NVM Express Technical Errata

Errata ID	**010**
Change Date	**4/26/2011**
Affected Spec Ver.	**NVM Express 1.0**
Corrected Spec Ver.	

Submission info

Name	Company	Date
Kevin Marks	Dell	4/21/2011

This erratum makes editorial changes to section 4.

Description of the specification technical flaw

Modify section 4.6 as shown:

Fused operations enable a more complex command by “fusing” together two simpler commands. This feature is optional; support for this feature is indicated in the Identify Controller data structure in Figure 65. In a fused operation, the requirements are:

- The commands shall be executed in sequence as an atomic unit. The controller shall behave as if no other operations have been executed between these two commands.
- The operation ends at the point an error is encountered in either command. If the first command in the sequence failed, then the second command shall be aborted. If the second command in the sequence failed, then the completion status of the first command is sequence specific.
- The LBA range, if used, shall be the same for the two commands. If the LBA ranges do not match, the commands should be aborted with status of Invalid Field in Command.
- The commands shall be inserted next to each other in the same Submission Queue. If the first command is the last entry in the Submission Queue, then the second command shall be the first entry in the Submission Queue as part of wrapping around. The Submission Queue Tail doorbell pointer update shall indicate both commands as part of one doorbell update.
- If the host desires to abort the fused operation, the host shall issue an Abort command separately for each of the commands.
- A completion queue entry is posted by the controller for each of the commands.

Whether a ~~the~~ command is part of a fused operation is indicated in the Fused Operation field of Command Dword 0 in Figure 6. The Fused Operation field also indicates whether this is the first or second command in the operation.

Modify section 4.7 as shown below:

A command is fetched when it is retrieved from host memory and stored internally to the controller. A command is launched when the controller begins executing that command.

Arbitration refers to the order in which commands submitted for execution by host software are launched for execution by the controller. The controller may access Submission Queues in any order. The controller fetches commands from memory for future execution in order from each individual Submission Queue it accesses. The controller may store commands internally for future execution. ~~The controller may fetch commands from memory for future execution in any order and may choose to fetch more commands than are launched for the Submission Queue in the near term~~**.** Arbitration does not imply command completion order, rather arbitration determines the order in which commands that are launched ~~(or started)~~ for execution by the controller. Since commands are of different types, of different sizes, and to different LBA ranges on the controller, the order of completion is likely to be different than the order of command launch. ~~However, using macro benchmarks with commands of similar type and size, performance should be observable to within a few percentage points of the assigned bandwidth to the associated Submission Queue.~~

All controllers shall support the round robin command arbitration mechanism. A controller may optionally implement weighted round robin with ~~an~~ urgent priority class and/or a vendor specific arbitration mechanism. The Arbitration Mechanism Supported field in the Controller Capabilities register (CC.AMS) indicates optional arbitration mechanisms ~~implemented~~ supported by the controller.

A command is ready for execution when a Submission Queue Tail Doorbell write has completed that moves the Submission Queue Tail Pointer value past the corresponding Submission Queue entry for the associated command. Within the same Submission Queue, ready commands may be launched in any order.

In order to make efficient use of the non-volatile memory, it is often advantageous to execute multiple commands from a Submission Queue in parallel. For Submission Queues that are using weighted round robin with urgent priority class or round robin arbitration, host software may configure an Arbitration Burst setting. The Arbitration Burst setting indicates the maximum number of commands that the controller may launch at one time from a particular Submission Queue. It is recommended that host software configure the Arbitration Burst setting as close to the recommended value by the controller as possible (specified in the Recommended Arbitration Burst field of the Identify Controller data structure in Figure 65), taking into consideration any latency requirements. Refer to section 5.12.1.1.

If required resources (e.g. ~~NVM regions~~ logical block locations) are not available that command(s) require, then those command(s) may be deferred for launch to the next arbitration round. In some cases, this may result in fewer commands being launched from a particular Submission Queue in an arbitration round.

Modify the first paragraph of section 4.7.1 as shown below:

~~When~~ If the round robin arbitration mechanism is selected, the controller shall implement round robin command arbitration amongst all Submission Queues, including the Admin Submission Queue. In this case, all Submission Queues are treated with equal priority. The controller may launch multiple commands from each Submission Queue per round based on the Arbitration Burst setting.

Modify the third paragraph of section 4.7.2 as shown below:

The next highest strict priority class is the Urgent class. Any I/O Submission Queue assigned to the Urgent priority class is serviced next after commands issued to the Admin Submission Queue, and before any commands issued to a weighted round robin priority level. Host software ~~Software~~ should use care in assigning any Submission Queue to the Urgent priority class since there is the potential to starve I/O Submission Queues in the weighted round robin priority levels as there is no fairness protocol with those I/O Submission Queues.

Disposition log

4/21/2011	Erratum captured.
4/26/2011	Update to command fetch language.
6/10/2011	Erratum ratified.