From Dave Guss SSI July 15, 1997

Background

- Configuration of a SCSI physical interface is typically not under the control of the device manufacturers. Misconfigurations, and/or inferior components on the interface, can result in a number of reliability problems, not the least of which is extra and/or missing REQ and/or ACK pulses caused by reflections and ISI effects.
- Less than robust implementations of the Data Pacing and/or Error Detection circuitry can compound this problem.
- The sharing of information concerning a robust implementation of this circuitry is in the collective best interest of the industry and the committee is therefore interested in including it in one of the SCSI documents.
- The following is my attempt to understand the effects of these errors and to identify a set of recommendations for the robust implementation of the detection circuitry.

SCSI Synchronous Transfer Data Pacing Overview

- The Initiator commands the Target to move data by sending a CDB containing the direction, length, etc. of the transfer.
- The Target orchestrates the movement of all data by sending the required number of REQ pulses to the Initiator. The Initiator responds to each REQ pulse by sending a corresponding ACK pulse back to the Target. This sequence is the same for read or write operations.
- For a read operation, the data bus is driven by the Target and each word of data (1, 2 or 4 bytes) is strobed from the bus by the Initiator on the active edge of the received REQ pulse (see Fig. 1).

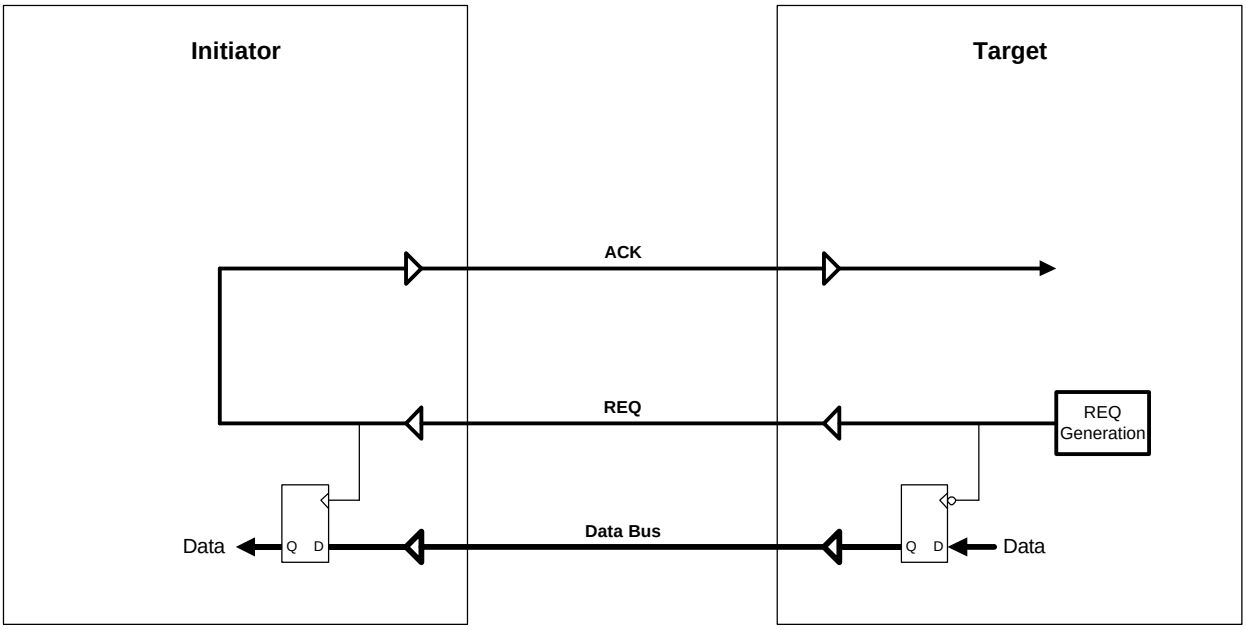

Figure 1 - SCSI Read Data Movement

- For a write operation, the data bus is driven by the Initiator and each word of data is strobed from the bus by the Target on the active edge of the received ACK pulse (see Fig. 2).

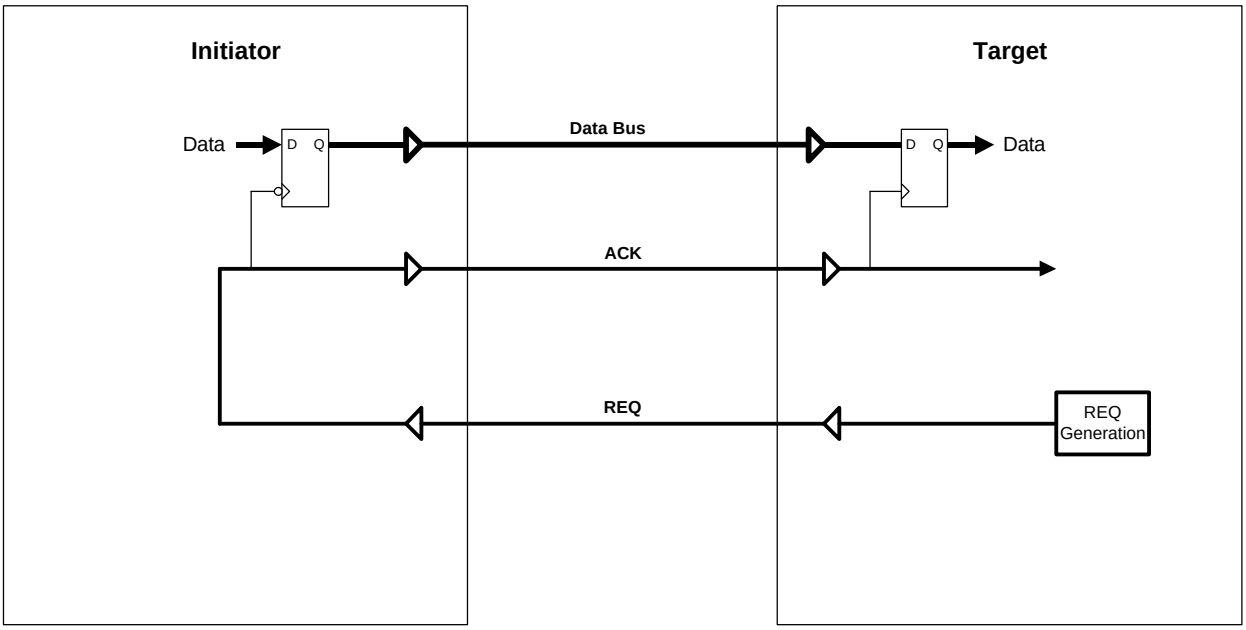

Figure 2 - SCSI Write Data Movement

- If all Initiators and Targets had guaranteed immediate access to buffer resources (available data to send or empty buffer space to receive data) during a Synchronous transfer, REQ/ACK counting, at least for the purpose of flow control, would not be necessary. However, this is not the case and a method of flow control is required to allow “throttling” of the transfer by the Initiator or Target when they temporarily run out of these resources.
- The following discussion on flow control describes the functional requirements without reference to specific hardware implementations (e.g. up counters, down counters, FIFO control ranks, etc). This should yield a set of recommendations that can be used regardless of the approach take in hardware.
- The method of flow control (a.k.a. "pacing mechanism") used in SCSI can be thought of as the circular flow of “tokens” in a loop from the Target to the Initiator and back to the Target. The Initiator and Target establish the number of tokens in the loop by negotiating the "Maximum Offset" they can tolerate. The Target starts a transfer with the Maximum Offset number of tokens available in its Available Token Count. When a word of data is to be transferred, the Target “spends” a token to generate the REQ pulse. When certain conditions in the Initiator are met it returns the token to the Target by sending an ACK pulse. When the Target receives the ACK pulse, and certain condition are met, the token is again made available to generate another REQ pulse. When no tokens are currently available, the Target’s REQ generation circuit must suspend the sending of REQ pulses. By this means, circuitry in the Initiator or in the Target can throttle the transfer by withholding the flow of tokens around the loop.
- The conditions necessary in the Initiator or the Target for “advancing” a token are different for read and write operations.
    - For SCSI reads (see Fig. 3) a REQ pulse is generated by the Target when the transfer count for this data phase has not been exhausted, there is data available to send and the

Available Token Count is not zero. The Available Token Count is then decremented by the generated REQ pulse. That REQ pulse results in the transferred word being placed in the buffer in the Initiator. When that buffer slot is again emptied, the token is released to be used to generate an ACK pulse (the token may be held in an Initiator Tokens Owed Count until the ACK can be generated at the current data rate). The Target uses the received ACK pulse to increment the Available Token Count, making the token available to generate another REQ.

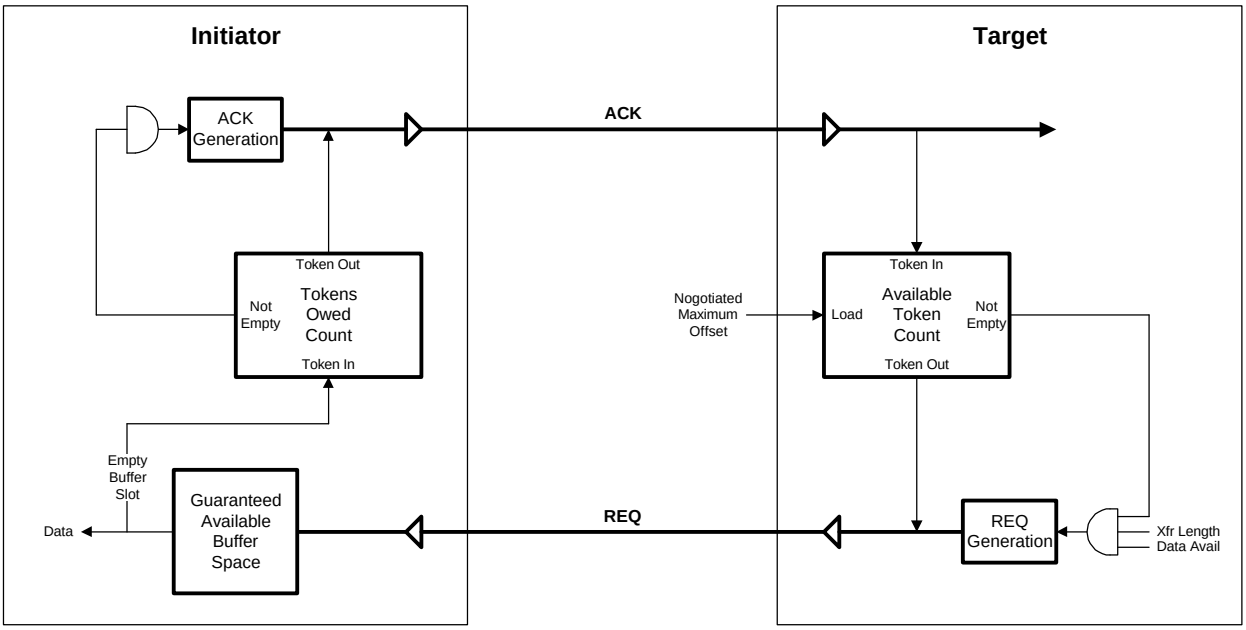

Figure 3 - SCSI Read Pacing Functionality

- For SCSI writes (see Fig. 4) a REQ pulse is generated by the Target when the transfer count for this data phase has not been exhausted and the Available Token Count is not zero. The Initiator uses the received REQ pulse to increment the Tokens Owed Count. When the Tokens Owed Count is not zero and data is available to send, an ACK pulse is generated. The Tokens Owed Count is then decremented by the generated ACK pulse That ACK pulse results in the transferred word being placed in the

buffer in the Target. When that buffer slot is again emptied, the token is released to be used to increment the Available Token Count, making the token available to generate another REQ.

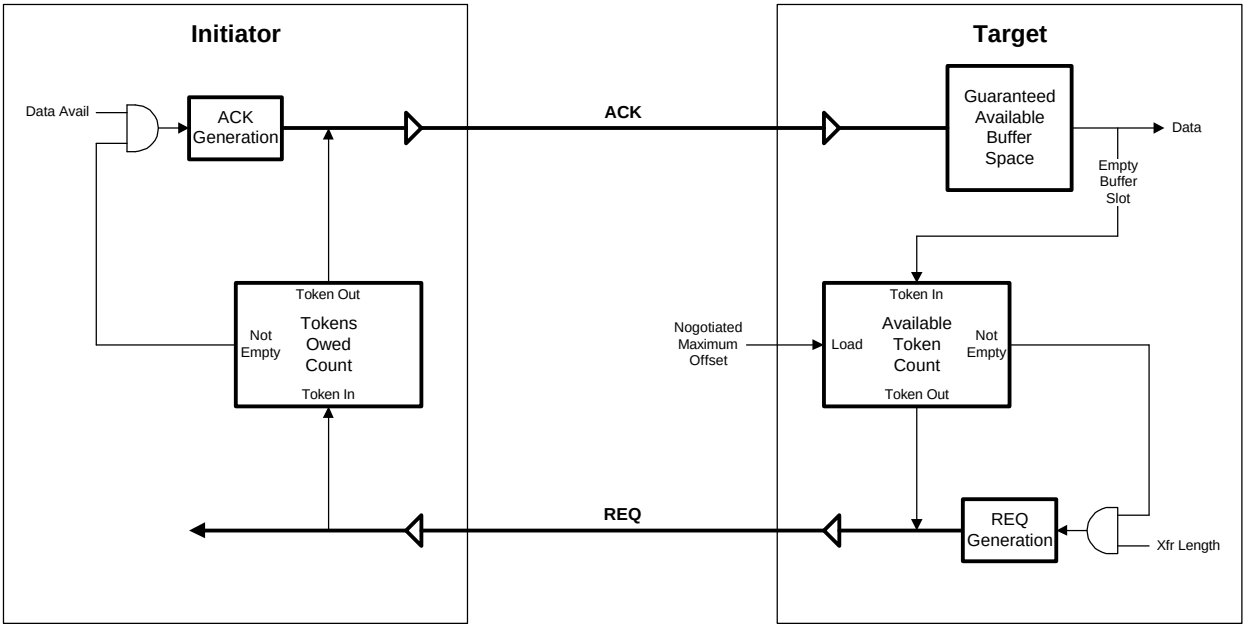

Figure 4 - SCSI Write Pacing Functionality

- The movement of a token around the loop is impeded by latencies. Examples are, the round trip delay through cable and electronics, the time to empty the receiving buffer slot and delays synchronizing events to Initiator and Target clock systems. It should be noted that if the total latency incurred around the loop exceeds the Maximum Offset number of word times, throttling (reduced data rate) will occur, even if adequate data bandwidth exists at each end of the nexus.

Failure Modes

- The failures modes of interest are:
    - Missing REQ or ACK active edges, typically caused by level shift due to ISI effects.

    - Extra REQ or ACK active edges, typically caused by leading edge distortion due to reflections or by induced noise.
- Of these, the dominate mode for any given transfer is undoubtedly a single type of failure (missing REQ(s), missing ACK(s), extra REQ(s) or extra ACK(s)), but it is also possible that any combination of these failures can happen with some reduced probability. To be thorough, here is a look at all combinations.
    1. Equal numbers of extra/missing REQs and/or equal numbers of extra/missing ACKs.
        - These two combinations of errors are not detectable by REQ/ACK counting because their net effect, at either end of the nexus, is zero. The effects of these errors can only be detected by some type of longitudinal redundancy on the data itself.
    2. Equal numbers of extra REQs and missing ACKs.
        - This error combination is not detectable by REQ/ACK counting in the Target, because its net effect at the Target is zero. It can be detected by the Initiator, at Command Complete only, provided that the Initiator can count the REQs received and detect that the count exceeded the total expected transfer count. Otherwise it falls into the first category.
    3. Equal numbers of missing REQs and extra ACKs.
        - This error combination is also not detectable by REQ/ACK counting in the Target, because its net effect at the Target is zero. It can be detected by the Initiator, at Command Complete only, provided that the Initiator can count the REQs received and detect that the count is less than the total expected transfer count and provided that the expected transfer count is deterministic (i.e. fixed block type device). Otherwise it falls into the first category.

4. All remaining errors, or combinations of errors, cause a net surplus or deficit of returned ACK pulses at the Target and can be detected by the Target's Available Token Count at the end of each Data Phase. A subset of these errors may also be detectable in the Initiator but such detection is redundant and appears to be of little incremental benefit.

Error Detecting Circuits

- The purpose of detecting extra or missing REQ/ACK edges is to prevent the undetected corruption of data caused by the resulting dropped or inserted interface words. To maximize the effectiveness of this error checking, care must be taken to ensure that the circuitry used to detect the REQ/ACK edges for the purpose of error detection "see" the same edges as the circuitry advancing the data path. This may be difficult if multiple circuits are used to detect the same edge because the perceived REQ/ACK pulses are likely to contain marginal energy and each circuit will have unique sensitivity to those pulses.
- There are three areas of functionality that have been identified as useful in the detection of extra or missing REQ/ACK edges:
    - Initiator's REQ counting mechanism - this optional function may already exist to protect buffer boundaries, etc. It can be used to check the number of REQs received and report if the number exceeds the requested transfer count. It can also be used to check if less REQs were received than the transfer count, but only for devices with deterministic block lengths. These checks can only be made at the end of an entire transfer, and not at the end of each Data Phase. Using this counter to detect missing/extra REQ pulses provides very little incremental benefit because the Target's Available Token counting mechanism will detect all the same errors, except for the very narrow case of equal and opposite REQ and ACK errors that cancel each others effect.

- Initiator's Tokens Owed counting mechanism - proper implementation of this required function is critical to the robust detection of extra REQs (detection of missing REQs or ACKs is not problematic). It must be able, under all conditions, to pass at least one extra token back to the Target as an extra ACK, so that the Target can detect the error condition. It should never reduce its count (e.g. rollover) in response to any number of extra REQ pulses. Using this circuit to detect extra REQs (as opposed to passing that indication back to the Target) has little benefit because it must be at the Maximum Offset count when the extra REQ arrives, giving very narrow coverage. Using this counter to check for more ACKs sent than REQs received is just a test of the Initiators internal circuitry and not part of this discussion.

- Target's Available Token counting mechanism - proper implementation of this required function is also critical to the robust detection of extra REQs or ACKs. It must be able to detect, under all conditions, that more ACKs were received than REQs were sent during any Data Phase (i.e. one, or more, ACKs received while the Available Token count was already equal to Max Offset). It should never decrease its count (e.g. rollover) in response to any number of extra ACK pulses.
    - The detection of less ACKs received is already done in properly implemented devices because the Target must not exit the Data Phase until the Available Token count is at Maximum Offset. Missing REQs or ACKs will then result in an interface "hang" waiting for ACKs that will never be sent. Because the maximum time to complete a SCSI Data Phase or a SCSI Command is not specified, this condition will typically be discovered by a system software timeout

which must then cause the Initiator to Reset the entire SCSI interface.

- The detection of more ACKs received has problems because the Target will assume the transfer is over after the proper number of ACKs are received and it will then leave the Data Phase. The remaining ACK(s) may come at any time after this, leaving open the possibility that they will go undetected as the Target moves on to send the Status or Message byte (see Figure 5). To minimize expose to this problem the Initiator should be able to detect, and report as an error, the condition of the interface leaving the Data Phase while ACK(s) are still owed. It should also abort sending any remaining ACK(s) to reduce the chance that they are confused with the subsequent Status or Message Phase. The Target should continue to monitor the ACK signal after it has left the Data Phase. Any active edges on ACK that occur before the Target asserts REQ for the Status or Message byte, should be reported as an error. The Target could wait longer before moving out of Data Phase or before asserting REQ for the Status or Message Phase, in hopes of increasing the chance of detecting this error. However, the increase in detection probability would have to be understood and traded off carefully against the reduced performance.

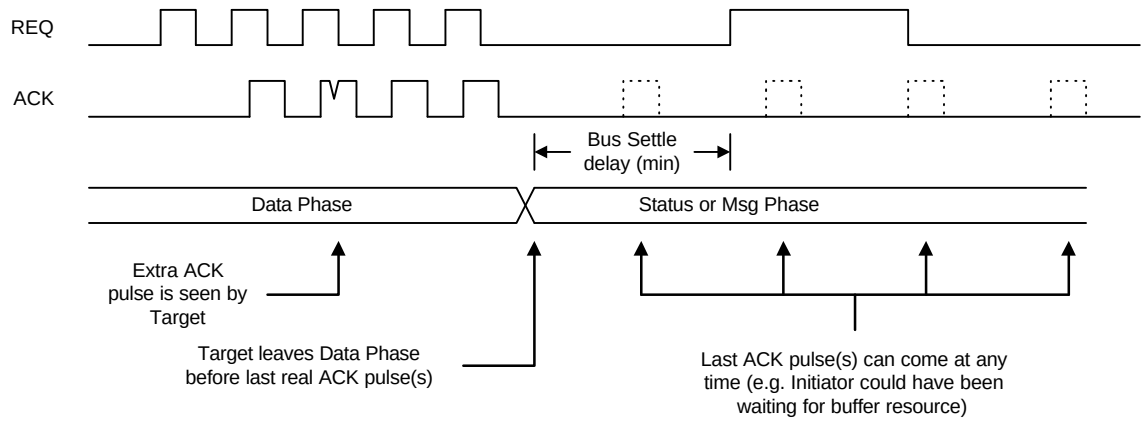

Figure 5 - Extra ACKs at the end of the Data Phase

Recommendations

1. A single circuit should be used in the Initiator and Target to detect the active edges of REQs or ACKs. This indication should be passed unambiguously to all other circuits requiring notification.
2. The Initiator's Tokens Owed counting mechanism must faithfully remember that at least one extra ACK is owed, regardless of how many extra REQs are received.
3. The Initiator should detect that the interface has left Data Phase while it still owes ACK pulses and report this as an error. It should also refrain from sending any remaining owed ACK pulses after the interface leaves the Data Phase.
4. The Target's Available Tokens counting mechanism must faithfully detect that at least one extra ACK was received while the Available Token Count was at Maximum Offset, regardless of how many extra ACKs are received.
5. The Target should continue to monitor the ACK signal after leaving the Data Phase to detect that additional ACK pulses have occurred. This monitoring should continue until as close as possible to the assertion of the REQ signal for the subsequent Status or Message Phase.