SQRL

Scsi
Queued
Reference
Lists

A proposal for SCSI on the P1394 Bus

Abstract

The current proposal for implementing SCSI on the P1394 bus, SBP (Serial Bus Protocol), describes a complicated method for managing the limited resource of tap slots on a target. This paper proposes a new command delivery mechanism for SBP that is very similar but does not incur the need to implement a complex algorithm for managing tap slots. It does not attempt to define many of the details, only the overall concept and a few considerations.

In designing this new command delivery mechanism several criteria were established and met:

- The protocol should not require specialized hardware to ensure minimum functionality. Ideally, full functionality could be achieved without any additional hardware.

- If a hardware implementation is desired, it should be feasible with a minimum of gates.

- During normal operation, the protocol should require the minimum amount of bus traffic. During exception handling, there should not be an "explosion" of traffic.

- No limit should be placed on initiators' ability to enqueue commands.

- The protocol should implement the SCSI-3 queuing model: Queue Empty, Round Robin, Head of Queue, Linked Commands, Abort by Command, Abort by Initiator, etc...

- The protocol should be as small a deviation to the current SBP proposal as possible to leverage the work already done there.

- Minimum implementation targets should not be burdened with excessive protocol.

SQRL Command Delivery Protocol

The proposed protocol is called SQRL (Scsi Queued Reference List) or - tongue in cheek - SQUIRREL (Scsi Queues Using Indirect References in Realtime Editable Lists) and is based upon a two dimensional queue structure. Basically, an initiator sends the first command block of a chain of commands to the target. The target is responsible for maintaining a queue of these chains. When the internal resources for maintaining the queue are exhausted, the target will use the originals located in the initiators' address space to continue maintainence.

Important to remember is that for each chain, the target only holds the first command block in the chain. As commands are prefetched, they are no longer a part of the queue structure.

In order to more easily explain how the protocol works, the concepts of *strict* and *loose* interpretations must be defined.

A strict interpretation of the protocol requires that as a chain is enqueued to the list, the link pointer in the previous chain is updated to point to the newly enqueued chain. In this interpretation, the original copies of all the chains are robustly linked in the initiators' address space. This interpretation requires that a write transaction follow every chain sent to the target.

A loose interpretation allows the drive to maintain an internal link between the chains for all chains it holds in memory. For chains received after the target has used up its internal storage, the target will have to begin writing updates to the link pointers in the original copies and discard the new chains for which it has no room. Under the normal case of not exceeding the target's internal storage capacity, this interpretation results in the absolute minimum of bus traffic for sending commands. This interpretation may be thought of as a "split queue" with part of the queue being maintained in the target and the rest being maintained in the initiators' address space.

First, the protocol will be described with a strict interpretation to avoid the confusing side issues and then the considerations of the loose interpretation will be explored.

Strict Interpretation

Target registers

To implement the SQRL queues, a target must contain in its CSR ROM a 1212 pointer to where command chains should be sent, called the Queue register. There is a seperate entry for each of the command queues supported by the target. An initiator will send command blocks to the Queue register.

Internal to the target and not available to the 1212 address space are two variables, pQueueHead and pQueueTail which point to the first chain in the queue and the last chain in the queue respectively. These are maintained by the target.

Command Block

A Command Block consists of two parts: a pair of 1212 pointers for maintaining the queue of chains and the chain itself, and the rest is devoted to the CDB, status block pointer, etc...

The first entry in a command block is the pNextChain pointer. This provides the link that queues chains together. It is initially set to point to itself so the target knows where the chain came from.

The second entry in a command block is the pNextCommand pointer. This points to the next command block in the chain. The last command block in the chain has its pNextCommand pointer equal to NULL.

Chains

To construct a chain of commands, an initiator first sets the pNextChain pointer to point to itself for all command blocks in the chain. It then links each of the commands through the pNextCommand pointers with the last pointer equal to NULL. To send this chain to the target, the initiator writes the first command block in the chain to the Queue register on the target.

Queue Structure

The target has two internal pointers, pQueueHead and pQueueTail, for each of the supported command queues. pQueueHead always points to the first command block of the first chain in the queue. For every chain in the queue except the last one the pNextChain pointer points to the next chain in the list. The last chain's pNextChain pointer points to itself. The pQueueTail pointer in the target points to the last chain as well.

Operations

Enqueue

An initiator constructs a new chain to send to the target as described in the section "Chains" above. The initiator then sends the entire first command block in the chain to the Queue register in the target.

If this is the first chain received, the target places the address contained in the pNextChain entry of the command block (which is a pointer to itself) into both the pQueueHead and pQueueTail pointers.

For subsequent chains, the target writes the newly arrived pNextChain entry value to the 1212 address pointed to by pQueueTail. pQueueTail is then updated to point to the new chain.

Dequeue

After a target has fetched all of the commands in a given chain, the target will remove the chain from the queue.

If the chain is the only member of the queue, the pQueueHead and pQueueTail variables are initialized to NULL, signifying empty.

Otherwise, the target puts into the pQueueHead pointer the pNextChain entry from the chain it is discarding.

Queue Empty Fetch

When the fetch policy in force is to empty the current chain, the target modifies the pQueueHead pointer to the value in the pNextCommand entry of the last fetched command block.

When the pNextCommand pointer is NULL, this chain is complete and can be dequeued.

Round Robin Fetch

When the fetch is to be round robin, the target will dequeue the chain on top and then enqueue it to the bottom.

First, the target saves the value of pNextChain for the current chain, writes the value in pQueueHead to where pQueueHead points (thus reestablishing the self reference in pNextChain for the current chain) and copies the saved pointer to the next chain into pQueueHead.

Next, the target writes the address of the newly dequeued chain to the location pointed to by pQueueTail and updates pQueueTail to point to the newly enqueued chain.

Loose Interpretation

Since a target is likely to have room for more than one command, it will be able to accept command blocks up to its limit without having to maintain the linked list in the initiator space. All queue operations occur within the target's internal storage. Of course, once its internal storage is used up it will have to begin using the strict interpretation.

The great advantage of this interpretation is that no unneeded traffic is present on the bus as long as the total number of outstanding chains is less than the target's internal storage. Its disadvantage is that a more complex algorithm is needed by the target to avoid breaking the queue.

An example of how complex it can get is when a fetch policy requires reading commands from the top chain thus "bumping" other chains out of the target's internal storage area. When this happens, the target will have to insert the bumped chain into the initiator's space and discard the chain locally. Another example of complexity arises when a round robin switch of the top chain to the bottom collides with an incoming chain.

Both the strict and loose interpretations lend themselves to a hardware implementation. For the strict case, the hardware would always write the link pointers on receipt of a new chain. The hardware for the loose case would be identical except for a counter that describes how much space is left in the target's internal storage. After the counter is exhausted, the automatic link updates would kick in.

Given the advantages of the loose interpretation, the standard should not require the target to maintain the strict interpretation to the extent that the target can maintain the queue locally.