

**Proposed Agenda
P1394b Working Group
March 23, 1999**

- 1. Review/Approve Agenda**
 - 2. Review of February Meeting Minutes**
 - 3. Procedures**
 - 3.1. Voting**
 - 3.2. Price/Pricing**
 - 3.3. Call for Patents**
 - 4. Meeting Schedule**
 - 4.1. April 29, San Jose (Sony)**
 - 4.2. May - NEED INVITATIONS!!!**
 - 5. Presentations**
 - 5.1. Cross Over [Ed McDonnell]**
 - 5.2. B.O.S.S. Updates[Mike Teener]**
 - 5.3. Upstarts/UTO [Colin Whitby-Strevens -as Required]**
 - 5.4. Copperheads [Max Bassler]**
 - 5.5. Standby [Steve]**
 - 5.6. PHY-Link [Tony Foster]**
 - 5.7. Electricals [Eric Hannah]**
 - 6. SCAT List**
 - 7. New Business**
 - 8. Review of Action Items**
 - 9. Adjournment**
-
-

MEETING MINUTES

Review of agenda and call for agenda additions.

Accept previous meeting minutes: John Fuller move to accept, seconded.
Passed without objection.

Voting, price/pricing, Call for Patents - all presented in the usual, professional manner by the group chair.

Review of Action Items from February & December meeting:

AR: David will generate a letter and have it delivered via registered postal service to Apple for the purpose of soliciting their position on any Apple intellectual property which may be contained within the current IEEE P1394b draft specification. 02/09/99 Status: David wrote and sent (unregistered) a letter to Apple. A response was received. The response is compliant with IEEE terms (e.g. "reasonable and nondiscriminatory"). A second "short" letter was sent to Apple. The nature of the short letter was not disclosed. A response for the second letter has NOT yet been received. This is closed as the result of the 1394 IP patent pool.

AR: Steve Bard to develop the SCAT list into a table and include perspiration and inspiration fields as well as status, dispensation and ownership fields. This table will be posted to the ftp repository. 03/22/99 Status: SCAT has been posted on ftp site. Soft copy on diskette being distributed today.

Meeting schedule reviewed.

Status from John Fuller (Microsoft possible host for meeting the second or third week in May or June). Status: Gerald Marazas (IBM) not able to host a meeting in May or June.

David announced an "after plenary" meeting for Loop healers (e.g. folks interested in discussing various methods of preventing loops).

Reports & Presentations

Auto-Crossover in 802.3ab: [Ed McDonnell]

- Automatic configuration eliminates need for crossover cables
- currently for 1000Baste-T; planned for future 10/100 products
- A node waits a specified time while evaluating its receive channel to determine if other end is sending link pulses or PHY-dependent data. if so then retains configuration otherwise decides to switch based on LFSR

A state machine of the Auto crossover state machine was shown and described.

EOR (end of report)

A discussion was led by Colin Whitby-Strevens regarding cross over. Bi-directional communication occurs on a single wire. There is an issue if, one two wires, both receivers determine at the same time that there has been a crossover (data coming in on "C" is really the data expected on "B" and vice versa) the algorithm could permit a double corrections – yielding the same original problem. Input from VESA home wiring group states that home wiring shall be a star implementation where all wiring in a room and wiring from the room to the wiring closet shall all be straight through ($1 \leftarrow \rightarrow 1$). Crossovers will occur at the patch panel in the wiring closet.

UTP has not determined if this is something which should be done or not. In order to decide, UTP needs a better feel for what the cost might be from the silicon folks. If the cost is zero, the decision is easy, but, if the cost is \$5.00 per chip (say) then the decision still will be easy. In fact, the information will be a bit different and some consideration of the trade-offs must be made prior to reaching a decision.

Based upon input from DC Sessions, it looks like Upstarts will include it in the draft provisionally based upon the "scream" test (e.g. it becomes part of the draft specification unless some one screams loudly that it costs too much). No objections from the group to provisionally adopting this for UTP use.

Copperheads: [Max Bassler]

Their agenda:

Review minutes of February 9th meeting

Presentations:

- 1) Test Data of enhanced 6 circuit
- 2) Beta only proposal

Review of action items

Open action items:

EMI/RFI test method – Report in April Chuck Brill

Cable detect pin for enhanced 6 circuit I/O – open pending plenary vote on S800 only 6 circuit enhanced

Enhanced 6 circuit I/O – test data post presentation; next meeting proposed wording; new 4.7 drawing option

Revise next values in the draft – Colin to advise – recommend 6 circuit enhanced next to be 5% at 250 ps rise time

NEW: Beta only connector proposal; post to web for review and comment

Verify S800 operation with cable + enhanced 6 circuit – Dave Bruner

EOR (end of report)

David Wooten presented the speed/connector developed at yesterday's SCAT meeting and reviewed the data on the foil with the group.

The last row in the matrix was eliminated (e.g. Beta only where beta to bilingual not possible). Max queried the group and sought concurrence that it be mandatory that the PHY support S800 beta mode. Max asked if there were any objections to making it mandatory that a PHY connected to an enhanced 6 circuit socket be required to support S800 – none were voiced.

David Wooten asked the opinion of the group as to whether the Copperheads task group should continue an effort in exploring cable/signal detect and a new enhanced cable assembly to support S1600 copper connections.

John Fuller moved to remove the requirement that there be a plug compatible solution for signaling rates above S800 on copper 6-pin circuit cable assemblies. Seconded by Max Bassler. An accepted friendly amendment by Colin Whitby-Strevens to add “1394a” as a distinguishing identifier to “copper 6-pin circuit cable assemblies”. New motion reads: “Remove the requirement that there be a plug compatible solution for signaling rates above S800 on copper 6-pin 1394a circuit cable assemblies.”

In Favor: 23 Opposed: 1 Abstained: 8

The reason stated for the one opposition to the motion was the implication of yet another 1394b connector. Gerald Marazas expressed awe at the complexity that has been brought to 1394b with the current number of connectors (among them: RJ45 for UTP, POF, GOF, 4-pin, 6-pin, bilingual and beta-only).

1394b Link-PHY Targets [Tony Foster]:

- 9 inch trace length between link and PHY
- 200 Mhz clock (data on both edges)
- 8 data, 6 control Bi-directional (not all are bi-direction)

- 1 clock each direction
- 200 MHz clock for all speeds if S1600 capable
- Process voltages compliance: +3.3, 2.5V 1.8 V, 1.5 V

STRATEGY:

- Accelerate definition
- Leverage the 802.3Z GMII specification

Link-PHY Challenges:

- Signals use a transmission line model
- Transit time on 9 inch traces
 - protocol impact of latency
- Signal skew
- Rise times
- Isolation in bilingual implementations

Transmission line model:

- Rise times at 200 Mhz
- Reflections and terminations
- 802.3Z specification
 - specify the signal/bus characteristics at receiver
 - measurement fixtures and loads

Bilingual issues:

- Capacitive isolation differentiates signal
 - creates higher frequencies on traces
 - causes spikes that are more difficult for process voltage compliance
 - signal droop on signal lines with 32:1 frequency range
 - charge in one clock cycle

- Isolation hampers Link-PHY integration

Electric Spec status:

- Actual text not started
- Obtaining a copy of the 802.3Z spec to edit
 - This model questionable for Bilingual operation
- Link and PHY pad definition?
- PC board materials and design?
 - Trace capacitance/impedance

PC Trace Delay Calculation:

- Rule of thumb
 - Strip line FR4/G10 2.26 ns/ft
 - Microstrip FR4/G10 1.77ns/ft
- Website for reference:
 - <http://www.polarinst.com> (demo CITS25)
- Long traces would require termination loads (50 ohm?)

Electrical Scope Questions:

- Link and PHY Pad definitions?
- PC board materials design [*ed. Note: lost slide – more material here not documented In minutes*]

Specification Process:

- Staffing shortage
 - work on the beta mode specification first
- Many tasks not allocated or covered
- IC process compliance Link-PHY
- Protocol review with the latency
 - multiple transitions in transit simultaneously

- Link-PHY spec in bilingual operation

IC Process Compatibility:

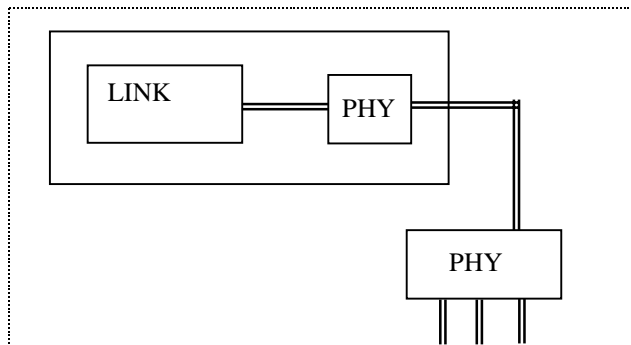
- Focus on adjacent IC process compliance
- The higher voltage process should target to comply with the lower voltage future process
- Possible implementations include
 - Pad reference for compliance

S3200 Timing Budget:

- Considered idea like PCI to minimize power consumption
- Settled on transmission line model
 - 9 in == 1.3 ns to 1.7 ns
 - Data cycle == 2.5 ns (1/400e6)
 - Rise time – 25% (1/400e6) = .635ns
 - Slew rate for 10-90% = $3.3V \cdot .8 / .625ns = 4.2V/ns$

EOR (end of report)

David Wooten suggested a possible implementation:



The link and PHY connected together in the solid box are integrated into a single piece of silicon. The integrated PHY is a Beta-only PHY. The four port PHY is a bilingual PHY. The whole thing would be in the “box” (whether that box is a PC or some other piece of equipment).

Tony suggested this concept should be posted on the reflector for further discussion.

The external PHY/Link interface serves no useful purpose. It makes more sense to have the PHY/Link integrated and expose a beta-only interface. A "PHY" would then simply become a "fan-out" device to add additional ports, provide bilingual capability, etc.

B.O.S.S. Updates [Mike Teener]

Updated changes to BOSS

(with contributions by Jerry Hauck, Dave LaFollette and David Wooten)
(Version 0.85, revised 03/22/99)

Changes include:

For 0.85

1. Added "no_request" options for asynch and isoch (thanks, Alistair).
2. Added no_request_odd/even pair as a way to avoid limiting the fairness cycles to subaction gap times (trying to avoid using specified gap times). (Thanks, Mr. Wooten.)
3. Lowered priority for out-of-phase asynch requests below the out-of-phase "no-request". This allows usto use the out-of-phase "no_request" to keep the fairness cycle from advancing. Useful for both border nodes and to remove the need for explicit minimum times for fairness intervals.

For 0.8

4. Hybrid of current/next and phase_0/1 for asynch arbitration. This hybrid allows pipelining of requests back to the link with minimal race conditions.
5. Various notes and improved explanations added. Some inconsistencies removed.
6. Resource required tables added.
7. Summary of BOSS arbitration added.
8. Simplification of border functionality (thanks Jerry!).

Resources required:

Tokens

The following is a list of tokens needed by the BOSS scheme and associated border functionality. Note that all tokens are transmitted on all ports pointing "away" from the current BOSS. In all cases except "grant", the BOSS will transmit these tokens out all connected ports.

Table 1 - Tokens

Token name	Comment
Cycle_start_odd	Follows cycle start packet where the low order bit of the cycle count is set.
Cycle_start_even	Follows cycle start packet where the low order bit of the cycle count is clear.
Arb_reset_odd	Sent by BOSS when no current or next_even requests are received.
Arb_reset_even	Sent by BOSS when no current or next_odd requests are received.
Bus_reset	Forces a bus reset. The cycle start phase is unchanged (all new nodes must wait for a cycle start to arrive to determine the phase), the fairness phase will be set to "odd".
Asynch_start	Sent by BOSS if cycle start has been received, and no isochronous requests of the current phase are received.
Grant	Sent by BOSS to the highest priority request (note that "highest priority" is context sensitive ... see below).

Isoch cycle start

An isochronous cycle is started by an explicit token (an arbitration code or control symbol of "cycle_start_even" or "cycle_start_odd" corresponding to the two different types of isoch requests), pipelining requires this. This means that a cycle start packet will look like:

...<DP><cycle start packet><DE><cycle start token(s)>...

(the cycle start token follows the packet to ease border functionality)

Arbitration reset

There need to be two types of arbitration resets to handle the queuing of fairness cycles.

Grants

There is a single grant token. I thought about having separate grant types for each request (for robustness), but the error modes were too complicated. Instead, there is a separate "asynch_start" token that is used by the BOSS to indicate then end of isochronous transmission. This is used to clear out any late transmissions of isochronous requests of the current phase.

Similarly, there is no "phase" in a grant. If a node doesn't correctly receive an arbration reset token, then it is possible that it will make the wrong type of request, but this will be momentary and will only cause a single fairness error.

Cable request types

We need the following cable request types:

(notes:

- asynch and iosch have independent priority lists since both requests are carried in parallel
- these are cable request types, there will be fewer link request types)

Table 2 - Asynch requests

Request name	Priority level	Comment
Cycle_start_req	1 (highest)	
Border_High	2	Keeps the 1394a/b boundary protocol working correctly, done at very high priority to handle legacy PHYs that have short request timeouts.
Next_odd	3 if the last arbitration reset token was arb_reset_odd, else 6	This is a queued request from the previous fairness cycle
Current	4	Used for all normal asynch requests by nodes that haven't used up their fairness budget.
No_req_even	5 if the last arbitration reset token was arb_reset_odd, else 7	Nodes that last saw an arb_reset_even send this request when they have no requests ... having a higher priority than the out-of-phase request keeps those the fairness cycle from advancing until all nodes receive the arb_reset of the current phase. There are two uses: 1) removes the need for programming a specific minimum fairness interval, and 2) allows a border node to synchronize the fairness cycles of the beta and legacy domains.
Next_even	6 if the last arbitration reset token was arb_reset_odd,	For queuing requests across the next fairness cycle

Request name	Priority level	Comment
	else 3	
No_req_odd	7 (lowest) if the last arbitration reset token was arb_reset_odd, else 5	

Table 3 - Isoch requests

Request name	Priority level	Comment
Isoch_odd,	1 (highest) if the last cycle_start was cycle_start_odd, else 2	Used if the last cycle start token was cycle_start_odd and the packet is intended to transmit in the current cycle. If the last cycle start token was cycle_start_even, this request is used if the packet is intended to transmit in the next cycle (which should be an even cycle).
Isoch_even.	2 if the last cycle_start was cycle_start_odd, else 1	Used if the last cycle start token was cycle_start_even and the packet is intended to transmit in the current cycle. If the last cycle start token was cycle_start_odd, this request is used if the packet is intended to transmit in the next cycle (which should be an odd cycle).
No_isoch_req	3 (lowest)	

Beta Bus Operation (normal BOSS mode)

All 1394beta PHYs have two modes of operation: BOSS (temporary bus master) and non-BOSS. A node becomes BOSS if is the last to transmit during a subaction: i.e., it transmitted the "ack" in a directed subaction (perhaps the "ack" to a concatenated response), the data packet in a broadcast subaction or some PHY packets. At any one time there is at most one BOSS node. During normal operation there are short periods of time when no node is BOSS. There are recovery algorithms to handle error conditions of no BOSS for too long, or multiple BOSSes.

The general rule is that if a port is not transmitting data, then it continuously transmits the highest priority request coming from any of the other ports or an attached link. The result is that each port not receiving data is receiving a continuously updated "snapshot" of the highest priority request sent by any nodes on that port's subtree. In the case of the BOSS, all ports are receiving requests so it should send a grant to the highest priority port (or its own link) immediately after finishing sending the ack or broadcast packet. If a BOSS doesn't receive any requests by the time it finishes transmitting data, then it sends an arbitration reset token of the appropriate phase (see next paragraph) and a grant to its parent. The root node ends up being the default BOSS in an idle bus.

Fairness is implemented by using a two priority/two phase approach. As long as a node has its "arb_enable" bit set, it will transmit "current" requests, if its arb_enable bit is reset, it will transmit a lower priority "next_odd" or "next_even" request, depending on whether the last arbitration reset token was "arb_reset_even" or "arb_reset_odd". If it has no requests, it sends a "no_request_odd" or "no_request_even" depending on the last arbitration reset token received. If the BOSS last received an "arb_reset_even" token, it will issue grants first to "next_even" requests (left over from the last fairness cycle), then to "current" requests. If it only sees "next_odd" or "no_request_even" requests, it shall transmit an "arb_reset_odd", then a grant to one of the "next_odd" requests.

The "no_request" of the opposite phase has a higher priority than the "next" of the opposite phase to keep the fairness cycle from advancing until all the nodes have received the current phase arbitration reset.

This process repeats for the "even" fairness cycle, with the "*-odd" and "*-even" signals inverted. Note that the priority of the "next_x" requests with respect to the "current" request inverts on each cycle.

Isochronous arbitration is similar to fairness, although there is no "current" request. When a node receives a cycle_start_odd, then current BOSS shall only respond to isoch_odd requests, and isoch_odd requests have priority over isoch_even requests for non_BOSS nodes repeating requests. The cycle_start_odd token shall be sent by a cycle master immediately after it transmits a cycle start with an odd-numbered cycle count.

Once again, this process inverts for cycle_start_even.

If two operating busses are joined together, then the one that does not have the new cycle master may receive two consecutive cycle starts of the same phase. This means that there may need to be a "cycle start inconsistent" notification.

Hybrid Bus Operation

Attributes:

- Almost all complexity is hidden in a border PHY (a PHY that has at least one port that may be connected in DS mode and at least one in Beta mode, also any PHY that supports 1394-1995 or 1394a Links will also need border functionality),
- Border functionality is only active when needed (i.e., when all ports in DS mode, then only -1995 or - a functionality is used; when all ports are in beta mode and link is also beta-capable, then only beta functionality is used).
- Allows considerable overlap between DS-mode arbitration overhead and beta data traffic in the most likely topology (the most important "beta cloud" is where the root is located) (a "beta cloud" is a subnet of the bus that only has beta connections active)
- Works for all topologies

Summary description, operation examined from the point of view of a border PHY.

Case 1: asynch arbitration with the root on the beta side

For a border phy with the root on a beta port, we normally do **not** put DP onto the DS ports if there is traffic on the beta side, instead we allow the DS side to arbitrate in parallel with data transmission on the beta side. In the abstract, we could just accept a request from the DS side, and forward requests from a DS port to the beta cloud using the special Border_High request, which will always win over normal beta requests. Unfortunately, old Apple Firewire PHYs will timeout after sending request for 20 usec and force a bus reset, so we must make sure that all outstanding requests get some kind of response in a timely way.

1. Request received on DS port
2. Request is forwarded to beta port(s) using Border_High priority and a 10 usec timer is started (to make sure no node on the DS side will timeout on a request)

Note: If beta side is not busy (not sending or receiving a packet), normal beta-mode protocols guarantee that a grant or a new packet will arrive within 10 usec (this timing is needed to keep an old Firewire PHY from timing out on its request)

3. While timer < 10 μ sec
 - 3.1. if a legacy packet arrives (one that can be sent out the DS port), then it is forwarded to the requesting DS port and the timer is canceled (the DP signal causes the request to be withdrawn; will happen with 10 usec, so Firewire PHY will be happy).
 - 3.2. If a beta packet arrives, continue sending idle on DS port(s) and continue 10 μ sec timer started in (2) above
 - 3.3. If the grant arrives, forward the grant to the DS cloud and cancel the timer.
4. if 10 usec timer expires, DP is transmitted on the DS port(s)

- 4.1. if a legacy packet arrives (one that can be sent out a DS port), then it is forwarded to the requesting DS port (the DP started in (4) is the data prefix for this packet).
- 4.2. If a beta packet arrives, continue sending DP on DS port(s)/
- 4.3. If the grant arrives, release the DP sent to the DS cloud and allow any attached nodes to re-arbitrate. Simultaneously transmit DP on the beta side to keep the root from timing out and assuming BOSS function. Start a timer.
- 4.4. While timer < subaction gap
 - 4.4.1. If the timer expires, release DP from the beta side
 - 4.4.2. If a request arrives from the DS side, respond with a grant, restart timer
- 4.5. While timer < state timeout
 - 4.5.1. If the timer expires do a bus reset (just like 1394a)
 - 4.5.2. If packet arrives from DS cloud, retransmit to beta side.

Case 2: Fairness with the root on the beta side:

To keep fairness working as expected, once a fairness cycle has started on one side, you don't want a new fairness cycle to finish on the other side. For this discussion, a fairness cycle starts on the first packet after an arbitration reset and finishes when an arbitration reset gap (for DS ports) or arbitration reset token (for beta ports) is received.

So, the requirement is to hold off the ending of a fairness cycle on one side until it can finish on the other. The basic idea is as follows:

If a fairness cycle must be extended on a DS-port, it is done by sending a DP on the DS port (a border node can detect the imminent occurrence of an arbitration reset gap on the DS side by some bus management magic, see below). If a legacy packet arrives on the beta side, it is repeated on the DS side (beta-side packets look to the DS side like a series of concatenated packets). This process is continued until an arbitration reset token is received (or generated) on the beta side.

Bus management magic: for all this to work, the arbitration reset gap on the DS side must be set to a "somewhat larger" value than normal (I think!). The border nodes would use a smaller value to guarantee that they detect when the arbitration reset gap is about to happen and take the appropriate action. If the fairness cycle must be extended on the beta cloud, the border node sends a Border_Low request, and holds the beta side (sending DP) until an arbitration reset gap appears on the DS side.

Case 3: Isoch operation with the root (cycle master) on the beta side:

The one concern here is that the isoch cycle on the DS side must not end until the isoch cycle is really done. This works almost the same way that the fairness cycle is extended: the border node detects the imminent subaction gap at the end of the DS-side isoch data, and keeps it from happening by sending DP. Note that this requires the bus management magic described above to set the gap count to a slightly larger value than normal.

It **continues to be** a requirement that we send cycle starts at S100, unless we can determine via some higher level mechanism that the minimum speed link through the network is higher.

Case 4: Normal arbitration with the root on the DS side:

The major concern here is that the beta cloud must follow the arbitration acceleration exclusion rules of 1394a ... i.e., the beta cloud cannot hold the bus too long after a cycle synch indication, it must allow the DS root to send a cycle start in the proper time. To do this, the border node must attempt to gain control of the beta cloud at cycle synch time. The "Border_High" priority level will allow it to do this, but this also requires that the border node have an active attached link. If the border node does not have an attached active link, it must send a Border_High request after every packet to pass control back to the DS port.

Case 5: Fairness with the root on the DS side:

Because of the nature of this configuration, there will always be an arbitration reset token on the beta cloud before one can appear on the DS side. This means that the only problem is to hold off the end of a new fairness cycle on the beta side if a fairness cycle has started on the DS side. This is done by the border node sending a Border_Low request whenever there is an arbitration reset token on the beta side. The border node will get a grant before the next arbitration reset token is sent on the beta side, and will send DP until it receives an arbitration reset gap on the DS side, at which point it will send the appropriate arbitration reset token itself.

Case 6: Isochronous with the root on the DS side:

There is no "cycle_start_even/odd" indicator on the DS side, it must be synthesized by one of the nodes on the beta side. Since PHYs cannot decode a cycle-start packet, this must be done by a link. To avoid too much complication in the various state machines, we would like a link to **always** send a message to its attached PHY saying something like "cycle_start_even/odd" received, and the PHY will synthesize the appropriate cycle start token.

***Exercise for the reader:** This means that it will be possible for nodes to receive multiple cycle start tokens of the same phase. We could suppress this by requiring that nodes do not repeat cycle start tokens of the same phase as the last one received. This may cause some problems (or may remove some!) when two operating busses are connected together.*

Note that there is not a concern with asynchronous packets slipping in after the cycle start is repeated into the beta cloud ... the border node always retains BOSS function during this period (see case 4).

Stand-by [Steve Bard]:

Standby is a term used to describe a low energy consumption mode of operation for a node with only one active port. A node in Standby does not participate in normal bus activity.

Characteristics of a Standby candidate Child Node

- A feature of Beta-mode operation only
- A beta-only leaf node does not have to implement Suspend/Resume provided it responds to Suspend symbol (0xx00101) by entering into Standby
- A bus reset does NOT occur as part of entering or restoring from Standby
- The active bus for which the Standby child node is a member of is not aware of any status change of the Standby child node (its parent node shall "proxy" the self-ID packet subsequent to any bus reset on the active bus)
- When a Standby child node restores from Standby, the latency from Standby to fully active is 3 milliseconds (+/- 5%.)
- A restoring Standby child node becomes active AFTER receiving the following information from the first PHY packet it detects:
 - Node-ID,
 - gap count
 - gap count "sticky-bit" status
- Upon detecting its first PHY packet, the restoring child node transmits a PHY response confirmation packet to its parent PHY.
- A Standby child node will begin monitoring for a PHY packet after detecting a resume tone or after generating a resume tone to its parent node

- A multi-port node with one connected port in Standby shall restore that port followed by a bus reset if, on any of its other ports, it detects a resume or a new connection
- A child node will enter into Standby when it detects a Standby PHY Command Packet containing its Node-ID and port address

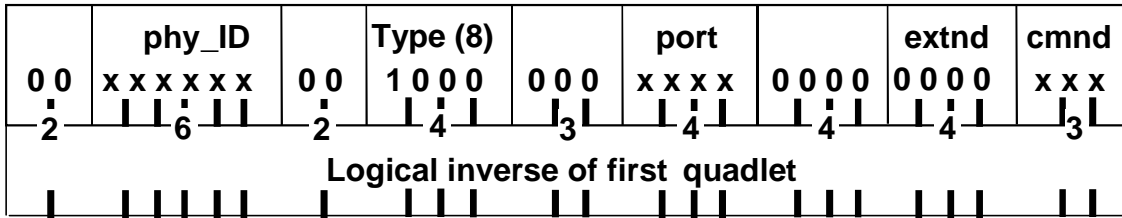
Characteristics of the Parent Node of a Leaf Node in Standby

- This is a feature of Beta mode operation only
- The parent PHY stores (for each port) a set of 12 bits from each self-ID sequence for which it will use to proxy the self-ID packet for its child port
 - Link_Active = 1 bit
 - Pwr_class = 3 bits
 - Parent_port_num = 4 bits
 - Highest_port_num = 4 bits
- When the parent node of a child port detects a Standby symbol (0xx00001), the parent node shall proxy the self-ID packet of the Standby child node when a bus reset occurs
- When either detecting or generating a resume tone from/to a Standby child node, the parent node will arbitrate for the bus after which it will generate a “Restore” NOP PHY command packet addressed to itself. The 17th bit through the 29th bit transmitted (bit 1 being the first bit transmitted) will contain the following data for the child:
 - Bit 17 = Gap Count “Sticky-Bit” status
 - Bits 18-23 = Child Node-ID
 - Bits 24-29 = Child Gap Count
- The parent node shall service only one restoring child leaf at a time
- A parent node with a Standby child node shall restore the child (assert a resume tone to the child) when it receives a Restore PHY command packet with the Node-ID of the parent node and the port address of the port connected to the Standby child node

PHY Packet Details

New PHY Command Seven (7) – Extended:

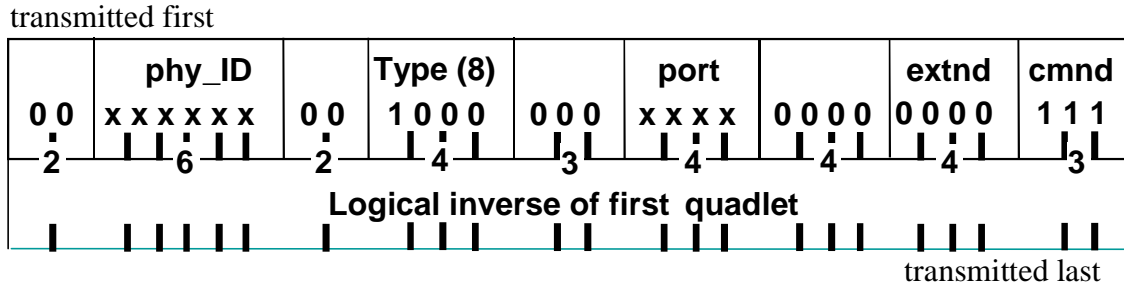
transmitted first



transmitted last

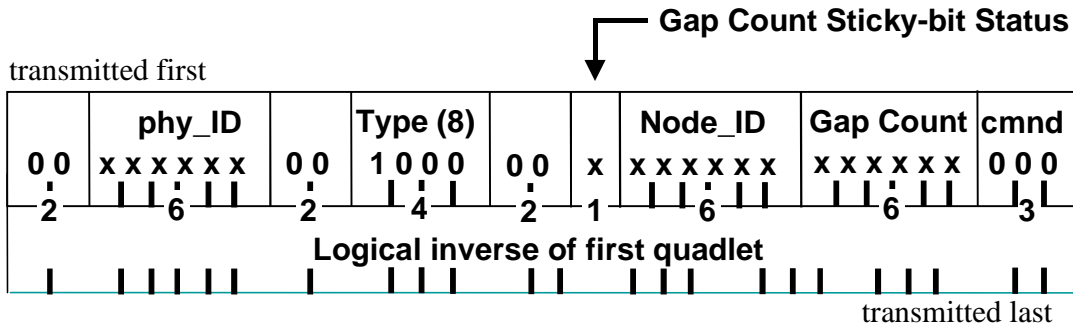
Field	Comment
phy_id	Physical node identifier of the destination of this packet
type	Extended PHY packet type (8 indicates command packet)
port	This field selects one of the PHY's ports
cmd	Command: 0 NOP 1 Transmit TX_DISABLE_NOTIFY then disable port 2 Initiate Suspend (become a suspend initiator) 3 Reserved 4 Clear the port's fault bit to zero 5 Enable port 6 Resume Port 7 Extended Command

Extended PHY Command Code:



Field	Comment
extnd	Command:
	0 NOP
	1 Initiate Standby with Connected Peer Port
	2 Restore from Standby with Connected Peer Port
	3-F Reserved

NOP PHY Command Packet containing "Restore" Data for Child Node:



CAT Simulations [Eric Hannah]

Eric's presentation is available on the web site for viewing/downloading.

The longest rise time determined by that value which will just trigger a worst case receiver's switching threshold under the highest case output voltage and cable length for the worst case signal, the clock waveform. The highest mV differential and the receiver's worst case sensitivity is 50 mV differential. Again the launch voltage is measured by generating a long series of 1's and 0's.

The clock waveform's fundamental has a frequency of 125 MHz/2=62.5 MHz, thus we can expect a cable loss of 18.4 db. Given the worst case system the worst case 62.5 MHz sine wave amplitude is 416 mV.

For the largest launch voltage of 525 mV the 80-20% voltage band is 315 mV. The smallest allowable clock waveform amplitude is 416 mV. Thus the logest

rise time is given by the time it takes this smallest clock waveform to pass through the 315 mV voltage band.

Jitter [Colin Whitby-Strevens]

S800 Copper shorthaul

UI (ns) 1017.3

Jitter output	ps				UI			
	DJ pk-pk	RJ RMS	RJ pk-pk	TJ pk-pk	DJ pk-pk	RJ RMS	RJ pk-pk	TJ pk-pk
TP1	102	8.7	122	224	0.10	0.009	0.120	0.22
TP1 to TP2	20	5.2	73	93	0.02	0.005	0.072	0.091
TP2	122	10.14	142	264	0.12	0.010	0.140	0.26
TP2 to TP3	264	8.4	118	382	0.26	0.008	0.116	0.376
TP3	386	13.17	184	570	0.38	0.013	0.181	0.56
TP3 to TP4	21	5.8	81	102	0.02	0.006	0.080	0.100
TP4	407	14.39	201	608	0.40	0.014	0.198	0.60

Notes:

Blue is the raw data, bold are the normative values, green are the values for the Spec

Jitter tolerance	ps					UI				
	DJ pk-pk	RJ RMS	RJ pk-pk	Sinusoidal pk-pk	TJ pk-pk	DJ pk-pk	RJ RMS	RJ pk-pk	Sinusoidal pk-pk	TJ pk-pk
TP1	102	8.7	122	102	326	0.10	0.009	0.120	0.100	0.32
TP2	122	10.14	142	102	366	0.12	0.010	0.140	0.100	0.36
TP3	386	13.17	184	102	672	0.38	0.013	0.181	0.100	0.66
TP4	407	14.39	201	102	710	0.40	0.014	0.198	0.100	0.70

S1600 Copper shorthaul

UI (ns) 508.65 1017.3

Jitter output	ps				UI			
	DJ pk-pk	RJ RMS	RJ pk-pk	TJ pk-pk	DJ pk-pk	RJ RMS	RJ pk-pk	TJ pk-pk
TP1	51	9	126	177	0.10	0.018	0.248	0.35
TP1 to TP2	15	1.5	21	36	0.03	0.003	0.041	0.071
TP2	66	9.12	128	194	0.13	0.018	0.252	0.38
TP2 to TP3	76	5.4	76	152	0.15	0.011	0.149	0.299
TP3	142	10.6	148	290	0.28	0.021	0.291	0.57
TP3 to TP4	15	1.5	21	36	0.03	0.003	0.041	0.071
TP4	157	10.71	150	307	0.31	0.021	0.295	0.60

Notes:

Blue is the raw data, bold are the normative values, green are the values for the Spec

Jitter tolerance	ps					UI				
	DJ pk-pk	RJ RMS	RJ pk-pk	Sinusoidal pk-pk	TJ pk-pk	DJ pk-pk	RJ RMS	RJ pk-pk	Sinusoidal pk-pk	TJ pk-pk
TP1	51	9	126	51	228	0.10	0.018	0.248	0.100	0.45
TP2	66	9.12	128	51	245	0.13	0.018	0.252	0.100	0.48
TP3	142	10.6	148	51	341	0.28	0.021	0.291	0.100	0.67
TP4	157	10.71	150	51	358	0.31	0.021	0.295	0.100	0.70

S800 MMF shorthaul

UI (ns) 1017.3

Jitter output	ps				UI			
	DJ pk-pk	RJ RMS	RJ pk-pk	TJ pk-pk	DJ pk-pk	RJ RMS	RJ pk-pk	TJ pk-pk
TP1	102	8.7	122	224	0.10	0.009	0.120	0.22
TP1 to TP2	102	7	98	200	0.10	0.007	0.096	0.197
TP2	204	11.17	156	360	0.20	0.011	0.153	0.35
TP2 to TP3	51	5	70	121	0.05	0.005	0.069	0.119
TP3	255	12.24	171	426	0.25	0.012	0.168	0.42
TP3 to TP4	172	5	70	243	0.17	0.005	0.069	0.239
TP4	428	13.22	185	613	0.42	0.013	0.182	0.60

Notes:
Blue is the raw data, bold are the normative values, green are the values for the Spec

Jitter tolerance	ps					UI				
	DJ pk-pk	RJ RMS	RJ pk-pk	Sinusoidal pk-pk	TJ pk-pk	DJ pk-pk	RJ RMS	RJ pk-pk	Sinusoidal pk-pk	TJ pk-pk
TP1	102	8.7	122	102	326	0.10	0.009	0.120	0.100	0.32
TP2	204	11.17	156	102	462	0.20	0.011	0.153	0.100	0.45
TP3	255	12.24	171	102	528	0.25	0.012	0.168	0.100	0.52
TP4	428	13.22	185	102	715	0.42	0.013	0.182	0.100	0.70

S1600 MMF shorthaul

UI (ns) 508.65 1017.3

Jitter output	ps				UI			
	DJ pk-pk	RJ RMS	RJ pk-pk	TJ pk-pk	DJ pk-pk	RJ RMS	RJ pk-pk	TJ pk-pk
TP1	51	9	126	177	0.10	0.018	0.248	0.35
TP1 to TP2	51	8.65	93	144	0.10	0.013	0.163	0.263
TP2	102	11.19	157	259	0.20	0.022	0.309	0.51
TP2 to TP3	25	4.35	61	96	0.05	0.009	0.120	0.169
TP3	127	12.01	168	295	0.25	0.024	0.330	0.58
TP3 to TP4	108	4.35	61	169	0.21	0.009	0.120	0.332
TP4	235	12.77	179	414	0.45	0.025	0.352	0.81

Notes:
Blue is the raw data, bold are the normative values, green are the values for the Spec

Jitter tolerance	ps					UI				
	DJ pk-pk	RJ RMS	RJ pk-pk	Sinusoidal pk-pk	TJ pk-pk	DJ pk-pk	RJ RMS	RJ pk-pk	Sinusoidal pk-pk	TJ pk-pk
TP1	51	9	126	51	228	0.10	0.018	0.248	0.100	0.45
TP2	102	11.19	157	51	310	0.20	0.022	0.309	0.100	0.61
TP3	127	12.01	168	51	346	0.25	0.024	0.330	0.100	0.68
TP4	235	12.77	179	51	465	0.45	0.025	0.352	0.100	0.91

Colin gave a fairly detailed background of jitter data and brought the following numbers to be placed on the web site for discussion for jitter numbers:

Output jitter requirements - comparisons

	S400 1394-1995 ps	S800 ps	S1600 ps
TP1	150	224	177
TP2	150	264	194
TP3	315	570	290
TP4	315	608	307

More Connector/Cable Discussion

Steve Bard proposed to David Wooten at the morning break that there only be ONE connector and that it be a beta only connector. David Wooten enhanced that proposal by suggesting the one new connector would have keying which would make it a bilingual connector.

Valid cables would be:

- A 4-pin ← → A 4-pin
- A 6-pin ← → A 6-pin
- A 4-pin ← → A 6-pin
- Beta ← → Beta
- A 4-pin ← → Bilingual
- A 6-pin ← → Bilingual

A beta plug will mate with a bilingual socket.

This would deprecate the need for an enhanced 6-pin circuit socket.

There was a side conversation suitable for inclusion in the minutes. Specifically, a platform containing a beta only socket may connect to “legacy” 1394 via a device (such as a power brick) which contains a “naked” bilingual PHY with two bilingual sockets (remembering that a bilingual socket is simply a keyed beta only socket). This may be an interesting product for “legacy” buses in that it would provide a source of cable power as well as providing interconnect to the beta only platform.

[Secretary's note: Steve Bard believes there is good value in a very small form factor beta only connector which does NOT provide power. Such a plug/socket could, in fact, be a smaller form factor than the existing 4-pin plug/socket and could be designed to be more robust as well as exhibit better EMI/RFI characteristics.]

ACTION ITEM LIST:

AI: Colin to announce provisional acceptance of crossover on the reflector. If no scream response, it will become part of the draft.

AI: Colin to review crossover proposal and evaluate implementation for upstarts.

AI: Tony Foster will post to reflector the PHY/Link discussion and Wooten's suggestion for an integrated PHY/Link.

AI: Colin to post jitter spreadsheet to web site and encourage discussion response on reflector.

AI: David Wooten will post to the reflector the beta mode connector (with bilingual keying) and socialize discussion.

Meeting adjourned 4:16 PM.

IEEE P1394b Working Group Plenary Attendees (March 23, 1999):

NAME	COMPANY	EMAIL	PHONE
Anderson, Bill	DDK	w.anderson@worldnet.att.net	408-980-8033
Bard, Steve	Intel	steve.bard@intel.com	503-264-2923
Bassler, Max	Molex	mbassler@molex.com	630-527-4490
Boucachard, Philippe	Canon CRF	boucachard@crf.canon.fr	81-3-5482-8269
Brill, Charles	AMP	cebrill@ix.netcom.com	717-533-1279
Brunker, Dave	Molex	dbrunker@molex.com	630-527-2622
Chen, Dao-Long	LSI Logic	dao-long.chen@lsil.com	970-223-5100 x5461
Coles, Alistair	HP	anc@hplb.hpl.hp.com	+44 117 922 8750
Datta, Tapas	Enthink	tapas@wipro.com	91-80-557-8227
Dorsey, Chris	ST Microelectronics	christopher.dorsey@st.com	972-466-7850
Farhoomand, Firooz	Panasonic	farhoomandf@panasonic.com	408-653-4059
Foster, Tony	Hewlett-Packard	tony_foster@hp.com	(916) 785-1092
Fuller, John	Microsoft	jfuller@microsoft.com	425-703-3863
Furuya, Nobuo	NEC	nobuo_furuya@el.nec.com	(408) 969-2479
Hannah, Eric	Intel	eric.hannah@intel.com	408-765-4441
Hauck, Jerry	Zayante, Inc.	jhauck@zayante.com	510-668-1006
Hill, John	AMP	jhill@amp.com	717-810-4651
James, David	Sony	dvj@alum.mit.edu	650-494-0926
Kelly, Campbell	ADS Technologies	kellyc@cadvision.com	403-255-0665
Killeen, Sean	SSL	sean.killeen@ssl.ie	+353 1 402 5700
Le, Thang	Hewlett-Packard	tl@rose.hp.com	(916) 785-4667
Lopata, John	Molex	jlopata@molex.com	(630) 579-4110
Ma, Te Khac	Lucent Technologies	tma@lucent.com	(610) 712-2195
Marazas, Gerald	IBM	marazas@us.ibm.com	919-543-6892
McDonnell, Edward	HP Labs	emcd@hplb.hpl.hp.com	117-922-8942 (UK)
Northey, Bill	FCI	northewa@bergelect.com	717-938-2119
Saito, Kyozo	Alps	kyozo_s@gw3.alps.co.jp	+81 229 23 5111

NAME	COMPANY	EMAIL	PHONE
Saito, Tomoki	NEC	saito@ccm.cl.nec.co.jp	+81 44 856 2082
Sessions, D. C.	VLSI	dc.sessions@vlsi.com	602-752-6545
Shergill, Robbie	National Semiconductor	robbie.shergill@nsc.com	408-721-7959
Skidmore, Jim	Texas Instruments	j_skidmore@ti.com	972-480-2094
Smith, David	Texas Instruments	desmith@ti.com	(972) 480-6345
Sroka, David	Philips	dave.sroka@abq.sc.philips.com	505-822-5070
Teener, Michael Johas	Zayante	mike@zayante.com	831-461-4901
Teng, Victoria	NEC	victoria_teng@el.nec.com	408-969-2861
Vonbank, Michael	3A Intern	mvonbank@3a.com	602-437-1751
Whitby-Strevens, Colin	Zayante	colin@zayante.com	831-461-4948
Yoshikatsu, Niwa	Sony	niwa@sm.sony.co.jp	+81 3 5448 4603