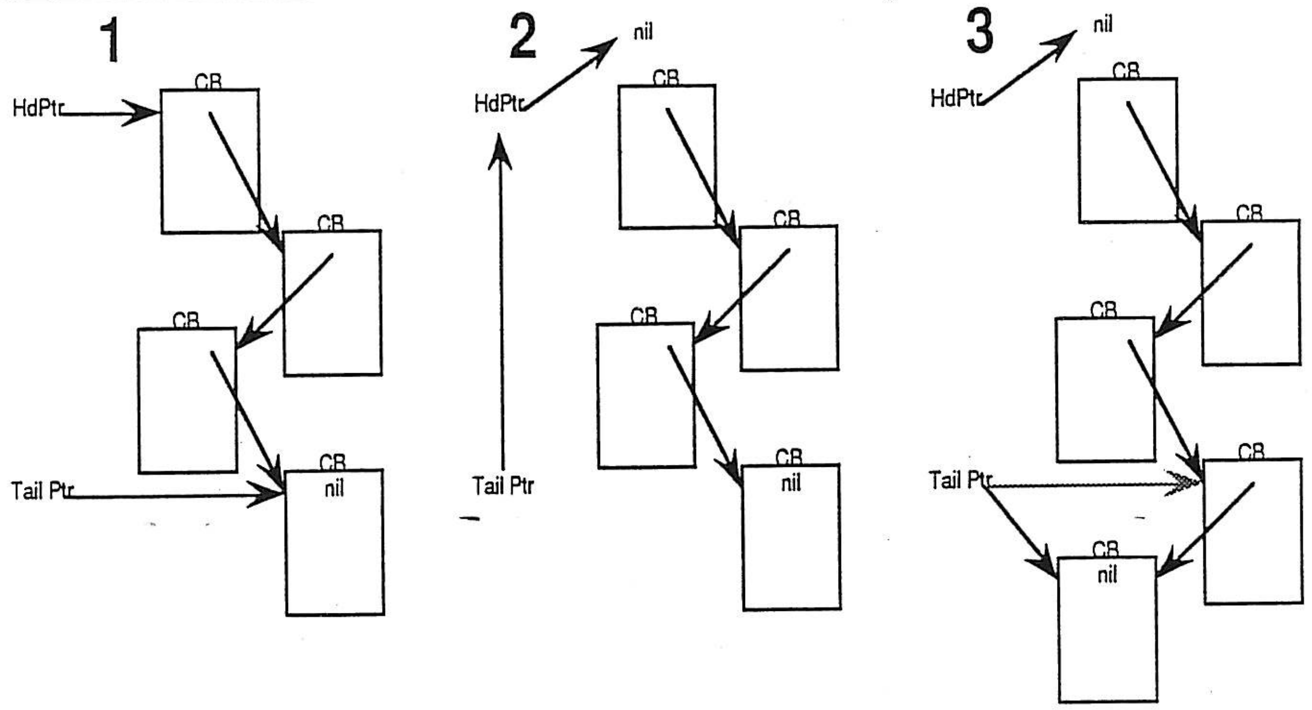
Step 8' Read the "Next CB Ptr" field of the last CB processed. This will be the address of the next CB to process.

Step 9' Return status for the last CB processed.

Step 10' Return to Step 2 above.

This alternative approach requires a differentiation between being woken up with an outstanding command, and being woken up with no outstanding commands. However, the alternative results in one less IEEE P1394 bus write.

The alternative algorithm is illustrated in the following diagram:



Label 1 is immediately after being woken up from a dormant state and there are no outstanding commands to process. .Assume that the target processes all the commands in the list and is about to attempt to progress to step 3 in the above algorithm.

In diagram 2, the target has replaced the head pointer with null, then successfully performed the compare and swap operation on the tail pointer. For this operation to succeed, the tail pointer must have been pointing to the last CB in the command list. At this point, the tail pointer points to the head pointer. The target then processes the last CB and returns status for it. At that point, the target may fall asleep.

In diagram 3, the target has replaced the head pointer with null and performed a compare and swap operation on the tail pointer. The tail pointer compare and swap failed because the tail pointer has been swapped by another enqueuing initiator and it no longer points to the last CB in the list that the target has completed. The target's compare and swap operation leaves the contents of the tail pointer un-modified.

The target waits for a write to it's wakeup register. When the target is woken up, it examines the location to which the tail pointer was previously pointing. If this location is null, the target goes to sleep again and waits for another wake up. If, however, the pointer is not null, the target processes the last CB in the list, then continues processing commands until it again reaches a CB whose "Next CB Ptr" field is null.

One important observation to make here is that a target can never return status for any CB while the tail pointer is pointing to it.

5.3. Priority Commands and List Management

In an IEEE 1212 device, lists are implemented in pairs. The enqueuing mechanisms described above applies to both lists in the pair.

The C list, or Command list, is the list which is normally used by initiators to present CBs to a target. The associated P list is used to control execution of commands in the C list.

This section describes the differences between the C list and the P list.

5.3.1. Priority Commands

The P list works the same as the C list except that any CB which is enqueued to a P list has a higher priority than any CBs in the associated C list.

When a CB or a list of CBs is enqueued to a P list, the target shall execute that CB or list of CBs at the earliest possible time, relative to commands in the associated C list. Note that a single target can implement multiple C list/P list pairs. The IEEE 1212 standard only defines the priority of commands in a P list relative to the associated C list, not relative to any other C list/P list pair that may exist.

Although the IEEE 1212 standard defines a separate wakeup register for the P list, this register may be at the same physical address as the C list wakeup register. In this case, the value written to the wakeup register determines whether the C list has changed or the P list has changed[3].

If a target implements more than one C list/P list pair, it must have unique list head and tail pointers for each list. In addition, it must have either a unique wakeup register for each list, or a unique wakeup value for each list, or both.

Commands placed in the P list are executed in a FIFO order, (i.e., the target shall not re-order commands in the P list and it shall execute them in the order in which they appear in the list).

SCSI CBs may be placed in either the C list or the P list. SCSI CBs placed in the P list will be execute at the next possible opportunity relative to commands in the C list. This is analogous to the functionality provided by the HEAD OF QUEUE TAG message in SCSI2.

5.3.2. Deleting Commands from a List

CBs in a list are uniquely identified by their 64 bit IEEE 1212 address. This is guaranteed to be unique for all CBs in the system. Note that this address is not contained in the CB itself. Therefore, targets which make internal working copies of CBs must remember the 64 bit IEEE 1212 address from which each CB is read.

The initiator deletes a CB from a list by issuing an ABORT CB to the P list. This ABORT command must be added to the P list which is associated with the C list containing the command to be aborted.

Contained in the ABORT command is the 64 bit IEEE 1212 address of the CB to abort. When the target receives an ABORT command on a P list, it searches through the associated C list until it finds the CB at the address indicated in the ABORT command. Note that the target may already have made a local copy of the CB to be aborted. In this case, the target must maintain an association between internal copies of CBs and the IEEE 1212 address from which they were read.

When the target finds the CB to be aborted, it removes it from the list and immediately returns a "command aborted" status using the normal status reporting algorithm described elsewhere in this document. The target is permitted to modify the "Next CB Ptr" field in the previous CB in the list as part of the activity associated with aborting a CB. Regardless of whether the target does this, the target cannot modify nor rely on the contents of any field in the CB which was aborted after it has returned the command aborted status for that CB.

After returning the command aborted status for the aborted CB, the target must return a successful completion status for the ABORT CB.

If the CB to be aborted is not found, the target shall report appropriate status for the ABORT command. If the CB to be aborted is currently in progress, the target shall return an appropriate status for the ABORT command. This status shall be different from the status returned for successful completion and for the situation where the command is not found.

[3] Note that the 1212 spec. defines a wakeup value for each wakeup register. In principal, the wakeup value could be the same for multiple wakeup registers at different physical addresses, or the physical address of the wakeup register can be the same for multiple operations that have different wakeup values, but never both.

5.4. Target use of CB

This section describes the target's activity when decoding the CB. This section does not describe how the target receives commands, but rather, it assumes that the target has already received a CB and is ready to perform the requested function. Details of the command delivery mechanism are described elsewhere in this document.

The target receives the entire CB and performs the following actions:

Examine the SCSI control bits to determine how this command should be handled in relation to other commands in the list. (For the sake of this example, assume that this is the only command and the target may begin operating on it immediately).

Decode the CDB (For the sake of this example, assume that the CDB is a disk read command).

Initiate the disk seek operation and prepare the data for transfer.

When data is available to return to the initiator, packetize the read data into packets of a size which is allowable by the maximum packet size, as currently defined.

Note:

Each data packet is preceded by an IEEE P1394 header which contains the destination address and length, in bytes, of the data field. The destination address of each packet is calculated by adding the offset of the data in the packet to the original "Initiator Buffer Pointer" for this command. Data may be transferred in any order.

If desired, the initiator can validate the data packet by examining the SRC ID, destination address and length fields of the packet header.

If the Status Block Pointer is valid (as determined from the SCSI control bits), write the status for this command to the status block.

Enqueue the status block into the status list using the Status Tail Pointer Pointer

Write the Initiator Wake-up Value to the Initiator Wake-up Register

The method of enqueuing status is the same as that used to enqueue commands.

6. Mapping SCSI Functionality onto IEEE P1394

This section discusses the SCSI messaging system and the manner by which the functionality of that system is mapped onto IEEE P1394.

This section also discusses SCSI commands whose functionality is implemented in IEEE P1394 without using a SCSI CDB.

6.1. SCSI Commands

With the IEEE 1212 command delivery mechanism, any SCSI CDB can be placed in a CB and added to a list for a given target. However, there are some SCSI commands whose functionality is implemented in the IEEE P1394 environment using the IEEE 1212 unit ROM architecture. This section describes these SCSI commands and the method by which their functionality is implemented in IEEE P1394.

For the purposes of this discussion, the SCSI command set is broken into 2 types of commands:

1. Commands which configure the target or retrieve operating parameters or status from the target
2. Commands which direct the target to perform a specific function or action

Examples of the first type of command are "MODE SELECT" and "MODE SENSE", while examples of the second type of command are "READ", "WRITE" and "FORMAT".

The IEEE 1212 specification describes a directory structured unit ROM architecture for accessing and modifying information about a unit on the IEEE P1394 bus. In other words, the IEEE 1212 specification provides the functionality of type 1 SCSI commands.

The following table contains commands which apply to all device types:

Command	Command type
CHANGE DEFINITION	1
COMPARE	2
COPY	2
COPY AND VERIFY	2
INQUIRY	1
LOG SELECT	1
LOG SENSE	1
MODE SELECT	1
MODE SENSE	1
READ BUFFER	2
RECEIVE DIAGNOSTIC RESULTS	2
REQUEST SENSE	1
SEND DIAGNOSTIC	2
TEST UNIT READY	1
WRITE BUFFER	2

6.1.1. Type 1 Commands (Configuration and Setup)

The commands in this section serve to provide information to the initiator about device characteristics. These commands also permit an initiator to configure a target, where changeable parameters exist. For IEEE P1394 devices this same functionality is provided by the IEEE 1212 unit ROM architecture.

The IEEE 1212 unit ROM is a directory based data structure containing information about device characteristics. Entries in the ROM are not directly modifiable, however, parameters which are changeable may be modified by other units on the IEEE P1394 bus through other means (i.e., not by directly writing to the IEEE 1212 ROM space, but through an alternative mechanism described in this section).

The IEEE 1212 unit ROM is a directly addressable read-only address space in each IEEE P1394 unit's node address space. The information in the unit ROM is directory based and is covered by CRC for data integrity. It is not necessary for a device to read or verify the CRC bytes in order to retrieve and interpret the information contained in the unit ROM.

The format of the IEEE 1212 unit ROM is described in the IEEE 1212 Unit Summary document. This document refers to entries in the unit ROM by name. The details of parsing the directory and retrieving the unit's configuration data are not covered in this document.

6.1.1.1. CHANGE DEFINITION Command

This command changes the operating definition of a logical unit or target.

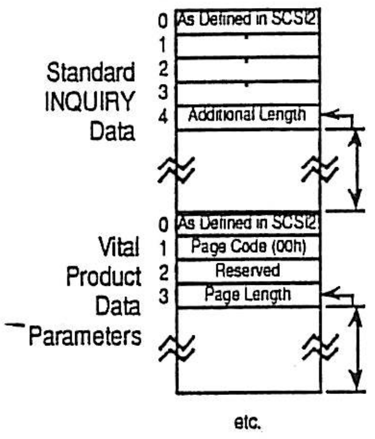
This command is not supported in IEEE P1394.

6.1.1.2. INQUIRY Command

This command returns information about the target.

In IEEE P1394, this is not implemented as a command. Instead, a pointer to the Inquiry data and the optional Vital Product Data pages is located in the unit ROM, as described in this section.

There is an entry in the Unit Directory table which contains a pointer to the INQUIRY data. The structure of the information that this pointer references is as follows:



The first block of data in the INQUIRY section of the unit information is the standard INQUIRY DATA. The format of the this data is the same as that defined in the SCSI2 spec. section 7.2.5.1.

Immediately following the standard INQUIRY data are one or more vital product data pages. The format of these pages is the same as that defined in the SCSI2 spec. sections 7.3.4.x. Vital Product Data Parameter page 0 shall always be supported by IEEE P1394 SCSI devices.

The first Vital Product Data Parameter page following the standard INQUIRY data shall be the Supported Vital Product Data Page (page 00h). The format of page 00h shall be the same as that defined in the SCSI2 spec. section 7.3.4.4. A page length of zero in page 00h indicates that the device does not support any Vital Product Data Parameter pages other than page 0.

Following Vital Product Data Parameter page 00h, there can be other Vital Product Data Parameter pages. These pages, if implemented, shall be located in numerically increasing order following page 0.

6.1.1.3. LOG SELECT/LOG SENSE Commands

These commands retrieve and set the unit's sense information.

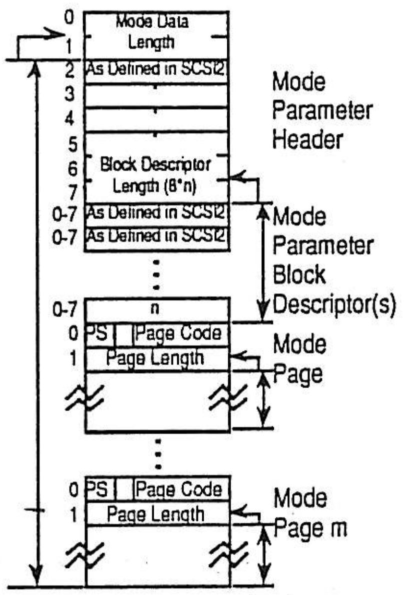
Implementation of the functionality of these commands has yet to be determined.

6.1.1.4. MODE SELECT/MODE SENSE Commands

These commands retrieve and set the unit's mode information.

In IEEE P1394, these commands are not implemented as commands. Instead, a pointer to the mode sense data is located in the unit ROM, as described in this section.

There is an entry in the Unit Directory table which contains a pointer to the mode sense data. The structure of the information that this pointer references is as follows:



The above figure is a representation of the mode sense data structure found in the units IEEE 1212 address space.

The first 8 bytes in this data structure has the same format as the 10 byte Mode Parameter Header defined in section 7.3.3 of the SCSI2 spec. Some of the elements in that data structure are discussed here for completeness. For a full description of this data structure, see the SCSI2 spec.

The first 2 bytes contain the length, in bytes, of the entire Mode Sense data structure. The 2 byte length field is not included in this size.

At offset 6 is a 2 byte field which contains the length, in bytes, of all the Mode Parameter Block Descriptors. Immediately following this field is the first Mode Parameter Block Descriptor. This is an 8 byte data structure which is completely defined in the SCSI2 spec. in section 7.3.3, table 7-63.

Immediately following the last Mode Parameter Block Descriptor are the Mode Pages. Within each Mode Page at offset 1 is a field containing the length, in bytes, of the information in the Mode Page following the page length field.

There can be any number of Mode Pages. There are no requirements about the order in which the mode pages appear in the unit's address space. The definition and format of each Mode Page is completely defined in the SCSI2 spec. in section 7.3.3.

The SCSI2 MODE SENSE command returns Mode Pages to the initiator, while the MODE SELECT command allows an initiator to change the Mode Page data. In IEEE P1394, the Mode Page data is directly accessible to the initiator. By accessing the above data structure, the initiator can read in all or part of the Mode Page data, or it can parse it one field at a time.

An initiator is free to modify the Mode Page data by writing to it. The initiator must determine the expect address of the data to be modified by first parsing the Mode Page data. If an initiator attempts to change information which is not changeable (for example, if an initiator attempts to overwrite the Page Code and Page Length fields of a Mode Page), the target shall return an error status to that initiator. The target may use an IEEE P1394 split transaction to do so. The target may also return an error status to an initiator who attempts to change a parameter in a Mode Page to an illegal value.

6.1.1.5. REQUEST SENSE Command

6.1.1.6. TEST UNIT READY Command

6.1.2. Type 2 SCSI Commands (Operation)

6.1.2.1. COMPARE Command

This command compares data on one target and LUN with that on another or the same target and LUN.

The operation of this command is the same as the COPY and COPY AND VERIFY commands described below.

6.1.2.2. COPY Command

This command copies data from one target and LUN to another or the same target and LUN.

The operation of this command is substantially different from the parallel SCSI COPY command. This section describes this command and its use. Note that the command operation and operating parameters are analogous with the COMPARE and COPY AND VERIFY commands described elsewhere.

6.1.2.3. COPY AND VERIFY Command

This command performs a copy of data between one target and LUN to another or the same target and LUN, then performs a verify of the data just copied.

The operation of this command is the same as the COPY and COMPARE commands described above.

6.1.2.4. READ BUFFER Command

6.1.2.5. RECEIVE DIAGNOSTIC RESULTS Command

6.1.2.6. SEND DIAGNOSTIC Command

6.1.2.7. WRITE BUFFER Command

7. Continuous Mode Data Transfer

8. Command Grouping

A. Minimal requirements for IEEE P1394 devices

The IEEE P1394 physical interface enables a new class of peripherals which did not previously exist. For example, devices which today must have massive buffers, such as laser printers, could be implemented with substantially smaller buffering if there existed a deterministic data delivery mechanism whereby the rate of data delivery could be matched precisely to the data delivery requirement. The IEEE P1394 physical medium provides such a mechanism. However, in order to exploit this and other intrinsic features of IEEE P1394, implementors of low cost products, such as laser printers, must be able to assume a baseline feature set.

Therefore, the baseline feature set outlined in the following sections will be required of all devices described in the SCSI document which could serve as data storage devices and which implement the IEEE P1394 physical interface. The second list of implementation requirements is a necessary set for all IEEE P1394 devices in general.

Note: The lists below are not complete and are subject to substantial revision!

A.1. Requirements for all IEEE P1394 devices

Required of all IEEE P1394 devices:

1) Minimal set of IEEE 1212 unit ROM entries (to be defined, but will be a small well defined set).

A.2. Requirements for IEEE P1394 mass storage devices

Required of all IEEE P1394 devices which could be used for mass storage in a system environment (e.g., rotating media storage devices, and others):

1) Must implement a minimum of 1 isochronous channel capable of continuous data transfer at a minimum fixed rate of 1.536Mbps while concurrently acting on unrelated asynchronous commands.
2) Must support asynchronous command time-out.
3) Must support a logical block size of 1 byte. Optionally, other logical block sizes may also be supported.
4) Must support spindle synchronization to the 1394 cycle start packet. Must be able to run in spindle synch and non-spindle synch operation.
5) Must have a minimum of 2 ports.
6) Must meet all minimal SCSI requirements (i.e., mandatory commands, etc., as per the appropriate SCSI3 standards documents).

B. IEEE P1394 Unit ROM

This section specifies an IEEE 1212 compliant address space in the form of RAM, ROM and register mapping for devices which are compliant with the IEEE P1394 Serial Bus Standard.

This address space specification is intended to provide a hierarchical template of RAM, ROM and register address definitions. These definitions should be sufficient to provide generic, rudimentary services to any device, regardless of type or class.

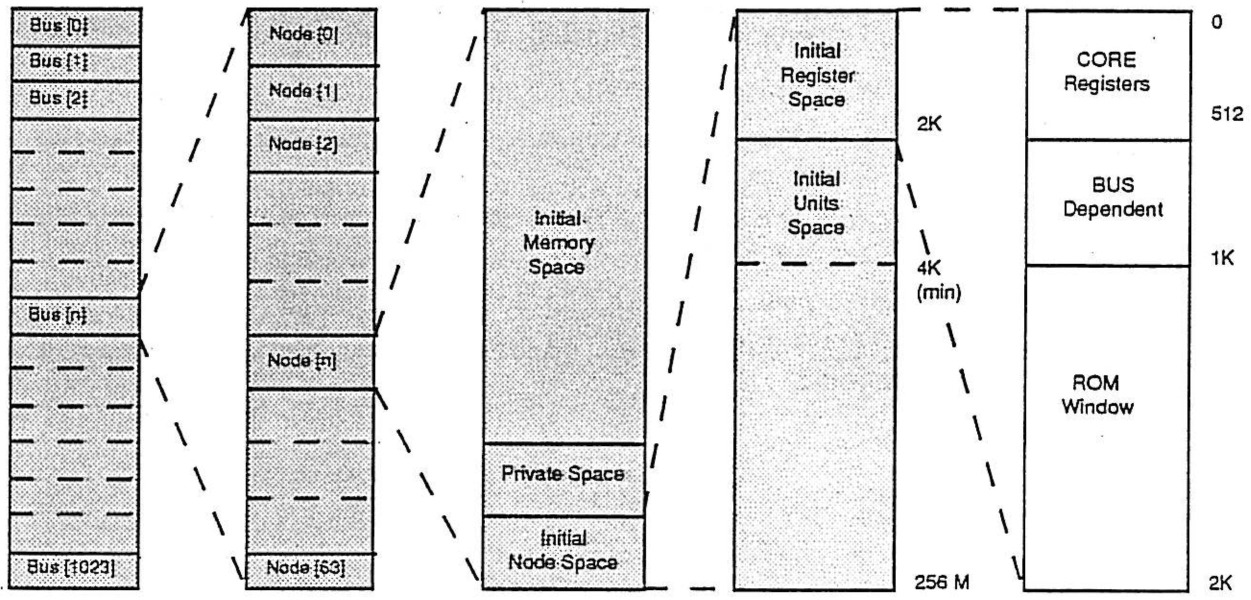
This specification defines node and unit management, Isochronous and Asynchronous I/O process and command/status support. It allows extensions in support of device type specific requirements not defined here. It ensures a level of compatibility between devices and driver software from multiple manufacturers.

B.1 Overview

This section provides a short overview of the IEEE 1212 CSR 64 bit address space. It describes bus management, Isochronous channel control and bus power control functions and register sets, each register's purpose and function, the values which may be written to or read from each register and the effects of certain actions associated with each register as appropriate. It describes the mapping of these registers and other parameters onto the 1212 CSR address space.

B.1.1 Address Space Overview

The following address space template has been structured to conform to the IEEE 1212 CSR (Control and Status Register) Architecture Standard while providing enough modularity to decrease memory requirements in the host or Monarch system. Instead of maintaining a complete copy of node and unit ROM entries, it is possible to maintain a table of references to various structures containing functionally related information and registers indexed by node and unit. Below is a diagram illustrating the IEEE 1212 64 bit fixed CSR address space.



64 Bit Address	Bus (10) Node (6)	Initial Memory Space (20)	Initial Node Space (28)

64 Bit Address	Bus (10) Node (6)	16 bits = 0	Local Memory Space (32)

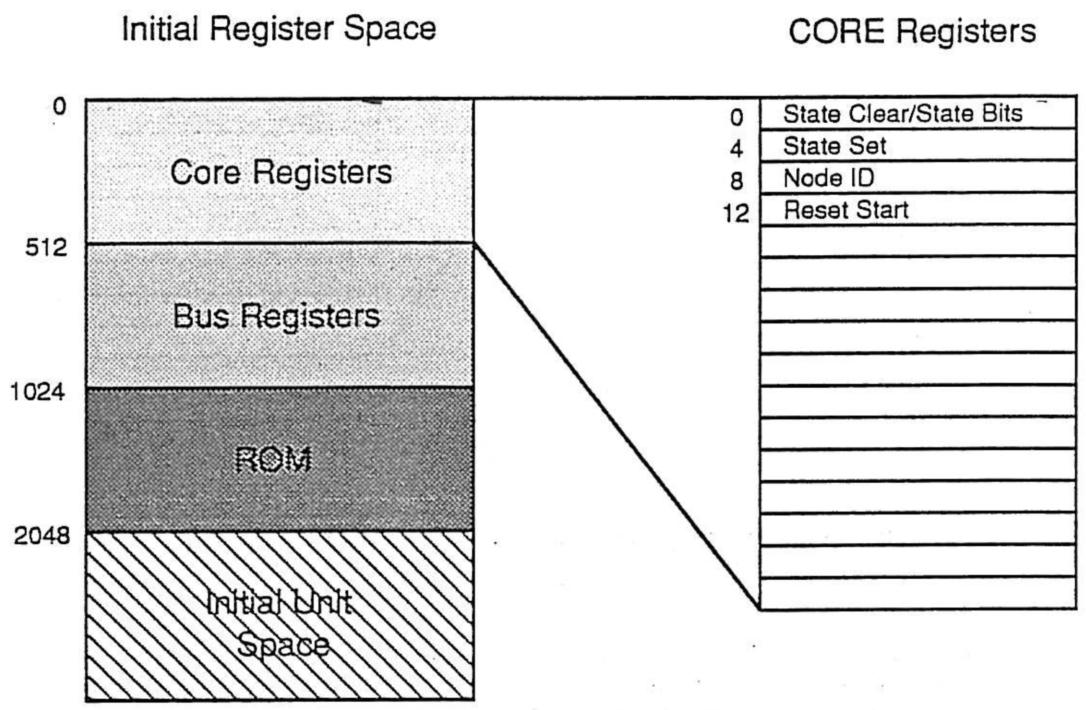
260

IEEE 64 Bit Fixed Address Space

The upper 16 bits are used to define a unit's bus and node. The remaining 48 bits are node address space. The upper 256 Mbytes are used for the node's CSR address space. This CSR address space is further divided into four areas. The Core and Bus areas are for CSR's. The ROM and initial Unit areas are for configuration ROM and unit specific register sets.

B.1.1.1 CORE - Bus Independent CSR's

The core CSR address space defines a uniform software interface which used during the initialization process. These CSR's function independently of any particular I/O bus implementation. The CSR's used in a IEEE P1394 implementation are illustrated in the diagram below.



IEEE 64 Bit Fixed Address Space - Core

State Clear Register - (R/W) see IEEE 1212 Standard section 7.4.1

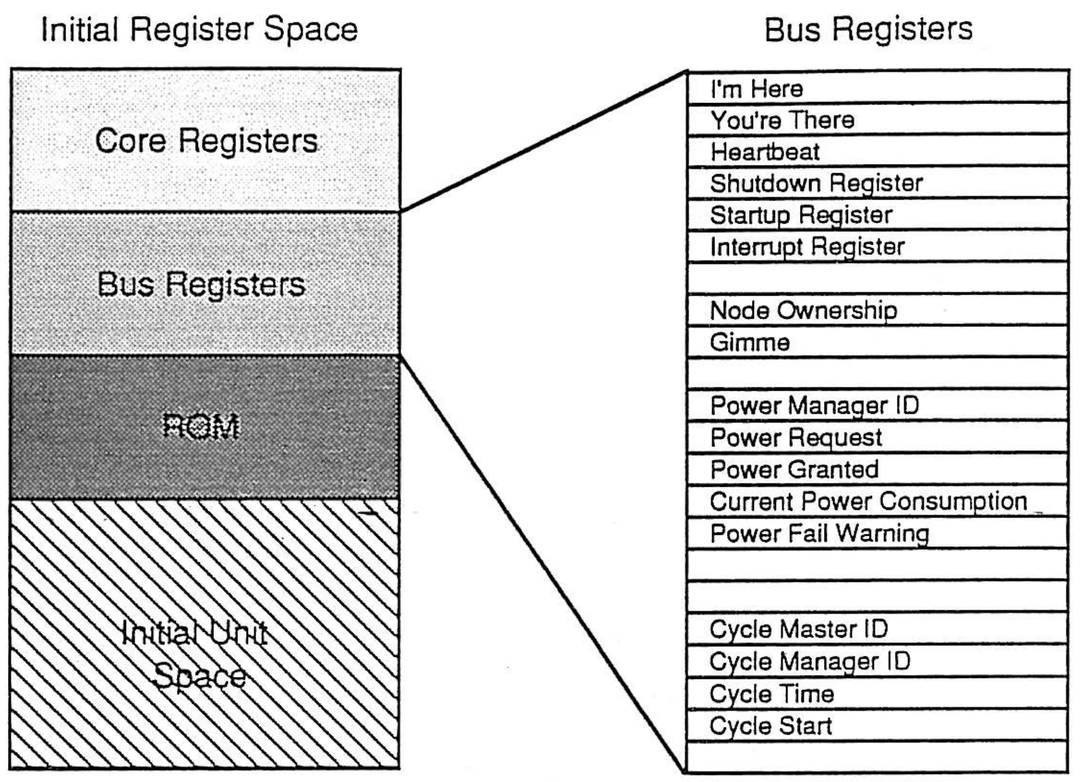
State Set Register - (W/O) see IEEE 1212 Standard section 7.4.2

Node ID Register - (R/W) see IEEE 1212 Standard section 7.4.3

Reset Start Register - (R/O) see IEEE 1212 Standard section 7.4.4

B.1.1.2 BUS - Bus Dependent CSR's

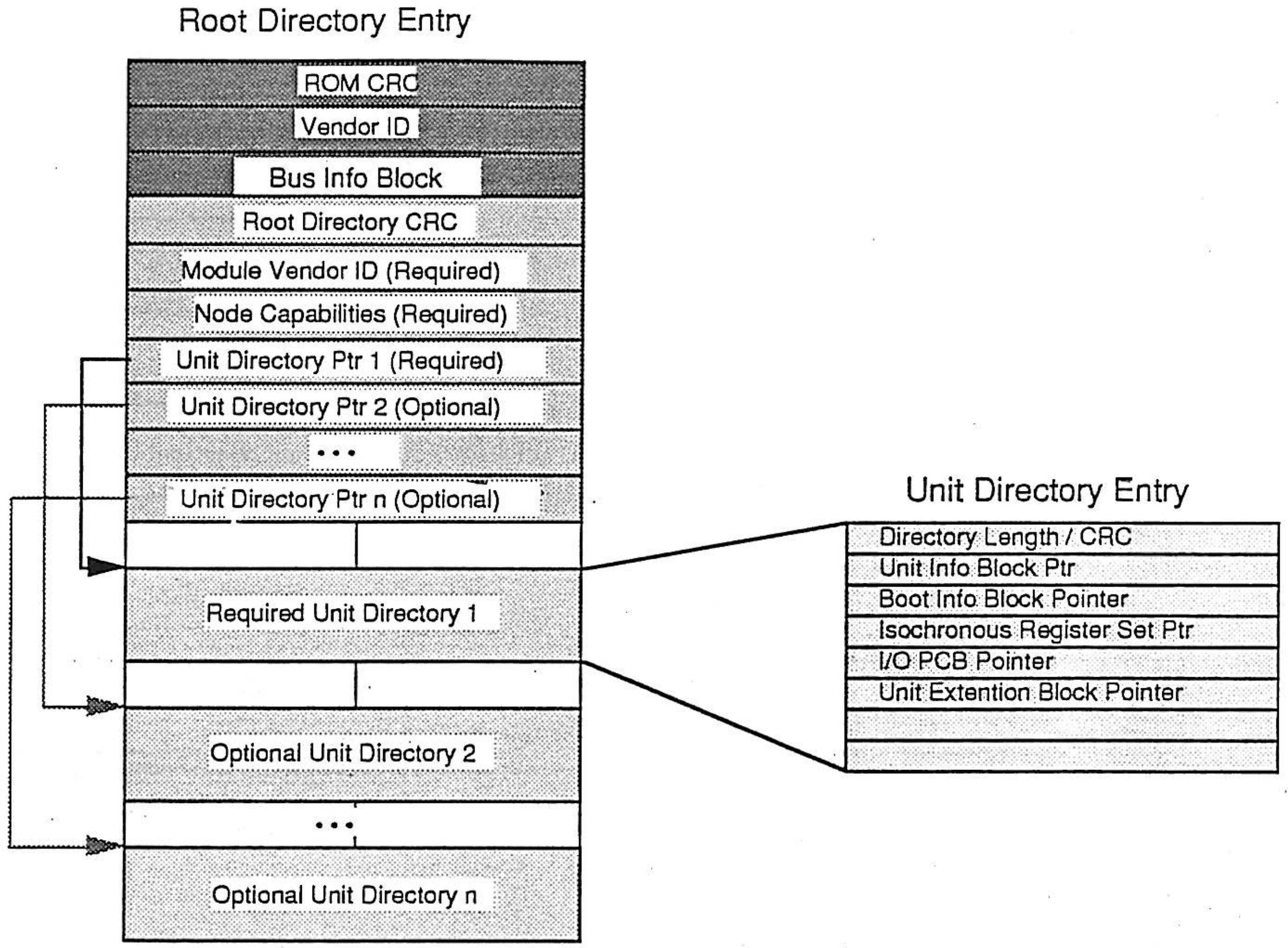
The bus CSR address space defines a set of registers used in all IEEE P1394 bus implementations. Included are CSR's for access to the Isochronous and power features of the IEEE P1394 bus. These Isochronous and power registers are described in a later section of this document. The Bus dependent CSR's are illustrated below.



IEEE 64 Bit Fixed Address Space - Bus

Note:
These CSR's are found at specific offsets from the top of the Bus dependent area. Their offsets will be defined at a later date.

ROM - Root Directory Entry
A node's Root Directory may contain entries of the following types There can be only one node directory pointer per node. One unit directory pointer per node is required. Other optional unit directories may be created. The top level of ROM address space is illustrated below.


Root Directory Entry
ROM CRC
Vendor ID
Bus Info Block
Root Directory CRC
Module Vendor ID (Required)
Node Capabilities (Required)
Unit Directory Ptr 1 (Required)
Unit Directory Ptr 2 (Optional)
• • •
Unit Directory Ptr n (Optional)
Required Unit Directory 1
Optional Unit Directory 2
• • •
Optional Unit Directory n
Unit Directory Entry
Directory Length / CRC
Unit Info Block Ptr
Boot Info Block Pointer
Isochronous Register Set Ptr
I/O PCB Pointer
Unit Extention Block Pointer

IEEE 64 Bit Fixed Address Space - ROM

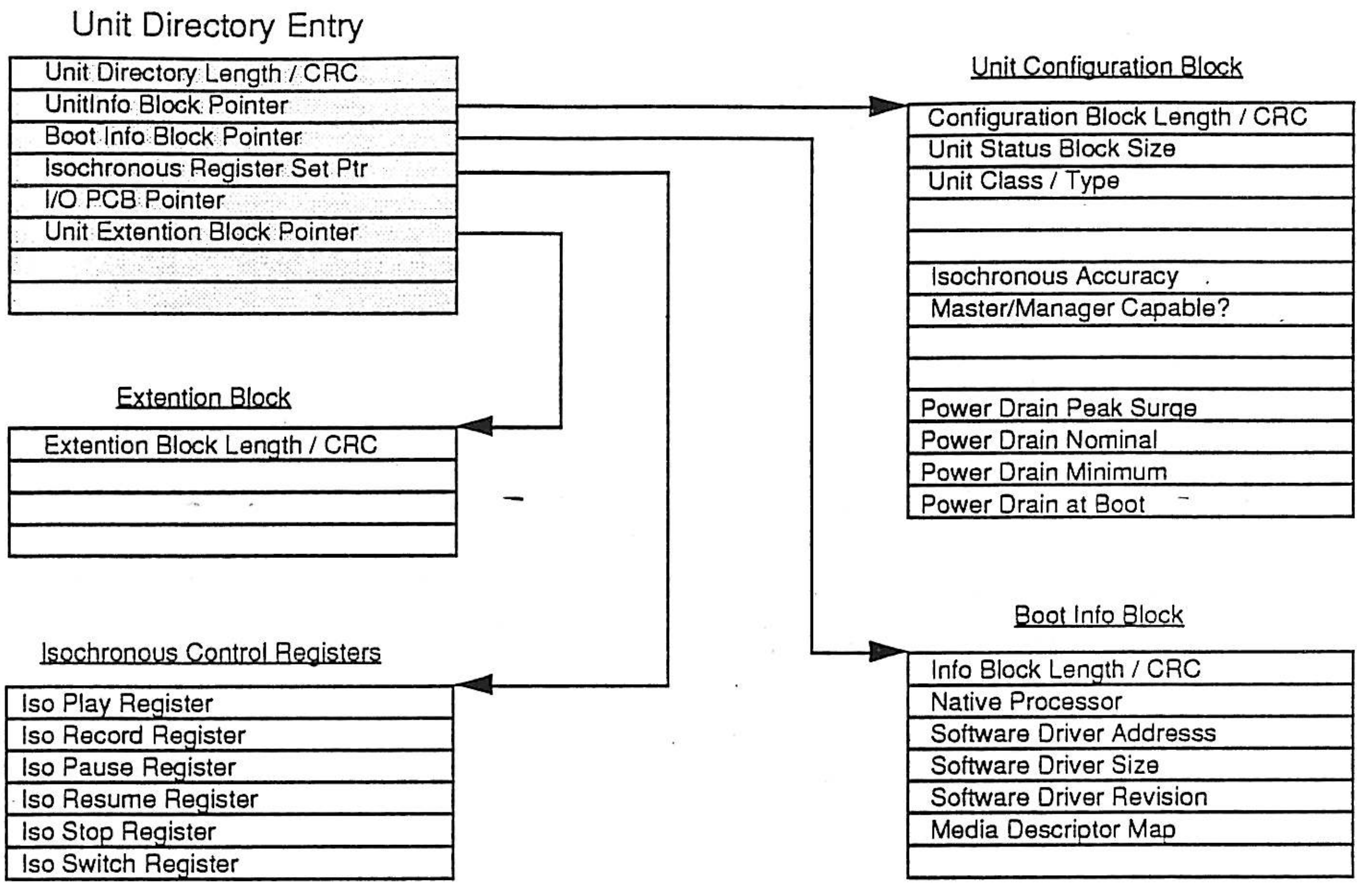
The entries illustrated above have the following structures. See IEEE 1212 Standard section 8.2

ROM - Root Directory Entry
The root directory entry is the top level description of a node's configuration. It provides a reference to the node and unit directories.

ROM - Node Directory Entry
In the Node Directory can be found those items which have no special significance to any units within the node. IEEE 1212 provides for many other entries which the vendor may optionally provide.

ROM - Unit Directory Entry
In the Unit Directory can be found those items which have special significance to units within the node. This includes offsets of vendor unique directory structures of no interest to a generic IEEE P1394 implementation.

UNIT - Unit Directory
A unit is implemented as a set of registers and ROM entries which describe the unit and its capabilities and provide a mechanism for access to unit functions. There are ROM entries for unit configuration, power profile, Isochronous attributes and in some units, a boot block. There are two defined register sets, one for unit control and asynchronous I/O, the other for control of the unit's Isochronous channel. The relationship of these ROM entries and registers sets to the unit directory entry is illustrated below.



UNIT - Configuration Block
This ROM entry contains information describing this unit as an available resource to the system...

T.B.D.

UNIT - Isochronous Info Block
This ROM entry describes various attributes of this unit's Isochronous capabilities including whether or not the unit is capable of being cycle master or manager, its Isochronous accuracy etc....

T.B.D.

UNIT - Boot Info Block
This ROM entry contains information needed by a system to boot from this device. This information would include the address and size of the unit's driver software, native processor, media descriptor maps, etc....

T.B.D.

UNIT - Power Info Block
This ROM entry provides a profile of the unit's power consumption/production capabilities. These entries are used by the power manager to quantify and allocate the power resources amongst various nodes on the bus ...

T.B.D.

UNIT - Isochronous Control Registers
This register set is used to control an Isochronous connection with this unit. It provides the ability to pause and resume the channel as well as switch this connection to another channel. These registers are defined in section 1.3 of this document.

B.2 Bus Management

IEEE P1394 provides features allowing very dynamic bus configurations. Devices may appear and disappear frequently, they may provide their own unit driver software. Units must be mounted and their driver software must be attached to the operating system. Units may be used by more than one CPU and may be owned by a CPU or shared with others. The task of managing this dynamic bus configuration is referred to as Bus Management.

IEEE P1394 supports **hot plugging** of peripherals. Hot plugging is attaching a module to the bus with power applied allowing units to appear or disappear at unpredictable times. Whenever possible, a change in bus configuration should not cause a catastrophic error condition. Of course if a hard disc unit is removed during an update of a disc file, there will be a catastrophic error. On the other hand, if a speaker unit is removed during audio playback it should not cause the system to crash. Whenever appropriate, changing the bus configuration should not cause distraction or distress to the user. It should be possible to present the user with options for dealing with a significant change in configuration.

Each unit on the bus has an associated device driver. In general, most device drivers will be loaded from their units and attached to the system during the boot process. When a unit appears on the bus, the unit's driver is loaded and attached to the system. This process is referred to as **unit registration**. The unit is now an available resource of the system.

There are two registers provided for support of the registration process. They are the **I'm Here** and **You're There** registers. When a node comes on-line it begins to **periodically** broadcast its node address and owner's node address to the I'm Here register address. This announces the node's presence on the bus. When the bus Monarch detects this broadcast, the node is recognized by a write to its You're There register with the monarch's node address. This causes the node to stop the I'm Here broadcast and informs it of the bus Monarch's address. The Monarch may now scan the node's ROM space for units to register with the system.

There are two basic unit behavior types, the well behaved unit and the renegade unit. Both well behaved and renegade units will appear at will and are registered as described above. However, the well behaved unit always announces its intention to disappear while the renegade unit does not. When a well behaved unit disappears, either the system has requested that the unit go off-line or the unit has announced its intention to go off-line. In both cases, the system is aware of the event and may remove the unit's driver from the system in a controlled manner. The problematic characteristic of the renegade unit is its tendency to disappear unexpectedly and without notice. There are several possible causes of unexpected disappearance including power failure and physical disconnection from the bus. In this dynamic environment, the challenge is to maintain a sense of coherency between units and driver software.

The LUN (logical unit number) is used to maintain coherency between unit's and their software drivers. During unit registration, the unit and driver software are assigned a LUN. Associated with each unit is a LUN register. This register is used to assign a LUN to the unit. <u>The host system is responsible for supplying and binding the LUN to driver software in a manner which allows the low level IEEE P1394 software to bind the driver software to the unit using the LUN.</u>

When a module is power-cycled, its nodes re-appear at any available bus address. The affected units have lost their context and coherency with their driver software. This leaves the system with a hanging software driver. Eventually, this driver will experience a failure from their now non-existent unit. Therefore, each unit is required to **periodically** announce it presence to the bus Monarch. The **Heartbeat** register is used for this purpose. Each unit periodically writes its bus / node address and LUN to this register. In this way, missing units are identified by their lack of a heartbeat and their software driver is removed from the system.

It is possible for the power cycled node to reappear at the SAME address it previously held - before it's units have been missed by the system. If the system loads and attaches another driver for these "new units", there would be two drivers in conflict over each unit. Fortunately, in this case, these "new units" do not have an assigned LUN and this identifies them as new units. Each new unit will broadcast I'm Here and be recognized by the Monarch as having a node address which is in use. The operating system may now be notified of the duplication and the old driver software can be removed from the system and the new driver software installed.

When a module (A) which supplies its own power has been physically disconnected from the bus it maintains its state. During the time this module is disconnected from the bus, it is possible for another node (B) to come on-line and acquire the bus address previously used by the now disconnected module (A). When the disconnected module (A) is reattached to the bus, there is an address conflict. This could cause catastrophic failure of the bus rendering it dysfunctional. However, hardware detects that the module's node has been disconnected from the bus and causes the node to clear its node ID valid bit. This causes the node to arbitrate for a new bus address. In this way any potential address conflict is avoided.

I'm Here Register
The I'm Here register is found in the bus dependent area of the node's initial register space. This register is broadcast to by each node to announce that it has come on-line or has a had a change in unit configuration. The announcing node broadcasts its node address and its owners node address and rank. The announcing node broadcasts repeatedly until it is recognized by the bus Monarch. Only the bus Monarch responds to this broadcast. The announcing node writes its node address and rank to the lower 16 bits and the node address and rank of its owner to the upper 16 bits. This register does not store a value. The address of this register is an indication that a specific node is announcing itself on the bus and the data written to this register contains identification data of the announcing node. A read of this register is undefined.

▨	owner's node ID	owner's rank	▨	node's ID	node's rank

You're There Register
The You're There register is found in the bus dependent area of the node's initial register space. This register is written by the bus Monarch in response to an I'm Here broadcast. The Monarch writes its node address and rank to this register. The effect of this write is to stop the announcing node from issuing the I'm Here broadcast as well as indicate to the announcing node the address and rank of the Monarch. This register does store the values written to it. Therefore a read of this register will return the node address and rank of the Monarch. At power up, this register contains a hex value of 0xFFFFFFFF. It is written by the Monarch with the Monarch's node address and rank in the upper 16 bits. This register is cleared on power up, node reset or whenever the node loses state.

▨	monarch's node ID	monarch's rank		

Interrupt Register
The Interrupt register is found in the bus dependent area of the node's initial register space. This register is used for wake up and other interrupts. It is a set of 32 bit flags which are used to qualify the cause of an interrupt generated at this node.. These bits are defined as follows. T.B.D.

	32 bit interrupt mask	

LUN Register
The LUN register is part of the I/O PCB found in the node's initial unit space. This register contains a 16 bit logical unit number (LUN) which uniquely identifies this unit on the bus. This register is set by the bus Monarch after the unit has been recognized by the Monarch and attached to a system (unit registration). At power up, this register contains a hex value of 0xFFFFFFFF. After the unit has been registered with the system, it is written with a value in the lower 16 bits. This value is the unit's LUN which is assigned by the system. This register is cleared on power up or when the unit loses state.

		logical unit number

B.2.1 Isochronous I/O

Isochronous data transfer is one of the features which make IEEE P1394 unique among inter-connect busses. Isochronous data has an associated temporal element and must be delivered within a specified time frame to be valid. Isochronous data which arrives late, is no longer of value and discarded. Examples of Isochronous data are sound and full motion video. Isochronous IEEE P1394 is fully explained in the CC 1394 ERS document by Mike Teener.

The **Isochronous cycle** has a period of approximately 125 µsec (8 KHz). Its is delimited by the highest priority bus packet, the **Cycle Start** packet. The cycle start packet is the highest priority asynchronous packet and provides a reliable timing reference for every node and unit on the bus. The cycle start packet contains absolute time for the IEEE P1394 bus. All Isochronous capable units on the bus recognize and may respond to the cycle start packet. Typically, the most accurate Isochronous capable node on a bus will be chosen to generate the cycle start packet. This node is referred to as the **Cycle Master**. The cycle master broadcasts the cycle start packet at 125 µsec intervals.

An **Isochronous port** is a hardware resource (most likely a FIFO) which supports Isochronous data transport. Each Isochronous capable node has at least one Isochronous transmit/receive FIFO pair. They are collectively referred to as an Isochronous port.

The Isochronous cycle can be divided into many time slices, each referred to as an **Isochronous channel**. There can be many Isochronous channels active at any time depending upon available Isochronous bus bandwidth. Each Isochronous channel is sequentially allocated and numbered Isochronous channels are assigned to a unit which then capture an available Isochronous port for use on the Isochronous channel. The units of a node may be assigned to as many Isochronous channels as there are available Isochronous ports in their node.

When a unit is assigned to an Isochronous channel, it is designated as either a **player** or a **recorder** (talker/listener). While there can be only one player on an Isochronous channel, it is possible to have many recorders on a single channel. When a cycle start packet is detected, each unit which has been assigned to play on an Isochronous channel takes its turn responding to the cycle start packet by issuing an Isochronous data packet.

An Isochronous data packet may contain zero bytes. A zero byte Isochronous data packet is sent in the case where the unit assigned to an Isochronous channel does not have any data to send. This is simply a place holder which maintains the correct number of idle gaps on the bus required as part of the arbitration sequence.

An **Isochronous connection** is a logical association between two or more units on the bus. When one unit is playing while one or more units are simultaneously recording on the bus, they are transferring data and said to be connected.

The **Cycle Manager** is a unit which generates zero data Isochronous packets when a player unit fails to issue an Isochronous data packet on its channel. The cycle manager knows how many Isochronous channels have been assigned. When no unit responds to the cycle start packet on an assigned Isochronous channel, the cycle manager will insert a zero data Isochronous data packet as a place holder for that Isochronous channel. This is a measure to prevent the complete collapse of the Isochronous system and indicates serious trouble on the bus.

By inserting a zero data packet, the cycle manager keeps the rest of the Isochronous system running. The system can then reassign units from the highest numbered Isochronous channel to the missing Isochronous channel. This is an attempt to buy time while the system identifies and corrects the cause of the missing Isochronous channel.

Cycle Start Register

The Cycle Start is an asynchronous packet broadcast every 125 µsec (8 KHz) by the Cycle Master. Its is the highest priority packet and therefore reliably wins arbitration. The cycle start packet provides a timing reference for every node on the bus. It contains universal time on the IEEE P1394 bus. All Isochronous capable nodes on the bus recognize and may respond to the cycle start packet. The Cycle Master broadcasts the cycle start packet to the Cycle Start register which is found in the bus dependent area of the node's initial register space.

Cycle #	offset value

Cycle Master ID Register

Typically, the most accurate Isochronous capable node on a bus will be chosen to generate the cycle start packet. This node is referred to as the Cycle Master . The cycle master broadcasts the cycle start packet at approximately 125 µsec intervals. The content of the cycle start packet defines time on the bus to an accuracy of 125 µsec ±12.5 nsec. The cycle master supplies the Isochronous pulse for the bus. This register contains the cycle master ID as bus/node in the lower 16 bits. The Cycle Master ID register is found in the bus dependent area of the node's initial register space.

▨	monarch's node ID	monarch's rank	master's node ID	master's rank

Cycle Manager ID Register
Each Isochronous cycle can be divided into several time slices, each slice referred to as an Isochronous channel. Typically, each Isochronous unit takes its assigned turn issuing an Isochronous data packet in response to the cycle start packet. When a unit fails to issue a packet in its turn, all subsequent Isochronous channels will not transmit. The Cycle Manager is a bus node which generates zero data Isochronous packets in these cases. The cycle manager knows how many Isochronous channels have been assigned. When a unit fails to respond on its assigned Isochronous channel, the cycle manager inserts a zero data Isochronous packet as a place holder for that Isochronous unit. This register contains the cycle manager ID as bus/node in the lower 16 bits. The Cycle Manager ID register is found in the bus dependent area of the node's initial register space.

▨	monarch's node ID	monarch's rank	manager's node ID	manager's rank

Cycle Time Register
This register contains the current Isochronous time at the cycle master node. The Cycle Time register is found in the bus dependent area of the node's initial register space.

Cycle #	offset value

Isochronous Channel Control
There is a set of registers associated with each Isochronous unit used to establish and control Isochronous connections among units. These registers provide the ability to start, stop, pause resume and switch Isochronous connections. These registers provide channel control for data transfer across Isochronous connections. There is an Isochronous control register set (and corresponding unit directory entry) for each active Isochronous channel of a given node. The Isochronous register set is referenced by the unit's directory entry and can be found in the node's initial unit space. The following paragraphs describe the Isochronous Control register set.

There are two registers used to create an Isochronous connection, the ISO Play (talk) and ISO Record (listen) registers. When a connection is desired, the Isochronous channel number and starting Isochronous cycle time are written into the ISO Play and ISO Record registers of the appropriate Isochronous units. Then, beginning with the specified Isochronous cycle, the units will connect. Recording units can be added at any time and may be synchronized by specifying the appropriate Isochronous cycle number on which to begin recording.

Recording or playback on an Isochronous channel may be paused and resumed using the ISO Pause and ISO Resume registers. The pause and resume can be specified to occur on a given Isochronous channel and cycle.

The ISO Stop register is used to close an Isochronous channel. The Isochronous channel and cycle on which to stop are written into the unit's ISO Stop register.

Sometimes it is desirable to reassign an Isochronous channel. This is the case when a low numbered Isochronous channel will be closed using the ISO Stop register. Now a higher numbered Isochronous channel must be re-assigned to the empty Isochronous channel or the Isochronous system will fail. The ISO Switch register allows an Isochronous channel to be reassigned or switched. It is possible to switch a unit to a new Isochronous channel by writing the new channel number and the cycle number at which the switch is to occur.

ISO Play (talk) Register
This register is used to instruct a unit to begin transmitting Isochronous data over an Isochronous channel beginning with a specified Isochronous cycle. At power up, this register contains a hex value of 0xFFFFFFFF indicating that it is currently not assigned to any Isochronous channel. A channel number is written into the upper 16 bits while a cycle number is written into the lower 16 bits. This register may be read to determine which channel this unit is currently assigned. This register is cleared at power up, unit reset and when the unit loses state.

Cycle #	Channel #

ISO Record (listen) Register

This register is used to instruct a unit to begin receiving Isochronous data over an Isochronous channel beginning with a specified Isochronous cycle. At power up, this register contains a hex value of 0xFFFFFFFF indicating that it is currently not assigned to any Isochronous channel. A channel number is written into the upper 16 bits while a cycle number is written into the lower 16 bits. This register may be read to determine which channel this unit is currently assigned. This register is cleared at power up, unit reset and when the unit loses state.

Cycle #	Channel #

ISO Pause Register

This register is used to instruct a unit to suspend transmission or reception of Isochronous data over a channel beginning with a specified Isochronous cycle. At power up, this register contains a hex value of 0xFFFFFFFF indicating that there is currently no paused Isochronous channel. A channel number is written into the upper 16 bits while a cycle number is written into the lower 16 bits. This register may be read to determine when the channel was paused. This register is cleared at power up, unit reset and when the unit loses state.

Cycle #	Channel #

ISO Resume Register

This register is used to instruct a unit to resume previously suspended transmission or reception of Isochronous data over a channel beginning with a specified Isochronous cycle. At power up, this register contains a hex value of 0xFFFFFFFF indicating that no paused Isochronous channel has been resumed. A channel number is written into the upper 16 bits while a cycle number is written into the lower 16 bits. This register may be read to determine when a paused channel was resumed. This register is cleared at power up, unit reset and when the unit loses state.

Cycle #	Channel #

ISO Stop Register

This register is used to instruct a unit to stop transmission or reception of Isochronous data over a channel beginning with a specified Isochronous cycle. This causes the unit to break its connection on this channel. The unit cannot restore this connection unless explicitly instructed to do so. At power up, this register contains a hex value of 0xFFFFFFFF indicating that no Isochronous channel has been stopped. A channel number is written into the upper 16 bits while a cycle number is written into the lower 16 bits. This register may be read to determine when an Isochronous channel was stopped. This register is cleared at power up, unit reset and when the unit loses state.

Cycle #	Channel #

ISO Switch Register

This register is used to instruct a unit to stop transmission or reception of Isochronous data over the current channel beginning with a specified Isochronous cycle and to begin transmission or reception on the newly specified channel. At power up, this register contains a hex value of 0xFFFFFFFF indicating that no Isochronous channel has been switched. A channel number is written into the upper 16 bits while a cycle number is written into the lower 16 bits. This register may be read to determine when an Isochronous channel was switched to a new channel. This register is cleared at power up, unit reset and when the unit loses state.

Cycle #	Channel #

Review Note:
This section currently does NOT address the asynchronous control aspects of Isochronous I/O. This proposal covers only the 'flow control' of an existing Isochronous channel.

B.2.2 Bus Power Control

Power is carried on the IEEE P1394 bus cable. This allows units with low or moderate power appetites to avoid the costs associated with an internal power supply. There are a set of registers provided in the Bus dependent address space for the purpose of controlling power consumption on the bus.

Power control is exercised at the node level. Since a node can be comprised of several units, the node function is required to manage the power consumption of all its units. Power is measured and allocated in watts. Therefore, a node which has three units each consuming 2 watts consumes a total of 6 watts. The power control registers are briefly described below.

B.2.3 Power Manager ID

The Power Manager is a bus node tasked with allocating bus power to nodes in response to power requests. The power manager knows how many watts of power are available and how many watts have been allocated. If a node requests power in excess of what is available, the power manager denies the request. This register contains the power manager ID as bus/node in the lower 16 bits. The Power Manager ID register is found in the bus dependent area of the node's initial register space.

▨	monarch's node ID	monarch's rank	manager's node ID	manager's rank

Power Request
This register is used by nodes to request power from the power manager. The requesting node writes a value representing the amount of power requested in watts.

Power Granted
In response to a power request, the power manager writes a value representing the amount of power granted in this register. The requested value is written if the request is granted. Zero is written if the request is denied.

Current Power Consumption
This register reflects current power consumption of this bus power dependent node.

Power Fail Warning
This register is broadcast to by the power manager giving all nodes a 4 msec warning that power is about to be removed from the bus.

C. Implementing a Shared List

The following Unit ROM entries are required to implement a shared list: