Document:	T10/00-309r0	Date:	October 23, 2001
To:	T10 Committee Membership		
From:	Edward A. Gardner, Ophidian Designs		
Subject:	SAM-2 Data Transfer Ordering Requirements		

Consider an example where a logical unit has two outstanding commands, a READ followed by a WRITE. Some circumstance requires that they be executed in order. Perhaps either or both are ORDERED commands. Perhaps the logical unit is a tape drive or other inherently sequential device.

Now suppose that both commands reference the same buffer in initiator memory. Does / should SAM-2 say anything about the order of the data transfers? Does / should SAM-2 dictate that the READ's data-in transfer complete before the WRITE's data-out transfer begins, resulting in a data copy operation? Or is SAM-2 supposed to allow any data transfer order, meaning the WRITE might record the original contents of the buffer, might copy the READ's data, or any combination of the two?

Many SCSI devices have dedicated hardware to automatically pre-fetch WRITE data. If those device implementations are valid, it implies that SAM-2 must allow any data transfer order.

Consider a more perverse example where we reverse the commands, so we have a WRITE followed by a READ. Either or both are ORDERED. Both reference the same buffer. Is it permissible for the logical unit to do these commands in the following order:

1. Perform the READ data-in transfer.
2. Perform the WRITE data-out transfer.
3. Execute the WRITE on the media.
4. Return STATUS for the WRITE.
5. Verify that the READ and WRITE did not overlap on the media so that the data that would be returned by the READ has not changed.
6. Return STATUS for the READ.

The consequence of this execution sequence is that we have again copied data, even though the command order is reversed. As an example of a device where this might actually occur consider a RAID array or the like. Assume the READ executes entirely on one component drive of the array and the WRITE executes entirely on a different component drive of the array.

I feel that SAM-2 should also allow this second example. In part because any straightforward wording that allows the first example will also allow the second example. And also because I wish to discourage any dependency whatsoever on data transfer order among concurrent commands. This emphatically applies to qualification or validation tests.

Precedent for this view already exists in SAM-2. Sam2r20 page 78, sub-clause 7.5.2, Enabled task state, says:

> Except for the use of target resources required to preserve task state, a task shall produce no effects detectable by the application client before the task's first transition to the enabled task state. Although, before entering this state for the first time, the task may perform other activities visible to lower layers – such as pre-fetching data to be written to the media – this activity shall not result in a detectable change in device state as perceived by an application client. In addition, the behavior of a completed task, as defined by the commands it has processed, shall not be affected by the task's states before it enters the enabled task state.

This paragraph states that data transfers (e.g. pre-fetching data) is below the level of the SCSI architecture model and irrelevant to that model. So long as device state is not affected, lower level operations such as data transfer may occur at any time.

Stated formally, I think SAM-2 should contain the statement:

> A logical unit may perform data-in transfers or data-out transfers for a command at any time during the command's lifetime. The data sent to the logical unit is device specific if the contents of a command's Data-out Buffer are altered during the command's lifetime.

This should be the only thing that SAM-2 states about data transfer ordering.

However, when I read SAM-2 I find some places that agree with this and others that disagree. That is, SAM-2 is inconsistent. This proposal is an attempt to reconcile those inconsistencies.

1 Related history

A similar issue arose in the discussion of T10/00-309r2, Bidirectional data transfers in SAM-2. Coincidentally, that is the same document number as this, only last year (2000) instead of this year (2001). In the discussion of that document Bob Snively expressed the sentiment that he wanted to prohibit read-modify-write transfers. That resulted in the following words within T10/00-309r2:

> In clause 5 the following statement should be added:
>
> The target may assume the content of the data-out buffer does not change during the lifetime of the task.

The implication of that statement is that the target may perform a data-out transfer at any time, since it may assume that the data-out buffer contents do not change. However other portions of SAM-2 constrain the timing of data-out transfers, rendering that assumption invalid.

Also note that while SAM-2 revision 20 claims to have incorporated T10/00-309r2, I cannot find the above statement anywhere in clause 5. The closest I can find is on page 50:

> The content of the Data-Out Buffer shall not change while the device server is processing the command.

This is very different. The statement in T10/309r2 implies that WRITE data pre-fetch (the first example at the beginning of this proposal) is valid. The statement in SAM-2 revision 20 does not.

2 Proposed SAM-2 changes

2.1 Sam2r20 page 50, sub-clause 5.1, the Execute Command remote procedure.

The crossed out sentence will be moved to sub-clause 5.4.3.1 and reworded.

> **Data-Out Buffer:** A buffer containing command specific information to be sent to the logical unit, such as data or parameter lists needed to service the command (see 5.4.3). ~~The content of the Data-Out Buffer shall not change while the device server is processing the command.~~

2.2 Sam2r20 page 56, sub-clause 5.4.2, Execute Command Request/Confirmation Protocol Services, Send SCSI Command.

Change cross-reference.

> **Data-Out Buffer:** A buffer containing command specific information to be sent to the logical unit, such as data or parameter lists needed to service the command (see 5.4.3 ~~5.1~~).

2.3 Sam2r20 page 58, sub-clause 5.4.3.1, Data Transfer Protocol Services Introduction.

Add the following, probably as the second paragraph of the sub-clause.

> A logical unit may perform data-in transfers or data-out transfers for a command at any time during the command's lifetime. The data sent to the logical unit is device specific if the contents of a command's Data-out Buffer are altered during the command's lifetime.

2.4 Sam2r20 pages 60-61, sub-clause 5.5, Task and command lifetimes.

This sub-clause proclaims in its title that it defines "task lifetime" and "command lifetime", yet it doesn't. It never even uses the word "lifetime". However, "task lifetime" and "command lifetime"

are used elsewhere. The SAM-2 editor should revise this sub-clause and/or the rest of SAM-2 to use consistent terminology (note this is an early submitted SAM-2 letter ballot comment).

2.5 Sam2r20 page 77, Terminology.

Clause 7 attempts to make a major distinction between “current task” and “pending task”. Those terms are obsolete. They derive from a time when the SCSI model was data group transfers on a parallel bus. Selection or reselection assigned the entire bus to a single task at a time, thus it was natural to speak of a current task.

This clause (and the glossary) first attempts to define “current task” in a generic fashion, then states that each SCSI protocol shall define the term. None of the protocol standards do so. Either it has a generic definition or is protocol specific, it cannot be both (note this is a potential SAM-2 letter ballot comment). “Pending task” is the opposite of “current task” and therefore has the same problem.

The distinction of current vs. pending task appears to have no merit. All it does is limit the times when a target can perform a data transfer. Among other things this makes any target that pre-fetches WRITE data non-compliant with SAM and SAM-2. Gateways, SCSI bridges and the like make the terms even more meaningless. The simpler remedy is to eliminate all use of the terms in SAM-2. If the terms are useful in any protocol (e.g. SPI-n), they should be defined in that protocol standard. The following changes eliminate all uses of “current task” and “pending task” in SAM-2.

Note that “pending” is used in a conflicting fashion throughout clause 4 to describe what I would call an outstanding command.

2.5.1 Sam2r20 page 7, sub-clause 3.1.21

Delete definition of “current task”.

2.5.2 Sam2r20 page 9, sub-clause 3.1.72

Delete definition of “pending task”.

2.5.3 Sam2r20 page 12, sub-clause 3.1.118

Change definition of “suspended information”:

> **3.1.118 suspended information:** Information stored within a logical unit that is not available to any ~~pending~~ task~~s~~.

2.5.4 Sam2r20 page 65, sub-clause 5.8.2, Overlapped commands

This is clearly an accidental misuse of “pending task”.

> An overlapped command occurs when a task manager detects a duplicate I_T_L_x nexus (see 4.9.1) in a command before another ~~pending~~ task holding that I_T_L_x nexus completes its task lifetime (see 5.5). Each SCSI protocol standard shall specify whether or not a task manager is required to detect overlapped commands.

2.5.5 Sam2r20 page 77, sub-clause 7.2, Terminology

Delete this sub-clause and editor’s note 2.

2.5.6 Sam2r20 page 78, sub-clause 7.5.2, Enabled task state

Note that the current wording is self-contradictory. The last sentence of the first paragraph says that the task “shall not become a current task unless it is enabled”, implying it may not transfer data, yet the second paragraph explicitly allows pre-fetching data before becoming a current task.

> A task in the enabled task state ~~may become a current task and~~ may send command status or complete at any time, subject to the task completion constraints specified in the Control mode page (see SPC-2). A task that has been accepted into the task set shall neither send command status nor complete ~~or become a current task~~ unless it is in the enabled task state.
>
> Except for the use of target resources required to preserve task state, a task shall produce no effects detectable by the application client before the task's first transition to the enabled task state. Although, before entering this state for the first time, the task may

perform other activities visible to lower layers – such as pre-fetching data to be written to the media – this activity shall not result in a detectable change in device state as perceived by an application client. In addition, the behavior of a completed task, as defined by the commands it has processed, shall not be affected by the task's states before it enters the enabled task state.

2.5.7 Sam2r20 page 79, sub-clause 7.5.3, Blocked task state

The second sentence is redundant with sub-clause 7.5.2.

A task in the blocked task state is prevented from completing due to an ACA or CA condition. A task in this state shall not send command status ~~become a current task~~. While a task is in the blocked task state, any information the logical unit has or accepts for the task shall be suspended. If the TST field in the Control mode page (see SPC-2) equals 000b the blocked task state is independent of the initiator. If the TST field equals 001b the blocked task state applies only to the faulted initiator.

2.5.8 Sam2r20 page 79, sub-clause 7.5.4, Dormant task state

The second sentence is redundant with sub-clause 7.5.2.

A task in the dormant task state is prevented from completing due to the presence of certain other tasks in the task set. A task in this state shall not send command status ~~become a current task~~. While a task is in the dormant task state, any information the logical unit has or accepts for the task shall be suspended.

2.5.9 Spc3r02 page 26, sub-clause 5.5.1, Reservations overview

I believe the following is a pre-existing error that is independent of this proposal.

A command that does not explicitly write the medium shall be checked for reservation conflicts before the command enters the enabled ~~current~~ task state for the first time. Once the command has entered the enabled ~~current~~ task state, it shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation.

A command that explicitly writes the medium shall be checked for reservation conflicts before the device server modifies the medium or cache as a result of the command. Once the command has modified the medium, it shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation.

Note that according to the current wording, a READ command to a tape drive would be allowed to alter the tape's position, although not return any data, before returning RESERVATION CONFLICT. The revised wording above fixes this.

I believe the above two paragraphs are equivalent to the following single paragraph:

A command ~~that does not explicitly write the medium~~ shall be checked for reservation conflicts before the command enters the enabled ~~current~~ task state for the first time. Once the command has entered the enabled ~~current~~ task state, it shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation.

I say this because a command is not allowed to alter the medium (or have any other application client visible effect) except in the enabled task state.

2.5.10 Spc3r02 page 42, clause 6, Model for processor devices

I believe this clause contains the same pre-existing error as above. However, there is no "medium" associated with a processor device. I believe the single paragraph proposed above is correct.

A command ~~that does not explicitly write the medium~~ shall be checked for reservation conflicts before the command enters the enabled ~~current~~ task state for the first time. Once the command has entered the enabled ~~current~~ task state, it shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation.

~~A command that explicitly writes the medium shall be checked for reservation conflicts before the device server modifies the medium or cache as a result of the command. Once the command has modified the medium, it shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation.~~

2.5.11 Spc3r02 page 239, sub-clause B.1, SBC commands

I believe this is the same pre-existing error discussed above:

> A command that does not explicitly write the medium shall be checked for reservation conflicts before the command enters the enabled ~~current~~ task state for the first time. Once the command has entered the enabled ~~current~~ task state, it shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation.
>
> A command that explicitly writes the medium shall be checked for reservation conflicts before the device server modifies the medium or cache as a result of the command. Once the command has modified the medium, it shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation.

I believe the one paragraph version cited above is preferable.

2.5.12 Spc3r02 page 243, sub-clause B.2, SMC commands

I believe this is the same pre-existing error discussed above:

> A command that does not explicitly write the medium shall be checked for reservation conflicts before the command enters the enabled ~~current~~ task state for the first time. Once the command has entered the enabled ~~current~~ task state, it shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation.
>
> A command that explicitly writes the medium shall be checked for reservation conflicts before the device server modifies the medium or cache as a result of the command. Once the command has modified the medium, it shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation.

I believe the one paragraph version cited above is preferable.

The above cite every occurrence of “current task” or “pending task” in Sam2r20 and Spc3r02.