Dave Guss, Silicon Systems
May 5, 1998

Problem 1
Missing or extra REQ/ACK pulses can corrupt data but not be detected when multiple circuits in a device are used to detect the same edge. This can happen because the perceived REQ/ACK pulses are likely to contain marginal energy and each circuit will have unique sensitivity to those pulses.

Advice to Implementers
Where possible, 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.

Problem 2
The REQ/ACK counting mechanisms in some **Initiators** and **Targets** can overflow when they receive extra pulses. This can cause the error to go undetected or be misinterpreted as missing pulses (which hangs the interface).

Advice to Implementers
The **Target's** REQ/ACK counting mechanism must faithfully detect that more ACKs have been received than REQs sent, regardless of how many extra ACKs are received. To accomplish this care must be taken to either detect, or prevent, counter overflows.
The **Initiator's** REQ/ACK counting mechanism must either:

- Be capable of faithfully holding an ACKs owed count of at least one greater than the maximum offset, regardless of how many extra REQs are received. This guarantees that the indication that one or more extra REQs were received will be signaled to the **Target** with at least one extra ACK. To accomplish this care must be taken to prevent counter overflows.
- Detect that it's current ACKs owed count is greater than the maximum offset and report it as an error.
- Keep a count of the total number of REQs received and compare it to the total transfer length, if known, and report a miscompare as an error.

Problem 3
Extra REQ or ACK pulses can cause problems at the end of the Data Phase because the extra ACKs received by the **Target** can come at any time after it assumes the transfer is over, confusing the downstream protocol.

Possible Protocol Change
The **Initiator** should detect that it has an ACKs owed count greater than zero at the time the **Target** leaves Data Phase. It should then stop sending any owed ACKs as quickly as possible.
The **Target** should continue to monitor the ACK signal after it assumes that the transfer is over and detect any active edges that precede it's raising of REQ for the following phase.
If a timing constraint were placed on the **Initiator** that limited the time it could take from seeing the end of Data Phase to when it can guarantee that no more ACKs will be put into the cable, then the **Target** could delay that period of time, plus the Bus Settle delay, before sending REQ for the following phase. This would make the end of Data Phase "clean up" more robust, but would have a minor impact on performance.

Problem 4
Missing REQ or ACK pulses cause the **Target** to wait forever for an even REQ/ACK count, hanging the interface. This condition is currently detected by a gross software timeout on the execution of the command.

Possible Protocol Change
If the **Target** had knowledge of the maximum time the **Initiator** could take to respond to the last REQ with it's last ACK, then the **Target** could run a timer while waiting for the REQ/ACK count to even up. The **Target** could leave Data Phase and report an error if it exceeds the maximum time.