To: X3T9.2 Committee Membership

From: Edward A. Gardner, Digital Equipment Corporation

Subject: Minutes of SBP Working Group, April 7-8, 1993

Attendance:

Jeff Stai	Western Digital	stai@wdc.com
Scott Smyers	Apple Computer	smyers.s@applelink.apple.com
Richard Mourn	Texas Instruments	5707151@mcimail.com
Ed Gardner	Digital Equipment	gardner@ssag.enet.dec.com
Jerry Marazas	IBM Boca Raton	marazas@bcrvmpc2.vnet.ibm.com
Jim McGrath	Quantum	
John Lohmeyer	NCR	John.Lohmeyer@ftcollinsCO.ncr.com
Nathan Hays	Quantum	
Paul Gramann	IBM Rochester	gramann@rchvmp3.vnet.ibm.com
Greg Floryance	IBM Rochester	gregf@vnet.ibm.com
George Penokie	IBM Rochester	gop@rchvmp3.vnet.ibm.com
Ralph L. Gee	IBM San Jose	geeman@vnet.ibm.com
Larry Lamars	Maxtor	71540.2756@compuserve.com

All present complained of the lack of coffee and suitable refreshments. Scott promised that caffeine would appear promptly.

Agenda:

Review minutes from March 1-2 meeting
64 byte messages
16 byte status block
Tap slot management
Isochronous overview and interaction with SCSI-3 queuing model
Data transfer control
Idempotence

This was the agenda proposed at the start of the meeting, the actual agenda as followed is reflected in the following section headings.

1. Isochronous, Tap Fetch Policy, and SCSI-3 Queuing Model

Isochronous commands need to be executed at prescribed points in real (wall clock) time. This implies that isochronous command blocks need to be fetched at (slightly earlier) prescribed points in real time. Does the primary responsibility for scheduling the fetch of isochronous commands reside with the target or the initiator? If it lies with the target, initiators would typically submit long isochronous chains (covering many minutes of activity) and the target would choose when to fetch the commands. This would require a liberalization of the current fetch policy rules for isochronous commands. Alternately the responsibility could lie with the

initiator, who would tap the target with short chains of commands every few seconds. There was general agreement that SBP was oriented towards targets controlling when commands would be fetched, implying that the fetch policy rules should be extended with special treatment for isochronous commands.

The interaction with the queuing model arises if a (disk) device is being shared by both an isochronous stream and asynchronous access. If the asynchronous access uses ordered commands (e.g., to update a directory after file updates are complete), that ordered command will occur at an unpredictable point within the isochronous stream. Many ordered commands may take sufficiently long to execute that the isochronous stream could be starved and miss its real-time transfer requirements. Discussion quickly became unruly due to caffeine withdrawal.

George proposed that SBP should provide multiple task sets per device to solve this problem. In terms of the current SAM queuing model, there would be multiple LUNs that each map the same device. Initiator software would use one task set for ordinary asynchronous operations and another for isochronous transfers. Note that SBP itself would not restrict either task set as to the type of operations it could perform, this would simply be an initiator programming convention, perhaps supported by a recommendation in SBP. Note that an initiator might want to send setup or exception handling commands to the isochronous task set even though they would be asynchronous commands.

Coffee and refreshments finally arrived. Following a brief interruption the meeting resumed in much improved temper. Scott gave an overview of isochronous streams and their operation so that everyone could better pretend to understand the subject.

George presented a suggestion whereby all the commands associated with an isochronous stream would be consider a single I/O process in terms of the queuing model. If an ordered command was sent after an isochronous stream were begun, the ordered command (and all following simple commands) could not be executed until the isochronous stream had completed. While appealing in the context of the queuing model, this was recognized as not satisfying the needs of mixed isochronous and asynchronous access.

John observed that the only ways to ensure that ordered commands and isochronous transfer do not interfere with each other is to either place them in different task sets, or to invent a new queue tag (for isochronous commands) that would be independent of ordered commands. Jim expressed concern that multiple task sets might lead to problems. George said that, without a lot of thought, just creating another task set was a tempting idea. He followed up with a discussion of temptation and retribution, and said we would regret giving in to the temptation of multiple task sets. There followed discussion of whether multiple task sets for the same device are truly independent, that is, whether an error in one task set blocks the other task sets on the same device.

Finally, discussion turned to the notion of a new queue tag, variously called “disordered” or “super simple”. Such commands could be re-ordered relative to all commands, including ordered commands. There was consensus that this would solve the problems of ordered and isochronous commands sharing a task set. Given the disagreement over multiple task sets, this seemed the solution to pursue.

Straw vote in favor of a single task set with a “super simple” queue tag: 7 yes (Ed, George, Ralph, Paul, Jim, Richard, Scott), 1 no (Jerry), 2 abstain (Greg, Jeff). George volunteered to prepare a proposal for this. Straw vote in favor of multiple task sets: 3 yes (Greg, Jeff, Jerry), 3 no (George, Jim, Ed), 4 abstain (Ralph, Paul, Richard, Scott). Subsequent discussion of what these votes meant led to requests from some people to change their votes, but the recording secretary was not present.

Scott claimed the following three points for isochronous operations:

1. Targets must have a dedicated command FIFO for each isochronous channel. Many people disagreed, no one other than Scott defended his claim. Scott was requested to prepare a written proposal explaining why he thought this might be necessary.
2. Chains of isochronous commands shall contain only isochronous commands directed to a single channel. This was explained as a requirement on initiators. Targets may do whatever they please if it is violated. Appeared to be unanimous consensus.
3. Isochronous commands may be fetched at the targets discretion. Unanimous consensus.

2. Review Changes from March 1-2 Meeting

Jerry reviewed his document to flag which changes were worth reading.

3. Message Review

Given that command blocks are 64 bytes, and command blocks are sent as tap messages, then simplicity argues that all messages should be 64 bytes. This led into discussion of types of messages and their properties:

			FIFOs to which it may be sent					
Kind of message	1/chn max	login	ACA	Normal	Urgent	Isoch	Directed to	Length
Asynch Cmd			yes	yes	yes		LUN	64
ACA commands			yes	yes	yes		LUN	64
iso setup			yes	yes	yes		target	64
iso append						yes	LUN	64
iso tear down						yes	target	64
login	yes	yes					target	64
tap slot rsv/rls			yes	yes	yes		target	64
sign in			yes	yes	yes		target	64
abort tag			yes	yes	yes		command	64
target reset	yes	yes					target	64
clear queue			yes	yes	yes		LUN	64

logout	yes	yes		target	64

Note: a blank entry in these columns means “no.”

The login FIFO is the FIFO used for login, logout, and recovery from 1394 bus resets or reconfigurations. There is one login FIFO per target. For a time the login FIFO was also being used for target resets, but later discussion replaced this with a separate CSR.

The ACA FIFO is the only command FIFO that continues to be fetched when an ACA condition exists. It is intended for the commands issued when an ACA condition exists.

The Normal and Urgent FIFOs are used for “ordinary” functions. The only difference between the two FIFOs is the order or priority of processing.

Any function (a.k.a. tap, chain, or command) that is acceptable to one of the ACA, Urgent, or Normal FIFOs is acceptable to any of the three. The only architectural rules set by SBP are the tap fetch ordering rules that dictate when commands are fetched from taps sent to the three FIFOs. The tap fetch ordering rules dictate when commands or functions are entered into the SAM queuing model. Certain combinations are nonsensical, in that there is apparently no reason why a sensible initiator would ever do them. For example, Abort Task functions sent to the Normal FIFO, ACA commands sent to a non-ACA FIFO, or non-ACA commands sent to the ACA FIFO. However, the outcome of such operations are fully defined by the tap fetch ordering rules and the queuing model. Thus there is no reason to architecturally preclude such operations in SBP. If an initiator doesn’t like the outcome, it doesn’t have it issue such operations, we need not go to the overt trouble of saying they are prohibited in the architecture and requiring that targets explicitly check and prohibit them. There was consensus on this, although it took some time to reach agreement on the subtle distinctions involved.

An Isochronous FIFO is assigned when an isochronous stream is assigned. Note that the former term was “port”, this was renamed “stream” at this meeting to avoid confusion with 1394 ports on devices. An initiator issues an Isochronous Setup operation to assign a stream. The target returns a stream identifier and the address of the isochronous FIFO to be used for that stream (assuming the target can accommodate the request). A separate FIFO address for isochronous operations simplifies the target dedicating resources (e.g. tap slots) to the isochronous stream, to make it easier to guarantee the required real-time operation.

In many ways an Isochronous FIFO or isochronous stream is like a distinct initiator. This was stated and agreed in passing, but was not discussed in detail until later in the meeting. No differences between isochronous streams and distinct initiators were noted.

Vote that all messages (sent by an initiator to a target FIFO) shall be 64 bytes, and that status blocks sent by a target to an initiator shall be 16 bytes. Unanimous agreement, with Scott, Jerry, Jeff, Ed, Jim, John, Paul, Greg, Ralph, and Nate in the room.

Unanimous agreement that the ACA, Normal, and Urgent FIFOs shall be “identical”, in that any operation (tap, command block, or chain) acceptable to one shall be acceptable to all three. The only difference shall be the tap fetch ordering rules. This is reflected in the above table.

Unanimous agreement that login, logout, and related operations (detailed list tbd) shall only be sent to the login FIFO, and that such operations shall not be chained. Unanimous agreement that

"normal" operations shall not be sent to the login FIFO, and that all such operations may be chained. This is reflected in the above table.

4. FIFO Addresses, Initiator IDs, Etc.

After some preliminary design investigations, several people or vendors have expressed concern about the current FIFO address structure. At present the initiator ID and tap request type are encoded in the FIFO address and only in the FIFO address. The concern was both that the multiple FIFO addresses might require more hardware than desirable for a low end implementation, while at the same time restricting or complicating hardware performance accelerators in higher end implementations. With further discussion it became clear that the concern was not with allowing the current FIFO address structure, since for some implementations it may be advantageous, but rather with enforcing that address structure for all implementations, since for other implementations it may be a liability. The desire is to have flexibility in FIFO addressing, so that each implementation can determine for itself how much decoding and command block steering to do in hardware versus firmware. FIFO address structure is typically useful for hardware decoding, but a potential liability for firmware.

A related but different topic is that of target resets (also called unit resets using 1394/1212 terminology). The present approach of using a function or command block is not well suited for hardware decoding. Many feel that a purely firmware detected reset may not be robust enough, and desire a more primitive, hardware oriented reset mechanism. 1394/1212 define a RESET_START register for this purpose, but that resets the entire node. That register is fine for a node that only contains an SBP target, but may be inappropriate for nodes that integrate an SBP target with non-SCSI functions. Again, the desire was to have flexibility to go either way. Following much discussion of the problem, the following solution was proposed and enthusiastically endorsed by everyone present.

An SBP target's configuration ROM contains two entries in its SBP unit dependent directory. The entries are the Target Reset CSR pointer and the Login FIFO pointer. Both point to their respective CSR or FIFO.

The Target Reset CSR has identical semantics to the 1394/1212 defined RESET_START CSR, except that a write to the Target Reset CSR is intended to reset just the SBP target rather than the entire node. If the node only contains an SBP target, then the Target Reset CSR pointer will typically point to the RESET_START CSR, since there is no difference between resetting just the target and resetting the entire node. If the node contains non-SCSI functions in addition to the SBP target, or it contains multiple SBP targets, then the target should have a separate Target Reset CSR distinct from the RESET_START CSR.

The Login FIFO pointer gives the address of the FIFO that initiators shall use for asynchronous login and logout, isochronous login and logout, and possible future control functions. These may only be singleton requests, that is, they may not be chained. The only additional control function that has been discussed to date would be used following a 1394 bus reset to direct the target to resume operation. The FIFO addresses for all other functions are determined when an initiator logs in.

A successful asynchronous login returns a one byte Initiator ID and addresses of the ACA, Normal, and Urgent FIFOs for that initiator. The Initiator ID shall be presented in subsequent command blocks from that initiator. That initiator shall use the three FIFO addresses for the corresponding three kinds of taps.

A successful isochronous login returns a one byte Stream ID, an Isochronous Command FIFO address, and an Isochronous Control Register (ICR) address. The Stream ID shall be presented in subsequent command blocks for that stream. The Isochronous Command FIFO shall be used for all taps for that stream. The ICR is used for real-time control of the stream, e.g. start, stop, suspend, resume, etc. Note that asynchronous login of initiators and isochronous login of streams are very similar.

Targets may choose how many distinct FIFO addresses they implement. A target might implement a single FIFO address and return that address for all of the FIFO address values. Or a target might implement separate FIFO addresses for each distinct use and return different addresses for all of the FIFO address values. Many targets might choose to implement separate FIFO addresses for performance critical uses and a single FIFO for all other purposes. To enable sending multiple types of commands to the same FIFO address, the information that was formerly encoded in the FIFO address will be added to the command block.

Initiators shall send taps to the proper FIFO address. If an initiator sends a tap to the incorrect FIFO address, the results are target implementation specific. The target is specifically not required to detect the inconsistency or behave in a predictable manner.

Targets shall assign unique Initiator IDs and Stream IDs and both types of ID shall be assigned from a single space. That is, a one byte ID value uniquely identifies either a single asynchronous initiator or a single isochronous stream, never both. This holds regardless of how the target assigns FIFO addresses. This is necessary to simplify the protocol (i.e., no other qualification is needed of the ID) and may assist higher level configuration management software. Vote that Initiator and Stream IDs shall be assigned from the same space: 9 in favor (John, Greg, Paul, Ralph, Nate, Scott, Jerry, Jeff, Ed), none opposed.

The tap type will be encoded in the high two bits of the SBP function code as follows: 00 Normal, 11 Urgent, 10 ACA, 01 Isochronous. The Initiator ID or Stream ID will be in byte 17 of the command block, immediately above the LUN:

Byte	0	1	2	3
16	Reserved	Init. or Stream ID	LUN	
20	Codes		Flags	
24	CDB			

Ed stated that isochronous streams should be considered distinct initiators for purposes of SAM and SCSI command processing. Several people took immediate exception. Ed pointed out that all SCSI commands must come from an initiator, as the identity of that initiator is needed for unit attention and reserve / release processing. Either we consider isochronous streams to be distinct initiators, or we associate each stream with a logged in asynchronous initiator, or we redesign SCSI command processing. There was objection to all three alternatives.

This led to a discussion of unit attention processing for isochronous streams, and isochronous error processing in general. With most errors the isochronous stream should continue

transferring. One of the isochronous documents from Apple discusses this in more detail. However some errors, such as unit attention conditions that report possible media changes, make it impossible to continue without higher level intervention. The room reached consensus that the target should automatically tear down or log out an isochronous stream if an error or unit attention made it impossible to continue the transfer. Higher level software would deal with the error, then re-log in and re-start the isochronous stream. There was brief discussion of defining an isochronous sign in, so that targets could notify host software that an isochronous stream had been torn down due to an error.

Returning to the question of whether isochronous streams are distinct initiators, there was agreement that isochronous streams would be torn down due to a unit attention condition, independent of any asynchronous initiator. Note that unit attention context does not need to be maintained for isochronous streams (unlike asynchronous initiators), as the stream ceases to exist as soon as a unit attention arises. There was no discussion of isochronous streams and reserve / release processing. There was continued objection to use of the term “initiator” as applied to isochronous streams. Clearly that term meant different things to different people among those present.

Jerry again asked whether login might be optional, which was resoundingly rejected (consistent with previous meeting).

SBP functions were reviewed to ensure they were consistent with the above, and the following list of SBP functions and function code assignments was agreed to:

0000	Asynchronous CDB
0001	Isochronous append
0010	Asynchronous log in
0011	Isochronous log in
0100	Asynchronous log out
0101	Isochronous log out
0110	Allocate tap slots
0111	Sign in
1000	Abort task
1001	Clear task set
1010	Clear ACA

5. Data Transfer Control Doublet

The SBP command block contains a doublet (16 bits) whose contents control how the target performs a data transfer. This doublet is presently in bytes 44-45 of the command block:

Byte	0	1	2	3
36	CDB			
40	Transfer Length			
44	Data Transfer Control		Reserved	Sense Length
48	Data Buffer Address (MSQ)			

The Data Transfer Control doublet will be structured as four 4-bit fields:

Rsvd	Speed	Align	Max Pkt

The Max Pkt field defines the maximum packet size that may be used for data transfers. The value n defines a maximum packet size of 2**n bytes, except for the value zero which defines a maximum packet size of 2**16 bytes (i.e., no limit). This maximum limits the amount of data that the target may transfer in a single 1394 block read or write transaction. Target implementations or bus speed may limit the maximum packet size to even smaller values.

The Align field defines the preferred data transfer alignment. The value n defines an alignment boundary of 2**n bytes, except for the value zero which defines no alignment. If a data transfer is long enough that it must be broken into multiple data packets, then each inter packet boundary should occur at an alignment boundary. Subject to that constraint, each data packet should be as long as possible (up to the maximum packet size). Note that this is a recommendation; targets may break transfers wherever they wish, but breaking transfers at alignment boundaries will improve system or host adapter performance.

The Speed field defines the 1394 transfer rate that the target shall use for the data transfer. It contains one of the following values:

0 Use the maximum transfer rate of which the target is capable.
1 98.304 megabits/second
2 196.608 megabits/second
3 393.216 megabits/second
4-15 reserved for future standardization

The Rsvd field is reserved for future standardization.

6. Idempotence

From Webster's New Collegiate Dictionary, 1977 edition:

> idem: pronoun: something previously mentioned: same
>
> idempotent: adjective: relating to or being a mathematial quantity which is not zero and every positive power of which equals itself

Ed described two "features" of 1394 that need to be considered in SBP. First, one failure mode of 1394 and similar systems is that a request may be successfully received but its response is lost. This may occur due to bus noise, but also may occur more systematically if (for example) the bus round trip time is set too low, so that another node's request collides with the response. When a response is lost the requestor will re-transmit the request, which appears to the requestee as a duplicate request. At a 1394 meeting earlier in the week it was decided that 1394 cannot reasonably prevent this scenario, so therefore application protocols such as SBP should be designed cope with duplicate requests.

The desired property is that SBP messages should be idempotent, that is, receiving multiple copies of a message has the identical result as receiving a single copy. The straightforward solution for achieving this is to include a small sequence number in tap messages (that is, the command block that is sent as a tap message). Solutions without sequence numbers are also

possible, but are more complex. It was agreed that SBP should go with the straightforward solution and include a sequence number.

The second problem results when a target resets without a host being aware of it. Consider a host A that has logged in as initiator ID 1. The target resets (for any reason), then a different host B logs in. The target has reset, it has therefore lost all context of which initiator IDs were in use, and assigns initiator ID 1 to host B. Subsequently host A issues a command using initiator ID 1 and bad things happen. Ed described a possible solution involving an identifier similar to an initiator ID but assigned by the hosts or initiators.

Detailed consideration of both problems was deferred, with the request that Ed prepare a detailed proposal addressing both these and related problems.