X3T9.2/91-207R0
December 5, 1991

To: X3T9.2 Membership

From: Edward A. Gardner, Digital Equipment Corporation
gardner@ssag.enet.dec.com

Subject: Periodic LRC/CRC Proposal .

During the November working group meeting there was a lively discussion of George Penokie's "SCSI Data Phase LRC Proposal" (91-176R0). One point that was discussed was whether a single LRC or CRC should be used for all data transferred by a command, versus inserting an LRC or CRC for each period of data. This is a proposal for how a periodic LRC or CRC might be implemented. It primarily discusses how the data phase transfer operates, leaving the details of initialization and parameter negotiation to a later document.

The SCSI LRC or CRC arena includes many controversial issues in addition to periodicity. While this proposal concept is general, I have included concrete examples that incorporate certain assumptions:

1. The LRC, CRC, or whatever error detection code is used is 32 bits long. I personally favor a 32 bit CRC, specifically the Autodin-II CRC used by Ethernet, FDDI, and many other busses. However this proposal is trivially adaptable to any error detection code.

2. The LRC, CRC, or whatever detection code is only calculated on data phase information. Again, this proposal can be extended if we can agree on what other information should be included.

1 Periodic CRCs

Through some yet to be defined negotiation process, the initiator and target agree to use an LRC or CRC during data transfers. As part of this negotiation they agree on two quantities, a period size and an alignment. The period size specifies the number of data bytes covered by each LRC or CRC. The alignment specifies the transfer boundary (e.g., 1, 2, or 4 byte boundary) at which the LRC or CRC must appear. If the period size is not an exact multiple of the alignment, pad bytes will be inserted. Note that the alignment is not necessarily the same as the bus width. For example, a device might have 32 bit internal data paths and wish to ensure 32 bit alignment even though it uses an 8 bit SCSI bus. Of course, it would simplify things if we could all agree to always use (for example) 32 bit CRC alignment, but lots of options and parameters are necessary to comply with existing SCSI practice.

The data transferred for a command (i.e., what would appear on the bus if all data phase information were transferred without disconnecting) consists of one or more periods. Every period except the last includes the agreed upon period size number of data bytes. The last period is shortened (if necessary) to accomodate arbitrary transfer sizes. Following the data bytes are pad bytes (if necessary) to achieve the agreed upon alignment and then the CRC. The CRC covers both the data bytes and the pad bytes. I believe it advantageous to require that CRC pad bytes be zero, despite the precedent of wide data transfer padding.

The number of data bytes, number of pad bytes, and overall size of every period except the last are determined from the agreed upon period size and alignment. If the last period is a full period (it contains the period size number of data bytes) then it too is formatted according to the period size and alignment. If the last period is shortened, then an IGNORE WIDE RESIDUE message is sent if any pad bytes are present. Note that the IGNORE WIDE RESIDUE message is sent after the CRC, although the pad bytes are before the CRC. Note also that this results in IGNORE WIDE RESIDUE being used on 8 bit SCSI busses. Also, if we want to allow the alignment to be smaller than the bus width, then there are ugly special cases that result from there being both CRC period padding and bus width padding. While there are several ways to deal with this, I choose to postpone writing anything down until persuaded that SCSI-3 should allow such a crock.

Examples (periodic without disconnection):

Transfer 1026 bytes with 513 byte period size and 4 byte alignment: Period 1 [513 data bytes, 3 pad bytes, 4 CRC bytes], Period 2 [513 data bytes, 3 pad bytes, 4 CRC bytes].

Transfer 1024 bytes with 513 byte period size and 4 byte alignment: Period 1 [513 data bytes, 3 pad bytes, 4 CRC bytes], Period 2 [511 data bytes, 1 pad byte, 4 CRC bytes], IGNORE WIDE RESIDUE specifying 1 byte.

Transfer 1021 bytes with 513 byte period size and 4 byte alignment: Period 1 [513 data bytes, 3 pad bytes, 4 CRC bytes], Period 2 [508 data bytes, 4 CRC bytes].

2 Non-Periodic CRCs

One particular value of period size, perhaps zero, will be reserved to indicate that the period size is arbitrarily large. This results in a non-periodic CRC. The entire data transfer is a single period consisting of data bytes, pad bytes if necessary for alignment, and a single CRC. The single period is always a shortened period as defined above.

Examples (non-periodic without disconnection):

Transfer 1024 bytes with no CRC period and 4 byte alignment: Period 1 [1024 data bytes, 4 CRC bytes].

Transfer 1021 bytes with no CRC period and 4 byte alignment: Period 1 [1021 data bytes, 3 pad bytes, 4 CRC bytes], IGNORE WIDE RESIDUE specifying 3 bytes.

3 Disconnection and Reconnection

Except for fatal or non-recoverable errors, targets shall not disconnect or change bus phase between the last data byte of a period and the last CRC byte of that period. Once the last data byte of the period has been transferred (strictly speaking, once REQ has been asserted for that data byte), the target shall continue the data phase transfer at least until the last byte of the period's CRC. Failure to do so is an illegal bus phase sequence.

The SCSI protocol as already defined allows disconnection, reconnection, and data pointer modification at arbitrary data byte boundaries, although such generality will probably not be supported by most devices. Those functions work the same as at present. In particular, MODIFY DATA POINTER only counts data bytes, it ignores or skips pad and CRC bytes, and therefore positions to the same data byte as at present.

There is no way to position the data pointer to a pad or CRC byte. The only way to transfer the pad and CRC bytes at the end of a period is to transfer the last data byte of the period and the subsequent pad and CRC bytes in one fell swoop.

One of the motivations for a periodic CRC is that it can substantially simplify implementation. The obvious approach is to always transfer entire periods (data, pad, and CRC) during a single data phase. Two separate restrictions accomplish this.

First, restricting MODIFY DATA POINTER to only reposition to a period boundary. While it's easy for MODIFY DATA POINTER to specify a location in the middle of a period, implementing such messages is likely to be impossible for most initiators. The initiator would need to be able to regenerate the CRC corresponding to an arbitrary point in the transfer rather than calculating it as the transfer proceeds. In the case of a Data In transfer the initiator might receive a CRC for which it has not yet received the data. In general, practical implementations will require that MODIFY DATA POINTER position the data pointer to a period boundary. The CRC setup negotiation must allow the initiator to specify such a restriction.

Second, restricting disconnection and reconnection to period boundaries further simplifies initiator implementation. If disconnection and reconnection may occur at arbitrary points, then the initiator must save the CRC context as well as the data pointer. If disconnection and reconnection only occur at period boundaries, then the initiator need only save the data pointer.

Combining these two we get the following possible combinations of restrictions that the initiator may need to impose on data transfers:

		Disconnection / Reconnection		
		Not Allowed	At Period Boundaries	Anywhere
MODIFY POINTERS	Not Allowed	E	*	E
	At Period Boundaries	-	*	*
	Anywhere	-	-	E

E Exists in SCSI today.
* Needs to be added.
\- Meaningless combination.

4 Unexpected Short Transfers

The initiator and target must agree on where CRCs are located, on the length of each period, for CRC checking to work. This is straightforward when the entire expected transfer length is transferred. The initiator knows the expected transfer length when it issues a command. The target determines the expected transfer length from the CDB, in which the length is explicit or implicit. So long as the entire expected transfer length is transferred, both agree on the actual transfer length and will agree on which bytes are data bytes, pad bytes, or CRC bytes.

Dealing with unexpected short Data Out transfers is also straightforward. The initiator always formats the transfer into periods and supplies CRCs as if the entire expected transfer length would be transferred. The target accounts for this and responds accordingly, even if an exception terminates the transfer prematurely. For example, if the target needs to verify the CRC of an unexpected short transfer, it would fetch and discard data from the initiator until the end of a period.

The difficult case is unexpected short Data In transfers. The initiator will not know a priori that the transfer will be shorter than expected. In some cases the target may not know a priori either. The Data In transfer must occur first, and the fact that the overall transfer is unexpectedly short communicated afterwards.

Consider what happens if a data period being transferred to the initiator is unexpectedly shortened. The target stops sending data, inserts pad bytes if necessary, sends a CRC, then changes the bus phase. The initiator cannot distinguish the pad and CRC bytes from data bytes until after the bus phase change. Thus the initiator will process the pad and CRC bytes as data bytes, presumably storing them in the application's buffer.

However, mistakenly storing pad and CRC bytes in the application's buffer is benign in this case, since the buffer is guaranteed to have room for them. Since the data transfer is unexpectedly short, the

application's buffer has room for more data bytes than were actually transferred. The pad and CRC bytes will be stored into those available but unused locations. (If there are fewer available locations than pad and CRC bytes, the pad and CRC bytes will only be stored up to the end of the data buffer. The normal initiator logic will process the remainder as if they really were pad and CRC bytes and either discard them or store them elsewhere).

An initiator adapter could, if its designer so wished, preserve the last few bytes of the data transfer in a FIFO until it knew they were valid data. The FIFO would have to be at least as large as the maximum number of pad bytes plus the size of the CRC. This is analogous to an adapter preserving the last word of a wide data transfer until it knows whether an IGNORE WIDE RESIDUE message will be sent. In practice such an approach may be difficult to retrofit to current adapter designs.

Checking if the CRC was valid after the fact is trivial for initiators that only allow disconnection and reconnection at period boundaries. When the target changes the bus phase, it denotes either the end of the transfer or a disconnection. Either way the initiator knows that the immediately preceeding four bytes were the CRC. The initiator need merely latch whether or not its CRC checker contains the unique pattern denoting a valid CRC when it sees the phase change.

Initiators that allow disconnection and reconnection anywhere must preserve the entire CRC checker contents. They do not know that it is a CRC (and therefore cannot determine if it was valid) until the target indicates the transfer is complete. This includes such complex cases as, for example, the target transferring the data and CRC, disconnecting, then later reconnects to say the command is complete. The initiator cannot determine that the last four bytes were the CRC and not data until the target reconnects.

This is likely to be somewhat complex to implement in SCSI protocol chips without the cooperation of higher level software. If needed, we can add some protocol to allow the initiator to immediately distinguish data from CRC. If any pad bytes were inserted to align the CRC, an IGNORE WIDE RESIDUE is already being sent after the CRC. This message not only specifies the number of pad bytes, it also implies that the last four bytes were in fact a CRC. It would be straightforward to always send an IGNORE WIDE RESIDUE message after an unexpectedly short transfer to inform the initiator that the last four bytes were the CRC. The IGNORE WIDE RESIDUE parameter would be zero if no pad bytes had been inserted. This is a tradeoff between the complexity of deferring the CRC validity check until the initiator knows the transfer has completed versus the complexity of an extra message. Presumably we should follow the SCSI precedent and allow both as options.