X3T9.2/88-3

Memo to: ANSI X3T9.2

Memo from: James McGrath
Quantum
1804 McCarthy Blvd
Milpitas, CA 95035

Date: January 4, 1988

Subject: Questions Concerning section 5 of SCSI-2 Rev 3.0

The following is a list of questions/comments/proposals (in ()'s) concerning various elements in the Rev 3 draft of the standard. I have not repeated concerns raised by bracketed comments embedded within the draft (although I have responded to a couple of them).

1.

5.1.3. SELECTION Phase
(skip 6 paragraphs)
IMPLEMENTORS NOTE: To maintain compatability with single initiator option of SCSI (X3.131-1986), it should not be an error to have only one ID bit present during selection.

(While not considered an error, is the SELECTION successful? This appears to be the intent, in which case more of the the single initiator option wording from X3.131 should be replicated here (i.e. SCSI-2 targets have to fully support single initiator mode, so it should be completely defined).)

2.

5.1.6. Information Transfer Phases
(skip 3 paragraphs)
IMPLEMENTORS NOTE: Between information transfer phases, C/D, I/O and MSG may change in any order. A new phase does not begin until REQ is asserted for the first byte of the new phase. The phase ends when C/D, I/O, or MSG changes after ACK goes false [or when a new phase begins].

(While technically correct, this note is misleading since there is silicon out there which will break if C/D, I/O, and MSG are changed in an arbitrary order. I suggest this point be noted.)

3.

5.1.10.2. MESSAGE OUT Phase
(4 paragraphs)
If the target receives all of the message byte(s) successfully (i.e., no parity errors), it shall indicate that it does not wish to retry by changing to any information transfer phase other than the MESSAGE OUT phase and transfer at least one byte. The target may also indicate that it has successfully received the message byte(s) by changing to the BUS FREE phase (e.g., ABORT or BUS DEVICE RESET messages).

(This last sentence conflicts with an earlier statement describing the conditions under which a BUS FREE phase may be reached. Specifically, it is perfectly possible on an error condition to go to BUS FREE from MESSAGE OUT (i.e. retries are exhausted, so the target gives up and decides to abort the entire command).)

4.

5.2.2.1. "Hard" RESET Alternative
(1) Clear all uncompleted commands [including queued commands?]
(5) Preserve all the information required to continue normal dispatching of commands queued prior to the RESET. [JBL: I'm not sure where item (5) came from. It seems strange to clear current commands, but preserve queued commands. Perhaps this was intended to be added to "soft" RESET?]

(When is a command queued? When it is identified (i.e. IDENTIFY and queue tag messages received)? See point 9 for more on this topic.)

5.

5.4. SCSI Pointers
(skip 4 paragraphs)
An additional pointer is used when the autosense data option is implemented, known as the autosense data pointer.

RESTORE POINTERS 03h. ... Pointers to the command, data, and status locations for the logical unit shall be restored to the active pointers.

(These two sections conflict, showing that the autosense data pointer concept is not fully integrated into the text. Why do we have this new pointer anyway? Why not just use the data pointer? We seem to be adding quite a bit of complexity in an unclean manner.)

6.

5.5.2. Messages
(skip 1 paragraph)
[Paul Boulay has requested a separate subsection be defined for each message. I didn't have time to do it for this rev. Any comments? JBL]

(I think it is a good idea.)

7.

CLEAR QUEUE 0Eh. ... If the target is currently connected for the command specified in the tag value, the target shall go to BUS FREE phase. [what does that last sentence mean???...P Boulay]

(This sentence should be removed (it is a holdover from a previous draft).)

8.

CLEAR QUEUE 0Eh. .... A UNIT ATTENTION condition shall be created for all other initiators with commands that either had been executing or were queued for that LU. The UNIT ATTENTION condition is received by each initiator upon transmission of the next command to that LU from that initiator. [AEN?]

(AEN appears to be a feature that adds considerable complexity to a target's implementation (which does not support initiator mode unless COPY is implemented) and yields little in value. Whenever an AEN might be invoked a UNIT ATTENTION condition can be posted instead.

If a command is currently active (executing or queued) from a device then, instead of issuing an AEN to the device, the LU can simply complete the command with CHECK CONDITION status and an UNIT ATTENTION key. If the LU wants to enter into extended error recovery, then it can send an INITIATE RECOVERY message and enter into ECA. AEN is simply not needed in this case.

If a command is not currently active (executing or queued) from a device then, instead of issuing an AEN to the device, the LU can simply begin a UNIT ATTENTION condition for that device. Since no command is active at the LU, the device should not be expecting communications from the LU and really should not care if an operating condition has changed. It will care when it has to access the LU again, but at that time the UNIT ATTENTION will be correctly noted. Once again, AEN is not needed.

AEN appears to be a solution in search of a problem. It is a technically cleaner than working through the UNIT ATTENTION mechanism, and would probably be faster. But conditions requiring an AEN are (hopefully) not frequent enough to support a performance argument, and the added complexity is considerable.

I would prefer to get rid of AEN entirely, but if a case can be made for it (perhaps for device types other than DASDs), then I would not object to leaving it. But the sections on UNIT ATTENTION and AEN should be integrated closely, the standard making it clear that they are two different mechanisms dealing with similar problems, and that either one (or both) can be implemented.

9.

HEAD OF QUEUE TAG 21h. ... The HEAD OF QUEUE tagged command shall not preempt commands already dispatched from the queue, but shall be executed next after a currently executing command. Subsequent HEAD OF QUEUE tagged commands shall be inserted at the head of the queue for execution in LIFO order, except where a previous HEAD OF QUEUE tagged command has already been dispatched from the queue for execution.

Although we have defined "command identification" (reception of IDENTIFY message and queue tag message) and "command completion" (receipt of the COMMAND COMPLETE message), we have not yet defined "command execution." Clearly receipt of the CDB by the target is not execution (otherwise the above section makes no sense). Perhaps this point should be defined as "command reception." Command execution would then be the time between dequeuing, which in turn must be after command reception, and command completion.

However, even this model leaves something to be desired. While a command is executing from the queue (say the seek portion of a READ command) the queued commands can also be "preprocessed" (syntax checking of the CDB, doing LBA to CHS conversion, scanning a cache table for a hit, etc...). Are these activities part of "command execution?" If so, then several commands are executing concurrently.

In a caching controller two commands can actually be executed at the same time. A seek for a READ command that generated a cache miss can be overlapped with executing another READ command that generates a cache hit. Here the seek for one command and the data transfer for another are overlapped. Even such activity as seeking can occur concurrently for two of more commands. Consider a DASD with two independently movable actuators.

In all of these cases, although there is still a single LU, there are now multiple servers rather than a single server. This has all sort of implications, particularly for error recovery situations. Although the command queuing proposal does not create this problem (queuing of commands for a single LU from multiple initiators is already allowed), it does make it much more likely to be a practical difficulty.

10.

IDENTIFY 80h to FFh.

(We need to clarify the error conditions and responses for the IDENTIFY message. MESSAGE REJECTion is not an appropriate response under some circumstances. When it is other legal for an initiator to send an IDENTIFY message, the following error conditions can occur:

Reserved bits (3 or 5) are set (response is not specified, some implementations reject the message while others treat this the same as setting a reserved bit in the CDB, generating a CHECK CONDITION).

Target/LUN bit is set, option not supported (response is specified - message is rejected).

LUN/Target Process specified not supported (response is not specified and, since some commands can execute through an illegal LUN, message rejection is not appropriate).)