T10/98-237r0

To: T10 Technical Committee
From: Chandru Sippy, QLogic Corp.
Subject: Packetized SCSI

The last line on T10/1302D rev. 0 Section 12.1 (Pgs. 127, 128) reads as follows:

On a write, SPI protocol does not allow the target to inform the initiator if the data was properly received, therefore, the initiator shall assume the data was properly received **and save the data pointers** as soon as the last CRC byte is sent. To retry a write operation a target shall send a MODIFY DATA POINTERS message then request that the SPI information unit be sent again.

The underlined bold text (for effect only) should really be removed from the specification.

My rationale for the removal of an implicit save data pointers is as follows:

Please refer to section 12.6 on SCSI Pointers. My understanding is outlined below:

- There is only set of active pointers that point to the next command, data, or status byte to be transferred.
- The saved pointers always point to the start the block to be transferred.
- In response to save data pointers, the initiator stores the value of the active data pointer into the saved data pointer for that task.
- The Modify Data Pointers (elsewhere in the specification) adds a 2s complement offset to the current active data pointer. It has no effect on the saved pointers.

For any given data transfer from the host system memory, we have a starting address (A) and a transfer length (L) for an ending address of A + L. If the target were to attempt modifying the data pointers outside this range between A and A + L with an invalid offset value, then an error should be flagged. The specification of course does not explicitly state this anywhere. But it reasonable to assume that this must be the case or else target data may get corrupted. The host memory is protected but the attached target medium is not. In other words, if the data transfer is just starting then a negative offset is illegal and the positive offset should be less than A + L.

Now assume that some data transfer has taken place and that D bytes have been transferred. Now the current address is at A + D and the remaining transfer length is L - D. The target may now issue a MODIFY DATA POINTERS with a negative offset not exceeding D and a positive offset less than L - D. Any other value should be illegal to ensure data integrity at either end.

Following the above logic, if a save data pointers was implicitly done, then the new starting address is A + D with a transfer length of L - D. The current pointers are also equal to the same values assuming no additional data transfer has taken place. Since the original values of A and L are not saved anywhere, the target may not modify the data pointers outside the new values. In other words, after transferring D bytes of data, with an implicit save data pointers one cannot go back. One can only go forward. If no save data pointers were assumed, then it would be okay subject to the restrictions that have been pointed out.

For the specification to be correct as stated one would have to save the actual original values also and then compare against these values all of the time. This requires having yet another copy of the original values (may be call it intermediate saved data pointers) besides the saved and active values.

Even though the specification does not state how many saved sets exist in the initiator, I contend that it **not** reasonable to maintain more than one saved value for a given task. All this does is add un-needed complexity to the initiator side f/w and the actual hardware that must notify the f/w that CRC has been sent and an implied save data pointers should be performed by the f/w. By simply striking out the underlined phrase, all of this just goes away as stated in the paragraph following the bulleted items.

However, George Penokie pointed out the following:

In packetized we need to have the pointers move at the end of each data IU because the next LQ IU may not be for the same IO process. So if they are not saved the pointer would be lost and the next time a transfer occurs for the original IO process the pointers would not be correct.

However, a small problem does seem to exist depending upon the combination of Save Data Pointers and Modify Data Pointers used by the targets. In an attempt to solve this dilemma, George further suggested the following:

There is a solution to the packetized version of this problem that an initiator can implement without any changes to the standard. The initiator when doing writes only need to monitor the phase lines to see if the data pointer should be saved. It could go something like this:

1. Phase is DT DATA OUT -- Initiator sends data and CRC in the data IU.
2. At the end of the CRC the initiator holds off saving the data pointers.
3. If the initiator detects a DT DATA IN phase it saves the data pointers before the L_Q IU forms the new nexus -- The target would not do this if there were a CRC or parity error.
4. If the initiator detects a Message In phase it looks to see what the message is. If it a Modify Data Pointers message then it does not save the pointers but modifies them per the message.
5. If the initiator detects any other phase or message (in the case of a Message in phase) it should save the pointers and do whatever.

There may be even simpler ways to solve the problem than the ones briefly outlined above. All Qlogic is looking for is some clarification and to alert other members that there is a small problem here. So do be careful before saving the data pointers blindly. It may create problems later!