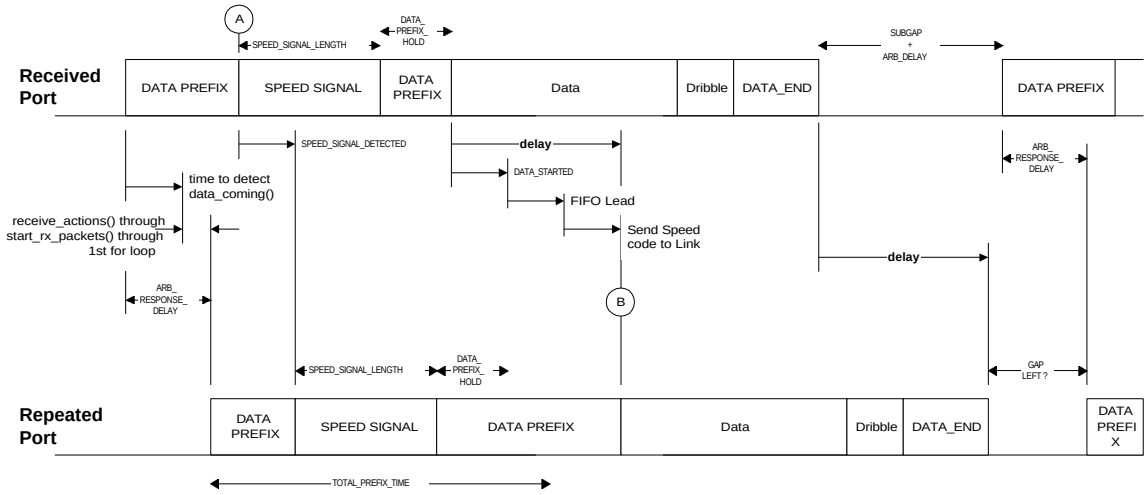
date : February 6, 1998
author: Steve Hamilton
company: Innovative Semiconductors, Inc.

I am by no means a PHY timing expert. But careful reading of the standard, and of draft 1.3 of the 1394a document, leaves me with the conclusion that 2 issues remain. The 1394A draft might want to provide additional clarification or restriction here. Discussion of the issues starts below. That discussion refers to the following drawing. I welcome correction if i have missed something pertinent.



PHY Timing Issues

Isssue #1 :

The first issue relates to which critical path determines point B in the above drawing. Point B is determined by the latest of 3 points. I believe the operating assumption of the specs to date has been that B is determined by the starting of data arrival. First transitions are detected, then enough of them have been counted to give the Resync buffer a sufficient lead (in case of clock tolerance issues), and finally moving the PHY-LINK interface from DATA_COMING to RECEIVE takes some time. These cummulatively are indicated by “delay” in the drawing. I believe this is what the standard’s PHY_DELAY refers to, although a precise reading leaves question as to whether the PHY_LINK action is included (i’ll assume it is).

Two other paths might determine B however. Starting from point A, some time is needed to recognize the presence of a speed signal (when there is one). The 1394A spec provides some good incites about how to do this, and why it may require some time. Then, once the speed signal is sent on the repeated port(s), data cannot be sent until SPEED_SIGNAL_LENGTH plus DATA_PREFIX_HOLD times elapse. These same 2 intervals (at a minimum) seperate point A from the arrival of data on the receiving port. Therefore it is the difference between “delay” and SPEED_SIGNAL_DETECTED which determines if this, or the data arrival, will determine point B. If SPEED_SIGNAL_DETECTED is greater than “delay”, then the path from point A dictates B. The 1394A spec requires PHY_DELAY to be greater than 60 ns. No spec is given for SPEED_SIGNAL_DETECTED. But clause 7.10.1 and table 7-19 show how it might easily take 3 or more 50 MHz clocks (> 60 ns).

So who cares which of those paths determines B? Well what happens to data if speed signalling delays its transmission on the repeated port? It accumulates in the Resync buffer. If the resync buffer is too full at the start of the packet, and clock tolerance works against you, it can overflow before packet end. Oops!

A third path may determine point B. The portion of data prefix which is prior to speed signalling is a different length on the receiving port and the repeating port. The length on the repeating port is reduced by ARB_RESPONSE_DELAY time, and increased by SPEED_SIGNAL_DETECTED time. If this turns out to be a net reduction, then TOTAL_PREFIX_TIME may determine point B. This is probably not likely, and in any case its impact is identical to the previous case.

Isssue #2 :

The second issue deals with idle gap reduction, and with Gap Count tables. The fact is that the gap period will be reduced through each repeating PHY. The arbitration and transmit start sequences introduce additional gap period called ARB_DELAY to compensate for this. One intent of the ARB_DELAY period is to ensure that even with repeaters' gap reduction, gaps are never reduced below SubActionGap length. ARB_DELAY is defined as (4 * GapCount * BasePeriod). The current GapCount tables do not allow sufficient time for this. I remember some reflector discussion on erros in the GapCount tables, but the proposed table revisions did not account for the effect described below.

No matter which path determines point B, once it is determined, a delay between the incoming and repeating port is established. To keep things simple lets count this delay in base clocks, rather than time so we don't have to worry about clock tolerance between nodes. Since the data, dribble bits, and data end intervals will have the same number of clocks on both ports, this delay will also characterize the relationship between the start of the IDLE periods on both ports. On the receiving port the start of the next packet can occur only after a minimum gap of SUBACTION_GAP plus ARB_DELAY. When it occurs, it will be reflected on the repeating ports after ARB_RESPONSE_DELAY. Therefore, the gap length actually seen on the repeating port, which we wish to always exceed a SubActionGap producesd the following equation which is then reduced.

SUBACTION_GAP + ARB_DELAY + ARB_RESPONSE_DELAY - delay > SUBACTION_GAP

ARB_DELAY + ARB_RESPONSE_DELAY - delay > 0

ARB_DELAY > delay - ARB_RESPONSE_DELAY

ARB_RESPONSE_DELAY is speced to be greater than 33 ns and less than delay (so right side never goes negative). Using PHY_DELAY (60 ns - 144 ns) for delay gives a range of values for the right side of this equation which is from (144 -33) to (delay - delay) or 111 ns to 0 ns. For the equation to be true we must use the maximum of these. Substituting for ARB_DELAY and reducing provides an ultimate restriction on GapCount.

ARB_DELAY > 111 ns

4 * GapCount * 10.17 > 111 ns

GapCount > 111 ns / 40.68 ns = 2.73

GapCount must always be at least 3, even with a hop count of 1. In fact, the above analysis convinces me that it must be at least (2.73 * H) where H is the number of hops. New style optimizing bus managers may use ping timing to set gap counts, which is good. However, old style bus managers may still be out there. If they use the table as is, problems may result.

Please note: The 2.73 multiplier is based upon using PHY_DELAY for delay. We already noted that considerations other than resync delays may establish delay. Are these other considerations covered under the definition of PHY_DELAY as "Worst-case repeater delay".?