PHY & Link designers:

In summary, I have a couple of scenarios which, based on my understanding, challenge whether CycleSync LREQ's really are sufficient to prevent Cycle Start starvation. I need some help correcting my scenarios or fixing the problem.

Fundamentals

If left unchecked, the use of ack-accelerated and fly-by arbitration can theoretically lead to arbitrarily long intervals (much greater than 125 usec) during which the root node never sees a full subaction gap. This can be particularly troublesome since Cycle Start packets can then be theoretically starved for as long as a fairness interval (especially if the root is a 1995 PHY).

The *fundamental* and sure fix is to make sure the aforementioned enhancements are not used between the time a cycle start is expected and the time the actual cycle start is received. Bill Duckwall's 1/3/97 "P1394A Enhancements" paper implemented this fix by introducing two new bus requests: AccPriReq and AccFairReq. It is my understanding that the link could use these requests in place of the corresponding PriReq and FairReq whenever accelerations where to be attempted. Consequently, these bus requests would *not* be issued between the time a cycle start is expected and the same cycle start is actually received by the link.

Simplification Offered: CycleSync LREQ

Rather than using two LREQ's to control use of the new enhancements, the P1394a specification defined the CycleSync LREQ to prevent Cycle Start starvation. Basically, a compliant non-root link issues the CycleSync LREQ whenever a cycle start packet is expected (determined by a rollover in the count field of the CycleTimer). Upon receiving the CycleSync LREQ, a compliant PHY refrains from using either ack-accelerated or fly-by arbitration until the next subaction gap appears on the bus.

As defined, the CycleSync LREQ prevents Cycle Start starvation as advertised provided that the root link is *always* able to queue the cycle start arbitration request in the root PHY *before* the end of the first subaction gap following the first cycleSync LREQ generated anywhere on the bus. If the root PHY is not yet ready to arbitrate to send a CycleStart by the time the first post-cycleSync subaction gap ends, there is no guarantee that another arbitration opportunity will occur within the next 125 uSec period.

Questionable Scenarios

The following scenarios (which need to be thoroughly validated) illustrate cases in which the root PHY may be unable to arbitrate to send a cycleStart packet until after the first subaction gap ends. If the scenarios are valid, the CycleSync LREQ as proposed doesn't absolutely prevent starvation. Pictures would be nice ... but text will have to do.

Scenario A

The root node consists of an legacy 1995 link and a legacy 1995 PHY.

1) Broadcast or ACK packet is being received into the link.
2) When the PHY drives the PHY/Link interface to IDLE (corresponding to DATA_END on the bus), the LINK issues a PriReq with a speed code of S200.
3) The 1394 bus goes to IDLE after DATA_END.
4) Internal CycleTimer's across the bus roll over (internal CycleSync).
5) All P1394a Links send CycleSync LREQ's to attached P1394a PHY's.
6) P1394a PHY's disable enhancements.
7) The 1394 IDLE bus state grows to a full subaction gap.
8) P1394a PHY's enable enhancements.
9) An ARB_DELAY after the subaction gap, the root PHY grants the root LINK the bus for transmission at S200.
10) Need some help here. Since the Cycle Start packet can't always be sent at S200, the link either needs to release the bus or go ahead and send the original S200 packet which caused the bus request. For this scenario, I will assume some legacy links send the S200 primary packet (not a cycle start).

11) The destination node of the S200 packet returns an ACK.
12) Seeing the ACK, P1394a PHY's begin concatenations or early arbitration. Once enhanced arbitration begins, the root may not see another subaction gap until the end of the fairness interval. Cycle Starts are possibly starved.

If, rather than sending the S200 packet, the link had released the bus and issued another LREQ immediately, the root would have been able to send the Cycle Start after the second subaction gap (which is guaranteed to occur if no ACK was issued after the first gap). Which way do legacy links operate?

Scenario B

The root node consists of a P1394a link and a P1394a PHY.

1) Root issues a FairReq and subsequently sends an asynchronous packet on the bus. This clears the arb_enable bit in the root PHY.
2) Sometime before the next fairness interval, a primary packet is being received into the link.
3) When the PHY drives the PHY/Link interface to IDLE (corresponding to DATA_END on the bus), the LINK issues a FairReq with a speed code of S200.
4) Soon after the FairReq, internal cycleSync events happen across the bus and in the root link. P1394a cycle slaves begin driving cycleSync LREQ's to the attached P1394a PHY's. These non-root PHY's disable all arbitration enhancements.
5) An ACK packet is received into the root link in response to the primary packet received in step 2 above. Because it is an ACK packet, the root's FairReq is *not* canceled.
6) Subsequent to the ACK, all non-root PHY's will not attempt arbitration until a subaction gap + ARB_DELAY expires since all enhancements are disabled. The root PHY, which has a pending FairReq, doesn't attempt arbitration either since the arb_enable bit is cleared.
7) As a consequence, a subaction gap appears on the bus and all P1394a PHY's enable enhancements. A non-root PHY will win the next arbitration and concatenations/accelerations continue for the next 125 uSec, causing the cycle start to be starved.

Some Possible Fixes (If required)

I feel pretty strongly that too much time is being spent trying to validate all of the corner cases for this CycleSync LREQ. Rather than apply another band-aid here, I'd recommend going back to the fundamental problem and solution. Have links turn off enhancements when a cycle start is expected. Have links turn enhancements back on after the expected cycle start arrives. This solution is trivial to validate and could be implemented several ways: Bill Duckwall's original two LREQ proposal, two different LREQ's which say enhancement on or enhancement off, or have the link directly write the enable_accel bit in the PHY.

If however, we'd like to apply some more band-aids:

— Have the cycleSync LREQ disable enhancements for two subaction gaps, not just one.

— Note that starvation only occurs for 125 uSec, worst case. (The link will persistently request the bus to send the late Cycle Start and the next subaction gap is guaranteed to happen with 125 uSec.) Worst case theoretical operation is one cycleStart gets through every 250 uSec. Statistically, this may be a rare event anyway. - Require links to anticipate the cycle sync and refrain from sending any other type of LREQ too close to the sync event.

Jerry Hauck Intel Corp
(408) 765-5528
jerry_hauck@ccm.sc.intel.com