

## *P1394b combined SCAT and C Code bug list*

Done	229
Not yet done	5
Recommend Accept	90
Recommend Accept in principle	21
Recommend Parially Accept	1
Recommend Reject	20
<b>Total</b>	<b>366</b>

<b>ID</b> 140	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot		<b>Fixed in version</b>	
<b>Title</b>	Active nodes and ports			<b>Status</b> Done	
<b>C code files</b>		<b>Owner</b> CWS	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0	<b>Recirc Clause</b> 04.7
<b>Problem description</b>					
no such thing as an active node, only active ports.					
<b>Bug fix description</b>					

<b>ID</b> 141	<b>Balloter</b>	<b>How submitted</b> Post ballot		<b>Fixed in version</b>	
<b>Title</b>	Standby nodes and ports			<b>Status</b> Done	
<b>C code files</b>		<b>Owner</b> CWS	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0	<b>Recirc Clause</b> 04.7
<b>Problem description</b>					
Standby is the state of a port, not a node					
<b>Bug fix description</b>					

<b>ID</b> 323	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	TX electrical spec	<b>Owner</b> EH	<b>Status</b> Recommend Reject
<b>C code files</b>		<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0 <b>Recirc Clause</b> 05

***Problem description***

May I ask around my question about transmitter electrical specifications. According to pp102-103 of P1394b Draft standard 1.00 'Transmitter electrical specifications', PHY transmitter(cable driver) should have a capability to apply appropriate current to ground level (see Figure6-3). This means the cable driver must be current push-pull type, not open-drain from VDD level. As I mentioned before, the open-drain driver has better compatibility to PECL of optical link. Furthermore, many highspeed interface LSI adopt open-drain configuration in driver circuit. I know P1394b should be finalized as soon as possible and I do not want to disturb any finished-matter of the draft, but I would like to have your comment of 'Why the 1394b treats the open-drain driver as an outcast?'.

***Bug fix description***

Hello Yoshikawa-San

We've further carefully reviewed your comments, but after due consideration we think that the specification should be left as it is, for the following reasons:-

1) The interface is designed primarily to drive electrical cables. We have really, really, tight differential skew requirements. These are driven by EMI reduction issues. Furthermore, we have had to tighten up the common mode specification to

Common mode output impedance: < 55 ohms

Max. common mode voltage: 2.415 V

We believe that the current specification provides by far the most appropriate way of meeting this requirement.

2) We are not aware of any onerous problems in the task of interfacing a transmitter designed to our specifications to a PECL optical transceiver. For example, a typical transceiver is Infineon's V23818-K305-L17/L57  
(\*[http://www.infineon.com/cmc\\_upload/0/000/009/500/Multimode\\_1dot0625\\_GBd\\_3dot3V\\_LC\\_2x5.pdf](http://www.infineon.com/cmc_upload/0/000/009/500/Multimode_1dot0625_GBd_3dot3V_LC_2x5.pdf))

This datasheet provides appropriate example circuits. (Note, this does not imply any form of recommendation, this is just an example which we can identify). The precise circuit and RC values may well vary from PHY to PHY, but in principle the interface can be achieved with a low number of passive components.

We note that Fibre Channel and Gigabit ethernet have very similar transmitter specifications to those in 1394b, and that interfacing their PMD chips to optical transceivers is commonplace.

3) Given (2) above, we believe that there will be significant marketplace advantage in providing PHYs in high volume which can either interface directly to copper cables or can be interfaced by a small passive network to optical transceivers, and that this is the most appropriate form for the 1394b specification to take.

4) None of the above prevent a manufacturer making a PHY with an interface optimised explicitly for coupling to an optical connector - but, given that the 1394b specifications are normative at the connector, this interface would be proprietary.

With best wishes

Colin Whitby-Stevens

<b>ID</b> 142	<b>Balloter</b>	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	Max. rise/fall time	<b>Owner</b> EH	<b>Status</b> Done
<b>C code files</b>		<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0 <b>Recirc Clause</b> 06.2

***Problem description***

Table 6-1 in Draft 1.00 lists max rise time at S800 as 600 ps. This rise time, if used, won't allow the eyd diagram to be wide enough. Comparing to FibreChannel, this should be 400 ps.

***Bug fix description***

---

<b>ID</b> 216	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	max. rise/fall time		<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> EH	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0 <b>Recirc Clause</b> 06.2

**Problem description**

Table 6-1 in Draft 1.00 lists max rise/fall time at S800 as 600 ps. This rise time, if used, won't allow the eye diagram to be wide enough. Comparing to FibreChannel, this should be 400 ps. The rise/fall times for the other speeds should be similarly scaled.

**Bug fix description**

---

<b>ID</b> 248	<b>Balloter</b>	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	LC connector drawings		<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> Max.	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0 <b>Recirc Clause</b> 07

**Problem description**

AR: Max Bassler find connector reference drawings for LC connector for use in the GOF chapter.

**Bug fix description**

---

<b>ID</b> 249	<b>Balloter</b>	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	POF tables		<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> Nakamura-San	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0 <b>Recirc Clause</b> 08

**Problem description**

AR: Kazuki Nakamura (Mitsubishi): In the POF Chapter, table 8-2/8-3, etc. will be modified to add a column. There will be a column for POF S100 and POF S200. Create just one table (not two tables).

**Bug fix description**

---

<b>ID</b> 250	<b>Balloter</b>	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	Jitter budget for POF		<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> Nakamura-San	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0 <b>Recirc Clause</b> 08

**Problem description**

AR: Kazuki Nakamura (Mitsubishi): The POF chapter must delineate where the jitter budget is allocated between TP1, TP2, TP3 and TP4.

**Bug fix description**

---

<b>ID</b> 251	<b>Balloter</b>	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	New HPCF chapter		<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> Nakamura-San	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0 <b>Recirc Clause</b> 08

**Problem description**

AR: Kazuki Nakamura (Mitsubishi) create material for an HPCF chapter (including an HPCF connector that is NOT able to be mated with the POF connector that has industry support).

**Bug fix description**

---

<b>ID</b> 143	<i>Balloter</i>		<i>How submitted</i> Post ballot		<i>Fixed in version</i>
<i>Title</i>	misuse of 1394-1995 and 1394a			<i>Status</i> Done	
<i>C code files</i>		<i>Owner</i> EH		<i>SCAT</i> <input checked="" type="checkbox"/> <i>BRAT</i> 0	<i>Recirc Clause</i> 08.1

*Problem description*

This bullet is wrong, the PMD does not support either 1394-1995 or 1394a!

*Bug fix description*

<b>ID</b> 144	<i>Balloter</i>		<i>How submitted</i> Post ballot		<i>Fixed in version</i>
<i>Title</i>	Mis-use of the word Active			<i>Status</i> Done	
<i>C code files</i>		<i>Owner</i> CWS		<i>SCAT</i> <input checked="" type="checkbox"/> <i>BRAT</i> 0	<i>Recirc Clause</i> 08.1

*Problem description*

(in two places)

*Bug fix description*

<b>ID</b> 284	<i>Balloter</i>	CWS	<i>How submitted</i> Post ballot		<i>Fixed in version</i>
<i>Title</i>	Text for UTP crossover function			<i>Status</i> Done	
<i>C code files</i>		<i>Owner</i> MT		<i>SCAT</i> <input checked="" type="checkbox"/> <i>BRAT</i> 0	<i>Recirc Clause</i> 09

*Problem description*

is missing

*Bug fix description*

CWS to write a description and send to EH for inclusion in the UPT5 chapter. (done)

MT to put a crossference to this in the services descriptions (done)

<b>ID</b> 147	<i>Balloter</i>		<i>How submitted</i> Post ballot		<i>Fixed in version</i> D103
<i>Title</i>	Confusing state name			<i>Status</i> Done	
<i>C code files</i>		<i>Owner</i> CWS		<i>SCAT</i> <input checked="" type="checkbox"/> <i>BRAT</i> 0	<i>Recirc Clause</i> 10

*Problem description*

Not the opposite of "active"

*Bug fix description*

Replace "Inactive" by "Off", replace "active" by "on"

<b>ID</b> 187	<i>Balloter</i>	MS	<i>How submitted</i> Post ballot		<i>Fixed in version</i>
<i>Title</i>	spelling			<i>Status</i> Done	
<i>C code files</i>		<i>Owner</i> CWS		<i>SCAT</i> <input checked="" type="checkbox"/> <i>BRAT</i> 0	<i>Recirc Clause</i> 10

*Problem description*

Resyn1

*Bug fix description*

**ID** 145 **Balloter** **How submitted** Post ballot **Fixed in version**  
**Title** misuse of "active" and "inactive" **Status** Done  
**C code files** **Owner** CWS **SCAT**  **BRAT** 0 **Recirc Clause** 10.2.1  
**Problem description**

**Bug fix description**

Reworded

---

**ID** 253 **Balloter** MJT **How submitted** Post ballot **Fixed in version** D104  
**Title** ARBRST\_EVEN/\_ODD terminology **Status** Done  
**C code files** **Owner** CWS **SCAT**  **BRAT** 0 **Recirc Clause** 10.2.3

**Problem description**

change to ARB\_RESET\_EVEN/\_ODD for consistency with Legacy

**Bug fix description**

---

**ID** 211 **Balloter** DW **How submitted** Post ballot **Fixed in version**  
**Title** Clarification of scrambling **Status** Done  
**C code files** **Owner** CWS **SCAT**  **BRAT** 0 **Recirc Clause** 10.2.6.1.  
A

**Problem description**

What is the significance of the D28.0 codeword? There is no place in the spec or here where I can find anything that indicates that the D28.0 codeword has any reason to occur during training. There is no explanation of how a continuous stream of Training requests (00000000b) will ever be scrambled into 00111000b (D28.0) so that it can be turned into a K28.5. Also missing is a description of how the synchronization of the scrambler works (lsb of the transmit represents the lsb of the scrambler and the decoded word is supposed to be 0 so if you get something other than 0 you should change the bit, or something like that.)

**Bug fix description**

Extra sentence added to 10.2.6.1.4  
Note added to 10.2.10 on how the descrambling works.

---

**ID** 146 **Balloter** **How submitted** Post ballot **Fixed in version** D103  
**Title** State name **Status** Done  
**C code files** **Owner** CWS **SCAT**  **BRAT** 0 **Recirc Clause** 10.4.2

**Problem description**

Confusing State Name (it is not the opposite of "active")

**Bug fix description**

Replace "Inactive" by "Off", replace "active" by "on"

---

**ID** 200 **Balloter** **How submitted** Post ballot **Fixed in version**  
**Title** table 10-15 inappropriate **Status** Done  
**C code files** **Owner** CWS **SCAT**  **BRAT** 0 **Recirc Clause** 10.4.2

**Problem description**

Table 10-15 has very little to do with the port state machines

**Bug fix description**

Replace with a table which describes the variables used in the port state machines

**ID** 221    **Balloter** CWS                      **How submitted** Post ballot                      **Fixed in version** D106  
**Title**            PTX3:PTX1 transition    **Status** Done  
**C code files**    **Owner** CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 10.4.2

**Problem description**

I was wondering what good the PTX3:PTX1 transition does given P1394b as we know it today.

Specifically, we no longer attempt to transparently (to higher layers) resync like we once did. I believe the current intent is that once sync is lost, force a disconnect and bring the port connection up from scratch. Nominally, this happens as follows (I think) Local end of peer-to-peer link is in P2:Active, PTX3:Transmit, and PRX4:Receive. After the invalid\_count reaches 4 (5 consecutive errored codewords), sync\_error\_signal is set forcing PRX4:PRX1. sync\_lost\_signal is set FALSE, forcing PTX3:PTX1. This causes the transmitter to send training requests and to set bport\_sync\_ok to FALSE. With bport\_sync\_ok set false, receive\_ok is cleared forcing the P2:P3 transition. In P3, the port notes that lost of sync caused the problem and sets sync\_fail. It then deactivates the port (bport\_active = FALSE), and heads off to P5. In suspended\_actions, sync\_fail causes connected to be cleared and, as a consequence, P5:P0 is taken. Now on the far end of the peer-to-peer link, the generation of TRAINING requests in step 3 may cause the far end invalid\_count to approach the limit 4 and cause the above cascade of events to occur on the far end. Even if the training requests don't trigger the above events, the fact that the local port goes to P0 and starts toning for a minimum interval will guarantee the far end visits P0 as well.

So I'm wondering what value the PTX3:PTX1 brings, particularly the part about sending TRAINING characters again. This seems to have no purpose, given that bport\_active will be turned off shortly and force both ends of the wire into P0. And sending TRAINING at an arbitrary instant may or may not be detected at the far end. For example, if sent outside of the packet context, then every TRAINING symbol causes the invalid\_count to be bumped up quickly to the limit. But if PTX1 is entered during the data context, then only 1 in 64 TRAINING symbols will be detected as invalid (the comma is detected as invalid, all others are interpreted as valid DATA symbols). So the invalid\_count probably doesn't build up to 4, and we never get the ending symbols required to take the far end receiver out of the data context. So just sending TRAINING at any ole time doesn't seem to work as expected ... we still require the visit to P0 (bport\_active being turned off) to guarantee recovery.

So I was tempted to delete PTX3:PTX1 and add "|| sync\_lost\_signal" to the PTX3:PTX0 transition. (Also, the bit of code which counts TRAINING as INVALID symbols could be removed as well, I think). However, such a fix is still "broken". If we run off to PTX0 before the connectivity management state machine had a chance to clear bport\_on, then we'd wind up back in PTX1 by virtue of the PTX0:PTX1 transition.

So I guess my real question is: should we avoid sending TRAINING requests immediately on loss of sync, or not bother since we know a visit to P0 is imminent? If we want to bother, any ideas for a simple code fix?

Have I missed something more subtle??

**Bug fix description**

Delete the PRX4:PRX1 transition, add "|| sync\_error\_signal" to the PRX4:PRX0 transition. PRX0 will set sync\_lost\_signal, which causes the TX side of the port also to transition to PTX0 (see next).

Delete PTX3:PTX1 and add "|| sync\_lost\_signal" to the PTX3:PTX0 transition. Also, remove the bit of code which counts TRAINING as INVALID symbols

Attempts to retrain (PRX0 to PRX1 and PTX0 to PTX1) while waiting for bport\_on to go false are benign. If this proves to be a problem then this can be fixed by a "while (bport\_on) ;" loop at the start of PRX0 and PTX0.

**ID** 188    **Balloter** MS                                      **How submitted** Post ballot                                      **Fixed in version**  
**Title**            Grammar - "an"    **Status** Done  
**C code files**    **Owner** DW    **SCAT**  **BRAT** 0    **Recirc Clause** 11

**Problem description**

Change to "When a Legacy . . ."

**Bug fix description**

<b>ID</b> 252	<b>Balloter</b>		<b>How submitted</b> Post ballot		<b>Fixed in version</b> D105
<b>Title</b>	Candidate Nephew Conditions (accept standby)			<b>Status</b>	Done
<b>C code files</b>		<b>Owner</b>	CWS	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0 <b>Recirc Clause</b> 11

***Problem description***

Various complications can arise from odd port states when a candidate nephew receives a standby request.

***Bug fix description***

Simplify the circumstances in which a node can become a nephew:- A nephew candidate must have only one active port (not counting the PHY/Link interface as a port) and all other ports must be disconnected. The active port will be the port connected to the uncle candidate.

Also need to caution/require A node shall/should (TBD) not command a standby candidate node to enter standby if the candidate node is performing bus management functions (e.g. Bus Manager, Isochronous Resource Manager, Cycle Master).

Need to change text and C code from proxy-root to root (C code changed - see SCAT #215).

<b>ID</b> 294	<b>Balloter</b>	CWS	<b>How submitted</b> Post ballot		<b>Fixed in version</b>
<b>Title</b>	C Code doesn't match loop test text			<b>Status</b>	Done
<b>C code files</b>		<b>Owner</b>	MT	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0 <b>Recirc Clause</b> 11

***Problem description***

11.7.8 states in item #2 : "The max\_occupancy\_timer and the test timer both expire. In this case, the Controlling node shall place the Test Port in the Active state and then send a long bus reset to all ports that are in the Active state."

C Code only looks for max\_occupancy\_timer reaching the max, rather than both timers. Think I talked with Wooten about this at one time, but not sure.

***Bug fix description***

- 1) change the text in the loop test area so that it is clear that the test is abandon when the received LTS has ATTACH\_IN\_PROGRESS set.
- 2) Eliminate COLLISION\_LIMIT, MAX\_OCCUPANCY\_TIMER and TEST\_INTERVAL from table 13-9. (these are in the 1.06 version but have been taken out of the working version).
- 3) Change table 11-3 as follows:
  - a) add a minimum value for MAX\_OCCUPANCY\_TIMER. I suggest that we use a tolerance value of 8256/BASE\_RATE for min and max. This makes it different from MAX\_DATA\_TIME but that's OK.
  - b) need a min and max for TEST\_INTERVAL. Use tolerance value of 768/BASE\_RATE.
- 4) Clause 13 needs to have something that says what the delay is in getting the received LTP values into a transmitted LTS.
- 5) I will change the loop test text to remove requirement for waiting until both MAX\_OCCUPANCY\_TIMER and test timer time out. Will only wait for MAX\_OCCUPANCY\_TIMER.
- 6) After table 11-3, make a note that the test values are chosen to insure that the LTP signaling ATTACH\_IN\_PROGRESS will be sent a long time before the MAX\_OCCUPANCY\_TIMER times out.

<b>ID</b> 298	<b>Balloter</b>	CWS	<b>How submitted</b> Post ballot		<b>Fixed in version</b>
<b>Title</b>	Range for DISCONNECTED_TONE_INTERVAL			<b>Status</b>	Done
<b>C code files</b>		<b>Owner</b>	CWS	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0 <b>Recirc Clause</b> 11

***Problem description***

Allow a coarse timer for DISCONNECTED\_TONE\_INTERVAL

***Bug fix description***

range of 42.66ms to 48 ms

**ID** 306     **Balloter**     CWS     **How submitted** Post ballot     **Fixed in version**  
**Title**     Connection Management state machine incorrectly references force\_retrain     **Status** Done  
**C code files**     **Owner**     CWS     **SCAT**  **BRAT**     0     **Recirc Clause** 11

**Problem description**

P2:p11 should reference force\_disconnect rather than force\_retrain.

**Bug fix description**

---

**ID** 313     **Balloter**     CWS     **How submitted** Post ballot     **Fixed in version**  
**Title**     RESET\_DETECT in wrong table?     **Status** Done  
**C code files**     **Owner**     MT     **SCAT**  **BRAT**     0     **Recirc Clause** 11

**Problem description**

Only referenced in port.c, so should it be in Table 11-3 instead of 13-9?

**Bug fix description**

Delete from 13-9, insert in 11-3

---

**ID** 352     **Balloter**     DW     **How submitted** Post ballot     **Fixed in version**  
**Title**     Clarify use of Legacy connector     **Status** Done  
**C code files**     **Owner**     MT     **SCAT**  **BRAT**     0     **Recirc Clause** 11

**Problem description**

Cannot be used on a Beta-capable port

**Bug fix description**

Add to Clause 5:-

The 4 and 6 circuit receptacles defined in IEEE std 1394-1995 and IEEE std 1394a-2000 may not be used on a port that is capable of operating in Beta mode. Only the receptacles defined in this clause are allowed for use for short-haul (< 5M) copper connections on a Beta-mode capable port.

---

**ID** 354     **Balloter**     CWS     **How submitted** Post ballot     **Fixed in version**  
**Title**     Bus reset in Standby on a Nephew     **Status** Recommend Accept  
**C code files**     **Owner**     CWS     **SCAT**  **BRAT**     0     **Recirc Clause** 11

**Problem description**

Clarify that the local link cannot initiate a bus reset on a nephew (write to ibr shall be ignored)

**Bug fix description**

---

**ID** 148     **Balloter**         **How submitted** Post ballot     **Fixed in version**  
**Title**     Remove definition of BIAS\_HANDSHAKE     **Status** Done  
**C code files**     **Owner**     CWS     **SCAT**  **BRAT**     0     **Recirc Clause** 11.3

**Problem description**

definition should be removed, and the text rolled into that of RECEIVE\_OK\_HANDSHAKE

**Bug fix description**

**ID** 149    **Balloter**    **How submitted** Post ballot    **Fixed in version**  
**Title**       Word wrap problem    **Status** Done  
**C code files**    **Owner**    CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 11.5  
**Problem description**  
word wrap caused a problem. P2:P11 condition should end with "|| force\_retrain"  
**Bug fix description**

---

**ID** 150    **Balloter**    **How submitted** Post ballot    **Fixed in version**  
**Title**       P12:P11 actions    **Status** Done  
**C code files**    **Owner**    CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 11.5  
**Problem description**  
P12:P11 should have the same actions as P12:P0. Both untested\_fault and loop\_disabled should be cleared.  
**Bug fix description**

---

**ID** 151    **Balloter**    Biva    **How submitted** Post ballot    **Fixed in version** D105  
**Title**       Disabled Standby Port?    **Status** Done  
**C code files**    **Owner**    CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 11.5  
**Problem description**  
Is disable allowed for a standby port? In that case, once it goes to disabled state and is thereafter enabled, should it go back to Standby state or Untested state? Please note that the in\_standby flag is not cleared on entering Disabled state from Standby. Currently the state machine only provides transition to Untested state for this condition. This will enable the port, but the restoration will not succeed, since Restore Configuration packets will not be exchanged. Can the port go to Restoring state directly when enabled?  
**Bug fix description**  
Force an apparent disconnect to the peer when entering disabled from standby - by turning off bport\_on for and holding off the background processing in a way exactly analogous to hard\_disable (but don't actually use the hard\_disable flag) for 2\*DISCONNECTED\_TONE\_INTERVAL. The in\_standby flag is (conveniently) used to determine that we entered disabled from standby.  
  
The peer will see an apparent disconnect, will go to disconnected. When the timer expires on the local port, we allow the background processing to start up a new connection. The peer will go to untested, and wind up in loop\_disabled with untested\_fault set.  
  
A bus reset will occur at the local node if the disable is given by a remote command packet but not if the disable is by a direct write to the PHY register bit (as in a). At the peer node, a bus reset will occur as a result of the apparent disconnection (see SCAT #247).

---

**ID** 152    **Balloter**    **How submitted** Post ballot    **Fixed in version**  
**Title**       P5:P1 notes    **Status** Done  
**C code files**    **Owner**    CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 11.5.1  
**Problem description**  
delete "as a result of failure to complete the previous suspend handshake"  
**Bug fix description**

---

---

**ID** 153    **Balloter**    *How submitted* Post ballot    *Fixed in version*

**Title**    P9:P10 notes    *Status* Done

*C code files*    *Owner* CWS    *SCAT*  *BRAT* 0    *Recirc Clause* 11.5.1

*Problem description*

delete "as a result of failure to complete the previous suspend handshake"

*Bug fix description*

---

**ID** 244    **Balloter**    CWS    *How submitted* Post ballot    *Fixed in version* D104

**Title**    removal of restore\_fault    *Status* Done

*C code files*    *Owner* JH    *SCAT*  *BRAT* 0    *Recirc Clause* 11.5.1

*Problem description*

If a restore fails, the current action is to go to P9, with no reporting on either uncle or nephew. It is not clear what can be done anyway under these circumstances. Better is to go to p0 disconnected, and remove all references to restore\_fault

*Bug fix description*

The port is forced to disconnected, using a new flag to synchronize this between restore\_actions() and connection\_status(). P10:p9 is replaced by P10:P0 on the condition !connected, and the condition on P10:P2 is now connected.

---

**ID** 154    **Balloter**    *How submitted* Post ballot    *Fixed in version*

**Title**    use of "active", Standby etc.    *Status* Done

*C code files*    *Owner* CWS    *SCAT*  *BRAT* 0    *Recirc Clause* 11.6

*Problem description*

Care needed, ports are active not buses, Standby applies to ports not nodes

*Bug fix description*

---

**ID** 220    **Balloter**    MS    *How submitted* Post ballot    *Fixed in version*

**Title**    Restore port command packet    *Status* Done

*C code files*    *Owner* CWS    *SCAT*  *BRAT* 0    *Recirc Clause* 11.6

*Problem description*

One of the things I needed to figure out (off line from PHYDOGS mtg.) was how would a nephew node react to restore\_port command, if the port was already active.

The scenario we were pursuing was this:

- . Nephew node link wants to restore the phy's standby port.
- . Link sends a bus request to send a restore port command.
- . PHY initiates a restore from standby upon detecting the bus request.
- . Interface is initialized, link gets the new node id, etc.
- . Link now re-requests and eventually sends the phy command packet restore port.

>From C code remote\_command, the PHY will set OK=False in the response packet if the port is active (which it should be after a successful restore).

*Bug fix description*

Text in connection management has been re-written, and does not suggest that the nephew link should try to do this.

---

<b>ID</b> 155	<b>Balloter</b>		<b>How submitted</b> Post ballot		<b>Fixed in version</b>
<b>Title</b>	Standby-restore				<b>Status</b> Done
<b>C code files</b>		<b>Owner</b>	CWS	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0 <b>Recirc Clause</b> 11.6.1

**Problem description**

A nephew node on restoring waits for the uncle to win control of its bus and send the restore config packet. This can take awhile which has the nephew's link interface tied up. Consequently, need to caution new link designs to avoid timing out and resetting the interface. Is there a way for the link to know the restore state?

**Bug fix description**

See #227

<b>ID</b> 215	<b>Balloter</b>	DW	<b>How submitted</b> Post ballot		<b>Fixed in version</b> D105
<b>Title</b>	Remove port field from standby packet				<b>Status</b> Done
<b>C code files</b>	port.c	<b>Owner</b>	MT	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0 <b>Recirc Clause</b> 11.6.1

**Problem description**

How about a minor modification to the first sentence to say:

"A node enters into Standby and becomes a nephew when it receives a Standby PHY Command Packet containing its Node-ID unless the ..."

Actually, there is a major modification hidden in there two. I don't see why there needs to be a port address embedded in a Standby command given that the command is being issued to a node that is only allowed to have one active port when it receives this command. Besides, if you leave the port address as part of the command, then you have to change the sentence to say that:

**Bug fix description**

removed from the C code, and also C code updated to reflect the final resolution of the accept/reject standby\_command debate (SCAT #252).

Description of e\_cmnd 1 in 15.5.1 (Remote command packet) needs to say that the port field is ignored for the Standby command

<b>ID</b> 156	<b>Balloter</b>		<b>How submitted</b> Post ballot		<b>Fixed in version</b>
<b>Title</b>	missed reset in standby				<b>Status</b> Done
<b>C code files</b>		<b>Owner</b>	CWS	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0 <b>Recirc Clause</b> 11.6.2

**Problem description**

The spec needs a healthy dose of clarification on the missed reset issues. For example, how does the phy register bit Standby\_reset (one per node) relate to missed reset (one per port)? Is Standby\_reset only meaningful at the nephew? How can it have meaning at the Uncle?

And how does all of this relate to the link interface? We had talked about telling the link if it missed a reset while the interface was asleep (even if none of the PHY ports were in standby), but I don't understand if this has any impact on the bit or not.

**Bug fix description**

missed\_reset is a node level flag in the nephew node, not a per-port flag. It applies to both the case when a reset was missed because the (only iotherwise active) port on the node was in standby, and to the case where a reset was missed because the link was not active at the time.

reset\_notify is the per port flag (only meaningful in an uncle), which tracks whether a reset occurred since the port (and its nephew) went into standby.

Remedy is to clarify this in the spec and to change the name of the PHY register bit to Missed\_reset.



*ID* 160    *Balloter*    Biva                                  *How submitted*    Post ballot                                  *Fixed in version*  
*Title*            Loss of Synchronization in Loop Prevention                                  *Status*    Done  
*C code files*    *Owner*            DW            *SCAT*  *BRAT*    0    *Recirc Clause* 11.7.11

*Problem description*

P.185, L.11-13 (Sec11.7.11) mentions "If,however,an LTP with a mode of ATTACH\_IN\_PROGRESS had been sent on the active bus,then a bus reset must be sent to the active bus (note:this bus reset is not sent on the port that had lost synchronization since it is no longer the test port.)". In order to ensure this we probably require that `untested_actions()` have a line "isbr\_OK = TRUE;" inserted just after the if (!bport\_sync\_OK) condition near Line 27, Page 299. Should it be a short or a long bus reset ?

[JH] This issue was addressed by the loop-breaking revamp. I hadn't realized this clause was already in the text, so I engaged wooten on it. See e-mail "RE: What if ATTACH\_REQUEST doesn't complete?"

Basically, if we lose sync after sending an ATTACH\_IN\_PROGRESS, we either issue another LTP with mode of LTP\_TEST in the case of a non-isolated node, or we simply set HR\_MODE to LTP\_TEST in the case of an isolated\_node. Comments to this effect are in the latest node.c.

But the text needs updating as well.

*Bug fix description*

---

*ID* 161    *Balloter*    Biva                                  *How submitted*    Post ballot                                  *Fixed in version*  
*Title*            No active ports in Loop Prevention                                  *Status*    Done  
*C code files*    *Owner*            DW            *SCAT*  *BRAT*    0    *Recirc Clause* 11.7.11

*Problem description*

Biva:-

The textual description in Sec11.7.12 (No Active Ports) indicates that the test interval may be set to 0 if the PHY currently has no ports in the active state. Is this supported by the C-code ?

[JH] I don't believe the described behavior is desired or necessary. The text should be changed. A related discussion is in the e-mail thread "RE: isolated nodes and such". Basically, we now allow single-port PHY's to skip the establishing dominance phase. Isolated nodes with multiple ports do not skip the dominance phase because they may still cause a loop or suffer from a race condition in which they send an attach out port #0 at the same time a connection to another isolated node reports an attach\_request inbound on port #1. Now we may form a loop, etc. if we aren't really careful. Since it doesn't take an isolated node long to establish dominance, we opted not to make the optimization and deal with all of the hazards. So I consider this closed provided Wooten updates the text.

*Bug fix description*

We amended the latest loop breaking text to discuss single port PHYs and isolated PHY behavior.

---

*ID* 191    *Balloter*    *How submitted*    Post ballot                                  *Fixed in version*  
*Title*            editorial    *Status*    Done  
*C code files*    *Owner*            CWS            *SCAT*  *BRAT*    0    *Recirc Clause* 12

*Problem description*

change "the" to "this"

*Bug fix description*

**ID** 192     *Balloter*     MS                     *How submitted* Post ballot                     *Fixed in version*  
*Title*             DC connected     *Status*     Done  
*C code files*     *Owner*             CWS     *SCAT*  *BRAT*     0     *Recirc Clause* 12  
*Problem description*  
How does a port know that it is DC connected?  
*Bug fix description*

---

**ID** 193     *Balloter*     MS                     *How submitted* Post ballot                     *Fixed in version*  
*Title*             Fault     *Status*     Done  
*C code files*     *Owner*             CWS     *SCAT*  *BRAT*     0     *Recirc Clause* 12  
*Problem description*  
change rwu to rcu  
*Bug fix description*

---

**ID** 194     *Balloter*     MS                     *How submitted* Post ballot                     *Fixed in version*  
*Title*             Port\_error     *Status*     Done  
*C code files*     *Owner*             CWS     *SCAT*  *BRAT*     0     *Recirc Clause* 12  
*Problem description*  
change rw?u to rcu  
*Bug fix description*

---

**ID** 195     *Balloter*     MS                     *How submitted* Post ballot                     *Fixed in version*  
*Title*             Receive\_OK     *Status*     Done  
*C code files*     *Owner*             CWS     *SCAT*  *BRAT*     0     *Recirc Clause* 12  
*Problem description*  
change to Note, Receive\_OK is set to false  
*Bug fix description*

---

**ID** 196     *Balloter*     MS                     *How submitted* Post ballot                     *Fixed in version*  
*Title*             editorial     *Status*     Recommend Accept in principl  
*C code files*     *Owner*             MT     *SCAT*  *BRAT*     0     *Recirc Clause* 12  
*Problem description*  
change "parameter be one" to "parameter shall be one"  
*Bug fix description*  
Superseded by new text

---

**ID** 355     *Balloter*     CWS                     *How submitted* Post ballot                     *Fixed in version*  
**Title**        Clarify Fault and Standby\_fault register descriptions                     *Status* Done  
*C code files*     *Owner*     MT     *SCAT*  *BRAT*   0     *Recirc Clause* 12

***Problem description***

The description of the register bits obscure the assumption by implementors that Fault = suspend\_fault || resume\_fault and that Standby\_fault = standby\_fault.

***Bug fix description***

Clarify

---

**ID** 368     *Balloter*     CWS                     *How submitted* Post ballot                     *Fixed in version* 1.12  
**Title**        Port event     *Status* Done  
*C code files*     *Owner*     MT     *SCAT*  *BRAT*   0     *Recirc Clause* 12

***Problem description***

Text and C code don't match, and both are slightly wrong. Text: reference to bias should be replaced by reference to rx\_ok, port\_event is set if connection\_unreliable goes TRUE, and if restore operations start and watchdog is 1. C Code: port event is set if the fault bit changes, and is not set if bias changes on a disabled port.

***Bug fix description***

Text: update description in the PHY register map to read Port event detect. The PHY sets this bit to one if any of Rx\_ok (unless the port is disabled), Connected, Disabled, Fault, Standby or Standby\_fault change or connection\_unreliable goes TRUE for a port whose Int\_enable bit is one. The PHY also sets this bit to one if resume or restore operations commence for any port and Watchdog is one. A write of one to this bit clears it to zero.

C code: implement with a background process which matches the text (but retain the resume/restore watchdog tests).

---

**ID** 166     *Balloter*                                     *How submitted* Post ballot                     *Fixed in version*  
**Title**        standby\_reset and missed\_reset                                     *Status* Done  
*C code files*     *Owner*     CWS     *SCAT*  *BRAT*   0     *Recirc Clause* 12.1

***Problem description***

relationship between this and missed\_reset?

***Bug fix description***

see SCAT #156

---

**ID** 167     *Balloter*                                     *How submitted* Post ballot                     *Fixed in version*  
**Title**        enab\_accel and enab\_multi                                     *Status* Done  
*C code files*     *Owner*     CWS     *SCAT*  *BRAT*   0     *Recirc Clause* 12.1

***Problem description***

Power reset values of enab\_accel (0) and enab\_multi (1), are different. However description of both indicates they are only used for backwards compatability with legacy links, in which case they should fit the 1394a spec. The 1394a spec has them both power up '0' unless hardware strap options select otherwise. Don't see how 1394b came up with different power up values for these two bits.

***Bug fix description***

---

**ID** 168    **Balloter**    **How submitted** Post ballot    **Fixed in version**  
**Title**    Encoding of Max.\_legacy\_path\_speed    **Status** Recommend Reject  
**C code files**    **Owner**    CWS    **SCAT**  **BRAT**    0    **Recirc Clause** 12.1

**Problem description**

pointless having 3 bits for this, and raises implementation red-herring questions

**Bug fix description**

No confusion as it is. All speeds are encoded in three bits for consistency.

---

**ID** 169    **Balloter**    **How submitted** Post ballot    **Fixed in version**  
**Title**    Power reset value of Pwr\_fail    **Status** Done  
**C code files**    **Owner**    CWS    **SCAT**  **BRAT**    0    **Recirc Clause** 12.1

**Problem description**

1

And add the words:- . . .or upon a PHY power reset.

**Bug fix description**

---

**ID** 170    **Balloter**    **How submitted** Post ballot    **Fixed in version**  
**Title**    untested\_fault visible to software?    **Status** Recommend Reject  
**C code files**    **Owner**    DW    **SCAT**  **BRAT**    0    **Recirc Clause** 12.1

**Problem description**

Currently, a port can be in P12 for two reasons ... one is loop\_disabled while the other is if some sort of fault situation occurred during which a loss of sync occurred. The loop\_disable case is visible to software via a per port register bit, the untested\_fault is not. Is this desirable, or should a second fault bit be provided?

As a reminder, when a loop is found, one side of an untested connection sets loop\_disabled to initiate the transition to P12. The far end sees this as an untested\_fault. So in topologies with loops, untested\_faults should be "common".

**Bug fix description**

Decide that this need not be visible to software.

---

**ID** 171    **Balloter**    **How submitted** Post ballot    **Fixed in version**  
**Title**    ref to 1394a    **Status** Done  
**C code files**    **Owner**    CWS    **SCAT**  **BRAT**    0    **Recirc Clause** 12.1

**Problem description**

replace this reference to 1394a by a reference to 15.1.1

**Bug fix description**

---

<b>ID</b>	172	<b>Balloter</b>	<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	
<b>Title</b>	Local_plug_present			<b>Status</b>	Done	
<b>C code files</b>		<b>Owner</b>	CWS	<b>SCAT</b>	<input checked="checked" type="checkbox"/> BRAT	0 <b>Recirc Clause</b> 12.1

**Problem description**

Description in table 14-2 states that lpp=0 if phy is dc connected to connector. This would mean that a bilingual port would not bother to tone -

This needs some thought. D0.17 used dc\_connected to force toning in this case - but this has been removed. I can't remember why we did this in the first place - it was something a bit subtle.

**Bug fix description**

PHY Dogs decided that the comment should be deleted. Should read:

Indicates that an external implementation dependent mechanism has determined that there is at least a physical connection from the local port (maybe not connected at the "far end"). If there is no such mechanism, then this flag shall be set permanently to 1.

<b>ID</b>	173	<b>Balloter</b>	<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	D104
<b>Title</b>	Negotiated_speed			<b>Status</b>	Done	
<b>C code files</b>		<b>Owner</b>	MT	<b>SCAT</b>	<input checked="checked" type="checkbox"/> BRAT	0 <b>Recirc Clause</b> 12.1

**Problem description**

No power\_reset value given for negotiated speed. C code / comments seem ambiguous on whether: value should start at 000, and go upwards as port is initialized, or value should start at max\_port\_speed, and go downwards as port is initialized

**Bug fix description**

Proposed response:-

Power reset value is implementation dependent, and negotiated\_speed is set no later than the appropriate point in self\_ID.

See definition of port\_speed on p280 l35-37

This is clarified by adding text into the description of S3:S0 on p266 l34 and S4:A0b on p267 l1 shall be no later than this for Beta mode operation.

The previous resolution ( adding comment p293 l38 // set Negotiated\_speed in port register map to port\_speed) is rejected.

<b>ID</b>	174	<b>Balloter</b>	<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	
<b>Title</b>	ref to 1394a			<b>Status</b>	Done	
<b>C code files</b>		<b>Owner</b>	CWS	<b>SCAT</b>	<input checked="checked" type="checkbox"/> BRAT	0 <b>Recirc Clause</b> 12.1

**Problem description**

This should be a ref to 15.5.1, not 1394a.

**Bug fix description**

<b>ID</b>	189	<b>Balloter</b>	MS	<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	
<b>Title</b>	Automatic Setting of IBR register			<b>Status</b>	Done		
<b>C code files</b>		<b>Owner</b>	CWS	<b>SCAT</b>	<input checked="checked" type="checkbox"/> BRAT	0 <b>Recirc Clause</b>	12.1

**Problem description**

change to rwu

**Bug fix description**

Same applies to ISBR

**ID** 190     **Balloter**     **How submitted** Post ballot     **Fixed in version**  
**Title** spelling     **Status** Done  
**C code files**     **Owner** CWS     **SCAT**  **BRAT** 0     **Recirc Clause** 12.1  
**Problem description**  
operations

*Bug fix description*

---

**ID** 203     **Balloter**     **How submitted** Post ballot     **Fixed in version**  
**Title** Int\_enable bit     **Status** Done  
**C code files**     **Owner** DW     **SCAT**  **BRAT** 0     **Recirc Clause** 12.1  
**Problem description**

Description of the Int\_enable bit in table 14-2 should include the Standby bit (i.e., changes in the standby bit should cause an interrupt if the Int\_enable bit is set for this port). More issues on standby\_fault and restore\_fault, and correct action on behavior leading to restore\_fault.

*Bug fix description*

(partially done, matters arising need dealing with)

Also requires a change to the description of the Port\_Event bit. Also should include Standby\_fault.

---

**ID** 210     **Balloter** DW     **How submitted** Post ballot     **Fixed in version**  
**Title** POR values for PHY registers     **Status** Done  
**C code files**     **Owner** CWS     **SCAT**  **BRAT** 0     **Recirc Clause** 12.1  
**Problem description**

I need all values in the PHY registers to have some power-on value even if it is vendor dependent. Particular are PHY-ID and R bit. They don't have any power on values specified. I suppose that this is because they are not used until a bus reset is sent and they would get set there. However, with the PIL-FOP model, the PIL might use the values in the FOP before any other port comes up. So, I would like the PHY-ID to initialize to zero and the R bit to be set to 1.

Would also like some definition of the gap count sticky-bit in the spec and think that PHY Register Chapter is as good a place as any (we refer to the sticky bit in several places but we don't define what it is anywhere.) When we define it, we should state what its POR value is.

*Bug fix description*

B\_link to vendor-dependent

Max\_legacy\_path\_speed to 000 (also logging a bug that this is not initialised - needs to be reset in R0)

Physical\_ID I think should be power reset to 63 (the bus is mal-configured until the first time it is configured!!) - this is different from your suggestion!

PS to vendor-dependent

R to 1 (PHY always comes up as an isolated node)

AStat and BStat as 00

Child 1 (interesting this, disconnected or disabled ports are always "children" - I'm clarifying the text on this)

Negotiated\_speed as 000

Receive\_OK as 0

**ID** 197     **Balloter**     MS                     **How submitted** Post ballot                     **Fixed in version**  
**Title**         BUS\_RESET\_COMPLETE     **Status** Done  
**C code files**     **Owner**         MT         **SCAT**  **BRAT**     0     **Recirc Clause** 13  
**Problem description**  
replace with SELF\_ID\_COMPLETE (and on line 24)  
**Bug fix description**

---

**ID** 198     **Balloter**     MS                     **How submitted** Post ballot                     **Fixed in version**  
**Title**         spelling     **Status** Done  
**C code files**     **Owner**         MT         **SCAT**  **BRAT**     0     **Recirc Clause** 13  
**Problem description**  
tranmitting -> transmitting (several places)  
**Bug fix description**

---

**ID** 199     **Balloter**     MS                     **How submitted** Post ballot                     **Fixed in version**  
**Title**         Bias\_bit     **Status** Done  
**C code files**     **Owner**         MT         **SCAT**  **BRAT**     0     **Recirc Clause** 13  
**Problem description**  
change to Receive\_OK bit (and on p254 line 23)  
**Bug fix description**

---

**ID** 218     **Balloter**     CWS                     **How submitted** Post ballot                     **Fixed in version** D107  
**Title**         Maybe should allow multiple ARBRST's in a row?                     **Status** Done  
**C code files**     **Owner**         JH         **SCAT**  **BRAT**     0     **Recirc Clause** 13  
**Problem description**  
Various possible corner cases, and lost token conditions (see email thread)  
**Bug fix description**

we must have the timeout for the case where the ARBRST\* token is lost and phases don't advance

Add the time-out at the BOSS. Yes, it uses the same timeout as the missing ACK, but I don't think the C Code today will re-generate ARBRSTs. After each passing of BOSS\_RESTART\_TIME (equal to arb\_reset\_gap time at senior borders) at the proxy\_root (PR) or at senior borders (SBs) while in IDLE, the PR/SBs will issue an ARBRST token if a) one hasn't been issued yet, or b) not everyone has acknowledged the current phase, or c) everyone has acknowledged the current phase, there are no in-phase requests, but there are next phase requests pending. In Case B, the ABRRST generated will be a repeat of the current phase. For C), the ARBRST will be one to advance to the next interval

Basic fix outlined above was accepted. After all "normal" gap timings have been done, the code will check every subaction\_gap or BOSS\_RESTART\_TIME to see if another arb reset is needed. Fix is a little kludgy ... it resets the arb\_timer in the middle of IDLE. Better ideas?

Noted that we weren't inverting the async phase in a hybrid bus. Assumed it was an error and corrected it ...

Also, no filtering is done on indications to the link ... so multiple arb reset gaps could be reported. Should this be addressed? Decided that it is already possible for software to see multiple arb resets with nothing inbetween. So can't be a problem for software. Also for links, its just a flag to allow arbitration. So not believed to be a problem. Conclusion, no filtering required.

---

*ID* 224     *Balloter*     DW                     *How submitted* Post ballot                     *Fixed in version* D106  
*Title*             Qualification of configuration requests                     *Status*     Done  
*C code files*     *Owner*     CWS     *SCAT*  *BRAT*     0     *Recirc Clause* 13

***Problem description***

Configuration requests need to be sent such that they can be 'debounced'. We can't have a configuration request sent as a single symbol. That symbol can be corrupted by a single-bit error and that could, in some cases, cause problems. On the other hand, there are cases where a node may receive a single configuration request symbol. That symbol cannot be accepted without qualification (it could be received as a result of noise on the line). Rather than go down every case of bad configuration token to decide if each of them leads to a correctable condition, it is better to debounce them. Debouncing would mean that they are only accepted in a specific context (i.e., only accept a disconnect request after seeing a PHY packet.) It might be easier, however, simply to state that each configuration request must be followed by another configuration request of the same type. If we couple this with a statement that each configuration request must be sent for at least 8 symbol times, then we have a secure way of getting the configuration requests through (it becomes just at least reliable as the two control tokens that we have at the beginning and end of each packet.)

***Bug fix description***

I will write a piece of text that describes why we are qualifying the configuration requests. Basically, we want the configuration requests that result in bus fragmenting to be as reliable as the start-of-packet or end-of-packet delimiters (mostly, when we miss either a SOP or EOP we end up fragmenting the bus.) Getting a double-bit error in an arbitration request that turns it into a valid configuration request is less likely than getting a double-bit error affecting two consecutive control codes. However, arbitration requests occur with much higher frequency so the chances of fragmenting the bus due to an erroneous configuration request is higher than fragmenting the bus due to loss of an SOF or EOF. However, if we require that the configuration requests come in pairs, the chances of two codes being changed in such a way that they descramble and decode into the same configuration request is *\_very\_ small*.

C code modified so that bport\_rx.c requires two identical consecutive config requests before pushing one into the FIFO, and only pushes one into the FIFO for any sequence of 2 or more identical config requests.

transmit\_functions.c modified to send four config requests. All other config requests correspond to unlocked Legacy arbitration, i.e. they are repeated indefinitely until some response is returned.

*ID* 259     *Balloter*                                     *How submitted* Post ballot                     *Fixed in version*  
*Title*             terminology in Clause 12                                     *Status*     Done  
*C code files*     *Owner*     MT     *SCAT*  *BRAT*     0     *Recirc Clause* 13

***Problem description***

Fix "local port" and "main bus" ... terms not known in this context.

***Bug fix description***

*ID* 261     *Balloter*                                     *How submitted* Post ballot                     *Fixed in version*  
*Title*             Service definitions                                     *Status*     Done  
*C code files*     *Owner*     MT     *SCAT*  *BRAT*     0     *Recirc Clause* 13

***Problem description***

make sure service definitions are limited to C code usages.

***Bug fix description***

*ID* 272    *Balloter*    CWS                    *How submitted* Post ballot                    *Fixed in version* D107  
*Title*            BORDER and Max\_Beta\_Time                    *Status* Done  
*C code files*    *Owner*    CWS    *SCAT*  *BRAT* 0    *Recirc Clause* 13

***Problem description***

At the last BRC I brought up a concern about MAX\_BETA\_TIME and whether it was sufficiently small to prevent old PHY's from pulling a bus reset during the isoch interval. In originally picking the value for MAX\_BETA\_TIME, we assumed that the max delay to servicing of a BORDER request would be based on a max async packet. However, I noted max isoch packets are twice as long, and can be concatenated together, effectively delaying service of a BORDER request for 100 us or more.

Having discussed at the BRC and looking at 1394a some more, I'm convinced that it is reasonable for receipt of data bits to reset the maximum arb state timer. (Witness the RX:RX transition ... can we assume that designers reset the timer on this transition?) So as long as the cycle start packet is sent with Legacy format at S100, then it seems that we wouldn't ever violate max arb state timeout during the isoch interval (max separation of CSP packets is ~165 us I think). So the calculation for MAX\_BETA\_TIME really only needs to consider the async case.

Does anyone have data to the contrary concerning the affect of receiving data bits on the max arb state timeout?? Is everyone comfortable making this assumption, or does someone want to volunteer to socialize at the TA or wherever?

In all of this, however, I remain convinced that MAX\_BETA\_TIME might be a tad to small. The number David proposed which is in the current draft (114.2 us) is calculated below and has a few arithmetic errors in it, I think. Specifically, the maximum net diameter is closer to 3.75 us. (A gap count of 63 allows for a roundtrip time of about 7.525 us according to the 1394a gap count analysis). And the max async packet is 512 bytes of payload, 20 bytes of header and data CRC, and ~400ns of delimiters giving a max async packet duration of ~43.7 us. Finally, I think David had a typo below (plugged in 2.5us for net diameter rather than the assumed 3.5 us). Allowing 5 diameters plus a max packet time for a BORDER request to take affect, the corrected calculation would be:

$167\text{us} - (43.7\text{us} + (5 * 3.75\text{us})) = 104.55 \text{ us}$ . So we're about 10 us off. I'd like to see 104.5 us for the spec.

Also, I'm not sure why this number is specified as a min rather than a max ... it hurts my head!

Another consequence of the above observation is that we may not need to heed BORDER requests in the iso interval. The conservative timer approach might indicate that we need to send a null packet in the iso interval ... but we can ignore it since we know the CSP did the trick. So we could mask out the BORDER usage during the iso interval in the C code.

Finally, if we believe data bits reset the max arb state timeout in all Legacy phys, we could change the DS\_stuck detection logic to match that assumption: namely, clear DS\_stuck if data bits are sent at S100 legacy or a DATA\_END is sent.

***Bug fix description***

There are PHYs which don't reset the arb state timeout timer on the RX:RX transition.

Needs both C code fixes and text change.

Need to protect two things:-

1) async for -95 PHYs which use 166us timeouts because isoch is not a problem (PHYs using this value are believed to reset the arb state timeout on the RX:RX transition)

2) 1394a PHYs which use 200us timeouts, but don't reset the arb state timeout timer when taking RX:RX, i.e for concatenated isoch packets. Two reflector checks have failed to identify any such PHYs. This is therefore deemed not to be a problem.

Turns out that the sums are approximately the same for the two cases. For the second case, would need to be able to unstick in the middle of a sequence of isoch packets. So BORDER request must take priority over isoch requests (C code update) - (now considered not needed - there are no such PHYs).

Note on the RX:RX transition that the arb state timer is reset on this transition, and a note on the All:RXb transition (done)

Agree the number of 104.5 microseconds as the max, and 83.324 as the min (allows it to be 8192/BASE\_RATE) (done in Ch 13).

Redundant setting of DS\_stuck removed from receive\_actions (duplicated at head of receive\_functions)

**ID** 289    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version**

**Title**            DATA\_NULL description                    **Status** Done

**C code files**    **Owner**            MT            **SCAT**  **BRAT**    0    **Recirc Clause** 13

**Problem description**

explain why data\_null was needed

**Bug fix description**

---

**ID** 296    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D107

**Title**            Should accelerating be set TRUE on PH\_ISOCH\_PHASE\_ODD/EVEN?                    **Status** Done

**C code files**    **Owner**            MT            **SCAT**  **BRAT**    0    **Recirc Clause** 13

**Problem description**

**Bug fix description**

Noticed in passing that we no longer had a phy service for a legacy link to tell the PHY that the cycle start had been received (the old accel lreq). So PH\_CYCLE\_START\_SEEN was added and needs to be included in the arb chapter.

---

**ID** 297    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D107

**Title**            Autocrossover\_supported                    **Status** Done

**C code files**    **Owner**            MT            **SCAT**  **BRAT**    0    **Recirc Clause** 13

**Problem description**

AUTOCROSSOVER\_SUPPORTED should be a flag in the PMD\_status\_indication() - it's a PMD property not a port property.

Will need documenting in the services section of the arbitration chapter as well

**Bug fix description**

---

**ID** 301    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D107

**Title**            DATA\_NULL and CSTs missing on A2:RX and A1:RX                    **Status** Done

**C code files**    **Owner**            MT            **SCAT**  **BRAT**    0    **Recirc Clause** 13

**Problem description**

DATA\_NULL indicates the start of a received packet just like SPEED or DATA\_PREFIX, so should force transission into RX. An important example would be when speed/format filtering prevents the SPEED or DP from arriving at the requesting /senior port and DATA\_NULL is the only indication that the GRANT has been acknowledged (A2:RX) or that the request is being denied (A1:RX).

Also, need to realize the CSTs start a packet as well and may need to be present. Probably best to define some time of data\_coming() call to simplify the state diagram.

**Bug fix description**

new boolean function data\_coming\_on(int port) defined

---

**ID** 303     **Balloter**     CWS                     **How submitted** Post ballot                     **Fixed in version** D107  
**Title**     arb\_timer shouldn't measure MAX\_ARB\_STATE\_TIME                     **Status** Done  
**C code files**     **Owner**     MT     **SCAT**  **BRAT** 0     **Recirc Clause** 13

**Problem description**

Traditionally, we have said that the C code doesn't explicitly describe the use of the arb\_timer to check for MAX\_ARB\_STATE\_TIME. However, in node.c, the loop detection logic contains "arb\_timer == MAX\_ARB\_STATE\_TIME".

We haven't thoroughly checked that arb\_timer counts and is reset as it should to implement that max timeout. Perhaps it would be safer to invent a signal which is declared to be true when a timeout has happened without actually referencing arb\_timer?

**Bug fix description**

boolean function invented

transition All:R0b can now use this function

---

**ID** 305     **Balloter**     CWS                     **How submitted** Post ballot                     **Fixed in version** D107  
**Title**     Why stretch symbols in a B\_Bus?                     **Status** Done  
**C code files**     **Owner**     CWS     **SCAT**  **BRAT** 0     **Recirc Clause** 13

**Problem description**

Possible optimization: one of the criticisms of 1394a was that it didn't scale with speed. This has been largely addressed with P1394b with the exception that DATA\_NULL is stretched to make sure it is sent for at least one symbol time at the slowest speed. This stretching amounts to some non-linear overhead.

Thought is that the stretching of DATA\_NULL may have only been done to lock of DS clouds. Consequently, the stretching may not be required for a B Bus.

**Bug fix description**

Already done for transmit actions, equivalent fix applied to receive\_actions

---

**ID** 312     **Balloter**     CWS                     **How submitted** Post ballot                     **Fixed in version**                     **Status** Done  
**Title**     DELETABLE\_SYMBOL\_TIME vs MIN\_DELETABLE\_SYMBOL\_TIME                     **Status** Done  
**C code files**     **Owner**     MT     **SCAT**  **BRAT** 0     **Recirc Clause** 13

**Problem description**

Both are defined in Table 13-9, but only MIN\_DELETABLE\_SYMBOL\_TIME is referenced by the text or C code.

**Bug fix description**

remove DELETABLE\_SYMBOL\_TIME from table 13-9

---

**ID** 314     **Balloter**     CWS                     **How submitted** Post ballot                     **Fixed in version** D107  
**Title**     T0:T0 re-entry without time elapse                     **Status** Done  
**C code files**     **Owner**     MT     **SCAT**  **BRAT** 0     **Recirc Clause** 13

**Problem description**

t0:t0 happens when arb\_timer==config\_timeout. No time elapses before the t0:t0 test happens again, so arb\_timer doesn't get a chance to advance any and the model gets into an infinite loop. Seems like this is a C code issue as well ... not sure. Suggested fixed would be to make t0:t0 test be !T0\_timeout && (arb\_timer == CONFIG\_TIMEOUT) with a corresponding change in tree\_ID\_start\_actions.

**Bug fix description**

---

**ID** 315    **Balloter**    Missing Services    **How submitted** Post ballot    **Fixed in version**  
**Title**    Missing services    **Status** Done  
**C code files**    **Owner** MT    **SCAT**  **BRAT** 0    **Recirc Clause** 13

**Problem description**

Seems like LPS and "PS" should be formal service indications from the link to complete the "black box" nature of the PHY code. Only service missing then would probably be some sort of reset service. See PRN on PH\_RESET for challenge in building that.

Then again, maybe the reset service isn't needed. The PHY core could be viewed as always putting out the PH\_ indications to the link "block". The link block would be the one responsible for block the indication to the link until a safe point has arrived on the bus?

Still, not sure if we can easily get the phy to make a call to cancel\_requests() whenever LPS reset is initiated. Rambling ...

**Bug fix description**

In principle, decided to take the approach indicated in the second para above - but the reset service is indeed still needed.

Detail:-

PH\_ARB\_requests added for LPS\_ACTIVE and LPS\_INACTIVE  
boolean PMD\_PS\_request added  
role of PH\_EVENT\_response changed  
missed\_reset flag eliminated

Clarified that PHY/Link block will track the arb state machine and deal with all the link interface reset and bringup issues.

- On link initiated interface resets, the PHY/link block will
- invoke cancel\_requests at the PHY (by issuing a PH\_RESET)
  - filter all LReqs until it has transferred Reg 0 to the link
  - issue Reg 0 from the last reg 0 issued by the arb block
  - issue PH\_ARB\_RESET\_ODD/EVEN as appropriate from last ditto issued by arb block
  - filter upward indications until the interface is idle
  - handshake with the PHY if granted during the reset interval

- On PHY initiator reset/restore, the PHY/Link block will
- handshake with PH\_EVENT\_response (after which the arb block will cancel requests)
  - filter requests until reg 0 has been transferred

MT to update the service descriptions

Need to check that this is OK in the PHY/Link chapter

---

<b>ID</b> 316	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	accelerating clean-up		<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> MT	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0 <b>Recirc Clause</b> 13

**Problem description**

Not broken ... but an opportunity to clean up.

accelerating is driven in the arb domain three times. In receive\_functions and transmit\_functions, it is cleared by cycle\_start\_tokens and granting a B cycle master. However, it is only tested for Legacy Link purposes (!B\_node\_root, or when breq != NO\_REQ, for example). So the clearing in receive\_functions and transmit\_functions isn't required.

Only other point is on the S4:A0b transition which, I believe, is a hold over from P1394a days to set accelerating TRUE in an async only bus.

This has questionable value in P1394 since we learn acceleration capabilities of a link in P1394b, and if async only, will never use accelerating. So pre-setting after a bus reset doesn't mean much ... if it wasn't already set then it means the link was expecting a cycle\_start packet and that cycle start will likely be the first packet out of reset.

So I could argue that it shouldn't be set at all on S4:A0b, or could argue that it should be set by cancel\_requests() which is called in R0.

In either case, motivation is one less driver on the signal.  
 ==== 01/13/2001 02:03:13 PM ==== Colin Whitby-Strevens =====

- 1) accelerating is set, not cleared, in transmit and receive functions, but point taken
  - 2) All three places where it is tested are pre-qualified by link\_CS\_indications being TRUE
  - 3) link\_CS\_indications is cleared on power reset, and, on the one occasion it is set true (PH\_CYCLE\_START\_DUE), accelerating is set FALSE
- The conclusion of (2) and (3) is that it is safe to remove it from S4:A0b. Also, it does not need to be initialised by power reset etc.

**Bug fix description**

removed

<b>ID</b> 317	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b> D107
<b>Title</b>	PMD Indications should be requests		<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> MT	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0 <b>Recirc Clause</b> 13

**Problem description**

As discussed with Mike, indications go up, not down. Requests head down.

**Bug fix description**

PMD\_STATUS\_indication and PMD\_DSPORT\_\*\_indications are now requests.

<b>ID</b> 319	<b>Balloter</b>	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	Service descriptions of link behavior for standby/restore		<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> MT	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0 <b>Recirc Clause</b> 13

**Problem description**

Put link requirements in service indications

**Bug fix description**

PH\_RESTORE\_NO\_RESET - The PHY layer is reporting that the port which was in Standby has commenced a restore operation, and that the previous Physical\_ID is invalid. The link shall not issue requests until it has sufficient information. This information includes having a valid Physical\_ID before making any bus request, knowing the isochronous interval before making an isochronous request and/or knowing the asynchronous interval before making a NEXT\_EVEN or NEXT\_ODD asynchronous request (it may make a CURRENT request). This indication shall only be generated in the Nephew.

"PH\_SELF\_IDENTITY. Given during bus initialization, after a bus reset or after a restore. The Physical\_ID and Root parameters will also be valid for this indication. "

<b>ID</b> 321	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	definition of MIN_IDLE_TIME	<b>Status</b> Done	
<b>C code files</b>	<b>Owner</b> MT	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0	<b>Recirc Clause</b> 13

**Problem description**

Description is very vague (does not match text requirements), and does not address subtlety concerning speed signal in advance of DP on Legacy ports.

**Bug fix description**

Minimum idle time (not including speed signalling on a Legacy mode port) after transmission of DATA\_END for a Legacy packet at either an originating or repeating port (~4/BASE\_RATE).

---

<b>ID</b> 324	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	A0:A1 text change	<b>Status</b> Done	
<b>C code files</b>	<b>Owner</b> MT	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0	<b>Recirc Clause</b> 13

**Problem description**

The state machine notes appear to be in error. What they should be communicating is that the A0:A1 transition is never taken at the proxy\_root. At non proxy\_root's, the transition is taken as follows:

- 1) a LEGACY\_REQUEST is received from any "junior" port (Beta or D/S). Note this is done at \_any\_ node which isn't proxy\_root (not just border nodes)
- 2) for any node (!proxy\_root) which also has a Legacy link: any link request other than immediate causes the A0:A1 when the timing constraints of arb\_OK() are met.
- 3) \_Only\_ at senior borders which are !proxy\_root: a boss arbitration request from a junior port or any request other than immediate from a local link causes the A0:A1 when the timing constraints for arb\_OK() are met.

These rules are probably confusing to read. The intent is any legacy launched requests are forwarded immediately up the tree, regardless of whether the senior\_port is D/S or beta (#1). Also, any request from a Legacy link gets launched on the bus and forwarded up the tree as a LEGACY\_REQUEST after timings have been met regardless of the senior port's type (#2). Finally, any beta cloud which doesn't contain the proxy\_root has to convert all requests to LEGACY\_REQUESTS and forward then, via D/S, to the proxy\_root's cloud. This is the responsibility of the senior border in each beta cloud: hence #3.

Note that each beta cloud in a hybrid bus has one senior border ... so you have more than one senior border in a network. In contrast ... you get one and only proxy\_root.

**Bug fix description**

text update

---

<b>ID</b> 344	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	PH_DATA_START doesn't have format parameter	<b>Status</b> Done	
<b>C code files</b>	<b>Owner</b> MT	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0	<b>Recirc Clause</b> 13

**Problem description**

The parallel PHY/Link interface needs to inform the link of the format for all received packets. Currently, a PH\_DATA\_indication of PH\_DATA\_START only communicates the speed, not the format. Requires changes to C code service model and the Arb clause to document the new parameter.

**Bug fix description**

---

<b>ID</b>	348	<b>Balloter</b>	CWS	<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	
<b>Title</b>	Suggest rename AUTOCROSSOVER_SUPPORTED			<b>Status</b>	Done		
<b>C code files</b>		<b>Owner</b>	MT	<b>SCAT</b>	<input checked="" type="checkbox"/>	<b>BRAT</b>	0 <b>Recirc Clause</b> 13

**Problem description**

to PMD\_AUTOCROSSOVER\_SUPPORTED in phy\_services.h for consistency with other events in PMD\_STATUS\_request

**Bug fix description**

<b>ID</b>	351	<b>Balloter</b>	CWS	<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	D1071
<b>Title</b>	Missing services			<b>Status</b>	Done		
<b>C code files</b>		<b>Owner</b>	MT	<b>SCAT</b>	<input checked="" type="checkbox"/>	<b>BRAT</b>	0 <b>Recirc Clause</b> 13

**Problem description**

Two inputs to the PHY remain which are not currently coming from services: cable\_speed and link. Seems like cable\_speed could be a PMD request while link is a little harder ... may need to define a PH\_BETA\_LINK and PH\_LEGACY\_LINK request from the LINK layer.

**Bug fix description**

```
speedCode PMD_CABLE_SPEED_request();
// Set by implementation-dependent means to the maximum speed
// at which the cable attached to the port is allowed to operate. If no
// cable-dependent speed indication is available, then the implementation
// shall return the maximum speed that the port is capable of.
// The encoding is the same as max_port_speed. This value is also
// maintained as a read-only register in the port register map.
```

linkType PH\_LINK\_TYPE\_response() - called in process\_requests immediately after power reset to determine link type

<b>ID</b>	205	<b>Balloter</b>		<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	
<b>Title</b>	Format for restore Packet			<b>Status</b>	Done		
<b>C code files</b>	node.c	<b>Owner</b>	MT	<b>SCAT</b>	<input checked="" type="checkbox"/>	<b>BRAT</b>	0 <b>Recirc Clause</b> 13.1

**Problem description**

C code comment says Legacy, C code builds a Beta format packet. Which should it be?

Issue widened out to all PHY packets.

**Bug fix description**

- 1) all PHY packets shall be sent at S100.
- 2) the PHY shall use legacy format for all PHY packets issued during Self-ID (because it can't know if the bus is hybrid or not)
- 3) the PHY may send a restore packet in either Legacy or Beta format
- 4) the link shall initiated PHY packets at S100. If the link doesn't know if the bus is a B Bus, it shall not mark the packet as a Beta format packet. (Observation: the PHY will upgrade if it can (i.e. only if a B bus).)
- 5) in a B Bus, the PHY shall upgrade all link requests to beta format, regardless of speed.
- 6) in a hybrid bus, the PHY shall upgrade the format of any packet issued by the link if and only if the packet speed exceeds max\_legacy\_path\_speed.
- 7) Observation: a PHY has to be able to receive PHY packets in either Legacy or Beta format

<i>ID</i>	175	<i>Balloter</i>	<i>How submitted</i>	Post ballot	<i>Fixed in version</i>	
<i>Title</i>	PH_DATA indication			<i>Status</i>	Done	
<i>C code files</i>		<i>Owner</i>	MT	<i>SCAT</i>	<input checked="" type="checkbox"/> <i>BRAT</i>	0 <i>Recirc Clause</i> 13.2.3.3

*Problem description*

There is an inconsistency in the description of Speed\_Code. Is the Speed\_Code valid with a phy data indication of PH\_DATA\_START or PH\_DATA\_BYTE?

*Bug fix description*

PH\_DATA\_START - chapter needs correcting

---

<i>ID</i>	177	<i>Balloter</i>	<i>How submitted</i>	Post ballot	<i>Fixed in version</i>	
<i>Title</i>	Description of self_ID packet needed			<i>Status</i>	Done	
<i>C code files</i>		<i>Owner</i>	MT	<i>SCAT</i>	<input checked="" type="checkbox"/> <i>BRAT</i>	0 <i>Recirc Clause</i> 13.3.1

*Problem description*

Need to include the self-ID packet definition here, particularly as there's a new meaning to the speed field 11 and the port status definition needs to be updated to deal with ports in standby.

*Bug fix description*

---

<i>ID</i>	202	<i>Balloter</i>	<i>How submitted</i>	Post ballot	<i>Fixed in version</i>	
<i>Title</i>	SPEEDb is used when SPEEDa was intended			<i>Status</i>	Done	
<i>C code files</i>		<i>Owner</i>	MT	<i>SCAT</i>	<input checked="" type="checkbox"/> <i>BRAT</i>	0 <i>Recirc Clause</i> 13.3.1

*Problem description*

*Bug fix description*

---

<i>ID</i>	178	<i>Balloter</i>	<i>How submitted</i>	Post ballot	<i>Fixed in version</i>	
<i>Title</i>	Cmnd 4 clears Standby_Fault			<i>Status</i>	Done	
<i>C code files</i>		<i>Owner</i>	MT	<i>SCAT</i>	<input checked="" type="checkbox"/> <i>BRAT</i>	0 <i>Recirc Clause</i> 13.3.3

*Problem description*

add "and Standby\_fault"

*Bug fix description*

---

<i>ID</i>	365	<i>Balloter</i>	CWS	<i>How submitted</i>		<i>Fixed in version</i>	
<i>Title</i>	abbreviations for "reserved"			<i>Status</i>	Done		
<i>C code files</i>		<i>Owner</i>	MT	<i>SCAT</i>	<input type="checkbox"/> <i>BRAT</i>	0 <i>Recirc Clause</i> 13.3.3.1	

*Problem description*

Figure 13-2 in the 1.11 draft standard has an "r"bit which means that the bit is reserved. Indeed, the figure uses "r", "rsv", and "reserved" yet Table 13-4 only defines "reserved". Consistency is a wonderful thing.

*Bug fix description*

---



**ID** 180     **Balloter**     **How submitted** Post ballot     **Fixed in version**  
**Title**     delete BIAS\_HANDSHAKE     **Status** Done  
**C code files**     **Owner**     MT     **SCAT**  **BRAT** 0     **Recirc Clause** 13.3.4

**Problem description**

Delete from this table, (a) because it is a connection management constant and (b) is superseded by RECEIVE\_OK\_HANDSHAKE

**Bug fix description**

**ID** 181     **Balloter**     CWS     **How submitted** Post ballot     **Fixed in version**  
**Title**     CONFIG\_TIMEOUT     **Status** Done  
**C code files**     **Owner**     JS     **SCAT**  **BRAT** 402     **Recirc Clause** 13.3.4

**Problem description**

The value of CONFIG\_TIMEOUT needs to be adjusted to prevent false positive loop detects (always was possible in the Legacy draft, but the effect now is less benign).

**Bug fix description**

Jim will check values and make recommendation .

The original recommendation was to increase the CONFIG\_TIMEOUT value to around 214us (if I remember correctly). However, as we briefly discussed before, I think there's a potential problem with this. If there is a loop in the network, then some nodes (not in the loop) will experience a state-timeout and generate a bus-reset before config-timeout is reached, and so no config-timeout interrupt occurs.

The recommendation for increasing CONFIG\_TIMEOUT was based on the fact that the FORCE\_ROOT\_TIMEOUT value could be as high as the CONFIG\_TIMEOUT value. I think that a better solution would be to limit the max value of FORCE\_ROOT\_TIMEOUT to something considerably less than 166.9us (the max for CONFIG\_TIMEOUT):

FORCE\_ROOT\_TIMEOUT    min: 83.3 us, max: 119.0 us (~8192/BASE\_RATE to ~11700/BASE\_RATE)

The max value is based upon Judi Romijn's analysis, which, as I understand it, states that the max startup delay for tree-id, including any wait time for packet transmission to finish and any wait time for force-root to timeout, must be less than the config-timeout time. This boils down to (force-root timeout + 47.17 us) < 166.9 us.

**ID** 212     **Balloter**     CWS     **How submitted** Post ballot     **Fixed in version**  
**Title**     Value for ROOT\_CONTEND\_FAST     **Status** Done  
**C code files**     **Owner**     MT     **SCAT**  **BRAT** 0     **Recirc Clause** 13.3.4

**Problem description**

ROOT\_CONTEND\_FAST is shorter than the maximum round-trip time. This may result (state T3:Root Contention) in both nodes of a contending pair seeing the peer's PARENT\_NOTIFY after the contention timeout, and concluding that they are the child, and moving into T1: Child Handshake. I think that this value should be doubled to 1.6 usec for a Beta link. However, when the peer is a Legacy port, it is important to use the current 0.8 usec value. Note that the current 3.2 usec value for ROOT\_CONTEND\_SLOW will give a bias towards the 1394b PHY becoming root in the case of contention between a Legacy PHY and a 1394b PHY.

**Bug fix description**

RCF should be 1.6, no need to distinguish on operating mode.

**ID** 201 **Balloter** **How submitted** Post ballot **Fixed in version** D103  
**Title** loose\_contention **Status** Done  
**C code files** **Owner** MT **SCAT**  **BRAT** 0 **Recirc Clause** 13.4.1  
**Problem description**  
Loose\_contention is misspelled, and not used anyway  
**Bug fix description**  
delete it

---

**ID** 182 **Balloter** **How submitted** Post ballot **Fixed in version**  
**Title** All:R0 transition **Status** Done  
**C code files** **Owner** MT **SCAT**  **BRAT** 0 **Recirc Clause** 13.4.4  
**Problem description**  
The All:R0a transition should not have a transition label per our convention that power\_reset is "special" and can interrupt state actions, etc. As a result All:R0b and All:R0c should be relabeled.  
**Bug fix description**

---

**ID** 183 **Balloter** **How submitted** Post ballot **Fixed in version**  
**Title** Bogus loop timeouts on T0:T0 **Status** Recommend Reject  
**C code files** **Owner** PhyDogs **SCAT**  **BRAT** 402 **Recirc Clause** 13.4.6  
**Problem description**  
There may be bogus "loop" timeouts - see TA discussion and presentation from Judi Romijn. Need to consider the impact of this study on 1394b.  
**Bug fix description**  
See BRAT  
David declared this redundant

---

**ID** 225 **Balloter** MS **How submitted** Post ballot **Fixed in version** D104  
**Title** Loop bit in T0:T0 transition **Status** Done  
**C code files** **Owner** MT **SCAT**  **BRAT** 0 **Recirc Clause** 13.4.6  
**Problem description**  
Loop bit is not set in T0:T0, and needs to be tested before giving the PH\_EVENT\_indication  
**Bug fix description**  
Action becomes  
T0\_timeout = TRUE;  
if (loop == 0) { loop = 1; PH\_EVENT\_indication(PH\_CONFIG\_TIMEOUT, 0, 0)}

---

**ID** 184 **Balloter** **How submitted** Post ballot **Fixed in version**  
**Title** A0:A1 and A0:TX contention **Status** Done  
**C code files** **Owner** CWS **SCAT**  **BRAT** 0 **Recirc Clause** 13.4.8  
**Problem description**  
When a collision occurs, arb\_OK() returns true immediately. If immediate\_request() is TRUE at the same time, both A0:A1 and A0:TX could test true.  
**Bug fix description**

---

**ID** 185    **Balloter**                                 **How submitted** Post ballot                                 **Fixed in version**  
**Title**        link\_concatenation now unnecessary                                 **Status** Done  
**C code files**     **Owner**        CWS                 **SCAT**  **BRAT** 0         **Recirc Clause** 13.4.8

***Problem description***

link\_concatenation is replaced with grant\_self on all state transitions. Fixes problems with mutual exclusions and guarantees that a legacy link concatenation is not interrupted. See C code bug # 77.

link\_concatenation is unnecessary since grant\_self is now set for cases of concatenation. The TX:TX transition can now be written as end\_of\_packet && grant\_self

***Bug fix description***

---

**ID** 186    **Balloter**                                 **How submitted** Post ballot                                 **Fixed in version**  
**Title**        RX:A0 and RX:RX     **Status** Done  
**C code files**     **Owner**        CWS                 **SCAT**  **BRAT** 0         **Recirc Clause** 13.4.8

***Problem description***

RX:A0 and RX:TX are not exclusive. Outer parentheses are not conventional. Suggested replacement is:

!concatenated\_packet && !(fly\_by\_OK || grant\_self) && requesting\_port == NPORT

***Bug fix description***

---

**ID** 222    **Balloter**    MS     **How submitted** Post ballot                                 **Fixed in version**  
**Title**        RX:A2 transition     **Status** Done  
**C code files**     **Owner**        MT                         **SCAT**  **BRAT** 0         **Recirc Clause** 13.4.8

***Problem description***

Has no description - needs to be provided

***Bug fix description***

---

**ID** 242    **Balloter**    CWS     **How submitted** Post ballot                                 **Fixed in version**  
**Title**        Withdrawn LEGACY request during iso period                                         **Status** Recommend Reject  
**C code files**     **Owner**        JH                         **SCAT**  **BRAT** 0         **Recirc Clause** 13.4.8

***Problem description***

What happens if a naked PHY receives a grant for a LEGACY child request during the iso period and, when the grant is received, the child request is no longer present? Have to be careful not to use for async traffic, if possible. Withdrawn request should probably cause a null packet unless the naked phy knows the phase for sure.

***Bug fix description***

resolved in dealing with cycle starts - see SCAT #207

Not a bug ... yes A1:TXa deals with this case. own\_request and converted\_request are set by arb\_OK() and therefor can't change while in state A1. So the only change can be a withdrawn child LEGACY\_REQUEST. If we went into A1 because of LEGACY\_CHILD\_REQUEST rather than own\_request or converted\_request, then only choice on way out is A1:A2 or A1:TXa, neither of which should interrupt isochronous traffic and accidentally launch async traffic.

---

---

**ID** 257 **Balloter** **How submitted** Post ballot **Fixed in version**  
**Title** PH\_RESTORE\* and register 0 **Status** Done  
**C code files** **Owner** DW **SCAT**  **BRAT** 0 **Recirc Clause** 14

**Problem description**

14.8.2.3 needs to elaborate relationship between PH\_RESTORE\* and register 0 when used for interface initialization.

**Bug fix description**

---

**ID** 260 **Balloter** **How submitted** Post ballot **Fixed in version**  
**Title** clarify PH\_RESTORE\_NO\_RESET **Status** Done  
**C code files** **Owner** DW **SCAT**  **BRAT** 0 **Recirc Clause** 14

**Problem description**

Does Clause 14 description need to clarify the PH\_RESTORE\_NO\_RESET doesn't always mean a bus reset didn't happen ... just means we don't know yet.

**Bug fix description**

---

**ID** 268 **Balloter** CWS **How submitted** Post ballot **Fixed in version**  
**Title** Isoch Mechanisms - review PHY-Link chapters **Status** Done  
**C code files** **Owner** MT **SCAT**  **BRAT** 0 **Recirc Clause** 14

**Problem description**

PHY\_link chapters need reviewing/updating to support the Isoch Mechanisms

Incorporate cycle start description into draft (MT)

- 1) define values for xx, yy, and zz
- 2) DW: make sure that text includes PH\_ISOCH\_EVEN/ODD indications from the link to the PHY
- 3) DW: make sure that the correct requirements for knowledge of the isoch phase are incorporated for link initialisation

**Bug fix description**

Expanded into a general review of the timing parameters for the PHY/Link interface (see draft for new details). One challenge was that unlike P1394a, more than one clock cycle can separate the link and PHY component. So for these timings to have meaning, needed to specify on which side of the PHY/Link interface the measurement datum should live to account for flight time across the interface. And this also required bounding that flight time which, until now, was not formalized. Used a fudge factor of 4 cycles to account for a one way delay across the interface.

And while looking into the flight time problem noted that idel is not relevant for a synchronous interface.

---

**ID** 270 **Balloter** CWS **How submitted** Post ballot **Fixed in version**  
**Title** How does a slow legacy link detect S400 ack packet? **Status** Recommend Reject  
**C code files** **Owner** PhyDogs **SCAT**  **BRAT** 0 **Recirc Clause** 14

**Problem description**

Really a possible bug in 1394a. Seems like an S400 PHY could receive an S400 ack packet and assume fly-by or ack accelerated arbitration would be possible.

However, an S200 or S100 link would simply see a null packet (?) and consider the request canceled. Thus, it would be confused when the PHY granted for the now canceled request. Since the PHY doesn't know the link speed, not sure how this would be fixed. Not an issue if links are always S400 (as I suspect).

**Bug fix description**

If the PHY speed is faster than the Legacy link speed of S100 and S200, then the link should not use accelerations.

**ID** 271    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version**  
**Title**        Should receive cancel an immediate request?                    **Status** Done  
**C code files**    **Owner**        DW        **SCAT**  **BRAT**    0    **Recirc Clause** 14

**Problem description**

For robustness, it might be good to have a received packet cancel any queued immediate\_request in the PHY/Link. This could happen, for example, if an isoch packet gets launched in the async interval, and interrupts the period where the ack would be sent. Or if an async request accidentally gets sent during the isoch interval, and a clueless link tries to ack it. Should never happen, but if it did, seems cleaner not to send the ack at a later time. But not the same as 1394a and might be difficult to describe or build. Certainly an immediate\_request made during packet receive should be honored ... so would have to say something about canceling at the top of receive\_actions or something. Maybe we can discuss

**Bug fix description**

Add to text:-

If link gets an indication of DATA\_ON (start of packet) after it has started sending an immediate request, then if it subsequently gets a GRANT it has to drop the bus (it can't send the ACK - its too late to send the ACK)

---

**ID** 286    **Balloter**    DW                    **How submitted** Post ballot                    **Fixed in version**  
**Title**        Restore packet handing, including PIL-FOP                    **Status** Done  
**C code files**    **Owner**        DW        **SCAT**  **BRAT**    0    **Recirc Clause** 14

**Problem description**

Write up description (four quadrant matrix) of PHY-Link initialisation and nephew restore vs Parallel Interface and PIL/FOP

**Bug fix description**

---

**ID** 310    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version**  
**Title**        PH\_RESET treats cycle start requests differently from LPS reset                    **Status** Done  
**C code files**    **Owner**        DW        **SCAT**  **BRAT**    0    **Recirc Clause** 14

**Problem description**

14.5.1.6 documents the link initiated reset (PH\_RESET) and states:

"When the PHY receives an Interface Reset request, it will begin the initialization completion sequence defined in clause 14.3.4.1"

Clause 14.3.4.1 states "When the Initialization Completion Sequence is started, the PHY shall not queue any requests other than CYLE\_START until the PHY has sent the last bit of the unsolicited PHY register 0 status transfer. After sending the last bit, the PHY shall be able to accept and queue any transfer request that does not start until after the last bit of the PHY register 0 status transfer has been sent."

But 14.5.1.6 seems to contradict this slightly by concluding: "Once the Interface Rest request is issued to the PHY, the link shall not issue further transfer requests (Asynchronous, Iso-chronous, Cycle Start or Immediate) until the PHY sends the unsolicited PHY register 0 status transfer. The PHY is not required to enforce this constraint and should the link violate the constraint, the results are unpredictable."

Not necessarily contradictory, but the specified behavior seems arbitrarily different? Why not have PH\_RESET act just like LPS reset: namely, that link requests are flushed (except cycle start) until the reg 0 is delivered?

**Bug fix description**

eliminated the last paragraph of 14.5.1.6 so that the initialization completion sequence is used for link initiated resets

**ID** 330     **Balloter**   CWS                     **How submitted** Post ballot                     **Fixed in version**  
**Title**         PH\_ASYNC\_START should be PH\_SUBACTION\_GAP (Table 14-19)                     **Status** Done  
**C code files**     **Owner**         DW             **SCAT**  **BRAT**    0     **Recirc Clause** 14

**Problem description**

D[5] Subaction Gap corresponds to PH\_DATA.indication of PH\_SUBACTION\_GAP like you originally had (not the PH\_ASYNC\_START I told you to put in there). Also, I'd change the next sentence to say "After a bus reset, this is an implicit PH\_EVENT.indication of PH\_BUS\_RESET\_COMPLETE.

**Bug fix description**

**ID** 343     **Balloter**   CWS                     **How submitted** Post ballot                     **Fixed in version**  
**Title**         S800 Legacy Format???

**C code files**     **Owner**         DW             **SCAT**  **BRAT**    0     **Recirc Clause** 14

**Problem description**

Table 14-10 has an encoding for S800 legacy. Probably should be a reserved encoding.

**Bug fix description**

**ID** 356     **Balloter**   CWS                     **How submitted** Post ballot                     **Fixed in version**  
**Title**         Status Transfers while LPS is going away                     **Status** Done  
**C code files**     **Owner**         JH             **SCAT**  **BRAT**    0     **Recirc Clause** 14

**Problem description**

Need to make sure that status transfers are clearly interpreted when LPS goes away. (Issue brought up at OHCI).  
The notification of a bus reset (either with a bus status transfer over Ctl/D or with a PINT of PH\_RESTORE\_RESET) is unacknowledged by the link. So we have to be worried about the race condition in which the Link has taken LPS away, but the PHY hasn't recognized that fact yet and sends a bus reset indication. The link must still accept that status transfer even if it thinks LPS is gone.

Question on the table which remains is if a PHY should/shall not abandon a status transfer in mid-flight when it first detects LPS deasserted? One argument is that it doesn't matter: a PHY can either "turn off" immediately or finish a status transfer first provided it is self\_consistent in the setting or clearing of the missed\_reset queue (don't clear it unless that status transfer actually happened). The other argument is that it might make link processing a little more difficult if status transfers can just end abruptly with PCLK stopping (before the stop bit is reached).

**Bug fix description**

TLPS\_DISABLE is 25 us minimum, which is >2k PCLKs. So once LPS is deasserted by the link, you'll get at least 2000 PCLKs before it stops. So any in-flight status has time to finish and I would hope links have no problems cleaning up.

In the case we are doing a reset rather than a disable, PCLK never stops, so no problem there either, I think.

**Summary:-**

- 1) Links shall continue to receive status even after LPS is disabled for as long as PCLK's are happening.
- 2) PHYs shall finish any PINT transfer in progress when LPS is first detected as being absent. PHYs shall not start any new PINT transfers once LPS is detected as being absent.

<i>ID</i>	357	<i>Balloter</i>	CWS	<i>How submitted</i>	Post ballot	<i>Fixed in version</i>	
<i>Title</i>	LinkOn after PHY power-on while LPS already active				<i>Status</i>	Done	
<i>C code files</i>		<i>Owner</i>	JH	<i>SCAT</i>	<input checked="" type="checkbox"/>	<i>BRAT</i>	0 <i>Recirc Clause</i> 14

***Problem description***

Table 14-4 says that the persistence time from the point at which both LPS is active and LCtrl is one, after which the PHY shall not signal LinkOn is 500 ns max. Last paragraph of 14.4.1 says that after a power cycle, PHY (with Power class <=4) should assert LinkOn for a minimum of ~166.6 us. These seem slightly contradictory. Is the 166.6 us minimum to be met even if LPS is already on? Will software get annoyed when it sees LinkOn but has no way to clear it for the 166.6 us min (max is implementation dependent) rather than the normal 500 ns provided for in 14.4.1? (I seem to recall adding the 500 ns language for Intel so that they would know how long to wait after vectoring to a LinkOn interrupt to wait before clearing the interrupt and re-enabling the event mask.) I wonder if the intent (wake up OHCI if the PHY loses it's brain) would be met equally well by doing the 166.6us dance only if LPS is inactive. Seems like if LPS is active, the link already say the PHY power cycle with the stopping of PCLK and is simply waiting for it to come back on line.

***Bug fix description***

Clarify that LinkON isn't required if LPS is already running. The intention is to assert LinkOn for 166us (in the absence of LPS) or until LPS was asserted. Once LPS is asserted, then PCLK would be generated and there'd be no need for LinkOn.

---

<i>ID</i>	358	<i>Balloter</i>	CWS	<i>How submitted</i>	Post ballot	<i>Fixed in version</i>	
<i>Title</i>	LinkOn behavior				<i>Status</i>	Done	
<i>C code files</i>		<i>Owner</i>	MT	<i>SCAT</i>	<input checked="" type="checkbox"/>	<i>BRAT</i>	0 <i>Recirc Clause</i> 14

***Problem description***

P1394b did not copy all of the text from the registers clause 5B1 of 1394a-2000. As a result, I believe normative behavior for linkOn is missing. Specifically, 1394a-2000 says in the paragraph describing Watchdog " ... An interrupt condition is created either when any of the Loop, Pwr\_fail, Timeout, or Port\_event bits transition from zero to one or when the PHY detects the absence of LPS and any of these same bits are one."

I can not find any normative language in 1394b which supports the second part of the OR clause above ... but I suspect it is normative behavior which dictates that LinkON would be autonomously asserted whenever these bits are set and LPS is gone.

***Bug fix description***

Text from 1394a copied over (text was added at a late ballot stage, and not synchronised into 1394b).

---

**ID** 367     **Balloter** CWS     **How submitted**     **Fixed in version**  
**Title** PHY/Link initialization     **Status** Not yet done  
**C code files**     **Owner** JH     **SCAT**  **BRAT** 0     **Recirc Clause** 14

**Problem description**

Corner case I was considering was if the PHY/Link interface did it's PH\_ARBRST\_EVEN to complete the initialization as required, and then began to send DATA\_ON since it detected the PHY was in the "middle of a packet transfer". What would happen, I was wondering, if a PH\_ARBRST\_ODD came from the serial bus at that time? Seems like I need to repeat that to the link somehow, but the language currently only lets me send an IDLE after a DATA\_ON, not a status transfer.

I could try to remember the indication for later, but that goes against the norm of not queuing status for later. So I don't like that so much.

Behavior which seems acceptable to me include two variants:

Go ahead and send the PH\_ARBRST\_EVEN in the initialization completion sequence and go into DATA\_ON if the serial bus is "busy". But if an ARBRST\_ODD is generated/received by the PHY, use that as the synchronization point to complete initialization and begin to immediately repeat to the Link. Seems contrary to the language which requires DATA\_ON to finish with an IDLE since a sequence observed on the PHY link interface might be ARBRST\_EVEN | DATA\_ON | ARBRST\_ODD | DATAON ....

Additionally, this type of approach can lead to consecutive bus status transfers to the link of alternating phase indications. For example, if after sending the first ARBRST\_EVEN the interface block sees that the phy core is repeating an ARBRST\_ODD, then the link interface would see back-to-back status transfers of arb reset. Don't know if this would "confuse" links or not. The second approach is to delay the PH\_ARBRST\_EVEN which concludes the initialization until after the serial bus is out of the packet. So the PHY/Link interface sees DATAON | ARBRST\_EVEN | IDLE. This approach seems to promise only one indication to the link and always ends with an IDLE cycle on the interface. But it doesn't comply with the language of 14.3.4.1 since the DATAONs are used before the ARBRST\*.

**Bug fix description**

Change the "Otherwise, the Ctl bus will indicate that the bus is idle" language. Don't enforce an idle ... if after the PH\_ARBRST\* is sent we are seeing anything other than DATA\_PREFIX, SPEED, or data bytes, we'll jump right into repeating. This meets the intent, but not the letter.

I also became uncomfortable with the language in 14.3.4 which says "the PHY shall continue to indicate DATA\_ON while it is in a state in which it would normally assert anything other than IDLE on Ctl[0:1] if initialization of the PHY-link interface were complete"

This statement was written for 1394a and assumes that the IDLE state occurs within some reasonable latency. But in a B bus with S3200 and S1600 nodes, our S800 node might see DATA\_NULL for an indefinite period of time (although the scenario is admittedly very rare), interrupted only occasionally by gap tokens. (I guess the cycle start packet, if any, would break things up at least as often as 125 us.) So waiting to synchronize with an IDLE state may take much longer than in 1394a.

Based on this realization, propose to change the language in 14.3.3, opting instead to go with the same idea ... repeat DATA\_ON until we can synchronize with the bus which we can do when we see anything other than DATA\_PREFIX, SPEED, or DATA\_BYTES.

---

**ID** 256     **Balloter** DW     **How submitted** Post ballot     **Fixed in version**  
**Title** bus reset interrupt on restore     **Status** Done  
**C code files**     **Owner** DW     **SCAT**  **BRAT** 0     **Recirc Clause** 14.5

**Problem description**

Need to add text in the PHY/Link interface description to caution that a bus reset may occur at any time during the restore process.

**Bug fix description**

Not exactly sure where this should be said

---

**ID** 227     **Balloter**     MS     **How submitted** Post ballot     **Fixed in version**  
**Title**     Bus request timeouts     **Status** Done  
**C code files**     **Owner**     DW     **SCAT**  **BRAT**     0     **Recirc Clause** 14.5.1

**Problem description**

Links must not timeout bus requests due to restore initiated by a nephew PHY upon receipt of a bus request. The sort of timeout that might be implemented in a link is when it has made a request and there's NOTHING RECEIVED nor has it been granted for TBD microseconds. This would be analogous to timeouts in 1394a (where something being received acts as a cancel of the request).

At any rate, it seems that a bus-request timeout needs to be rather large (> 3ms) to be compatible with the restore mechanism, and be operationally OK. This needs to be advised so that shorter timeouts are not implemented.

**Bug fix description**

Overtaken by the re-work of the bring up of the interface - now the link receives a PH\_NO\_RESTORE immediately, and the PHY cancels the request, so there is no request to timeout.

---

**ID** 232     **Balloter**     DW     **How submitted** Post ballot     **Fixed in version** D104  
**Title**     New link cancellation rules     **Status** Done  
**C code files**     **Owner**     DW     **SCAT**  **BRAT**     0     **Recirc Clause** 14.5.1

**Problem description**

sometime during the period between BUS\_RESET or PH\_RESTORE\* and the register 0 transfers, the PHY needs to ignore or cancel most requests with cycle start being the exception.

**Bug fix description**

The text as David presented is a sufficient description ... the interface behaves as if the requests are continuously ignored and not queued by the PHY.  
C code fixed - basic approach was to have any PH\_BUS\_RESET\_START or PH\_RESTORE\_NO\_RESET clear all link requests and set a flag to ignore subsequent non cycle start bus requests until register zero is delivered.

---

**ID** 162     **Balloter**     **How submitted** Post ballot     **Fixed in version**  
**Title**     PHY Link handover     **Status** Done  
**C code files**     **Owner**     DW     **SCAT**  **BRAT**     0     **Recirc Clause** 14.6.3.2

**Problem description**

(text on p209, l10)

when phy hands bus over to link, spec leaves open the possibility that the link may drive CTL/DATA pins on cycle following PHY driving CTL and DATA to 0s. <"The link indicates ... asserting HOLD or TRANSMIT on the Ctl[0:1] lines as soon as possible ...> This revives the old bug for isolated systems - that the phy and link drivers could collide for some brief overlap period. This would foul up the interface if one of the isolation networks were in place (both annex J and bus holders get broken by overlap).

Propose to reinvent having the link start by driving 0s to both CTL and DATA lines - see A spec, Figure 6-8 in section 6.5.

**Bug fix description**

Change the spec to ensure a clash is avoided

---

**ID** 163    **Balloter**    **How submitted** Post ballot    **Fixed in version** D106  
**Title**    Upgrading requests    **Status** Done  
**C code files**    **Owner** DW    **SCAT**  **BRAT** 0    **Recirc Clause** 14.6.3.4

**Problem description**

When a beta link makes a request of the PHY, the format of the request can be "upgraded" from unspecified to BETA. Later, when transmit\_actions.c executes, the link is granted using this "upgraded" format.

The link uses the format returned in the PH\_ARB\_confirmation to match it's outstanding requests so that it knows which has been granted. For example, assume the link made a NEXT\_EVEN S400 BETA request and then a CURRENT S400 unspecified request. If a grant for a BETA format is returned, the link knows to send the NEXT\_EVEN packet rather than the current packet. So the link is essentially checking the format.

Now if a format gets upgraded, the link may be confused or will need extra checking. For example, if a CURRENT S400 unspecified format is requested, and the grant returned is for ASYNC, S400, BETA ... the link will need to understand what has happened (that unspecified became beta). Is this what we intended to do?

**Bug fix description**

The link needs to do the extra checking, and this is now documented in the PHY-Link interface

C code upgraded to send the right information.

Format 0 in table 14-12 needs to say "Unspecified" instead of "Legacy", and the words under Table 14-14 adjusted

**ID** 164    **Balloter**    **How submitted** Post ballot    **Fixed in version**  
**Title**    PHY cancellation of late link requests    **Status** Done  
**C code files**    **Owner** DW    **SCAT**  **BRAT** 0    **Recirc Clause** 14.6.3.4

**Problem description**

PHY cancellation of "late" link requests (not sure where this comment should go)

Setup: At link in the even phase, link issues a next\_odd (Req #1) and sometime later issues a current (Req #2). Subsequently, the link receives from the PHY an ARB\_RST\_ODD followed by a GRANT. Assuming Req #1 and Req #2 have the same speed and format, the link can not determine whether the GRANT was for Req #1 or Req #2.

Scenario A: Phy issued ARB\_RST\_ODD and GRANT after receiving Req #1 but before Req #2. Consequently, when Req #2 arrives, it is queued by the PHY.

Scenario B: Phy issued ARB\_RST\_ODD before receiving Req #1. Req #1 and Req #2 both received by PHY before it issues a GRANT. In this case, Req #2 has updated and replaced Req #1. After issuing the GRANT, no requests are pending in the PHY.

From the link's perspective, scenario A and scenario B look identical in terms of order of arriving indications from the PHY. But in scenario A, a request is still pending in the PHY and the link should not reissue, and in Scenario B no requests are pending and the link does need to re-issue.

Proposed fix is that after sending a GRANT to the link, the PHY discards all LREQs for the same "pipe" (async or isoch) as the GRANT until the link acknowledges the GRANT by taking possession of the PHY/Link interface. If so, then Scenario A would end up with no request pending in the PHY since it ignored Req #2 from the link.

**Bug fix description**

Agreed

<b>ID</b> 165	<b>Balloter</b>		<b>How submitted</b> Post ballot		<b>Fixed in version</b>
<b>Title</b>	Remembering request format				<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> JH	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0	<b>Recirc Clause</b> 14.6.3.4

**Problem description**

Must remember requested format when returning grant

Even if PHY choses to upgrade an unspecified request format to a beta format, the grant issued to the parallel link must be tagged with the requested format, not the actual format. This allows the link to match the grant and LREQ exactly, and avoid some error inducing confusion in some small corner cases.

**Bug fix description**

Done in the description

---

<b>ID</b> 229	<b>Balloter</b>	Biva	<b>How submitted</b> Post ballot		<b>Fixed in version</b>
<b>Title</b>	Simultaneous transmission of inband status transfers				<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> DW	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0	<b>Recirc Clause</b> 14.8.1.2

**Problem description**

I can't figure out how any of the inband status transfers can be sent simultaneously, but the text in 12.8.1.2 before Table 12-18 sure confuses this point. Recommendation is to clarify in 12.8.1.2 that either the bits don't happen simultaneously, or what interpretation should be given to various ambiguous bit pairs.

**Bug fix description**

Clarified

---

<b>ID</b> 204	<b>Balloter</b>		<b>How submitted</b> Post ballot		<b>Fixed in version</b>
<b>Title</b>	PIL-FOP negotiation				<b>Status</b> Recommend Accept in principl
<b>C code files</b>		<b>Owner</b> DW	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> <sup>^</sup> 0	<b>Recirc Clause</b> 15

**Problem description**

PIL-FOP negotiation should be performed at the time of speed signaling, rather than by straps etc. Also an "external" FOP should be supported.

**Bug fix description**

Extra bits are used in the speed negotiation. ("Done" in Connection Management Chapter)

---

<b>ID</b> 258	<b>Balloter</b>		<b>How submitted</b> Post ballot		<b>Fixed in version</b>
<b>Title</b>	PH_RESTORE for PIL/FOP				<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> DW	<b>SCAT</b> <input checked="" type="checkbox"/>	<b>BRAT</b> 0	<b>Recirc Clause</b> 15

**Problem description**

Where is the use of PH\_RESTORE for PIL/FOP described? 15.2.3 needs gchanged drastically. Add SCAT to make section match descisions from August BRC.

**Bug fix description**

---

*ID* 231 *Balloter* DW *How submitted* Post ballot *Fixed in version*  
*Title* FOP/nephew restore condition *Status* Done  
*C code files* *Owner* DW *SCAT*  *BRAT* 0 *Recirc Clause* 15.2.3  
*Problem description*

A FOP which is also a nephew (when it's only active port on the serial bus is in standby as a nephew) needs to initiate a restore if it receives a bus request from the PIL.

*Bug fix description*

Something like if the best request to forward on the senior port (which is in standby) is non-null, then restore? I'll try this route unless someone has a better idea ...

---

*ID* 230 *Balloter* DW *How submitted* Post ballot *Fixed in version*  
*Title* we need PH\_RESTORE\* defined for P2P *Status* Done  
*C code files* *Owner* DW *SCAT*  *BRAT* 0 *Recirc Clause* 15.4  
*Problem description*

To be added to Restore Notify encoding.

*Bug fix description*

*ID* 320 *Balloter* DW *How submitted* Post ballot *Fixed in version*  
*Title* Bus number register *Status* Done  
*C code files* *Owner* MT *SCAT*  *BRAT* 0 *Recirc Clause* 16  
*Problem description*

At the OHCI meeting, talking about 1394.1. OHCI is going to make the bus number register read-only, 0x3FF. There is a request that, since 1394b is a supplement/amendment to 1394, that 1394b might be a good place to indicate that the bus\_number register on nodes should not be set. This would make things compatible with 1394.1 use of this value.

I volunteered to get this into the 1394b specification. Now, just have to figure out where to put it.

David Wooten  
CWS:-  
Is this the bus\_id field of the NODE\_IDS register?

In which case, where to put it is simple. It goes in Clause 8.3.2.2.3 after P1394b has been renumbered to align to 1394-1995 (smile).

*Bug fix description*

Added "higher layer updates" section after PHY-link interfaces sections. Has new NODE\_IDS register descriptions

---

*ID* 364 *Balloter* CWS *How submitted* Post ballot *Fixed in version* 1.12  
*Title* Cycle start request to be ignored if not root *Status* Done  
*C code files* *Owner* CWS *SCAT*  *BRAT* 0 *Recirc Clause* 16  
*Problem description*

After a bus reset, the link is allowed to issue a cycle start request in the expectation that it will end up as root. The PHY is required to ignore this request if it turns out that the node is not root. Code to implement this needs to be added.

*Bug fix description*

*ID* 11     *Balloter* CWS     *How submitted* Post ballot     *Fixed in version* D1001  
*Title* layout     *Status* Recommend Accept  
*C code files* all     *Owner*     *SCAT*  *BRAT*     *Recirc Clause* 17

***Problem description***

line wraps, untidy comment positioning and excessive indentation

***Bug fix description***

reduce indentation to two spaces, tidy comment positioning, check to ensure no line-wraps  
Version D1001 created to facilitate dif's between ballot draft and subsequent versions  
All other C code changes are post-D1001

---

*ID* 207      *Balloter*      Biva      *How submitted* Post ballot      *Fixed in version* D106  
*Title*      Cycle Start Issues      *Status* Done  
*C code files*      *Owner* PhyDogs      *SCAT*  *BRAT* 0      *Recirc Clause* 17

*Problem description*

Hello Colin,

I am not sure whether I have already asked this before ... you may find it's a repetition.

I am a bit confused with the point at which a cycle start token appears on the bus and correspondingly the point at which "cycle start detected" status indication goes on the Link interface.

My understanding as per the Draft 1.00 c-code is as follows:

1. If the cycle master is a B node, then it embeds the CS (cycle start) token within the data prefix of the cycle start packet.  
Question: Does the cycle master need to send a status indication back to the Link indicating cycle start detected in this case? If it has to send the indication, then at what point should it send, after sending the token or after completion of cycle start packet transmission? Or is the indication not needed at all for the B cycle master Link?

2.

2A. The c-code does not support repetition of cycle start tokens during packet prefix repetition (vide wait\_Betaport\_event() function), but the send\_cycle\_start\_token flag gets set if a CS token is detected. The token gets retransmitted only as a part of boss\_end\_packet\_actions() [boss\_management\_actions] or in idle actions. The former case probably means retransmission by the current boss which has a cycle start token queued to be sent. The latter case is retransmission attempt in the Idle state, but liable to be stomped in a bus if DP is detected on a port. Is this the intended way a cycle start token should get propagated?

2B. Seems to me like a forwarding means which will probably maintain the protocol and move the B PHYs into isoch phase one after another, but does not guarantee any timing relation between the cycle start packet and the cycle start token. It seems to me that if a particular node quite far away from the root has a queued Isoch request for the next phase, it may also happen that the remote node

does not see the token at all, if a sufficiently wide Idle does not open up and the bus is beta-only (quickly starts off and continues with the async phase). Can there be a problem in such a case?

Are there any further implications here? The provision to tolerate cycle start token stomping is probably anyway needed atleast to handle the case of cycle start packet originating from a DS parent cloud, so the B PHYs have to generate the token only after their Links provide a cycle start packet received indication (provided such a function is supported).

#3) A related question here is that "shouldn't a B Link, which is generating an isochronous transmission request necessarily send a cycle start received indication if it receives a cycle start packet"? The reason I am asking this is for the situation where the cycle start packet is sent from a DS cloud and only one node has a B Link. This B Link has queued isoch requests, but if it does not generate the "cycle start packet received indication" over Lreq then the CS token will not get generated at all and hence the own PHY

arbitration phase will not change. Consequently the queued isochronous request will never get granted. Shouldn't it be mandated that a B Link, which has isochronous transmission active, should necessarily generate a CS pkt received indication over Lreq on detecting TCODE 8 (& 5 quadlet pkt). The Lreq start bit should also maintain the same max response timing as specified for generation of Isochronous requests over Lreq following pkt receive over D/Ctl lines in 1394a Link interface section. Is that right ?

#4) A further confusion is there with the status indication of async\_start and cycle\_start going at the same cycle to the B Link, using the bus status transfer format of Sec12.8.1.2. What will the B Link imply? Will it assume a null iso cycle (i.e. CS immediately followed by Async start, possible for beta only bus) or start of iso phase immediate after an async start (i.e Async Start immediately followed by a cycle start)? To me it seems that the current code tries to move CS token far away from a preceeding async start, by making it follow a packet. Hence the Link, observing both CS and AsyncStart indications coming together will treat it only as a null iso cycle. Is that right?

Am I missing some more fundamental implications behind the mechanism of iso cycle initiation in a betaonly or a hybrid bus? It will be very helpful to get some more light on this issue,

hence seeking your help one more time,

*Bug fix description*

The description has been annotated with reference numbers to ease the fix description.

#1. For a B Link which is cycle master, I believe it is easier to send the token indication to the link to help it keep it's LREQ's in phase. This "bug" has also been captured as C code bug #134 and will be resolved there.

#2A. Just a comment: I think we didn't attempt to forward CS tokens in start\_rx\_packet() because it may have been difficult to stomp (having to detect data\_coming concurrently with generating token at this point in the C code) and because there are cases in which we'd have to wait until IDLE anyway (such as CS being received while we already started repeating the CS packet payload). No action required for #2A.

#2B. Yes, there is a problem for an all B bus, if the cycle master doesn't send the CS token at the slowest port speed in the \_network\_ AND if the amount of time in A0:IDLE before the end of the isoch period doesn't exceed the symbol time of the slowest network port. The fix in the C code is to require the CS token to always be generated at an S100 symbol time for a B Bus, and to refrain from stomping in a B Bus. This means that PPM differences won't having us accidenatly stomping the guaranteed S100 CS token and we can regenerate the S100 timings in a B Bus without fear of overflow.

#3. Correct, it must be a requirement for B links with pending isoch traffic to generate cycle start token requests when a cycle start packet is received. Furthermore, this request needs to follow the same timings as a P1394a issued IsoReq to ensure that the token reaches the "root" before a subaciton gap pops. Recommended action is to have DRW improved the language in 12.5.2.1 to clarify that it is a must (not a may) if the link has isoch traffic queued, and that the timings must be obeyed.

#4) I can't figure out how any of the inband status transfers can be sent simultaneously, but the text in 12.8.1.2 before Table 12-18 sure confuses this point. Recommendation is to clarify in 12.8.1.2 that either the bits don't happen simultaneously, or what interpretation should be given to various ambiguous bit pairs.

New Isochronous Mechanisms defined and documented in e-mail "Isochronous Mechanisms Overhauled". This resolves all known cycle start issues, including the stomping problems noted in #2B. CST's are no longer stomped ... only repeated at top of RX for fixed time, etc.

---

<b>ID</b> 245	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b> D104
<b>Title</b>	Update references to Bport active in comments		<b>Status</b> Done
<b>C code files</b>	<b>Owner</b> JH	<b>SCAT</b> <input type="checkbox"/>	<b>BRAT</b> 0 <b>Recirc Clause</b> 17

**Problem description**

need to scrub the C code comments for references to bport active and change to bort on

**Bug fix description**

---

<b>ID</b> 246	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b> D105
<b>Title</b>	Untested port when all others are suspended		<b>Status</b> Done
<b>C code files</b>	<b>Owner</b> CWS	<b>SCAT</b> <input type="checkbox"/>	<b>BRAT</b> 0 <b>Recirc Clause</b> 17

**Problem description**

Need to resume all suspended ports and wait for the resumption to complete before starting to test the new connection.

**Bug fix description**

Do this at the same time as the restore test (when a new connection invalidates nephew qualification)  
 Also corrected the bug that resume actions needs to restore a nephews port in standby

---

**ID** 262     **Balloter**     CWS     **How submitted** Post ballot     **Fixed in version** D106  
**Title**     units for wait\_times     **Status** Done  
**C code files**     **Owner** JH     **SCAT**  **BRAT** 0     **Recirc Clause** 17

**Problem description**

change all wait\_times in the C code to use canonical S400 symbol times.

**Bug fix description**

New fundamental function wait\_PHY\_databit\_time - which waits for a time equal to one databit (related to PHY speed, databit before 8B10B encoding)  
new function wait\_Legacy\_time(n), where n is in units of 2 \* 100ns/BASE\_RATE (approx. 20 ns)  
all Legacy-related timings assumed to be expressed in these units  
wait\_next\_symbol replaced by wait\_symbol\_time  
stomping code improved to use packet speed quantisation, rather than PHY\_SPEED

Further tidying up of constants done in C Code version D107

---

**ID** 363     **Balloter**     CWS     **How submitted** Post ballot     **Fixed in version** 1.1  
**Title**     Power reset drivers on some signals in wrong process     **Status** Done  
**C code files**     **Owner** JH     **SCAT**  **BRAT** 0     **Recirc Clause** 17

**Problem description**

receive\_ok, rx\_ok, and arb\_reset are being reset after power\_reset by a different process than the one that normally drives the signals.

**Bug fix description**

receive\_ok and rx\_ok are now reset in receive\_ok\_monitor() while arb\_reset is reset in arb\_power\_reset()

---

**ID** 2     **Balloter**     CWS     **How submitted** Ballot comment     **Fixed in version** D1005  
**Title**     tpSig     **Status** Recommend Accept  
**C code files** data\_structures.h     **Owner**     **SCAT**  **BRAT** 414     **Recirc Clause** 17.1

**Problem description**

needs only the values L and H (no Z)

**Bug fix description**

remove Z:-  
typedef enum {L, H} tpSig;     // Differential signal on twisted pair

---

**ID** 3     **Balloter**     CWS     **How submitted** Not submitted     **Fixed in version**  
**Title**     register read/write should not be PHY\_arb\_req services     **Status** Recommend Reject  
**C code files** phy\_services, process requests     **Owner**     **SCAT**  **BRAT**     **Recirc Clause** 17.1

**Problem description**

**Bug fix description**

Formal reject, as a duplicate of #138

---

**ID** 4      **Balloter** CWS      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** loop\_to\_detect declaration      **Status** Recommend Accept  
**C code files** shared.h      **Owner**      **SCAT**  **BRAT** 426 **Recirc Clause** 17.1  
**Problem description**  
loop\_to\_detect declaration is alphabetically out of place  
**Bug fix description**  
move down four lines

---

**ID** 5      **Balloter** CWS      **How submitted** Post ballot      **Fixed in version** D1005  
**Title** bportT declaration      **Status** Recommend Accept  
**C code files** shared.h      **Owner**      **SCAT**  **BRAT**      **Recirc Clause** 17.1  
**Problem description**  
bportT is not used in process\_req.c  
**Bug fix description**  
change the comment  
// shared between main arb state machine, port.c, node.c, bport\_tx.c and dsport\_tx.c

---

**ID** 6      **Balloter** CWS      **How submitted** Post ballot      **Fixed in version** D1005  
**Title** reference to process\_req.c      **Status** Recommend Accept  
**C code files** shared.h      **Owner**      **SCAT**  **BRAT**      **Recirc Clause** 17.1  
**Problem description**  
references are made to the c code file name process\_requests.c  
**Bug fix description**  
change the comments (several places) to process\_req.c

---

**ID** 7      **Balloter** CWS      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** arbreqTdeclaration      **Status** Recommend Accept  
**C code files** shared.h      **Owner**      **SCAT**  **BRAT** 427 **Recirc Clause** 17.1  
**Problem description**  
arbreqT is used in port.c  
**Bug fix description**  
add port.c to the comment

---

**ID** 8      **Balloter** CWS      **How submitted** Post ballot      **Fixed in version** D1005  
**Title** inconsistent use of "the" and "main" in comments      **Status** Recommend Accept  
**C code files** shared.h      **Owner**      **SCAT**  **BRAT**      **Recirc Clause** 17.1  
**Problem description**  
"the" and "main" are used inconsistently in section comments  
**Bug fix description**  
amend appropriately

---

**ID** 16      **Balloter** CWS                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**        Move PH indications    **Status** Recommend Accept  
**C code files** various    **Owner**    **SCAT**  **BRAT** 420 **Recirc Clause** 17.1

**Problem description**

Move enum typedefs of PH indications to phy\_services.h

**Bug fix description**

Make the move. On further investigation, discovered that PH\_DATA\_indications of DATA\_PREFIX, DATA\_START, etc are not consistent with the definitions, which also do not permit a consistent description (both the servie model descriptions and the C code need changing)

General clean-up and scrub of the C code  
Note: Also need to clean up 15.2.3.3 on p245

---

**ID** 22      **Balloter** CWS                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**        PMD services need defining properly    **Status** Recommend Accept  
**C code files** data\_structures.h, phy\_services.c, port.c                      **Owner**    **SCAT**  **BRAT** 419 **Recirc Clause** 17.1

**Problem description**

A proper service interface should be used for PMD, rather than the use of variable shared between several modules and the PMD.

It needs to be made clear which port machine is controlling the PMD at any given time (i.e. the control signal for a PMD mux).

**Bug fix description**

Also BRAT #s 434, 435, 436, 437, 438, 440, 442, 447, 449, 450, 451, 453, 454  
Various C code modifications proposed for "to PMD" direction, similar changes also required for the "from PMD" direction.

---

**ID** 23      **Balloter** CWS                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**        DS arbitration signal encodings    **Status** Recommend Accept  
**C code files** data\_structures.h, dsport\_rx.c, dsport\_tx.c                      **Owner**    **SCAT**  **BRAT** 415 **Recirc Clause** 17.1

**Problem description**

DS arbitration signals are overloaded, and the encodings do not reflect this

These will have to be assigned numerical values to deal with the overloading  
Decode does not reflect overloading, encode with have to be updated

Overloading of ds arb states can not be easily resolved at the cable port; it needs to be at the arb state machine side of the FIFO per skid's recommendation. Additionally, there is no RX\_DATA arb state ... a stopped clock detector is needed.

**Bug fix description**

Also BRAT #s 416, 460, 463  
Process\_req has been modified to use PHY\_state to filter and resolve the overloading. And a caution about filtering of spurious RX\_DATA\_PREFIX indications has been added to dsport\_rx.c. Also, RX\_DATA\_PREFIX kicks off the extract clock circuit and a stop clock detector is used to determine when to push the ending arb state into the FIFO.

---

**ID** 24      **Balloter** Jim Skidmore                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**        RESET\_WAIT    **Status** Recommend Accept  
**C code files** data\_structures.h    **Owner**    **SCAT**  **BRAT** 418 **Recirc Clause** 17.1

**Problem description**

(Comment #11)  
Add RESET\_WAIT to enumerated list of wait\_times.

**Bug fix description**

Agreed. Note, while not strictly referenced in the C code, RESET\_WAIT is referenced on the state machine. However, it is not used directly ... it is summed with a variable named reset\_duration.

---

**ID** 25      **Balloter** CWS                      **How submitted** Ballot comment                      **Fixed in version** D103  
**Title**      Arb\_state enum values    **Status** Recommend Accept  
**C code files** data\_structures.h    **Owner**    **SCAT**  **BRAT** 417 **Recirc Clause** 17.1

**Problem description**

These values will have to be re-assigned, to deal with the overloading which is used for DS arbitration line states. Also one or two new ones will have to be introduced to deal with the fact that in DS mode, different encodings are used for RX and TX versions of the same arbitration states

**Bug fix description**

See # 23 (BRAT #415)

---

**ID** 26      **Balloter** CWS                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**      reset\_notify clearing    **Status** Recommend Accept  
**C code files** phy\_services.h, port.c    **Owner**    **SCAT**  **BRAT** 422 **Recirc Clause** 17.1

**Problem description**

Deal more sanely with the issue of reset\_notify clearing on successful transfer to the link,

**Bug fix description**

Also BRAT #s 456, 457, 495

insert

```
void PH_EVENT_response(boolean event_OK);
```

amend port.c

```
PH_EVENT_indication(missed_reset ? PH_RESTORE_RESET : PH_RESTORE_NO_RESET, 0, 0);  
// PHY/Link interface is required to provide PH_EVENT_response of event_OK  
// to clear the missed_reset flag  
}  
in_standby = FALSE;  
}
```

```
void PH_EVENT_response(boolean event_OK) {  
if (event_OK) missed_reset = FALSE;  
}
```

amend comment in reset\_start to

```
PH_EVENT_indication(PH_BUS_RESET_START, 0, 0); // Optional upon reentry to R0 from R1  
// PHY/Link interface is required to provide PH_EVENT_response of event_OK  
// to clear the missed_reset flag
```

---

**ID** 31      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** ds\_portR, rx\_S200, and rxS400 variables      **Status** Recommend Accept in principl  
**C code files** phy\_services.h      **Owner**      **SCAT**  **BRAT** 421 **Recirc Clause** 17.1

**Problem description**

(Comment #01)  
The ds\_portR, rx\_S200, and rxS400 variables should be arrays.

Change line 40-41 from:

```
RX_signal ds_portR;  
  
boolean rx_S200, rx_S400;
```

to:

```
RX_signal ds_portR[NPORT];  
boolean rx_S200[NPORT], rx_S400[NPORT];
```

**Bug fix description**

Noted that connect\_detect and bias\_detect were also per-port variables. #ifdef unsubscribed ... #else ... #endif region added to phy\_services.h

---

**ID** 32      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** sync\_error\_signal and sync\_lost\_signal variables      **Status** Recommend Accept  
**C code files** shared.h      **Owner**      **SCAT**  **BRAT** 423 **Recirc Clause** 17.1

**Problem description**

(Comment #02)  
The sync\_error\_signal and sync\_lost\_signal variables are per-port-variables, and should be declared as arrays, or moved to the port.c module (sec 16.2.2, pg. 290).

**Bug fix description**

Not appropriate in port.c since shared among modules ... #ifdef subscripted region added to shared.h

---

**ID** 33      **Balloter** Clay E Hudgins      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** bport\_active      **Status** Recommend Accept  
**C code files** shared.h      **Owner**      **SCAT**  **BRAT** 424 **Recirc Clause** 17.1

**Problem description**

Consider replacing "should" with "shall."

**Bug fix description**

Note, this is a mechanism, not a requirement  
Amend comments to:-  
boolean bport\_active;      // Set to true to instruct the Beta port to commence operating  
                                 // Set to false to instruct the Beta port to cease operating

and similar change made to the dsport\_active signal

---

**ID** 35      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** bport\_active      **Status** Recommend Accept  
**C code files** shared.h      **Owner**      **SCAT**  **BRAT** 425 **Recirc Clause** 17.1

**Problem description**

(Comment #03)  
The bport\_active variable is a per-port-variable, and should be declared as an array, or moved to the port.c module (sec 16.2.2, pg. 290).

**Bug fix description**

Not appropriate in port.c since shared among modules ... #ifdef subscripted region added to shared.h. Also corrected for newly added dsport\_active signal.

---

**ID** 88      **Balloter** CWS      **How submitted** Post ballot      **Fixed in version** D1005  
**Title** BIAS\_HANDSHAKE -> RECEIVE\_OK\_HANDSHAKE      **Status** Recommend Accept  
**C code files** data\_structures.h, port.c      **Owner**      **SCAT**  **BRAT**      **Recirc Clause** 17.1

**Problem description**

One occuence of BIAS\_HANDSHAKE remains in port.c and data\_structures.h which should be replaced/removed. Also, Table 11-3 should have the comments merged and BIAS\_HANDSHAKE removed. Also, remove from Table 15-8

**Bug fix description**

replace/remove

---

**ID** 137      **Balloter** CWS      **How submitted** Post ballot      **Fixed in version** D106  
**Title** Missing PMD services      **Status** Recommend Accept  
**C code files** phy\_services.h      **Owner** JH      **SCAT**  **BRAT**      **Recirc Clause** 17.1

**Problem description**

For posterity, the following signals should be replaced with PMD services. Extremely low priority ... entered to allow sticky to be removed but not forgotten.

bias\_detect, connect\_detect, ds\_portR, local\_plug\_present, rx\_S200, rx\_S400, signal\_detect, ds\_portT, ds\_portTarb, ds\_portTspeed, tpBias

**Bug fix description**

---

**ID** 138      **Balloter** CWS      **How submitted** Post ballot      **Fixed in version**  
**Title** Register read/write services      **Status** Done  
**C code files** phy\_services.h      **Owner** MT      **SCAT**  **BRAT**      **Recirc Clause** 17.1

**Problem description**

Ideally, register read/write requests should be a separate service indication since not all implementations use LREQ. Not a very high priority, but it is nice to have a separate service model for the register block. Additionally, the read/write services, regardless of how they are defined, are not currently described in Clause 15 service model.

Recommendation would be to remove PH\_REG\_READ and PH\_REG\_WRITE from C code completely, and optionally have Mike add a new service description in the text.

**Bug fix description**

removed from C code

Service description TBD

---

**ID** 304     **Balloter** CWS                     **How submitted** Post ballot                     **Fixed in version** D107  
**Title**         arb\_timer units are not consistent                     **Status** Done  
**C code files**     **Owner**         MT         **SCAT**  **BRAT** 0     **Recirc Clause** 17.1

**Problem description**

arb\_timer is declared as type timer and is compared against some items which are declared as Legacy\_wait\_times (units of S400 symbols) and some declared as wait\_times (units of seconds).

Given an interpretation that arb\_timer should time seconds, then the conflict is limited to timing SPEED\_SIGNAL\_LENGTH, DATA\_PREFIX\_HOLD, and MIN\_IDLE\_TIME. Speed signal timing happens on S3:S0 and could be rewritten such that S3 was given state actions that did a wait\_Legacy\_time(SPEED\_SIGNAL\_LENGTH), thereby removing the arb\_timer usage. However, the usage of SPEED\_SIGNAL\_LENGTH and DATA\_PREFIX\_HOLD in start\_rx\_packet is harder to avoid. We need some way to exit the function early if a Legacy packet never materialized while enforcing the min time (quantized to Legacy) if a Legacy format does occur. Since wait\_Legacy\_time can not be interrupted, it is difficult to address. Solutions might be to 1) define a conversion function from Legacy\_wait\_times to units of seconds, or 2) redefine arb\_timer to count S400 symbols and quantize the ~13 other constants (normatively), or 3) invent a new timer which counts S400 symbols exactly.

MIN\_IDLE\_TIME needs some thought ... not sure if it must be quantized or not.

Also, noted that DATA\_PREFIX\_TIME is not referenced in the C Code.

**Bug fix description**

Decided to build a function, Legacy\_time which returns the duration (in seconds) of an exact number of specified S400 symbols. The function was written using BASE\_RATE. In doing so, realized the wait\_PHY\_databit\_time didn't really need to be an abstract function ... could express in terms of BASE\_RATE as well. So wait\_symbol\_time and wait\_Legacy\_time were re-written to use BASE\_RATE.

Finally, decided that MIN\_IDLE\_TIME did not need quantization, and removed DATA\_PREFIX\_TIME from enumerations.

SCAT items:

- 1) remove wait\_PMD\_databit\_time from any text (not in 1.06 anyway)
- 2) change the S3:S0 transition as noted in the pdf ... basically use the Legacy\_time function to convert SPEED\_SIGNAL\_LENGTH units

---

**ID** 318     **Balloter** CWS                     **How submitted** Post ballot                     **Fixed in version** D107  
**Title**         PH\_EVENT\_Response in wrong place                     **Status** Done  
**C code files**     **Owner**         CWS         **SCAT**  **BRAT** 0     **Recirc Clause** 17.1

**Problem description**

Probably should be defined in phy\_services.h rather than port.c (it is one per link, not one per port for sure). And process requests should really be the thing which sits and spins on it and sets missed reset accordingly.

**Bug fix description**

---

**ID** 331     **Balloter** CWS                     **How submitted** Post ballot                     **Fixed in version** D1071  
**Title**         Remove link type of no\_link                     **Status** Done  
**C code files**     **Owner**         CWS         **SCAT**  **BRAT** 0     **Recirc Clause** 17.1

**Problem description**

Doesn't do anything useful

**Bug fix description**

*ID* 335    *Balloter*    CWS                    *How submitted* Post ballot                    *Fixed in version* D1071  
*Title*        rename Legacy\_child\_request                    *Status*    Done  
*C code files*    *Owner*        CWS    *SCAT*  *BRAT*    0    *Recirc Clause* 17.1  
*Problem description*  
as Legacy\_junior\_request  
*Bug fix description*

---

*ID* 342    *Balloter*    CWS                    *How submitted* Post ballot                    *Fixed in version* D1071  
*Title*        arb\_timer no longer needed in node.c                    *Status*    Done  
*C code files*    *Owner*        CWS    *SCAT*  *BRAT*    0    *Recirc Clause* 17.1  
*Problem description*  
shared.h can be updated ... perhaps arb\_timer simply moves to arb.h  
*Bug fix description*

---

**ID** 96      **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D1005  
**Title**        revamp of loop healing                      **Status**    Recommend Accept  
**C code files** various                      **Owner**                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2

**Problem description**

Skid raised a number of issues with loop healing, Wooten added some features as part of DevCon presentation, and Hauck uncovered some possible errors in related pieces of code. The C code is being updated to match the revised DevCon presentation and does include optimizations for single port PHY's and isolated nodes.

**Bug fix description**

1) in\_test\_interval is no longer required. Effectively, the node controller can simply check !need\_new\_LTP && test\_interval\_timer >= TEST\_INTERVAL to track if a particular holding register value has been in effect for the required time.

2) In initialize\_arbitration(), it is incorrect to clear restore\_request and con\_mgmt\_request. Doing so causes these flags to be reset with each bus reset, possibly before they have been serviced. For example, if we still need to process a restore when an unrelated bus reset occurs, we will forget that we have a restore to do later. Nothing at that point causes restore\_request to be set again.

3) con\_mgmt\_request has been renamed to loop\_test\_request. And rather than a set/clear flag used as a semaphore between otherwise async processes, it is asserted to continuously by the loop\_tester until the need to test is removed (either by completion of the test or by loss of the test port). Consequently, it is only driven in loop\_test\_interval\_actions().

4) tx\_format and tx\_speed are presumed to hold the format and speed of the current or last packet transmitted. This is important so that the rule re: S100 downshifts can be implemented correctly in boss\_end\_packet\_actions(). So any time we transmit \_any\_ packet on the bus, we should set these correctly to enable the downshift test. Consequently, tx\_format and tx\_speed are now set within con\_mgmt\_granted().

5) restore\_request is now dealt with independently from loop\_test\_request. A restore\_request will cause arbitration without needing to also set loop\_test\_request. It was much simpler to handle them as separate cases.

6) Processing of received ATTACH\_REQUESTS is now done differently. Each port implements an attach variable which is set when the port is ready to attach for any reason, including receipt of an ATTACH\_REQUEST. This ATTACH flag "arms" the port and is used in reset actions to begin propagation the reset, much like resume is used. The node controller will ask to release the bus as a direct or indirect result of any ATTACH bit being set. So con\_mgmt\_granted simply waits for the indication from the loop\_test\_interval\_actions() before releasing the bus. received\_attach is obsolete.

7) Skid's comment #12 is basically accepted.

8) As discussed in Skid's #13, test\_port\_selector had several race conditions. It was rolled into loop\_test\_interval\_actions(). However, the part that monitored whether the arb state machine was stuck in a loop was retained but renamed to loop\_detector().

9) loop\_test\_interval\_actions() now uses a new function attach\_pending() to determine if there are unfinished attaches to complete.

10) loop\_test\_interval\_actions() completely revamped. Based on skid's #14 and David's DevCon presentation. All asynchronous handshakes strengthened. Special support for single port PHY's added per DevCon.

-----

11) Untested\_actions() completely revamped per Skid's #18 and the Devcon presentation. The concept of a single\_port\_node was introduced and allowed to head straight to attach\_request (no LTS sent). And isolated\_node arms the attach immediately and then lets the node controller release the bus. This fixes a bug in which we were tying up the PHY/Link interface for >> a max packet time. Most asynchronous handshakes strengthened to avoid feared race conditions. An the concept of an attach flag was introduced to arm the port such that reset.c would propagate the joining reset on those ports (use to be just active or resuming ports).

-----

12) In process\_req.c, separately specified the requests for connection management: now loop\_test\_request and restore\_request. D1.01 code didn't properly cause a request to be asserted for restores.

-----

13) For loop testing, single\_port\_phy's and isolated\_nodes no longer hold onto their local bus after sending an attach\_request. Much like for resume, reset\_detected() has been modified to detect the incoming reset on the test port in these cases.

-----

14) To propagate resets on newly attaching ports, reset\_start\_actions() and reset\_wait\_actions() were modified to work on ports with the attach flags set. There is still as sort of race condition here ... once a port sees bus\_initialize\_active, it will clear attach and head off to active. So for a brief period, both attach and active are false. But before reset\_start\_actions can complete, we should make it to active. As an observation, we shouldn't ever find ourselves in reset\_wait with the attach flag set.

-----

15) transmit\_actions() was modified to account for canceled link requests and loop\_test\_requests. It was observed that Beta link requests could be cancelled at any moment such as just after boss\_end\_packet\_actions decided to jump to TX, or after start\_tx\_packet, etc. So we have to be careful not to use our entry into TX for the wrong packet type. (Like we entered for an immediate\_request which got canceled and then tried to do a loop test packet. Invalid because we didn't wait for a subaction gap.) So at entry into TX and at each wait thereafter until the link is granted, the various possible transmit requests are prioritized and considered. This mechanism also allows the removal of loop\_test\_request by the node controller without worrying about what the arb state machine is doing. If we came into TX because of a loop\_test\_request which is no longer present, we gracefully terminate with a null packet if we can't make use of the visit to TX for anything else.

-----

16) process\_requests would only clear in\_packet if the last symbol had set packet\_ending AND there was something else in the FIFO. As a result, the clearing could be delayed for some time and, if the port status changed from !active to active, the FIFO would start throttling itself preventing the reset machine from seeing the bus reset which would follow an attach request, for example.

---

<b>ID</b>	341	<b>Balloter</b>	CWS	<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	
<b>Title</b>	On restore, nephew should hold the bus with a DN rather than a DP			<b>Status</b>	Recommend	Reject	
<b>C code files</b>		<b>Owner</b>	CWS	<b>SCAT</b>	<input type="checkbox"/>	<b>BRAT</b>	0 <b>Recirc Clause</b> 17.2

**Problem description**

Based on e-mail exchange entitled "Change to Standby-Restore text", it appears that the upcoming draft will say that the nephew picks up the bus after the uncle transmits a restore packet by sending a DP. This should probably be DN (Wooten admitted that he man not have the proper symbols.)

==== 02/16/2001 08:21:30 PM ==== Jerry Hauck =====

Oops ... wasn't quite correct on this one. Resolution on SCAT #285 shows that intention has changed to take control by transmitting an S100 Beta packet with no payload. So text may need modified in all related clauses (connection mgmnt, arb, link).

**Bug fix description**

Current text in Connection Management is OK (matches SCAT #285)

No relevant text in either arb or link

---

**ID** 34      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** restore\_port      **Status** Recommend Accept  
**C code files** node.c      **Owner**      **SCAT**  **BRAT** 428 **Recirc Clause** 17.2.1

**Problem description**

(Comment #04)  
Table 16-6, pg. 284, function restore\_port  
For-statement incorrect.

**Bug fix description**

Change pg. 284 line 23 from:

for (j = 0; j++; j < NPORT)  
to:

for (j = 0; j < NPORT; j++)

---

**ID** 36      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** loop healing - in\_test\_interval      **Status** Recommend Accept in principl  
**C code files** node.c      **Owner**      **SCAT**  **BRAT** 429 **Recirc Clause** 17.2.1

**Problem description**

(Comment #12)  
Table 16-6, pg. 286-287, function con\_mgmt\_granted  
The in\_test\_interval variable not needed. The test\_interval\_timer variable should only be reset upon sending a new LTP packet. If a short-bus-reset is to initiated, there should be no packet-end sequence.  
Change pg. 286 lines 28-55, pg. 287 lines 2-3 from:

```
} else { // request for loop test
...
}
to:

} else { // request for loop test
if ((link == Legacy_Link) && (breq == FAIR_REQ || breq == PRIORITY_REQ)) {
breq = NO_REQ; // Cancel the request
PH_ARB_confirmation(PH_LOST, 0, 0); // And let the link know
}
PH_DATA_indication(PH_DATA_PREFIX, 0, 0); // Send notification of bus activity

for (i = 0; i < NPORT; i = i + 1)
portTarb(i, DATA_PREFIX); // Ensure the bus is held
with DATA_PREFIX.

max_occupancy_timer = 0; // Start the occupancy timer.
release_bus = FALSE;

in_control = TRUE;
while (!release_bus)
if (need_new_LTP) {
send_LTP;
test_interval_timer = 0;
need_new_LTP = FALSE;
}
received_attach = FALSE; // Clear flag if set

start_tx_packet(S100, LEGACY); // Send null packet to clean up,
in_control = FALSE; // then release the bus.
if (!isbr_OK) // The isbr_OK flag is set in untested_actions.
stop_tx_packet(DATA_END, S100, DATA_END_TIME, LEGACY, NPORT);
}
```

**Bug fix description**

JH] Basically agreed. Need to add a PH\_DATA\_indication of DATA\_END before the call to stop\_tx\_packet, and the received\_attach flag is no longer required. release\_bus became loop\_test\_request and is only tested, never set. Also, the port sets isbr and allows this routine to set isbr\_OK. The port can't know the arb phases and understand if isbr\_OK is allowed.

See #96

---

**ID** 37      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** test\_port\_selector - reselection of aborted port      **Status** Recommend Accept in principl  
**C code files** node.c      **Owner**      **SCAT**  **BRAT** <sup>^</sup> 430 **Recirc Clause** 17.2.1

**Problem description**

(Comment #13)  
Table 16-6, pg. 287, function test\_port\_selector  
There's no mechanism to keep an aborted port from being re-selected.

There's a code race condition which could result in the while loop getting stuck. When a port completes the loop-test, its port\_under\_test flag is cleared by loop\_test\_interval\_actions. At the same time, the release\_bus flag is set, which causes con\_mgmnt\_granted to terminate and the arb-state-machine to transition to the R0 reset-start state. But this, in turn, causes the con\_mgmnt\_request flag to be cleared.

Now suppose that this code (test\_port\_selector) executed before the arb-state-machine transitioned to the R0 state. A new test\_port would be selected and con\_mgmnt\_request would again be set. But con\_mgmnt\_request would be cleared upon entry into R0 (see initialize\_arbitration), and there's no mechanism for it to be set again. The following while loop would be stuck. It's waiting for port\_under\_test to be cleared by loop\_test\_interval\_actions, which is waiting for in\_control to be set, which is set upon entry into con\_mgmnt\_granted, but which won't be entered because the con\_mgmnt\_request flag was cleared by reset\_start\_actions (see initialize\_arbitration).

Also, considers what happens if a node has more than one untested port, and while the selected test-port is waiting for the test-interval to elapse, one of the other untested ports receives an attach-request symbol? This will cause the bus to be released and a short-bus-reset to be started, but the port\_under\_test flag won't be cleared, so the test\_port\_selector routine will be stuck in the while loop.

To fix this, add code to exit the while loop if a bus-reset is started.

Change lines pg. 287 15-25 from:

```

for (i = 0; i < NPORT; i++) { // selects ports in turn - important if testing a particular port
    // is aborted
    ...
}

```

to:

```

for (i = 0; i < NPORT && !bus_initialize_active; i = i + 1) {
// Selects ports in turn - important if testing a particular port is aborted.
// Select unique port to test if none already selected (do not need to search from port 0),
// but don't select a port that is currently receiving a dominant LTS value.
if (untested[i] &&
    !((test_interval_timer >= TEST_INTERVAL) && !need_new_LTP &&
      ((portRarb[i] == DATA_BYTE) &&
        (current_data[i] & 'h3F) > HR_test_value) || ((current_data[i] & 'h80) != 0))) {
    received_LTS = 0; // once in 64, this will cause a phantom potential collision
    port_under_test[i] = TRUE;
    test_port = i;
    con_mgmnt_request = TRUE; // and request the bus

    while (port_under_test[i] && !bus_initialize_active)
        ; // and sit here while the port is tested
    test_port = NPORT;
    port_under_test[i] = FALSE;
    con_mgmnt_request = FALSE; // don't need to request the bus now
}
}

```

*Bug fix description*

[JH] Agreed in principle, but implementation has radically changed. The test\_port\_selector routine has been merged in with the loop\_test\_interval\_actions() routine and the asynchronous handshakes revised. Consequently, I believe the race conditions have been corrected. Specifically, loop\_test\_interval\_actions() selects a valid test port based on some refined criteria to make sure we don't select a port which will have to be aborted. port\_under\_test is then set true which allows the given test\_port to know it has been addressed by any send\_attach or loop\_disable commands. The port, once commanded to attach, is responsible for scheduling the attach in a fashion that is synchronized with the rising edge of bus\_initialize\_active. Only after the port has successfully synchronized with the reset can loop\_test\_interval\_actions continue and select a new test port. The request to gain control (loop\_test\_request) is now a level signal, not a set/clear semaphore. So as long as the test port needs service, con\_mgmt\_granted will eventually be visited. No more deadlock. And in all cases, testing of a given port is guaranteed to conclude with clearing port\_under\_test.

[JH] The desire to keep an aborted port from being reselected was addressed by Wooten's DevCon presentation. So this is accepted in principle. As written, the proposed fix relies on current\_data[i] holding the last received LTS. This inspiration allowed us to always use current\_data[i] for the comparison rather than relying on received\_LTS to be set (the source of other race conditions). Consequently, received\_LTS is removed.

See #96

---

**ID** 38      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** loop\_test\_interval\_actions      **Status** Recommend Accept in principle  
**C code files** node.c      **Owner**      **SCAT**  **BRAT** <sup>^</sup> 431 **Recirc Clause** 17.2.1

**Problem description**

(Comment #14)  
Table 16-6, pg. 287, function loop\_test\_interval\_actions  
Loop-healing logic needs to wait for the test\_interval\_timer to increment to at least TEST\_INTERVAL time before comparing LTS values. If an attach-request is received on any port, then processing in this function should be skipped (the processing in untested\_actions should control the release of the bus, etc.).

I can't see any need for the in\_test\_interval flag--all references to this flag may be deleted.

I suggest the following to replace the loop\_test\_interval\_actions function in its entirety.

```
void loop_test_interval_actions() { // continuously running
    static int collision_count;

    if (power_reset) {
        collision_count = 0;
        in_control = FALSE;
        HR_test_value = new_test_value();
        while (power_reset)
            ;
        return;
    }

    if ((test_port != NPORT) && port_under_test[test_port]) {
        // Delay testing until the test port selector has initialized a new port to test.

        if (received_attach) // Attach-request symbol rcv'd on some untested port (not necessarily the test_port).
            ; // Do nothing

        // Check if the test-port is receiving a dominant LTS from its peer port.
        else if ((test_interval_timer >= TEST_INTERVAL && !need_new_LTP &&
            (received_LTS & 'h3F) > HR_test_value) || (received_LTS & 'h80) != 0) {
            // We've waited long enough for the last xmitted LTP to reach all
            // nodes in the network, and the received LTS is dominant, so abort
            // testing on this port (go to next).
            port_under_test[test_port] = FALSE;
            release_bus = TRUE;
            while (in_control)
                ;
            return;
        }

        // Check if there's a collision between the xmitted LTS and the rcv'd LTS.
        else if (in_control && test_interval_timer >= TEST_INTERVAL && !need_new_LTP &&
            (received_LTS & 'h7F) == ((HR_G << 6) | HR_test_value)) {
            // There's a collision between xmitted LTS and received LTS.
            if ((USING_EUI) && (collision_count == 0))
                EUI_sequence = 0; // reset the EUI sequence
            collision_count = collision_count + 1;
            if (collision_count == COLLISION_LIMIT) {
                loop_disabled[test_port] = TRUE; // place port in loop disabled state
                port_under_test[test_port] = FALSE;
                release_bus = TRUE;
                while (in_control)
                    ;
                collision_count = 0;
                return;
            }
            HR_test_value = new_test_value(0);
            HR_G = (HR_G == 0) ? 1 : 0;
            need_new_LTP = TRUE;
            return;
        }
    }
}
```

```

}

// Check if the test-port can be attached.
else if (in_control && test_interval_timer >= TEST_INTERVAL && !need_new_LTP) {
    // Test interval has expired, and we're in control, without a collision
    collision_count = 0; // Clear any bogus collisions
    HR_mode = ATTACH_IN_PROGRESS; // Should not receive attach-request on any port
    // once this is set.
    need_new_LTP = TRUE;
    while (need_new_LTP)
    ;
    send_attach = TRUE; // Flag for the test port
    while (send_attach) { // Wait for the port to complete
    if (!(untested[test_port] || (!in_control))) {
        // If the port has shut down for some reason or on bus reset...
        send_attach = FALSE;
        port_under_test[test_port] = FALSE;
        release_bus = TRUE;
        while (in_control)
        ;
        return;
    }
    }
    release_bus = TRUE;
    while (in_control)
    ;
    port_under_test[test_port] = FALSE;
    return;
}
} else {
    collision_count = 0;
    send_attach = FALSE;
}
}

```

**Bug fix description**

[JH]Disagree that we need to wait so long to compare LTS to HR. Generation bit was included so that we could quickly detect collisions without having to wait for full intervals. However, can't abort too quickly or assume no collision too quickly. Basically, a new\_ltp guarantees that the generation number has changed and that the old HR <> the new HR. So if we get LTS=HR, we note a collision immediately. But as long as LTS <> HR, we have to wait a full test interval before drawing a conclusion. This change on aborting is captured in the DevCon presentation and included in the latest implementation.

[JH] Agreed on processing outstanding attaches before selecting a port to test. No need to test if attach is going to complete and force a new test interval anyway.

[JH] Agreed on in\_test\_interval. It has been removed.

See #96

---

<b>ID</b> 84	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b> D1005
<b>Title</b>	send_LTP() does not produce LTP format defined in 15.5.4	<b>Status</b>	Recommend Accept
<b>C code files</b>	node.c	<b>Owner</b>	<b>SCAT</b> <input type="checkbox"/> <b>BRAT</b> <b>Recirc Clause</b> 17.2.1

**Problem description**

15.5.4 shows 3F in the PHY\_ID position for an LTP. However, send\_LTP() fills that field with zeros.

**Bug fix description**

Insert  
tx\_phy\_pkt.phy\_ID = 0x3F;

---

**ID** 90      **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D1005  
**Title**            "standby\_reset" should be "missed\_reset" in comment                      **Status** Recommend Accept  
**C code files** node.c                                      **Owner**                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2.1  
**Problem description**

**Bug fix description**

change it

---

**ID** 95      **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D106  
**Title**            Wait more times through the reset loop before declaring a loop?                      **Status** Done  
**C code files** node.c                                      **Owner**    CWS                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2.1  
**Problem description**

Current code declares a loop must be present if we get to reset\_count > 2. Since we count from 0, I think this means that we go into R0 4 times before assuming a loop. This may be okay.

My fear was that we weren't waiting long enough. For suspend/resume and loop breaking, lots of code defers an inbound bus reset while bus\_initialize\_active is true due to an ongoing reset. This period could last a long time (twice a long bus reset plus some?). So it seems like the node sending the "inbound bus reset" may go through R0 a few times until it is recognized. Maybe the first time is a short bus reset, the second is a long, the third is long again and hopefully seen in a worst case non-looping topology. But we are getting pretty close to that magic number of 4 ....

**Bug fix description**

Set the test to > 3

---

**ID** 106      **Balloter**    Biva                      **How submitted** Post ballot                      **Fixed in version** D1006  
**Title**            restoring multiple ports                      **Status** Recommend Accept in principle  
**C code files** node.c                                      **Owner**                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2.1  
**Problem description**

I have a doubt with the following portion of the c-code in con\_mgmt\_granted function:

```
(Draft1.01 Page 286, Line 25 - 27)
"restore_request = FALSE;
for (i = 0; i < NPORT; i++) // check if any more to do
    if (restore[i]) restore_request = con_mgmt_request = TRUE;"
```

In case the restoration of one port gets over at a time when the restore flag for another port has got set, then this would immediately set the restore\_request and con\_mgmt\_request flags. It may even get a grant immediately. But, the restore flag does not necessarily indicate that that port is ready for transmitting the restore config packet. It may be waiting in the start\_port() routine in P.300 L.51 (possibly still in the process of training). In that sense, shouldn't we use the bport\_sync\_OK flag to qualify that the port is ready for resume?

Can the code be changed to:

```
if (restore[i] && bport_sync_OK[i])
    restore_request = con_mgmt_request = TRUE;
```

**Bug fix description**

fix is necessary at top of con\_mgmt\_granted as well, to make sure that the port we pick to restore is ready as well. So all tests of restore[i] in this routine are qualified with bport\_sync\_ok[i] as well.

---

<b>ID</b>	111	<b>Balloter</b>	Biva	<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	
<b>Title</b>	test interval for no active ports				<b>Status</b>	Done	
<b>C code files</b>	node.c	<b>Owner</b>	DW	<b>SCAT</b>	<input checked="" type="checkbox"/>	<b>BRAT</b>	<b>Recirc Clause</b> 17.2.1

**Problem description**

The textual description in Sec11.7.12 (No Active Ports) indicates that the test interval may be set to 0 if the PHY currently has no ports in the active state. Is this supported by the C-code ?

**Bug fix description**

I don't believe the described behavior is desired or necessary. The text should be changed. A related discussion is in the e-mail thread "RE: isolated nodes and such". Basically, we now allow single-port PHY's to skip the establishing dominance phase. Isolated nodes with multiple ports do not skip the dominance phase because they may still cause a loop or suffer from a race condition in which they send an attach out port #0 at the same time a connection to another isolated node reports an attach\_request inbound on port #1. Now we may form a loop, etc. if we aren't really careful. Since it doesn't take an isolated node long to establish dominance, we opted not to make the optimization and deal with all of the hazards. So I consider this closed provided Wooten updates the text. See SCAT #161

---

<b>ID</b>	114	<b>Balloter</b>	CWS	<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	D1006
<b>Title</b>	Sending ATTACH_IN_PROGRESS LTP can negate short bus reset benefit				<b>Status</b>	Recommend	Accept
<b>C code files</b>	node.c	<b>Owner</b>		<b>SCAT</b>	<input type="checkbox"/>	<b>BRAT</b>	<b>Recirc Clause</b> 17.2.1

**Problem description**

When a PHY decides to send an ATTACH\_REQUEST, it does so at the same time it requests to send the ATTACH\_IN\_PROGRESS LTP. The time to send the LTP in the current spec is 160ns + 660ns + 160ns. Additionally, before a BUS\_RESET could be sent, another call to start\_tx\_packet for 160ns is required. As a result, once ATTACH\_REQUEST is scheduled, the PHY can't head off to R0 for another 1.14 us.

If the node receiving the ATATCH\_REQUEST responds with a short bus reset quickly, it will expect the peer PHY to be in R0 within 1.26 us. Otherwise, the short RESET turns into a long reset.

This leaves 100 ns margin at best. Consequently, the ATTACH\_REQUEST should be delayed until the LTP has been sent.

**Bug fix description**

Fix was slightly different than proposed. The send\_LTP and con\_mgmnt\_granted loop routines were rewritten such that send\_LTP always concludes with a call to start\_tx\_packet. As a consequence, when a node is ready to send the ATTACH\_IN\_PROGRESS LTP, it will only take 660ns + 160ns + 160ns before R0 can be visited. This leaves >260 ns margin which seems sufficient.

---

<b>ID</b>	119	<b>Balloter</b>	CWS	<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	
<b>Title</b>	restore				<b>Status</b>	Recommend	Reject
<b>C code files</b>	node.c	<b>Owner</b>	JH	<b>SCAT</b>	<input type="checkbox"/>	<b>BRAT</b>	<b>Recirc Clause</b> 17.2.1

**Problem description**

Restore needs some work. For example, the uncle doesn't start sending DP or SPEED until it wins arbitraiton. But the nephew expects it right away.

**Bug fix description**

No issue - Nephew does in fact wait.

---

*ID* 124    *Balloter* CWS                    *How submitted* Post ballot                    *Fixed in version* D104  
*Title*        node\_status\_monitor() accounting                    *Status* Done  
*C code files* node.c                                    *Owner*        JH                    *SCAT*  *BRAT*                    *Recirc Clause* 17.2.1

*Problem description*

Gut feel on this one is that the counting of suspended\_ports in node\_status\_monitor is not as desired. Specifically, it seems that some ports which are in P7 or P8 are counted as suspended ports (since in\_standby isn't true yet). Is that desired?

Also, a port in P12 isn't counted as suspended if untested\_fault is true, but is counted if loop\_disabled is true.

*Bug fix description*

change in\_standby to be set on entry to P7 and P8 (rather than P9) so that all of P7, P8, P9 and P10 count as "in\_standby".

Add untested\_fault into the suspended test

Set untested\_state in untested\_actions() and clear untested\_state immediately before exiting untested\_actions

connection\_status - implement correctly the comment which says  
// check for continuous signaling or bias in relevant port states (P1, P2, P3, P4, P7, P8 and P11)  
currently the test does not include P8, and it needs to, and it needs to include P10 to prevent falling thru to the next block, also it does not capture P4. Fix by replacing these tests by tests on explicit port states.

Issue of turning on toning after turning off the port - this needs to be done relatively quickly, to prevent a disconnect from being detected at the far end. Proposal is to do this in the same block of code if receive\_ok goes away (in Beta mode).

Fix the comment in the next section to include the standby state

Also, resume\_all\_ports should test for P3, P4 and P5

---

**ID** 127    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D106  
**Title**            Legacy loop while already in P11:Untested                    **Status** Done  
**C code files** node.c                                    **Owner**            JH            **SCAT**  **BRAT**            **Recirc Clause** 17.2.1

**Problem description**

If a border node finds a loop in the a cloud, it sets force\_retrain on any beta port which is in P2:Active. The intent is to force the P2:P11 transition to P11:Untested. In P11:Untested, the first thing we do is force the far port to resync with us (causing him to break out of whatever he was doing). If successful, we then as a border send and LTS with ATTACH\_IN\_PROGRESS until the Legacy loop is resolved. Then we move into our normal testing loops.

The question, what should happen to a port which is already in P11: Untested when a legacy loop is discovered? Currently, the C code will do nothing. We'll continue to send whatever LTS we were sending, etc. And the far end may actually send us an ATTACH\_REQUEST to which we will never be able to respond since we don't see bus\_initialize\_active ever go false. Consequently, the far end will be forced to send the bus reset which we can't hear either.

Bottom line seems to be we need to make sure the port exits early from any wait actions if it won't be able to process/hear the incoming ATTACH\_REQUEST.

**Bug fix description**

A fix has been detailed on paper, but not yet entered into the C code.

We decided that this might be okay as is, mostly. Specifically, a node may be stuck in P11 due to a legacy loop with bus\_initialize active high. So an inbound ATTACH\_REQUEST or BUS\_RESET can't be serviced. If a peer port (also in P11) sends a BUS\_RESET, it will put itself into P2 ad expect the reset to finish. But since the reset handshake isn't received from the "stuck" PHY, the peer effectively gets stuck in bus reset itself since reset\_complete() never test true. So enough trips through bus reset and the peer will force all beta ports to retrain. The original stuck device will see the loss of sync and exit, P11 heading off to the loop\_disabled\_state.

Need to check that this sequence really works. Specifically, the P2:P11 with force\_retrain actually results in the peer port heading from P11 to P12. (Not clear if it currently does, since it attempts to retrain immediately at the top of P11. Just retraining doesn't guarantee a peer will see loss of sync (Comma/training symbols received during the packet phase don't cause errors!). Once in P12, we need to make sure entry back into P11 occurs. Maybe need a longer wait of not sending signaling to force the visit to P12?

Also, force retrain is cleared in the top of PRX1 and also forces a PRX4:PRX1 transistion. So PRX1 could clear it before P11 sees it. Recommended solution is to keep PRX1 from trying to manipulate/test force\_retrain. Instead, it is simply used to force P2:P11 and then is cleared in P11 after the retrain is forced.

Or we could simply have force\_retrain set by the loop detector even if the port is in P11 and use it to exit all of the waits early?

Fix is to remove all pretence of "retraining" (which has been, in effect, already disabled in the port as it did not work properly) and explicitly to force a disconnect, using the mechanism recently invented for the case of a restore fail. The previous code did this for the case of ports in P2 - which went to P11 and immediately turned off bport. The far side would see the sync fail and go to disconnected, which would in turn force the local side eventually to wind up in disconnected. The fix is more explicit, and covers all cases, wherever the port state happens to be.

---

**ID** 133    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D1006  
**Title**            False dominance detection in loop healing                    **Status** Recommend Accept  
**C code files** node.c                                    **Owner**                                    **SCAT**  **BRAT**            **Recirc Clause** 17.2.1

**Problem description**

If HR\_MODE == ATTACH\_REQUEST because a remote controlling node is performing an attach, chances are that a local test port will be selected and stay selected because it thinks it is dominant if a full 8 bit compare is done with the received LTS. Then when the HR\_MODE is changed back to LTP\_TEST, the selected test might not be dominant any longer, but will take a full TEST\_INTERVAL to determine that.

**Bug fix description**

Fixed in revised C code

---

**ID** 237     **Balloter**    DW                                      **How submitted** Post ballot                                      **Fixed in version** D104  
**Title**         restore packet from Uncle                                      **Status**    Done  
**C code files**                                      **Owner**         JH                      **SCAT**  **BRAT**    0    **Recirc Clause** 17.2.1

**Problem description**

when PHY receives restore packet from Uncle. This is not repeated on the PHY-link interface

**Bug fix description**

Amend C code (change TRUE to FALSE, and change the comment)

The collective fix formalized the PH\_RESTORE\_\* indications and register 0 transfers, making sure they were only sent to the Nephew link as early as possible, etc.

**ID** 263     **Balloter**                                      **How submitted** Post ballot                                      **Fixed in version** D106  
**Title**         removal of EU164 and COLLISION\_LIMIT change                                      **Status**    Done  
**C code files**                                      **Owner**         JH                      **SCAT**  **BRAT**    0    **Recirc Clause** 17.2.1

**Problem description**

remove the associated C Code for EU1 64. Also, as a consequence, COLLISION\_LIMIT has been reduced to 7 which is a C Code and text change

**Bug fix description**

**ID** 339     **Balloter**    CWS                                      **How submitted** Post ballot                                      **Fixed in version** D1071  
**Title**         bias should be false for a beta\_mode\_only\_port                                      **Status**    Done  
**C code files**                                      **Owner**         CWS                      **SCAT**  **BRAT**    0    **Recirc Clause** 17.2.1

**Problem description**

The lines in bias\_status

```
if (Beta_mode_only_port[i])
```

are, I think, a hangover from some ancient code which was trying to be clever about receive\_OK. I can't find any use in the code which takes advantage of this forced state. In particular, in a bi-lingual port, bias is FALSE unless we have incoming bias generated by a peer DS port. Given that we're AC connected, and Beta mode only, I think that bias will "naturally" always be FALSE, and if we're going to force it to anything, then we should force it to FALSE.

port status bit: I wonder if this should be receive\_OK rather than bias . . . this seems to be the only possible reason for setting bias to TRUE all the time on a Beta mode only port (see 1). Just a thought in passing.

**Bug fix description**

test removed, so that bias is now always FALSE for a Beta mode only port  
Also allowed some clean-up of use of Beta\_mode\_only\_port

port status bit is already defined as receive\_OK  
But PHY response packet still used bias, contrary to its documentation, so this was changed to receive\_OK

**ID** 18       **Balloter**    CWS                                      **How submitted** Ballot comment                                      **Fixed in version** D1005  
**Title**         definition of autocrossover\_supported                                      **Status**    Recommend Accept  
**C code files**    port.c                                      **Owner**                                      **SCAT**  **BRAT**    433 **Recirc Clause** 17.2.2

**Problem description**

This is an implementation dependent constant

**Bug fix description**

Change from boolean to a #define

**ID** 19      **Balloter**    CWS                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**        Min\_port\_speed comment                                      **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT**    432 **Recirc Clause** 17.2.2

**Problem description**

Min\_port\_speed was deleted from the port register map during SCAT review. This comment needs to be updated to reflect this decision.

**Bug fix description**

update comment

---

**ID** 20      **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D1005  
**Title**        use of Beta\_mode\_only                                      **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2.2

**Problem description**

comment does not reflect the name change of Beta\_mode\_only to Beta\_mode\_only\_port

**Bug fix description**

update comment

---

**ID** 27      **Balloter**    CWS                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**        Delay in activate\_connect\_detect                                      **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT**    439 **Recirc Clause** 17.2.2

**Problem description**

Clarify use of delay in activate\_connect\_detect

**Bug fix description**

add comment  
// also ensures handshake times for both modes

---

**ID** 28      **Balloter**    CWS                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**        receive\_speed\_indication                                      **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT**    446 **Recirc Clause** 17.2.2

**Problem description**

receive\_speed\_indication looks for a start bit. If it finds one, then it samples the next 6 bit positions and constructs "rs" which holds the decimal equivalent of the binary tones. If a start bit tone was followed by no speed encoding, then rs = 0 after the 6 samples. As a result, received\_speed is set to -1 (received\_speed = rs - 1;) and EVT\_RECEIVED\_SPEED is signaled.

It may be argued that listening\_for\_speed will never be true unless both ends of a connection have already shifted into sending a speed\_code. If so, then we should never run into this problem (i.e., at least one speed bit will always be set when we are listening). But the proof escapes me and it seems real implementations may end up sending a few empty tone frames before getting their speed signals out.

**Bug fix description**

Obviously correct solution would be to bracket the last two executable lines of received\_speed\_indication with if rs > 0 . That is, only calculate received\_speed and trigger EVT\_RECEIVED speed if some real speed bits were found.

---

**ID** 29      **Balloter**    CWS                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**            error in comment in standby\_initiator\_actions()                      **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT** 455 **Recirc Clause** 17.2.2

**Problem description**

comment says suspend\_target instead of standby\_target

**Bug fix description**

amend comment

---

**ID** 39      **Balloter**    Jim Skidmore                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**            Toner    **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT** 443 **Recirc Clause** 17.2.2

**Problem description**

(Comment #05)  
Table 16-7, pg. 291, function toner  
For-statement incorrect.

**Bug fix description**

Change pg. 291 line 47 from:

```
for (i = 0; i++; i < 3) { // send three bits or spaces  
to:
```

```
for (i = 0; i < 3; i++) { // send three bits or spaces
```

---

**ID** 40      **Balloter**    Jim Skidmore                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**            signal\_detect race condition    **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT** 444 **Recirc Clause** 17.2.2

**Problem description**

(Comment #15)  
Table 16-7, pg. 291, function signal\_detect\_OK  
Eliminate possible race conditions with signal\_detect\_monitor, avoids setting  
sd\_detected FALSE for a short period (which adversely affects other code which  
monitors this flag).

**Bug fix description**

Change line 30 from:  
if (x) sd\_detected = FALSE;

to:  
if (x) sd\_detected = signal\_detect;

[JH] Agreed, but changed comment as well.

---

**ID** 41      **Balloter**    Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title**      toner - code imporvement      **Status**    Recommend Accept in principl  
**C code files** port.c      **Owner**      **SCAT**  **BRAT** <sup>^</sup> 445 **Recirc Clause** 17.2.2

**Problem description**

(Comment #16)  
Table 16-7, pg. 291, function toner  
A suggestion for code simplification which (IMHO) better illustrates the intended purpose.

Change lines 42-54 to:

```
while (toning || send_speed) {
    t_ack = we_agree && send_speed;
    t_speed = send_speed ? (PHY.port_speed[pn] + 1) : 0;
    for (i = 0; i < TONE_INTERVAL; i = i + 1) {
        if (i == 0 ||                // start bit
            i == 1 && t_ack ||        // ack bit
            i >= 2 && i <= 4 && (t_speed & (1 << (i - 2)) != 0)) // speed bits
            send_tone;
        else
            wait_time(TONE_DURATION);

        wait_time(SPEEDCODE_BIT_INTERVAL); // inter-bit gap

        if (i == 4) {
            if (t_speed != 0) // speed code bits sent
                signal(EVT_SENT_SPEED);
            if (t_ack) // ack bit sent
                signal(EVT_SENT_ACK);
        }
    }
}
```

**Bug fix description**

[JH] Agreed in principle, but the specific code above doesn't meet the intent of waiting until the last possible point to sample the latest we\_agree and port\_speed signals. So the spirit of the change was accepted, but the actual code is slightly different. Parens also added to send\_tone().

---

**ID** 42      **Balloter**    Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title**      received\_speed\_indication      **Status**    Recommend Accept  
**C code files** port.c      **Owner**      **SCAT**  **BRAT** 448 **Recirc Clause** 17.2.2

**Problem description**

(Comment #06)  
Table 16-7, pg. 292, function receive\_speed\_indication  
For-statement incorrect.

**Bug fix description**

Change pg. 292 line 22 from:

```
for (i = 0; i++; i < 6) { // now look for six speed code bits
to:
```

```
for (i = 0; i < 6; i++) { // now look for six speed code bits
```

---

**ID** 43      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** connection\_status race condition      **Status** Recommend Accept  
**C code files** port.c      **Owner**      **SCAT**  **BRAT** 452 **Recirc Clause** 17.2.2

**Problem description**

(Comment #17)  
Table 16-7, pg. 294, function connection\_status  
Avoid race condition in transition to untested state.  
At pg. 294 line ~45, insert following just before return statement:

```
while (!untested_state) // wait till in untested_state  
;
```

**Bug fix description**

[JH] Accepted. Removed comment about race condition from code as well.

---

**ID** 44      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title**      untested\_actions - various      **Status** Recommend Accept in principle  
**C code files** port.c      **Owner**      **SCAT**  **BRAT** 458 **Recirc Clause** 17.2.2

**Problem description**

(Comment #18)

Table 16-7, pg. 299-300, function untested\_actions

1) Not enough time given for current S100 symbol (already in portT) to be transmitted; the data-prefix symbols may not be transmitted.

At pg. 299 line ~3, insert following just after "portT.arg = ARB\_STATE;" statement:

```
wait_next_symbol(S100);
```

2) The con\_mgmnt\_request signal should be set only when node is not already in control of bus.

Change pg. 299 line 46 from:

```
con_mgmnt_request = TRUE; // request the bus  
to:
```

```
if (!lin_control)  
con_mgmnt_request = TRUE; // request the bus
```

3) The while-loop (waiting for node to gain control of bus) should exit when sync is lost.

Change pg. 299 line ~48 from:

```
while (lin_control && !seen_reset && !bport_sync_ok) {  
to:
```

```
while (lin_control && !seen_reset && bport_sync_ok) {
```

4) Need to clear port\_under\_test flag is attach-request received.

At pg. 299 line ~43, insert following just after "untested = FALSE;" statement:

```
port_under_test = FALSE;
```

Need to make sure that port\_under\_test flag is cleared if we lose sync.

At pg. 300 line ~13, insert following just before return statement:

```
port_under_test = FALSE;
```

5) If port becomes loop-disabled, need to ensure that flags are cleared and bus released.

At pg. 300 line ~14, insert following just before "} // end of loop while untested":

```
if (loop_disabled) {  
untested = FALSE;  
port_under_test = FALSE;  
release_bus = TRUE;  
return;  
}
```

6) The untested\_fault flag is redundant. Replace all occurrences of untested\_fault with loop\_disabled. Affects the P11:P12 transition in Figure 11-2, pg. 176, as well.

**Bug fix description**

1) [JH] Point is taken, but preferred approach is to not schedule S100 arb symbols in start\_port() or start\_resume\_port(). Instead, starting arb indication will be given a speed of DEFAULT, marking the starting arb states as purely elastic.

2) [JH] This fix is no longer required. The port does not directly request the bus. Instead, attach is used to indicate when the node controller should cause a bus reset to happen.

3) [JH] Current implementation is different but agrees in spirit. If the port ever loses sync, untested\_actions must conclude.

4) [JH] Current implementation uses a combination of attach and untested to communicate to the test controller when the port is "done". Basically, once a command is sent to the port, the controller waits for either attach to be true or untested to be false to conclude testing. A new test port can only be selected if any pending attach is also completed. So the test\_port is completely done if attach and untested are both false.

5) [JH] Agreed in principle, but current implementation has a different structure. The node controller instructs the port that a loop has been found ... the port responds by clearing untested and sets loop\_disabled. On seeing !untested, the node controller clears the port\_under\_test, etc. and releases the bus.

6)[JH] Not entirely true. Intent was that loop\_disabled indicates if we are in P12 because of a loop we found. Untested\_fault was intended to indicate if start\_port() failed or we loss sync. This would happen, for example, if the far end was in the disable

state or could happen if the far end went to P12 first (initiated the loop disable). On a bus reset, we would only force ports out of P12 which were there because of a loop.  
(This point now accepted, see SCAT #337)

[JH] I was concerned with the asynchronous nature of clearing loop\_disabled by reset.c. Would it be possible for port to set loop\_disabled but, before the state machine transitions were evaluated, a reset cleared it? If so I would end up in P2 rather than P12! I decided not to worry about it. The found\_loop signal from the node controller causes the port to set loop\_disabled. found\_loop in turn is only asserted when we have control of the bus, and we don't release control of the bus until the port acknowledges found\_loop. At that point, there is a call to start\_tx\_packet() before we can get into R0. This should be plenty of time for the state machine to move to P12.

See #96

---

<b>ID</b> 45	<b>Balloter</b> Jim Skidmore	<b>How submitted</b> Ballot comment	<b>Fixed in version</b> D1005
<b>Title</b>	loop_disabled_actions	<b>Status</b> Recommend Accept	
<b>C code files</b> port.c	<b>Owner</b>	<b>SCAT</b> <input type="checkbox"/> <b>BRAT</b> 459	<b>Recirc Clause</b> 17.2.2

**Problem description**

(Comment #19)  
Table 16-7, pg. 300, function loop\_disabled\_actions

In the P12: Loop-Disabled state, the port is inactivated (see activate\_connect\_detect). The port will continue to send tones in order to maintain connectivity with its peer port.

The peer port, in turn, will lose synchronization, detect that it is no longer receiving a continuous signal, set its untested\_fault flag and also transition into the P12: Loop-Disabled state.

In order that the the local port not immediately transition back into the the P11: Untested state, the port must wait for its peer port to become inactive before exiting.

**Bug fix description**

At pg. 300 line ~23, insert the following just after the "activate\_connect\_detect(0);" statement:

```
while (receive_ok && loop_disabled)
; // Wait til peer become inactive or loop_disabled flag is cleared.
```

---

<b>ID</b> 80	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b> D1005
<b>Title</b>	Loss of DC Connection Not Noticed, never tones again	<b>Status</b> Recommend Accept	
<b>C code files</b> port.c	<b>Owner</b>	<b>SCAT</b> <input type="checkbox"/> <b>BRAT</b>	<b>Recirc Clause</b> 17.2.2

**Problem description**

Consider Node B which is connected to Node D with a DC path. Furthermore, assume Node D is off. Node B will attempt toning. After debouncing the DC connection and hearing no return tones, it will try bias. After a return bias is not received, the current C code leaves Node B listening for tones, but not sending tones. (Is this intentional ... I can't remember?)

Now if the DC path is removed, I don't think the C code takes note. It will still not initiate toning. Now a new connection (DC or otherwise) can be made to Node B without any notice and again without cause Node B to tone. If the new connection were made to another Node in the same state as Node B, then neither would initiate toning and the connection wouldn't become active until a power reset.

Seems like either Node B needs to revert to toning if bias is unsuccessful, or the loss of a DC connection should be noted and somehow cause toning to be turned back on.

All of this assumes local\_plug\_present was not implemented.

**Bug fix description**

Insert two lines before line 30 on p294

```
if (!dc_connected) toning = TRUE; // might have been waiting for dc_connected
// peer to wake us up
```

**ID** 86      **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D1005  
**Title**        dsport\_active and PMD...(SELECT\_DS\_PORT) not symmetric with bport\_a    **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2.2

**Problem description**

dsport\_active not being set and cleared symmetrically with bport\_active. Both are intended to control operation of FIFO and data/arb transmitter. Neither affects toning or bias/connection detection.

**Bug fix description**

adjust to make symmetric - see also #79

---

**ID** 87      **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D104  
**Title**        To Interrupt or not on Restore ... and what indication to Uncle?                      **Status**    Done  
**C code files** port.c                                      **Owner**                      SB                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2.2

**Problem description**

restore\_actions() sets port\_event to true, but fails to give a PH\_EVENT\_indication(PH\_INTERRUPT) which is the norm. But it does give a PH\_EVENT of PH\_RESTORE\_\*. Both the PH\_INTERRUPT and PH\_RESTORE\* have encodings for the parallel PHY/Link interface.

**Bug fix description**

See SCAT #157

---

**ID** 89      **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D1005  
**Title**        crossover can now be a local variable within connection\_status                      **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2.2

**Problem description**

Only used locally, now that we have a PMD service for it.

**Bug fix description**

move definition

---

**ID** 91      **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D1005  
**Title**        con\_mgmnt\_request set for restore?                      **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2.2

**Problem description**

Couldn't figure out how a restore would cause con\_mgmnt\_request to be set. Seems like at the time restore\_request is set by restore\_actions, con\_mgmnt\_request would need to be set as well??

**Bug fix description**

See #12 in C Code bug #96

---

**ID** 97      **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D1005  
**Title**        receive\_ok late when taking P0 -> P11 -> P2                      **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2.2

**Problem description**

For single port PHY's, the time spent in P11 is quite short (just the time to achieve synchronization and to send the ATTACH\_REQUEST). If two such peer ports enter P11 at maximally different times (one port goes to P11 after sending the last bit of it's speed, the other port is waiting for all 6 speed bits before declaring EVT\_RECEIVED\_SPEED ... so diff is 3 tone bit times or so), then receive\_ok may not yet be set on when P11:P2 is taken.

Note that receive\_ok is set when the local port heads to P11, but depending on the time constants, etc. receive OK may shut off before the far port starts signaling. And the start up time for receive\_OK once signaling is heard was specified to be 2 calls to signal\_detect\_OK.

This might only be a sim problem when connectivity time constants are shortened (allowing receive\_ok to be cleared quickly), but isn't obviously correct as written.

**Bug fix description**

Basic change was to connection\_status(). When in the active, untested, resuming states where we have already determined we have (or had) a continuous signal, let receive\_ok simply be a function of bport\_sync\_ok. This means it will be set properly before heading to P2:Active and also means that loss of sync will quickly send us out of P2.

---

**ID** 98      **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D103  
**Title**        LTS "Packet" violates padding and elasticity assumptions.                      **Status**    Done  
**C code files** port.c                                      **Owner**        JH                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2.2

**Problem description**

As documented in e-mail thread "Problem with LTS's", the lack of a speed code before the leading DATA\_PREFIX of an LTS causes the received to see the "packet" as an S100 legacy packet. And when pad bytes don't occur in the proper places, error counts increment and some real LTS values are ignored.

Additionally, the LTS sequence as defined can be extremely long (several isoch periods). As such, it can cause FIFO overrun since it contains no deletable symbols.

**Bug fix description**

Fix was to define LTS to start with SPEEDb followed by DP, and to have bport\_rx.c only push different LTS's into the FIFO. Finally, the FIFO overrun has been changed so that the last thing (like an ATTACH\_REQUEST) isn't lost.

---

**ID** 99      **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D103  
**Title**        No need to check for ATTACH\_REQUEST after sending one anymore                      **Status**    Recommend Accept  
**C code files** port.c                                      **Owner**                      **SCAT**  **BRAT**                      **Recirc Clause** 17.2.2

**Problem description**

Can't think of any scenario which would result in a multi-port PHY sending an ATTACH\_REQUEST on the test\_port and then receiving an ATTACH\_REQUEST on the test\_port?

Seems to me that before an ATTACH\_REQUEST is sent by a non single port phy, dominance is first established. And if it is, then by definition the neighboring PHY connected to the test port could \_not\_ have established dominance and scheduled an ATTACH\_REQUEST.

So why in the code/algorithm have we put the effort into checking for either BUS\_RESET or ATTACH\_REQUEST to return after we sent and ATTACH\_REQUEST?

Sure, it might be more robust, but I can't think of a valid topology which would let me test it.

**Bug fix description**

Test for ATTACH\_REQUEST removed

---

**ID** 102    **Balloter**    CWS    **How submitted** Post ballot    **Fixed in version** D1005  
**Title**    negotiated\_speed - no power\_reset value given    **Status** Recommend Accept  
**C code files** port.c    **Owner**    **SCAT**  **BRAT**    **Recirc Clause** 17.2.2

**Problem description**

No power\_reset value given for negotiated speed. C code / comments seem ambiguous on whether: value should start at 000, and go upwards as port is initialized, or value should start at max\_port\_speed, and go downwards as port is initialized

**Bug fix description**

Power reset value is implementation dependent, and negotiated\_speed is set only once speed negotiation is complete. See definition of port\_speed on p280 I35-37

Propose clarifying by

1) adding comment

p293 I28

// set Negotiated\_speed in port register map to port\_speed and

2) possibly also add some text into the description of S3:S0 on p266 I34 and S4:A0b on p267 I14 to say that this is the point at which Negotiated\_speed is set to port\_speed[xxx]

**ID** 104    **Balloter**    Biva    **How submitted** Post ballot    **Fixed in version**  
**Title**    Beta port startup latency    **Status** Done  
**C code files** port.c    **Owner**    JH    **SCAT**  **BRAT**    **Recirc Clause** 17.2.2

**Problem description**

When the speed handshake successfully ends for a bport, it immediately sets bport\_active TRUE and the beta port starts training. This point of switching from (speed) toning to training may fall right after the last speed tone bit position during the last lap of speed handshaking. Now, assuming that the tone sampling points of the peer is delayed and the waveshape of signal\_detect for the peer is distorted (meaning tone durations are stretched and inter-tone gaps shrunk), it may be possible that the peer sees a false tone at the last speed bit sampling point.

For illustration, assume that the local node sends a speed code of S100. The tone/signal transmission sequence from the local node is:

```
HLLLHLLLHLLLLLLLLHHHHH...
1 2 3 4 56
* * * * *
```

- 1-> reference tone
- 2-> ack
- 3,4,5-> S100 tone pattern
- 6-> switching to training phase.
- \*-> tone sampling positions for the peer.

In case of the above sequence, the speed pattern detected by the peer may possibly be 101 (instead of 100 for S100), because the last "\*" happens to fall after the local port has started off with the training phase. Hence it may incorrectly detect a received speed of S1600.

I am not sure if such a scenario will actually happen, since the c-code implies that the sampling points remain locked on closely to the starting edge of tone positions of the peer. But since the positions 5 & 6 are just adjacent, I am a bit apprehensive of a false speed tone detection here.

Wouldn't it be safer to give a SPEED\_TONE\_BIT\_INTERVAL gap after the last speed bit position before the port can initiate continuous bitstream transmission? Will it have an adverse effect on the max latency in beta port startup?

**Bug fix description**

It may be a bit worse than this, we probably should ensure that all 6 bits are sent before signaling EVT\_SENT\_ACK, particularly considering the alterations put in to allow interoperability with non-1394 devices. Fix for SCAT#41 fixes this too.

**ID** 105    **Balloter** Mamezaki-San    **How submitted** Post ballot    **Fixed in version** D103  
**Title** LTS speed    **Status** Recommend Accept in principle  
**C code files** port.c    **Owner**    **SCAT**  **BRAT**    **Recirc Clause** 17.2.2

**Problem description**

In `untested_actions()` function, LTS symbols are transmitted at port speed, but receiver will receive it at S100 speed because of no speed code.

**Bug fix description**

Fixed. `bport_rx.c` was also updated such that when an LTS is being received, only changed data bytes are pushed into the FIFO. This solves the elasticity problem as well ... An LTS sequence looks like a packet  $\gg$  max packet size, so the FIFO could easily overflow. Doesn't really have any significant consequence if the FIFO is changed to always accept the most recent push as has been suggested, but it feels cleaner to specifically document that LTS's are long and should be treated, for FIFO pushing reasons, as deletable symbols.

---

**ID** 110    **Balloter** Biva    **How submitted** Post ballot    **Fixed in version** D1005  
**Title** missing `isbr_OK` in `untested_actions`    **Status** Recommend Accept in principle  
**C code files** port.c    **Owner**    **SCAT**  **BRAT**    **Recirc Clause** 17.2.2

**Problem description**

P.185, L.11-13 (Sec11.7.11) mentions "If,however,an LTP with a mode of `ATTACH_IN_PROGRESS` had been sent on the active bus,then a bus reset must be sent to the active bus (note:this bus reset is not sent on the port that had lost synchronization since it is no longer the test port.)". In order to ensure this we probably require that `untested_actions()` have a line "`isbr_OK = TRUE;`" inserted just after the `if (!bport_sync_OK)` condition near Line 27, Page 299. Should it be a short or a long bus reset ?

**Bug fix description**

This issue was addressed by the loop-breaking revamp. Basically, if we lose sync after sending an `ATTACH_IN_PROGRESS`, we either issue another LTP with mode of `LTP_TEST` in the case of a non-isolated node, or we simply set `HR_MODE` to `LTP_TEST` in the case of an `isolated_node`. Comments to this effect are in the latest `node.c`.

---

**ID** 115    **Balloter** CWS    **How submitted** Post ballot    **Fixed in version** D1006  
**Title** Compatibility of increased speed toning with future standards    **Status** Recommend Accept  
**C code files** port.c    **Owner**    **SCAT**  **BRAT**    **Recirc Clause** 17.2.2

**Problem description**

For compatibility with future inteconnects (serial ATA), the current spec increased the number of speed tones which would be collected for the received speed from 3 to 6. However, the number of speed bit tones sent before signaling `EVT_SENT_SPEED` is still 3. As a consequence, it is possible for a node to send its last speed handshake with the ACK bit and, after the 3 speed bit, call `start_port()`. This would cause the receiver to see garbage in the last 3 bits of the expected 6 bit speed field. Seems undesired, so perhaps `EVT_SENT_SPEED` should not be sent until the 6th speed bit (defined to be zero in our standard) is sent.

On a similar note, the `received_speed` variable in `port.c` is defined to be of type `speedCode` which, in turn, is an enumerated type with 7 values. However, the code attempts to load `received_speed` with 6 bits. Don't know if this is legal in C, but it isn't legal for VHDL. Seems like `received_speed` would have to be declared as an int, or that `speedCode` be defined to have upto  $2^6$  values.

**Bug fix description**

Fix was to wait until after all 6 bits were sent before `EVT_SENT_SPEED` or `EVT_SENT_ACK` are issued. Nothing done about the C code issue.

---

**ID** 122    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D105  
**Title**            Danger in overloading of immediate\_request                    **Status** Done  
**C code files** port.c                    **Owner**            JH            **SCAT**  **BRAT**            **Recirc Clause** 17.2.2

**Problem description**

Probably a nit: suspend\_target\_actions() sets immediate\_request to force suspend related packets and signaling on the bus. However, immediate\_request can be asynchronously cleared by the PHY/Link interface. Consequently, it is technically possible for the variable to be cleared before being serviced, causing a delay (and problems?) before the suspend actions are completed properly. Would it be safer to define an independent request which, like immediate\_request, causes an unconditional transition into TX?

**Bug fix description**

Fixed by introducing an immediate\_link\_request flag and an immediate\_phy\_request flag. Process\_requests maintains immediate\_request as the or of the two flags. transmit actions now tests the immediate\_link\_request flag before granting the link. Also helps to fix SCAT #238

---

**ID** 128    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D104  
**Title**            connection\_status and port state machine contend on toning and start\_port( **Status** Done  
**C code files** port.c                    **Owner**            JH            **SCAT**  **BRAT**            **Recirc Clause** 17.2.2

**Problem description**

When suspended, loop\_disabled, or disabled, connection\_status() is "stuck" in a bit of code which can consume long periods of time during which it is toning out and listening for tone or continuous tone coming in.

When the port state machine is in one of these states, an event asynchronous to connection\_status can cause a port state machine transition. For example, clearing of the disabled flag could allow P6:P11. Clearing of the loop\_disabled flag by a bus reset could cause P12:P11, and setting of the resume bit can cause P5:P1. And I think Standby to restore can happen by the restore flag causing the P9:P10 transition.

So while the connection\_status() routine is stuck toning for finite time, the port state machine can easily enter P11, P10, or P1 and start to signal continuously. As a result, we have two drivers on the tpB output conflicting with each other.

**Bug fix description**

Fix is to introduce a new flag, toner\_active, to check for the change in state in connection\_status in the code which maintains connectivity during suspend or standby and to turn off the toner if the state changes to P11, P10 or P1, and to wait for toner\_active to go FALSE in start\_port() and start\_resume\_port() before setting bport\_on to TRUE.

Also the latency is reduced by eliminating the wait at the end of the connection\_status code, so RESUME\_CHECKS now has the value of 65, and that RESUME\_SAMPLING\_INTERVAL is deleted.

Also noted that disconnection detection latency can be improved by breaking out of the loop if a tone is detected, so that the next time round is only just over a DISCONNECTED\_TONE\_INTERVAL before a disconnection is inferred.

Seems like an issue with P12 -> P11 taken because of an untested\_fault. Since receive\_OK is FALSE, loop\_disabled\_actions finishes immediately and then the P12:P11 goes quickly as well. I feel that the transition terms weren't designed with this scenario in mind ... we probably thought more about a loop disable or a startup with the far end in disconnected or something.

Fix is to set loop\_disabled = TRUE on the P11:P12 transition

Also noted that we don't want to start the connect\_timer in start\_port and start\_resume\_port until after the wait for toner\_active, and we should turn off the toner\_active sooner when toning goes away ... it currently waits until boundaries of the 42 ms frame - all now fixed in the C code.

Comment: Now that the disconnect time has been improved, then does the standard 2\*DISCONNECTED\_TONE\_INTERVAL time used everywhere need to be that big? Its clearly harmless to leave it the way it is.

---

**ID** 135    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D103  
**Title**            Remove declaration of time\_expired from untested\_actions()                    **Status** Recommend Accept  
**C code files** port.c                                    **Owner**                    **SCAT**  **BRAT**                    **Recirc Clause** 17.2.2  
**Problem description**  
time\_expired no longer used  
**Bug fix description**

---

**ID** 240    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version**  
**Title**            Incorrect term in standby\_actions()                    **Status** Recommend Reject  
**C code files** port.c                                    **Owner**    CWS                    **SCAT**  **BRAT**    0    **Recirc Clause** 17.2.2  
**Problem description**  
The following conditional statement in standby\_actions() is incorrect:  

```
if (!(Beta_mode && receive_ok) || (Beta_mode && !bport_sync_ok))
    activate_connect_detect(0);
```

When Beta\_mode is TRUE, the first term, !(Beta\_mode && receive\_ok), evaluates the condition to TRUE. The second term, (Beta\_mode && !bport\_sync\_ok), would have no effect upon the condition regardless of the state of bport\_sync\_ok.  
**Bug fix description**

Indeed, the whole section is wrong. The model of what is correct is a bit higher up in suspend\_actions(). However, standby can only happen if we're operating in Beta mode!!  

```
while ((!bport_sync_ok)
    && (connect_timer < 12 * RESET_DETECT))
;
if (!bport_sync_ok)
    activate_connect_detect(0);
```

Why do we need 12\*RESET\_DETECT??? This number was in 1394a suspend/resume for the Legacy camcorder problem. But this is now moot, as the whole section is removed in resolution of SCAT #244, so the comment is formally rejected.

---

**ID** 247    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D105  
**Title**            Disconnection during standby                    **Status** Done  
**C code files**                                    **Owner**    CWS                    **SCAT**  **BRAT**    0    **Recirc Clause** 17.2.2  
**Problem description**  
disconnection during standby should cause a bus reset  
**Bug fix description**  
code added to the end of standby\_actions

---

**ID** 264    **Balloter**                                    **How submitted** Post ballot                    **Fixed in version** D106  
**Title**            Border node in untested actions                    **Status** Done  
**C code files**                                    **Owner**    JH                    **SCAT**  **BRAT**    0    **Recirc Clause** 17.2.2  
**Problem description**  
change a border nodes behavior at the top of P11 to match text: refrain from sending any LTS until the border has finished bus\_initialize\_active, rather than sending an LTS with AIP. Just as easy to stall the far side by withholding the LTS.  
**Bug fix description**

---

*ID* 266    *Balloter* CWS                      *How submitted* Post ballot                      *Fixed in version* D105  
*Title*          clearing of found\_loop    *Status* Done  
*C code files* port.c    *Owner* CWS    *SCAT*  *BRAT* 0    *Recirc Clause* 17.2.2

*Problem description*

found\_loop is not cleared once the port under test has been forced to loop\_disabled

*Bug fix description*

clear it at the end of untested\_actions

---

*ID* 267    *Balloter* Biva                      *How submitted* Post ballot                      *Fixed in version* D106  
*Title*          sync loss and disconnection    *Status* Done  
*C code files*    *Owner* CWS    *SCAT*  *BRAT* 0    *Recirc Clause* 17.2.2

*Problem description*

There seems to be a problem with the forcing of disconnection due to detection of sync loss on a beta port with the present c-code. If I remember correctly, the intention here has been to force disconnection and "start from scratch" implying re-start from speed negotiations. In turn this means both connected and Beta\_mode flags need to be cleared when the disconnection is forced due to "sync\_fail".

The issue we are facing is that the connected flag is forced to 0 in suspended\_actions, but the Beta\_mode flag continues to remain set, since the connection\_status() function is doing the RESUME\_CHECKS loop. Since the peer will also be sending connection tones during this time, the flag know\_still\_Beta\_mode will remain set, hence Beta\_mode flag also remains set. Resultant, both the peer ports keep toning, but without detecting a re-connection or starting with speed handshaking again. Only a physical disconnection of the cable will cause the portMode to get cleared and take the ports out of this deadlock.

Kindly check this issue, and whether my understanding is proper. I feel the disconnection forcing may be effective if we modify L.55, P.298, Draft1.01 to

```
if (know_still_Beta_mode && connected) Beta_mode = TRUE;
else Beta_mode = FALSE;
```

*Bug fix description*

Agreed

---

*ID* 281    *Balloter* DW                      *How submitted* Post ballot                      *Fixed in version* D107  
*Title*          Need to add P bit into the PH\_Restore packet    *Status* Done  
*C code files*    *Owner* DW    *SCAT*  *BRAT* 0    *Recirc Clause* 17.2.2

*Problem description*

*Bug fix description*

MT to update the spec of the packet to allocate the P bit. Note, a Q bit (copy of uncle or FOP's R) is added to the format. (done)  
DW to update the description  
C code to include the P and Q bits for the Uncle (ignored by the nephew). (done)

---

**ID** 283    **Balloter** CWS                    **How submitted** Post ballot                    **Fixed in version** D107  
**Title** RESUME\_SAMPLING\_INTERVAL                    **Status** Done  
**C code files**    **Owner** CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 17.2.2

**Problem description**

is no longer defined, but is in the C code - needs fixing  
RESUME\_CHECKS is 65 in the table and 33 in the C code - needs fixing

**Bug fix description**

Intent was to get rid of RESUME\_SAMPLING\_INTERVAL and increase RESUME\_CHECKS - comments also fixed

**ID** 285    **Balloter** DW                            **How submitted** Post ballot                    **Fixed in version** D107  
**Title** Update spec for restore packet handling                    **Status** Done  
**C code files**    **Owner** CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 17.2.2

**Problem description**

Clause 11 and C code change such that uncle ends transmission of restore packet with no grant (not end of subaction). Nephew after receiving restore packet takes control of bus by sending DN on senior/uncle port. When done transferring stuff to link, release bus and mark as end of subaction (call BEPA).

**Bug fix description**

some minor differences in practice:-

- 1) As the uncle is, in effect, giving a "quiet grant" to the nephew, and the nephew is going to be boss and do BEPA, then there's no need for the uncle to wait until it sees the correct phase. The nephew will call BEOP after it has received the restore packet, and will thereby be in correct phase;
- 2) As the nephew is going to take control and start sending DN to the uncle, there's no way that the NONE\_\* indications are going to get through to the uncle anyway.

Consequence:- the uncle simply sends the restore packet, with the normal two symbols of DE, and releases the bus. No long waiting, no long DE.

3) On the nephew, as soon as the packet has been received, I've coded the taking control of the bus as start\_tx\_packet(S100, BETA). This has the effect of sending DP rather than DN, but also does things like get the DS-stuck flags set right, etc, etc. I feel more comfortable in starting a packet through normal mechanisms rather than trying to do something special. Also better for terminating the packet with BEPA.

**ID** 322    **Balloter** CWS                            **How submitted** Post ballot                    **Fixed in version**  
**Title** Overloaded use of connect\_timer                    **Status** Recommend Reject  
**C code files**    **Owner** CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 17.2.2

**Problem description**

Using connect\_timer in dc\_connection\_detector could be a problem since connect\_timer is also used in other concurrently running processes which use activate\_connect\_detect, start\_resume\_port, etc.

**Bug fix description**

I can't see that we're doing anything fundamentally different from 1394a.

As I understand it, the argument for 1394a is that the result of sharing connect\_timer is that timeouts are liable not to "pop" until sometime after their nominal value (because another port, or even another event on the same port, reset it), but that this is benign in all cases.

I've had a check through, and can't find a problem. In the particular case that Jerry cites, the effect is to delay the recognition of a DC connection.

So why might this cause a problem? Well, if we're DC connected to a bilingual (or beta??) port, then we're going to be trying toning anyway. Toning will succeed, and the DC connected flag is ignored. If we're connected to a Legacy port, then, when toning fails, then, if dc\_connected, we'll try bias. But if we're tardy in discovering the DC connection, then we do nothing very much, keep toning, and try again. Eventually, the DC connection will be detected, and we'll try bias. Of course, we may get incoming bias during this time and all will be well.

**ID** 337    **Balloter**    CWS                      **How submitted** Post ballot                      **Fixed in version** D1071  
**Title**            Deadlock in P12    **Status** Done  
**C code files**    **Owner**    CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 17.2.2

**Problem description**

I observe the following:  
Phyc:- finishes start\_port() in P11 with bport\_sync\_ok == FALSE;  
          untested\_fault == TRUE resulting in P11:P12. loop\_disabled = TRUE;receive\_ok = FALSE;  
Phya:- is in P11 and transmitting a continous DATA\_PREFIX. (More on this below.)  
Phyc:- completes loop\_disable\_actions and stops. connected == TRUE, loop\_disabled == TRUE,  
          receive\_ok == FALSE.  
- detects continous signal; receive\_ok = TRUE; P12:P11.  
- finishes start\_port() in P11 with bport\_sync\_ok == FALSE;  
          untested\_fault == TRUE resulting in P11:P12. loop\_disabled = TRUE; receive\_ok = TRUE;  
          will not complete loop\_disable\_actions() unless loop\_disabled == FALSE or  
          receive\_ok == FALSE

The reason phya is transmitting continous DATA\_PREFIX is it is a border\_node and waiting for bus\_initialize\_active to go FALSE.

From port.c:

```

else { // not a single port phy
portT.arb = SPEED;            // start LTS with normal packet start at the port speed
portT.speed = DEFAULT;
portT.pkt = BETA;
portT.tag = ARB_STATE;
wait_symbol_time(port_speed);    // send speed signal (single SPEEDb)
portT.arb = DATA_PREFIX;
wait_symbol_time(port_speed);    // send DATA_PREFIX
if (border_node)
    // strictly the following test is required to remain true until tree_ID completes on all Legacy ports
    while (bus_initialize_active) <-- STUCK HERE.
;

```

The reason our implementation is stuck with bus\_initialize\_active TRUE, is the self\_ID code has not been completed. But, this can happen with a compliant implementation if it can't get to a0:Idle. It could be stuck in T0 or in a state for MAX\_ARB\_STATE\_TIME causing a transition to R0.

JH:-

The P11:P12 transition is taken for one of two reasons: either the port was commanded to take the transition because a loop was found (loop\_disabled = TRUE), or because we lost or never achieved sync with the far end (untested\_fault = true).

**Case 1 : Found an actual loop**

I claim the intent when a loop is found is for the local port (the one detecting the loop first) to head off to P12 and cause the remote port to land in P12 as well. Both sides should stay in P12 unless a disconnect happens or a bus reset occurs (indicating a topology change).

So how well did we capture this intent in C? Well, we start out with our local port and the remote port both in P11, happily signaling away. When the local port is given the loop\_disabled signal, it heads off to P12 and starts to tone. Since bport\_sync\_ok was true as we took P11:P12, receive\_ok will be true upon arrival into P12. Intent per SCAT #45/ PRN #353 was that we would wait in loop\_disabled\_actions() until the remote end took the P11:P12 transition as witnessed by our local receive\_ok going away (remote end stops signaling and starts toning). The problem/bug with this scenario is that receive\_ok can never go false. Taking a peek at the portion of connection\_status() running in P12 shows that receive\_ok can never be set FALSE, only TRUE (for beta connected ports).

```

// here if P5, P6, P9 or P12 and connectivity established
// look for disconnect or report bias/continuous tone
if (Beta_mode) {                      // Beta mode - send a tone at periodic intervals
    know_still_Beta_mode = FALSE;
    toning = TRUE;                      // turn on the autonomous toner
    signal_detect_OK();                // flush any stale values (no need to wait)
    for (i = 0; i < RESUME_CHECKS; i++) {
        if (signal_detect_OK()) {

```

```

if (port_state == P1 || port_state == P10 || port_state == P11) {
    toning = FALSE;          // port changed state, so stop toning
    return;
} else {
    know_still_Beta_mode = TRUE;
    if (signal_detect_OK()) { // still true - continuous tone
        if (!disabled) {     // in P5:suspended state
            receive_ok = TRUE; // to start resuming if suspended
            toning = FALSE;    // turn off the autonomous toner
        }
        return;
    }
    break;                    // connected, but no continuous tone
}
}
}
// loss of sync in suspend forces disconnection, so reset Beta_mode as well
Beta_mode = know_still_Beta_mode && connected;
if (!Beta_mode) {           // new disconnection when in Beta mode
    connected = FALSE;
    toning = FALSE;         // ensure the far end sees a "disconnection"
    wait_time(2 * DISCONNECTED_TONE_INTERVAL);
}
} else {                    // DS mode
    if (!disabled) {        // in P5:suspended state
        receive_ok = bias;
        if (receive_ok) return;
    }
    connected = dc_connected; // see if still connected
}
}

```

Consequently, I predict we'll get stuck in `loop_disabled_actions()` for ever in this scenario. (And by the way, the inability to clear `receive_ok` means that P5:P5 can never happen either).

(Possible Fix #1.1a) Now one "fix" might be to make sure `receive_ok` can be set false by adding `receive_OK = FALSE` at the point where the comment says "connected, but no continuous tone".

(Alternate Fix #1.1b) Jim, however, has a different approach which we might consider. He maintains `receive_ok` on a beta port as an indication of continuous signaling; i.e., he doesn't try to change the meaning of `receive_ok` based on port state from continuous signaling to `bport_sync_ok`, etc. On a Legacy port, `receive_ok` equals `bias`. And he maintains a new signal called `rx_ok = beta_mode ? bport_sync_ok : bias`. For exits from P2 on the state machine, `rx_ok` replaces `receive_ok`. If adopted, `receive_ok` probably comes out of `connection_status()` and therefore will continuously update (to TRUE or FALSE) all the time.

Another problem I see is that the condition within `loop_disabled_actions()` which waits for `receive_ok` to go false is actually:

```

while (receive_ok && loop_disabled)
    ; // Wait til peer becomes inactive or loop_disabled flag is cleared.

```

If `loop_disabled` gets cleared asynchronously by software or a bus reset, then we might not stay in P12 long enough to make sure the far end loses sync and follows us into P12.

(Possible Fix #1.2) Remove `loop_disabled` from while loop at end of `loop_disabled_actions`.

Still, I think there can be a race condition in which one port is stuck waiting for `receive_ok` to go false while the connected port heads back into P11 and starts signaling again. It will eventually fail and head back to P12, but doesn't need to wait there since `receive_ok` is already false. If `loop_disabled` gets cleared ... it's back to P11 again and the original port is still stuck. This can oscillate for a while if `loop_disabled` keeps getting cleared.

(Possible Fixes #1.3) We can decide that the oscillation isn't infinite, or we can add some hysteresis by requiring a port to stay in P12 for a minimum amount of time `_after_ receive_ok` is seen as FALSE to make sure the connected port also sees `receive_ok` as false.

Case 2 : Lost or never achieved synchronization  
Let's further divide Case 2.

Case 2a: Never achieved sync because remote end is disabled  
Normally, we never achieve sync when the far end is in P6:Disabled. We were able to establish connectivity via speed

exchange, but start\_port fails because the far end doesn't want to play. I claim our intent was to head off to P12 and sit and wait for the far end to become disconnected or become not disabled as advertised by starting to signal continuously.

How did well did our C capture this intent? Well, as the local port fails start\_port, untested\_fault gets set causing the P11:P12 transition. And the connection\_status routine, while we were in P11, set receive\_ok = bport\_sync\_ok which is known to be FALSE in this case. So our local port makes it to P12 with untested\_fault set and receive\_ok cleared. With or without the fix above, loop\_disabled\_actions() completes quickly. In the current C code, we avoid heading back to P11 immediately because receive\_ok is false and loop\_disabled was set on the original P11:P12 transition. The setting of loop\_disabled was done as part of PRN #391/SCAT #128. However, I think this violates the intent. By setting loop\_disabled on P11:P12, the next bus reset will clear loop\_disabled and force us to head back into P11 and try to start\_port() again. Since the far port is still disabled, we'll fail again and head back to P12. So with each bus reset, we pointlessly try and talk to the remote disabled port. I think this violates the intent, as was argued in SCAT #44 point 6 against collapsing untested\_fault and loop\_disabled into a single signal.

However, a fix is not offered for this "violation of intent". Turns out our best fix for Case 2c discussed below is to change our intent, and allow a bus reset to force us out of P12 each and every time. As such, we observe that loop\_disabled and untested\_fault are now identical ... so SCAT #44 point 6 should be reversed and untested\_fault can be replaced with loop\_disabled.

(Possible Fix #2a.1a) Replace untested\_fault with loop\_disabled. (Loop\_disabled can then be removed from the actions on P11:P12.)

(Alternate Fix #2a.1b) If for some reason we decide to stick with our original intent, then a new fix for PRN#391 will be needed. I think the original intent would be met by making the P12:P11 condition read "connected && (!loop\_disabled && !untested\_fault) || (untested\_fault && receive\_ok)". Note that it might even make things a bit cleaner to split up P12:P11 into two transitions. The first deals with Case 1 when we entered P12 because of a loop. As such, the exit from P12 would be connected && !untested\_fault && !loop\_disabled. (The untested\_fault term effectively tells us if we entered P12 because of case 1 or case 2.) The second transition would then be connected && untested\_fault && receive\_ok which says that if we end up in P12 because of an untested\_fault ... we'll wait for the other end to initiate a transition out.

Case 2b: Lost sync because remote end headed to P6 or P12

We "expect" to lose sync if the far end initiates a loop\_disable or a disable. I think the intent is the same as Case 2a. Since the far end initiated, we head off to P12 and wait for the far end to do something to bring us back out of P12. So comments/conclusions for Case 2a apply here as well.

Case 2c: Lost sync because of multiple bit errors or failure to train

This case has raised the most issues since I'm not sure of the intent. Once a port has lost sync, it need to go to P12 and tone until the remote connected port breaks out of P11. Toning is the only way to force a resync as we've learned in the past (just sending training characters doesn't force a re-train ... if the far end was in a data phase, the training character simply look like data with an annoying invalid symbol (the comma) once in 64 valid characters). So our intent must be (is?) to have the local port head to P12 and force the remote end into P12 as well.

But assuming that as the intent raised some C code problems. We can determine that the far end entered P12 by detecting the lack of continuous signaling. So at first blush, it seemed like waiting for receive\_ok to be false in loop\_disabled\_actions() would mean the far end was in P12. Although true for case 1, it doesn't hold for this case. Problem is that on entry to P12, receive\_ok is already false (because of bport\_sync\_ok being false) even though the far end is still signaling continuously in P11. As such, we might get a false indication that the far end went to P12 even though it never left P11. So a fix would be required to stall in P12 or get receive\_ok to reflect the true state of signaling upon entry to P12. Alternate Fix #1.1b above would address this issue by having receive\_ok reflect the true state of continuous signaling at all times.

Having done such a fix, we could succeed in getting both ports into P12 with untested\_fault set. But now what? The individual ports can't differentiate between this Case2c and Case2b. In Case2b, I thought the intent was for the port which detects the loss of sync to head to P12 and quietly wait until the remote port initiates a transition back to P11 by starting continuous signaling. But if we implement that intent for Case 2b, Case 2c becomes a deadlock. Both ports are waiting for the other to initiate ... and untested\_fault wasn't provided to software for possible recovery. Only way out of this would be to have the ports become disconnected or have one of them become disabled. Seems close to fatal ... but is it frequent enough to worry about?

Well as it turns out, today's C code didn't implement the original intent for Case 2a and 2b as discussed. The transition into P12 set loop\_disabled and P12:P11 allows an exit when loop\_disabled is cleared. So if we did get stuck with two ports in P12 unexpectedly, the next bus reset will force them back into P11.

So Jim's preference is that we leave that bit of the C code alone, effectively changing the intent for Cases 2a and 2b. Any and all bus resets will now cause an eventual transition out of P12 no matter why the port was in P12. Are we okay with this approach? (Intent #2c.1a)

(Alternate Intent #2c.1b) One alternative could be to leave these ports in P12 and not attempt any recovery. Argument would have to be that it was too rare to occur and if it did, the user would yank a cable or something. If we go this route, then we could

return to the original intent for Cases 2a and 2b (namely wait for far end to initiate).

(Alternate Intent #2c.1c) A more robust approach would be to deal with the heart of the matter; namely, that a single port can't distinguish between Case 2c and Cases2a-b. In some sense, Case 2c is an unexpected loss of sync while 2a and 2b are expected losses of sync (the far end is causing the loss intentionally). So why not have the far end advertise said fact in some fashion. For example, when a loop is discovered, send a control symbol to the far end to say "loop discovered, I'm about to detach". Similarly for disable. Then, if a port is told that a loss of sync is coming, it will know that when it gets to P12, it should sit quietly and wait for the far end to initiate a trip back to P11. Conversely, if the loss of sync happens without forewarning, then the port can automatically come back to P11 after sitting in P12 for the min time. With this approach, we don't have to rely on bus resets to save us. Admittedly, some extra work is needed for Case 2a since the port which is already disconnected won't be able to send a "TX\_DISABLE". In this case, however, receive\_ok never goes true while in P11. This could be the clue that the other end is intentionally not playing ... so go to P12 and wait.

(Alternate Intent #2c.1d) Which suggests a final possibility. A port which is in P12 or P6 "intentionally" should not respond to incoming signaling. They prefer to wait until some other event (bus\_reset or disable being cleared) wakes them up. So when a port finds itself losing synchronization (because of either 2b or 2c), it heads to P12 and, after a sufficient delay, automatically head back to P11. If receive\_ok stays false while in P11, then the port knows the far end intentionally broke sync and the near end can head to P12 and wait patiently. If, however, receive\_ok fires up, the local port knows the loss of sync was unintentional ... and it does it's best to acquire sync. So an intentional disable event causes an extra trip to P11 for the "clueless" port.

**Bug fix description**

Implementation comprises Alternate Fix #1.1b, Fix #1.2, Fix #1.3 is not implemented, as this seems to be covered by #1.1b Fix #2a.1a, and leave the code as it is by changing the intent to #2c.1a

---

<b>ID</b>	340	<b>Balloter</b>	CWS	<b>How submitted</b>	Post ballot	<b>Fixed in version</b>	
<b>Title</b>	P2:P11 and P2:P3 are not mutually exclusive				<b>Status</b>	Done	
<b>C code files</b>		<b>Owner</b>	CWS	<b>SCAT</b>	<input type="checkbox"/>	<b>BRAT</b>	0
		<b>Recirc Clause</b>	17.2.2				

**Problem description**

Seems like for beta mode in P2, receive\_ok = bport\_sync\_ok. So it is possible for bport\_sync\_ok = receive\_ok = FALSE with loop\_to\_detect = TRUE. Then both P2:P11 and P2:P3 can be taken.

Also can force\_disconnect be true while P2:P3 is also valid? Do we need a !force\_disconnect worked into P2:Px transitions?

**Bug fix description**

P2:Px transitions updated

---

<b>ID</b>	1	<b>Balloter</b>	CWS	<b>How submitted</b>	Ballot comment	<b>Fixed in version</b>	D1005
<b>Title</b>	Speed filtering frequency error				<b>Status</b>	Recommend Accept	
<b>C code files</b>	speed.c	<b>Owner</b>		<b>SCAT</b>	<input type="checkbox"/>	<b>BRAT</b>	465
		<b>Recirc Clause</b>	17.3.1				

**Problem description**

Currently waits for 10 ns

```
for (i = 0; i < (1<<DS_PHY_SPEED); i++)
    wait_event(PH_DS_BIT_CLOCK); // Wait for 50 MHz clock
```

**Bug fix description**

```
for (i = 0; i < (2<<DS_PHY_SPEED); i++) // Wait for 50 MHz clock
    wait_event(PH_DS_BIT_CLOCK);
```

---

**ID** 10      **Balloter**    CWS                      **How submitted** Not submitted                      **Fixed in version** D104  
**Title**            DS port receive FIFO    **Status** Done  
**C code files** dsport\_rx.c                                      **Owner**                      **SCAT**  **BRAT**                      **Recirc Clause** 17.3.1

**Problem description**

The FIFO is going to get filled in the local clock domain for arb states, and in the received clock domain for data. This is not captured in the C code (in particular, switching between the two).

**Bug fix description**

Fixed as a side-effect of fixes in C code bugs #23, #25 and #46

---

**ID** 21      **Balloter**    CWS                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**            FIFO overflow test    **Status** Recommend Accept  
**C code files** dsport\_rx.c                                      **Owner**                      **SCAT**  **BRAT** 467 **Recirc Clause** 17.3.1

**Problem description**

Consider the starting condition of `fifo_rd_ptr = fifo_wr_ptr`. This means the FIFO is empty, yet the code refuses to push onto the FIFO.

**Bug fix description**

The proper test should be:

$(\text{FIFO\_DEPTH} + \text{fifo\_wr\_ptr} - \text{fifo\_rd\_ptr}) \% \text{FIFO\_DEPTH} < \text{FIFO\_DEPTH} - 1$

The left hand side is the expression for how many elements are currently in the FIFO, and the right hand side is the maximum number of elements the FIFO can hold.

---

**ID** 46      **Balloter**      Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D103  
**Title**      Ds arb state decode      **Status**      Recommend Accept in principl  
**C code files** dsport\_rx.c      **Owner**      **SCAT**  **BRAT** <sup>^</sup> 461 **Recirc Clause** 17.3.1

**Problem description**

(Comment #20)  
Table 16-18, pg. 301, function decode  
Decoding of DS arb states is dependent upon the arb control state-machine state.  
Change pg. 301 lines ~16-28 to:

```
// Note: During tree-ID, it is possible to receive short periods of
// RX_DATA_PREFIX, e.g., while in T1: Child_Handshake waiting for the
// peer port to transition to the S0:Self_ID_Start state.
switch(rx_arb) {
  case RX_DATA_PREFIX: return(DATA_PREFIX);

  case RX_DATA_END: return(PHY.PHY_state < S0) ? PARENT_HANDSHAKE : DATA_END;
  // RX_PARENT_HANDSHAKE is same as RX_DATA_END

  case RX_PARENT_NOTIFY: return(PHY.PHY_state < A0) ? PARENT_NOTIFY : REQUEST_CANCEL;
  // RX_REQUEST_CANCEL is same as RX_PARENT_NOTIFY

  case RX_SELF_ID_GRANT: return(PHY.PHY_state < A0) ? SELF_ID_GRANT : LEGACY_REQUEST;
  // RX_REQUEST is same as RX_SELF_ID_GRANT

  case RX_ROOT_CONTENTION: return(PHY.PHY_state < A0) ? ROOT_CONTENTION : GRANT;
  // RX_GRANT is same as RX_ROOT_CONTENTION

  case RX_IDENT_DONE: return(PHY.PHY_state < A0) ? IDENT_DONE : DISABLE_NOTIFY;
  // RX_DISABLE_NOTIFY is same as RX_IDENT_DONE

  case RX_IDLE: return(IDLE);

  case RX_BUS_RESET: return(BUS_RESET);

  case RX_CHILD_HANDSHAKE: return(CHILD_HANDSHAKE);

  case RX_SUSPEND: return(SUSPEND);
}
```

**Bug fix description**

See #23 (BRAT #415) Agreed in principle ... some of the above overloads are incorrect and RX\_SUSPEND isn't isolated. Still, process\_req has been modified to use PHY\_state to filter and resolve the overloading. And a caution about filtering of spurious RX\_DATA\_PREFIX indications has been added to dsport\_rx.c

---

**ID** 47      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version**

**Title** decode\_bit      **Status** Recommend Reject

**C code files** dsport\_rx.c      **Owner**      **SCAT**  **BRAT** 462 **Recirc Clause** 17.3.1

**Problem description**

(Comment #21)  
 Table 16-8, pg. 301, function decode\_bit  
 Must ensure that received arb state is IDLE when in a transmit state.  
 Change pg. 301 line from:

```
new_signal = ds_portR; // Get signal
to:

if (PHY_state == S4 || PHY_state == TX || PHY_state == PH)
    new_signal.RX_arb = RX_IDLE;
else
    new_signal = ds_portR; // Get signal
```

**Bug fix description**

I'd like to reject this comment for two reasons. The first is that I don't think all of the proper PHY states have been captured. For example, when the PHY is in the RX state, many ports are in the TX state and one could argue that those ports want to see RX\_IDLE as well to prevent the pushing of bogus arbitration indications into the FIFO. But we certainly don't want to ignore the receive ports arb comparators during the time we are in the RX state .... so adding PHY\_state == RX to the above isn't a sufficient fix. And the C code, as has been noted by some, complicates some things by pushing arb states through the FIFO.

The second reason is that the most recently published C Code had some weasel words in decode\_bit to deal with this:

```
// not expecting data, process arb comparators
// Note : While the local port is transmitting serial data, the arbitration
// comparators are presumed to be disabled or ignored to prevent false queuing of
// data patterns as received arbitration indications. While not explicitly
// described in the C code, such operation is presumed by this implementation.
```

**ID** 48      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D104

**Title** encode      **Status** Recommend Accept

**C code files** dsport\_tx.c      **Owner**      **SCAT**  **BRAT** 464 **Recirc Clause** 17.3.1

**Problem description**

(Comment #22)  
 Table 16-9, pg. 303, function encode  
 Add IDLE and DATA\_PREFIX to case table.  
 At line ~ 19, add the following:

```
case IDLE: return(TX_IDLE);
case DATA_PREFIX: return(TX_DATA_PREFIX);
```

**Bug fix description**

Agreed

**ID** 49      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D104  
**Title** dsport\_transmit\_actions      **Status** Recommend Accept in principle  
**C code files** dsport\_tx.c      **Owner** JH      **SCAT**  **BRAT** 466      **Recirc Clause** 17.3.1

**Problem description**

(Comment #23)

Table 16-9, pg. 304, function dsport\_transmit\_actions

1) Typographic/spelling error in comment--change line ~8 from:

```
} else { // tag == ARB_START  
to:
```

```
} else { // tag == ARB_STATE
```

2) When SPEED is transmitted, don't necessarily send data-prefix (e.g., during self-ID).

Delete pg. 304 line ~22:

```
ds_portTarb(TX_DATA_PREFIX);
```

3) When DATA\_END is transmitted, don't necessarily send dribble bits (e.g., null packet).

Change pg. 304 line ~26 from:

```
tx_dribble_bits(TX_DATA_END);  
to:
```

```
if (ds_in_packet) tx_dribble_bits(TX_DATA_END);
```

**Bug fix description**

#1) above agreed and done

#2) above - see SCAT #206 for a complete fix

#3) above done

---

**ID** 79      **Balloter** CWS      **How submitted** Post ballot      **Fixed in version** D1005  
**Title** Initialization of portT, introduction of dsport\_active      **Status** Recommend Accept  
**C code files** dsport\_rx.c, dsport\_tx.c, port.c, shared.h      **Owner**      **SCAT**  **BRAT**      **Recirc Clause** 17.3.1

**Problem description**

portT is not initialized anywhere for a DS port ... all of the upstarts code initializes just for beta ports. Furthermore, dsport\_transmit\_actions() runs all of the time after bus reset, even if the port hasn't yet been marked as active, connected whatever. In simulations, the combined consequence is that at the instance the port is marked connected, D/S bits come rushing out of port immediately.

bports don't suffer because portT is properly initialized, and because the bport\_tx machines stay in an inactive state until bport\_active is set true.

**Bug fix description**

A symmetric solution is proposed. Introduce dsport\_active which starts and stops dsport\_transmit\_actions. This is easily done by changing the opening while to "while (power\_reset or !dsport\_active) "

(I used the word "or" above rather than the vertical bars because the bug tracker complained.)

Then, in every circumstance in port.c where bport\_active is assigned, also assign dsport\_active as required symmetrically. And just prior to asserting dsport\_active, make sure portT is set to a valid value.

---

**ID** 9      **Balloter** CWS      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** encoding of BORDER      **Status** Recommend Accept  
**C code files** bport\_tx.c, bport\_rx.c      **Owner**      **SCAT**  **BRAT** 469 **Recirc Clause** 17.3.2

**Problem description**

encoding and decoding of BORDER is missing

**Bug fix description**

Also BRAT # 476  
replace 0b111 (INVALID) with BORDER

---

**ID** 30      **Balloter** CWS      **How submitted** Ballot comment      **Fixed in version** D104  
**Title** pkt and pkt\_prefix      **Status** Recommend Accept  
**C code files** bport\_rx.c      **Owner**      **SCAT**  **BRAT** 471 **Recirc Clause** 17.3.2

**Problem description**

Notice the use of pkt and pkt\_prefix. Requests are only expected when we aren't inside a packet (pkt) and when we aren't expecting a packet to begin (pkt\_prefix).

Data is the logical inverse ... we are in a packet or expecting one to begin.

But comma is a little strange ... the code implies we expect a comma sometimes when we are in pkt\_prefix (since we only ignore the comma inside of actual data).

Why isn't comma detection restricted in the same fashion as request types (since commas can only be inserted in a string of requests)? And if it does no harm to detect commas during the packet prefix, why not detect during "data" as well?

From Alistair:-

In normal operation a comma should not be received during packet or packet prefix or any request other than TRAINING or OPERATION. See 10.3.1.2.

So it would be consistent to make the comma decode conditional on !pkt and !pkt\_prefix, same as requests.

You might want to think about what happens in abnormal operation if it is possible for the receiver to think it is pkt or prefix when the transmitter is trying to send a TRAINING request with commas. The comma will appear as an invalid to the rx but the rest of the TRAINING request will appear as valid, so does the rx ever get to realise it has lost sync??

Regards,  
Alistair

**Bug fix description**

For consistency, comma can only be detected where it is expected ... in a request position. So qualification for detecting a comma has been enhanced to be conditional on !pkt and !pkt\_prefix.

Alistair's second question is deeper and touches on a related topic ... why PTX3:PTX1 is needed. A new bug will be logged to deal with this aspect of the problem.

---

**ID** 50      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title**      fifo error      **Status** Recommend Accept in principl  
**C code files** bport\_rx.c      **Owner**      **SCAT**  **BRAT** 468 **Recirc Clause** 17.3.2

**Problem description**

(Comment #24)  
Table 16-11, pg. 306, function push  
The if-statement condition is incorrect (it will not evaluate to TRUE if the fifo\_rd\_ptr is equal to the fifo\_wr\_ptr, i.e., the FIFO is empty).  
Change pg. 306 line ~54 from:

```
if (((fifo_rd_ptr-fifo_wr_ptr + FIFO_DEPTH) % FIFO_DEPTH) > 1) {
```

**Bug fix description**

See 21 for equivalent fix

---

**ID** 51      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title**      rx\_character      **Status** Recommend Accept  
**C code files** bport\_rx.c      **Owner**      **SCAT**  **BRAT** 470 **Recirc Clause** 17.3.2

**Problem description**

(Comment #25)  
Table 16-11, pg. 309, function rx\_character  
Parentheses required (as written, rx\_scam\_ctrl is ex-or'ed with just one bit of descram).

**Bug fix description**

Change pg. 309 line 27 from:

```
rx_control=(rx_scam_ctrl^((descram&0x80)>>4) | ((descram&0x20)>>3) |  
((descram&0x8)>>2) | (descram&0x2)>>1);  
to:
```

```
rx_control=(rx_scam_ctrl^(((descram&0x80)>>4) | ((descram&0x20)>>3) |  
((descram&0x8)>>2) | ((descram&0x2)>>1)));
```

For aesthetic consistency with other code, change line 34 from:

```
} else if((rx_scam_req=rx_comma_decode(character_in, rx_rd))>=0 && !pkt ) {  
to:
```

```
} else if(((rx_scam_req=rx_comma_decode(character_in, rx_rd))>=0) && !pkt ) {
```

---

**ID** 52      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title**      train\_character\_sync      **Status** Recommend Accept  
**C code files** bport\_rx.c      **Owner**      **SCAT**  **BRAT** 472 **Recirc Clause** 17.3.2

**Problem description**

(Comment #26)  
Table 16-11, pg. 310, function train\_character\_sync  
The rx\_bits variable must be kept to just 16-bits of significant data.

**Bug fix description**

Change pg. 310 line 29 from:

```
rx_bits = rx_bits << 1;  
to:
```

```
rx_bits = (rx_bits << 1) & 0xFFFF;
```

---

**ID** 53      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** rx\_sync\_lost\_actions      **Status** Recommend Accept  
**C code files** bport\_rx.c      **Owner**      **SCAT**  **BRAT** 473 **Recirc Clause** 17.3.2

**Problem description**

(Comment #27)  
Table 16-11, pg. 310, function rx\_sync\_lost\_actions

The char\_check variable must be initialized prior to calling the character\_sync function.

**Bug fix description**

Add following at pg. 310 line 55 just after "rx\_rd=negative\_rd;" statement:

```
char_check = 0; // initialize char_check count
```

---

**ID** 54      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** scrambler\_sync\_actions      **Status** Recommend Accept  
**C code files** bport\_rx.c      **Owner**      **SCAT**  **BRAT** 474 **Recirc Clause** 17.3.2

**Problem description**

(Comment #28)  
Table 16-11, pg. 311, function scrambler\_sync\_actions  
The check for inverted polarity (by looking for either of two special characters) also requires that the rx\_req\_code variable be TRUE (i.e., the characters are request codes).

**Bug fix description**

Change pg. 311 line 28 from:

```
} else if((rx_req == 0xF8) || (rx_req == 0x98)) {  
to:  
  
} else if(((rx_req == 0xF8) || (rx_req == 0x98)) && rx_req_code) {
```

---

**ID** 55      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** pad\_count      **Status** Recommend Accept  
**C code files** bport\_rx.c      **Owner**      **SCAT**  **BRAT** 475 **Recirc Clause** 17.3.2

**Problem description**

(Comment #29)  
Table 16-11, pg. 313, function bport\_receive\_actions  
Incrementing of the pad\_count variable is incorrect.

**Bug fix description**

Change pg. 313 line 21 from:

```
if (pad_count==rx_speed_ratio) pad_count=0;  
else pad_count++;  
to:  
  
pad_count = (pad_count + 1) % (1 << rx_speed_ratio);
```

[JH] Used Dave Scott's suggestion to rename rx\_speed\_ratio to rx\_speed\_diff to avoid some of the confusion. And then used Skid's mod operator for the calculation.

---

**ID** 56      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** bport\_transmit\_actions comments      **Status** Recommend Accept  
**C code files** bport\_tx.c      **Owner**      **SCAT**  **BRAT** 477      **Recirc Clause** 17.3.2

**Problem description**

(Comment #30)  
Table 16-12, pg. 316, function bport\_transmit\_actions  
Extraneous spaces in comment.

**Bug fix description**

Change pg. 316 line 5 from:

int tx\_speed\_ratio; // number of symbols per byte for given pkt\_speed and port\_speed  
to:

int tx\_speed\_ratio; // number of symbols per byte for given pkt\_speed and port\_speed

---

**ID** 92      **Balloter** CWS      **How submitted** Post ballot      **Fixed in version** D106  
**Title** Inconsistent clearing of FIFO pointers      **Status** Done  
**C code files** bport\_rx.c      **Owner** CWS      **SCAT**  **BRAT**      **Recirc Clause** 17.3.2

**Problem description**

The FIFO pointers fifo\_wr\_ptr and fifo\_rd\_ptr are not cleared simulatenously in all cases. The fifo\_rd\_ptr is always cleared at power\_reset. In ds mode fifo\_wr\_ptr is also cleared at power\_reset. However, in beta mode fifo\_wr\_ptr is also cleared when bport\_active goes false.

**Bug fix description**

Both pointers only be cleared at power\_reset.  
in inactive\_actions(),

fifo\_wr\_ptr = 0;

is moved up into the power\_reset loop

Change C code for DS port in decode\_bit() to reset fifo\_wr\_ptr only on power\_reset

---

**ID** 100    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D103  
**Title**            Stretching bug in bport\_transmit\_actions                    **Status** Recommend Accept in principl  
**C code files** bport\_tx.c                    **Owner**                    **SCAT**  **BRAT**                    **Recirc Clause** 17.3.2

**Problem description**

I believe there is a bug in stretching request symbols. Unless portT.speed is assigned the value DEFAULT whenever a packet is not being transferred, the request symbols will be stretched according to the present value of tx\_speed\_ratio. While a request symbol is being stretched, portT can change at a higher packet speed rate and the state machine will miss the first of these new symbols.

I recommend the following change to the code at the end of bport\_transmit\_actions:

FROM:

```
tx_character(localT);
for(i=1; i<tx_speed_ratio; i++)
    tx_character(localT);            // stretch as required
```

TO:

```
tx_character(localT);
if localT.tag == CTRL {
    for(i=1; i<tx_speed_ratio; i++)
        tx_character(localT);            // stretch control symbols as required
}
```

**Bug fix description**

Preferred fix was to make sure portT.speed had been appropriately initialized before assigning to portT.tag. In many places, it turned out the assignment to portT.speed was redundant, but it was retained/added for clarity to emphasize that the speed field must be defined for each transmit symbol.

**ID** 101    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version**  
**Title**            Second speed signal ignored by bport\_rx.                    **Status** Done  
**C code files** bport\_rx.c                    **Owner**    CWS                    **SCAT**  **BRAT**                    **Recirc Clause** 17.3.2

**Problem description**

When a bport receives a concatenated null packet it will ignore the second speed code. Speed\_known is set upon receipt of the first speed symbol and it is not cleared until a non data prefix symbol and a non speedc is received. Therefore the speed\_known will not be cleared until the speeda or speedb symbol of the concatenated packet is received. This is too late for bspeed\_filter to do its job.

**Bug fix description**

**ID** 117    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D103  
**Title**            Unnecessary assignment to tx\_speed                    **Status** Recommend Accept  
**C code files** bport\_tx.c                    **Owner**                    **SCAT**  **BRAT**                    **Recirc Clause** 17.3.2

**Problem description**

bport\_tx.c, inside of bport\_transmit\_actions(), assigns port\_speed to tx\_speed immediately before a while loop. However, tx\_speed gets reassigned immediately inside of the while loop, and is never referenced outside of the while loop, I believe.

**Bug fix description**

Assignment removed

**ID** 121    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D103  
**Title**            Extraneous code in bport\_rx.c for setting context                    **Status** Recommend Accept  
**C code files** bport\_rx.c    **Owner**                    **SCAT**  **BRAT**                    **Recirc Clause** 17.3.2

**Problem description**

The routine rx\_sync\_actions() in bport\_rx.c contains a while loop with the comment "Do not report DATA\_END symbols that have been sent for context info only"

I don't believe we are "setting context" anymore and, consequently, this while loop can be removed.

**Bug fix description**

while loop removed

---

**ID** 131    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D103  
**Title**            variables used by rx\_character not properly initialized                    **Status** Recommend Accept  
**C code files** bport\_rx.c    **Owner**                    **SCAT**  **BRAT**                    **Recirc Clause** 17.3.2

**Problem description**

rx\_character uses a number of variable/signals including pkt and pkt\_prefix before they are initialized.

**Bug fix description**

bport\_rx scrubbed to see if all global signals within the file were initialized before use. Several were not and were added to rx\_inactive\_actions. Additionally, rx\_scram\_req was not being reset between each character. Consequently, update\_scrambler might think a received character was a request symbol when it really was a control symbol. Probably doesn't happen in a real system ... but it felt safer to initialize the lot of rx\_scram\_x at the start of each character decode.

---

**ID** 139    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D105  
**Title**            Does bport\_rx deal with concatenated speed codes?                    **Status** Done  
**C code files** bport\_rx.c    **Owner**                    JH                    **SCAT**  **BRAT**                    **Recirc Clause** 17.3.2

**Problem description**

Withdrawn link requests can cause a speed code to be sent on an otherwise empty packet. Concatenate a few of these together, and you get speed code followed by DP followed by another speed code. Doesn't look like bport\_rx handles that too well.

**Bug fix description**

Modified bspeed\_filter() to handle the case. Fixed by enabling the speed code recognition filter except in packet context rx\_speed\_diff now only used inside the filter

---

**ID** 213   **Balloter**   CWS   **How submitted** Post ballot   **Fixed in version** D103  
**Title**   Misleading comments in end of bport\_receive\_actions()   **Status** Recommend Accept  
**C code files**   **Owner**   CWS   **SCAT**  **BRAT**   0   **Recirc Clause** 17.3.2

**Problem description**

bport\_receive\_actions() had some older comments regarding "control signal" and "config request symbol". Control and config requests have all been mapped into ARB\_STATES, and ARB\_REQUESTS remain independent.

```
// Buffer first occurrence of each new control signal....  
  
(stuff deleted)  
  
// Buffer first occurrence of each new config request signal....
```

**Bug fix description**

Code changed to:

```
// Buffer first occurrence of each new arbitration state....  
  
(stuff deleted)  
  
// Buffer first occurrence of each new arbitration request....
```

---

**ID** 214   **Balloter**   CWS   **How submitted** Post ballot   **Fixed in version** D103  
**Title**   Bport receiver misses first request.   **Status** Recommend Accept  
**C code files**   **Owner**   CWS   **SCAT**  **BRAT**   0   **Recirc Clause** 17.3.2

**Problem description**

there is a case where the bport receiver will miss the first request sent by its peer.

The case is when the peer has been sending requests before the receiving port enters PRX4:Receive. The variables localR and last\_localR have been identical for more than one symbol.

When the code evaluates these variables at the end of receive\_actions it determines that there is no first occurrence and does not push the arb request onto the FIFO.

**Bug fix description**

Line added to top of bport\_receive\_actions().

---

**ID** 217   **Balloter**   CWS   **How submitted** Post ballot   **Fixed in version** D103  
**Title**   incorrect comment in rx\_sync\_error declaration   **Status** Recommend Accept  
**C code files**   **Owner**   CWS   **SCAT**  **BRAT**   0   **Recirc Clause** 17.3.2

**Problem description**

```
boolean rx_sync_error; // FALSE if failed to complete synchronization  
  
should be // TRUE if failed to complete synchronization
```

**Bug fix description**

*ID* 234 *Balloter* DW *How submitted* Post ballot *Fixed in version* D1041  
*Title* Full FIFO behavior *Status* Done  
*C code files* bport\_rx.c *Owner* JH *SCAT*  *BRAT* 0 *Recirc Clause* 17.3.2

***Problem description***

Need to allow the port state machine to push ending symbols into the FIFO, no matter what is already there.

***Bug fix description***

Change the C-code for the fifo between the port and arb state machines so that the port will write a value and then check to see if the fifo is full. If it is, it will not advance the pointer.

Originally, the sequence of LTS could cause the FIFO to overflow. David didn't care if LTS's were lost, but the final BUS\_RESET or ATTACH\_REQUEST must never be dropped. I'm not sure the LTS can overflow anymore since we have bport\_rx.c only pushing in changed LTS. Still, I think the behavior requested may be good for the general overflow situation. We're always guaranteed that after a very long run of stuff, we'll still always be able to put the last known arb/control state in the FIFO. So even if data gets scrambled/corrupted during the long sequence, clean-up should be a bit simpler since we know current stuff will always be present.

The actual change text was wrong, in my opinion. It stated that the write to the FIFO would be done first, and then the pointer conditionally advanced. If the pointer didn't advance because the FIFO was full, and subsequently no more pushes happened, then I don't think the last written element would be ever read. This is because the write pointer is still pointing at it, and when the read pointer becomes equal, it assumes the FIFO is empty and won't read the location. Said differently, the current FIFO will never read from the same location the write pointer is pointing to.

So to get this fixed, I found it more natural to treat the pointers differently. I think the Code as originally written was implementing a fifo in which the pointers always pointed to the "next" location to be written to or read from. Consequently, the write or read is done first, and then the pointer is advanced.

I change the meaning of the pointers to such that they always point to the "last" location read from or written to. As such, the pointers are incremented first, and then the read or write is performed. Empty is still the same ... rd\_ptr = wr\_ptr. And full is when the wr\_ptr is one "less" than the rd\_ptr. So I think this is a compatible model.

The benefit was that I could conditionally update the pointer when pushing, and then write the data. In all cases, the write pointer is left pointing to the last written location. And the empty test can never evaluate to true unless a read was performed on the same location as the write pointer. So the last written thing will always be read.

---

*ID* 235 *Balloter* DW *How submitted* Post ballot *Fixed in version* D1041  
*Title* Bport error counter behavior *Status* Done  
*C code files* bport\_rx.c *Owner* JH *SCAT*  *BRAT* 0 *Recirc Clause* 17.3.2

***Problem description***

Don't bump the error counter if padding is in the wrong place, or data is in the wrong place. This would most likely happen if the speed code was corrupted. If so, then we'll get tons of misplaced data and padding, causing the error count to run amuck even though we only had a single bit error.

***Bug fix description***

Only bump the error count for invalid 8/10 decodes ... not protocol layer stuff.

---

*ID* 236 *Balloter* DW *How submitted* Post ballot *Fixed in version* D1041  
*Title* Bport FIFO Data behavior *Status* Done  
*C code files* *Owner* JH *SCAT*  *BRAT* 0 *Recirc Clause* 17.3.2

***Problem description***

Must push one data byte into the FIFO for each expected data byte. So if we were expecting data and got either padding or invalid symbol, we want to still push into the fifo to prevent underrun. What we push is somewhat arbitrary ... the C code currently uses push\_data\_zero(). David prefers repeating the pad, but can live with any arbitrary data.

***Bug fix description***

Change push\_data\_zero to add weasel comments that what is pushed isn't important. As a corollary, we affirmed that we don't push at all when we weren't expecting a data byte.

**ID** 77      **Balloter** CWS                      **How submitted** Post ballot                      **Fixed in version** D1005  
**Title**      link\_concatenation, grant\_self, and TX:A0 and TX:TX mutual exclusion      **Status** Recommend Accept  
**C code files** various                                      **Owner**                      **SCAT**  **BRAT**      **Recirc Clause** 17.4

**Problem description**

If TX:A0 = TRUE && grant\_self = TRUE, then TX:TX is true as well. While investigating, it became clear that the distinction between link\_concatenation and grant\_self wasn't clear. In BEOP, link\_concatenation was being set if the link was to be granted, but grant\_self was not being set. As a result, the RX:TX transition would not be taken as intended if BEOP was called from within receive\_actions.

Also, it was noted that a con\_mgmnt\_request could be granted during a true legacy link concatenation. It is unclear if this would confuse legacy links.

**Bug fix description**

The proposed fix is detailed and included in the modified C code. In essence, link\_concatenation will only be used/set when a legacy link uses the hold cycle to request a concatenation. transmit\_actions will then prefer a link\_concatenation to other transmit requests. Grant\_self will replace link\_concatenation on the state machines for transitions to and from TX/RX.

Link\_concatenation is no longer required as parameter to be passed into beop or con\_mgmnt\_granted.

Note State machine changes required as per above

---

**ID** 291      **Balloter** CWS                      **How submitted** Post ballot                      **Fixed in version** D1071  
**Title**      PH\_Data indications cleanup                                      **Status** Done  
**C code files**                                      **Owner** CWS      **SCAT**  **BRAT** 0      **Recirc Clause** 17.4

**Problem description**

PH\_REQ\_SPEED is not used  
remove Beta\_format from PH\_data\_req\_service

**Bug fix description**

---

**ID** 293      **Balloter** CWS                      **How submitted** Post ballot                      **Fixed in version** D107  
**Title**      Elasticity in C code                                      **Status** Done  
**C code files**                                      **Owner** JH      **SCAT**  **BRAT** 0      **Recirc Clause** 17.4

**Problem description**

C code currently tries to take elasticity out of the 140ns of DP - needs to be in addition for all originated Legacy format packets.

**Bug fix description**

---

**ID** 12      **Balloter** CWS      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** read\_phy\_reg specification      **Status** Recommend Accept  
**C code files** decode\_phy\_pkt.c      **Owner**      **SCAT**  **BRAT** 485 **Recirc Clause** 17.4.1

**Problem description**

read\_phy\_reg does not distinguish between base registers and port registers, neither does its call

**Bug fix description**

Also BRAT #486  
add an extra flag to make this distinction  
byte read\_phy\_reg(int page, int port\_num, int reg, boolean base\_reg); // Not defined in C code  
// Return current value of specified PHY register

```
void remote_access(int page, int port_num, int reg, int ext_type) {  
// Current value of remotely read register  
phy_resp_pkt.dataQuadlet = 0;  
phy_resp_pkt.phy_ID = physical_ID;  
phy_resp_pkt.ext_type = (ext_type == 1) ? 3 : 7;  
phy_resp_pkt.page = page;  
phy_resp_pkt.port_num = port_num;  
phy_resp_pkt.reg = reg;  
phy_resp_pkt.data = read_phy_reg(page, port_num, reg, ext_type == 1);  
phy_response = TRUE;
```

**ID** 14      **Balloter** CWS      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** setting accelerating      **Status** Recommend Accept  
**C code files** transmit\_functions.c      **Owner**      **SCAT**  **BRAT** 480 **Recirc Clause** 17.4.1

**Problem description**

accelerating is not set after sending CYCLE\_START\_X

**Bug fix description**

```
set it  
if (!immediate_request && !iso_cycle && cycle_start_request) {  
portTarb(i, odd_isoch_phase ? CYCLE_START_ODD : CYCLE_START_EVEN);  
accelerating = TRUE;  
send_cycle_start_token = FALSE; // should not be TRUE anyway
```

**ID** 57      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version**  
**Title** send\_control      **Status** Recommend Reject  
**C code files** transmit\_functions.c      **Owner**      **SCAT**  **BRAT** 478 **Recirc Clause** 17.4.1

**Problem description**

(Comment #31)  
Table 16-14, pg. 318, function send\_control  
Control symbols should be sent to active ports only.  
Change pg. 318 line 28 from:

```
if (Beta_mode[i]) {  
to:
```

```
if (active[i] && Beta_mode[i]) {
```

and change line 35 from:

```
if (out_of_packet && i != not_to_port && Beta_mode[i]) {  
to:
```

```
if (out_of_packet && i != not_to_port && active[i] && Beta_mode[i]) {
```

**Bug fix description**

The active qualification is present in portTarb and portTarb\_at\_speed

---

**ID** 58      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D106  
**Title** data-prefix padding following speed-signal sequence      **Status** Recommend Accept in principle  
**C code files** transmit\_functions.c      **Owner** JS      **SCAT**  **BRAT** 479 **Recirc Clause** 17.4.1

**Problem description**

(Comment #32)

Table 16-14, pg. 318-319, function start\_tx\_packet

I don't think data-prefix padding following speed-signal sequence is correct. The fork process is unnecessary and doesn't really accomplish the goal of ensuring enough deletable symbols when a cycle-start symbol is to be sent. The min\_OK\_port\_speed variable is unnecessary (symbol stretching/padding is done at the pkt\_speed). I suggest the following.

Delete all references to min\_OK\_port\_speed.

Change pg. 319 lines 19-55 from:

```
fork
...
join
```

to:

```
#define MIN_DATA_PREFIX_TX 180 // Per 1394a, min data-prefix time preceding packet data
// (SPEED_SIGNAL_LENGTH + DATA_PREFIX_HOLD + 40)
```

```
if (sent_cycle_start)
    wait_next_symbol(min_operating_speed);
```

```
for (i = 0; i < NPORT; i = i + 1) {
    if (speed_OK[i]) {
        portTarb(i, DATA_PREFIX); // Ensures DS ports get data-prefix.
        if (!(pkt == LEGACY) && (link == LEGACY_LINK) && (pkt_speed == S100))
            portTspeed_general(i, pkt_speed, pkt);
    } else
        portTarb(i, DATA_NULL);
}
```

```
wait_next_symbol(pkt_speed); // Wait till speed-sig symbol completed.
```

```
for (i = 0; i < NPORT; i = i + 1)
    // Now send the next symbol on the Beta mode ports.
    if (Beta_mode[i])
        portTarb(i, speed_OK[i] ? DATA_PREFIX : DATA_NULL);
```

```
// Ensure timing for starting packet format, plus timing for deletable symbols. The
// requirement is that the originating node shall ensure that there is at least 17ns
// of deletable symbols at the start of each packet since the last deletable symbol
// was transmitted. The following code achieves this conservatively. Optimized
// implementations which recognize that deletable symbols have already been
// transmitted and reduce the time here accordingly are compliant.
```

```
if (pkt == LEGACY) {
    // Leave the speed signal for longer, send the next symbol on the DS ports.
    wait_time (SPEED_SIGNAL_LENGTH - symbol_time(rx_speed));
    for (i = 0; i < NPORT; i = i + 1)
        // Harmless writing this to the beta mode ports as well!
        portTarb(i, speed_OK[i] ? DATA_PREFIX : DATA_NULL);
    wait_time (convert(MIN_DATA_PREFIX_TX + DELETABLE_SYMBOL_TIME, pkt_speed) -
        SPEED_SIGNAL_LENGTH);
} else {
    wait_next_symbol(pkt_speed);
    wait_time (convert(DELETABLE_SYMBOL_TIME, pkt_speed));
}
```

**Bug fix description**

Agreed in principle

Minor adjustments to include the latest elasticity algorithm. Proposed mod to send DP with speedcode on Ds ports not included,

as this issue is fixed in dsport\_tx.c

C code checked in as PRN #571

---

<b>ID</b>	59	<b>Balloter</b>	Jim Skidmore	<b>How submitted</b>	Ballot comment	<b>Fixed in version</b>	D104
<b>Title</b>	stop_tx_packet			<b>Status</b>	Done		
<b>C code files</b>	transmit_functions.c	<b>Owner</b>	MT	<b>SCAT</b>	<input type="checkbox"/>	<b>BRAT</b>	481 <b>Recirc Clause</b> 17.4.1

**Problem description**

(Comment #33)  
Table 16-14, pg. 320, function stop\_tx\_packet  
Must add 20ns to end time to account for dribble bits on DS ports.

Change pg. 320 line 20 from:

```
if (pkt == LEGACY) wait_time(hold_time);
```

to:

```
if (pkt == LEGACY) wait_time(hold_time+20); // account for dribble bits on DS ports
```

**Bug fix description**

Opened a few issues. We realized that dribble bit time must be provided for whenever an actual data payload is sent, regardless of whether the local PHY is transmitting on a Legacy port or not. (Reason is that somewhere in the network, the packet may go across a Legacy port and needs the extra time to generate dribble bits and keep the DATA\_END time in spec.) Conversely, a null packet should not add the dribble bit time in a repeater since it may cause the repeater to fall behind during a closely spaced packet sequence.

Investigation of this bug also revealed that ports which had speed-filtered a payload would start sending DE while other ports were doing dribble bits. Consequently, some of the DE times would be out of spec.

Code was fixed with the addition of the non\_null\_packet flag which marks if an actual payload is present. If it is, ending packet actions allows for dribble bit times, and ports which speed filtered would delay heading to DE for the dribble time.

Additional action is requested in the text.

1) In the section on packet spacings, need to state that legacy packet "endings" shall include dribble bit time if and only if the transmitted packet had a data payload. Specing this may be a challenge. I guess we could require the DATA\_END on a beta port to be DATA\_END\_TIME + dribble bit time quantized appropriately, and to make sure that data bit separation from one legacy packet to a concatenated one shall always be > dribble bit time + MIN\_PACKET\_SEPARATION.

2) State the DATA\_END on speed-filtered legacy ports shall not start prior to the entirety of the dribble bit time, if any.

---

<b>ID</b>	60	<b>Balloter</b>	Jim Skidmore	<b>How submitted</b>	Ballot comment	<b>Fixed in version</b>	
<b>Title</b>	portR_next_arb			<b>Status</b>	Recommend Reject		
<b>C code files</b>	receive_functions.c	<b>Owner</b>		<b>SCAT</b>	<input type="checkbox"/>	<b>BRAT</b>	482 <b>Recirc Clause</b> 17.4.1

**Problem description**

(Comment #34)  
Table 16-15, pg. 320, function portR\_next\_arb  
I don't think the "if (packet\_ending[port\_num]) { ... }" is needed.

**Bug fix description**

The FIFO handshake is subtle, I admit and could be cleaned up. But the lines Jim is referring to are needed so that when the trailing DE of a packet is pulled from the FIFO, the FIFO will begin auto advancing. If the lines were removed, process\_req would be stuck with advance\_OK false and in\_packet true ... so the C code won't advance the FIFO. Effectively, the lines of code in question are used as an acknowledgement to say that receive\_actions have seen the ending symbol and it is safe to auto advance.

**ID** 61      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version**  
**Title** wait\_fifo\_fill\_time      **Status** Recommend Reject  
**C code files** receive\_functions.c      **Owner**      **SCAT**  **BRAT** 483 **Recirc Clause** 17.4.1

**Problem description**

(Comment #35)

Table 16-15, pg. 320-321, function wait\_fifo\_fill\_time

I don't think this function is correct (or even necessary). The requirement is that we wait at least 17 ns so as to guarantee that the rx FIFO won't underflow. To accomplish this, we need merely wait a couple of S100 tics. This function could simply be replaced with:

```
wait_time(20);
```

**Bug fix description**

[CWS] I think that there's a chance that the FIFO may already be sufficiently full, and if a blind wait is done, then it might overflow . . .

---

**ID** 62      **Balloter**      Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D104  
**Title**      multiple received speed-signals      **Status**      Recommend Accept in principl  
**C code files** receive\_functions.c      **Owner**      **SCAT**  **BRAT** 484 **Recirc Clause** 17.4.1

**Problem description**

(Comment #36)  
Table 16-15, pg. 321, function start\_rx\_packet  
Just as in 1394a, this function does not correctly handle the case of multiple received speed-signals (still a possibility).  
I suggest the following to replace the start\_rx\_packet function in its entirety.  
The wait\_Betaport\_event function can be eliminated.

```
#define MIN_DATA_PREFIX_RX (SPEED_SIGNAL_LENGTH + DATA_PREFIX_HOLD)

void start_rx_packet() { // Send data prefix and do speed signaling
                        // receive on DS or Beta, repeat to both

    int i;
    ArbState arb;
    boolean data_started; // actual data now in rx fifo

    arb_timer = 0;

    if (!DS_stuck) {
        max_beta_timer = 0; // Timer only needs to be implemented in border capable nodes,
        DS_stuck = TRUE; // and note that this will have to be released by sending
                        // a Legacy format packet.
    }

    portTarb(receive_port, IDLE); // Immediately send idle on the receive port
                                // (i.e., remove grant, if any).
    data_started = FALSE; // no data yet

    format = LEGACY; // Assume legacy format until proven otherwise.

    // Wait for data to begin. While waiting, process any speed-signaling received or
    // any gap events (subaction gap or arb-reset gap) which occur.

    // Upon entry into start_rx_packet, the incoming arb state on the receive_port
    // will be either DATA_PREFIX, DATA_NULL, or SPEED. If receive_port is a DS port,
    // then the arb state will be DATA_PREFIX.

    // Also ensure that for legacy packet the min data-prefix requirements are met.

    while (!data_started ||
           (arb_timer < convert(MIN_DATA_PREFIX_RX, rx_speed) && format == LEGACY)) {

        arb = portRarb[receive_port];

        // Process any gap events which may have occurred.
        if (send_async_start_token || arb_reset) begin
            gap_repeat_actions(FALSE); // This may or may not advance one S100 symbol.
            portTarb(receive_port, IDLE); // Restore ports to previous state.
            tx_control(DATA_PREFIX, receive_port); //
        end

        // Now process the last arb state read from rx fifo.

        if (data_started) // If data already started, then we must just be killing
            ; // time till enough data-prefix sent.
        else if (arb == SPEED) {
            rx_speed = portRspeed[receive_port];
            format = current_pkt[receive_port];
            tx_control(DATA_PREFIX, receive_port); // Take care of DS ports, which must set
                                                // the DP before the speed-signal is sent
                                                // (but no harm to Beta-mode ports).
            advance_OK[receive_port] = TRUE; // Allow next symbol to be read from rx FIFO.
        }
    }
}
```

```

// Note: There must necessarily be another symbol in the rx FIFO by the time
// the following processing completes.

// Repeat the speed signal.
for (i = 0; i < NPORT; i = i + 1)
  if (i != receive_port && active[i]) {
    speed_OK[i] = (rx_speed <= port_speed[i]) && (Beta_mode[i] || format == LEGACY);
    // Do not repeat Beta format packets on DS ports!
    if (speed_OK[i])
      portTspeed_general(i, rx_speed, format);
      // format only needed for Beta-mode ports
    else
      portTarb(i, DATA_PREFIX);
  }
wait_next_symbol(rx_speed); // Wait for speed symbol/DP to be sent.

// Now send next symbol on Beta-mode ports.
for (i = 0; i < NPORT; i = i + 1)
  if (i != receive_port && Beta_mode[i])
    portTarb(i, DATA_PREFIX);

    if (format == LEGACY) {
      // Extend the speed signal.
      wait_time(SPEED_SIGNAL_LENGTH - symbol_time(rx_speed));
      // Now send next symbol on DS ports.
      for (i = 0; i < NPORT; i = i + 1)
        if (i != receive_port)
          portTarb(i, DATA_PREFIX); // Harmless writing this to beta-mode ports as well!
      // Advance to next symbol boundary beyond MIN_DATA_PREFIX.
      wait_time(convert(MIN_DATA_PREFIX, rx_speed) - SPEED_SIGNAL_LENGTH);
    } else
      wait_next_symbol(rx_speed); // Wait for DP/DN symbol to be sent.
  }
else if (arb == DATA_BYTE)
  data_started = TRUE; // Exit loop (when enough DP).
else if (arb == DATA_PREFIX || arb == DATA_NULL) {
  tx_control(arb, receive_port); // Repeat the arb state.
  advance_OK[receive_port] = TRUE; // Allow next symbol to be read from rx FIFO.
}
else
  return; // There's no packet, so bomb out.
}
tx_speed = rx_speed;
wait_time (20);
}

```

#### **Bug fix description**

The spirit of the proposed fix was accepted, but several issues were noted with the actual code. Additional bugs in the original `start_rx_packet()` were noted and fixed as well. In summary:

- 1) multiple speed signals (concatenated) are now accepted
  - 2) packet format changes are tracked (sequence of null packets can go back and forth from beta to Legacy, etc.)
  - 3) If last packet format received was beta, the Legacy min prefix restrictions are ignored
  - 4) Minimum legacy prefix is enforced at all times for legacy formats, even if packet prefix terminated with IDLE, BUS\_RESET, DATA\_END, ARB\_CONTEXT, etc.
  - 5) Data\_prefix is not sent to a beta port until the format is definitely a legacy format.
-

**ID** 63      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D106  
**Title** decode\_phy\_packet      **Status** Recommend Partially Accept  
**C code files** decode\_phy\_pkt.c      **Owner** CWS      **SCAT**  **BRAT** 487      **Recirc Clause** 17.4.1

**Problem description**

(Comment #37)  
Table 16-16, pg. 324, function decode\_phy\_packet  
Must process only received self-ID packets.  
Change pg. 324 line 18 from:

```
if (!Beta_mode[receive_port] || !received_speed_signal) { // originated from node with Legacy link or DS node
```

to:

```
if (receive_port < NPORT &&  
    (!Beta_mode[receive_port] || !received_speed_signal)) { // originated from node with Legacy link or DS node
```

Setting of senior\_border and senior\_port variables should only be done after transmission of own self-ID.  
Change pg. 324 line 26 from:

```
if (Beta_mode[receive_port]) {
```

to:

```
if (Beta_mode[receive_port] && PHY_state == RX) {
```

Brackets needed.  
Change pg. 324 lines 45-49 from:

```
else if (phy_pkt.ext_type == 0xF) // Resume packet?  
...  
else if (phy_pkt.ext_type == 0xE) { // LTP packet?
```

to:

```
else if (phy_pkt.ext_type == 0xF) { // Resume packet?  
...  
} else if (phy_pkt.ext_type == 0xE) { // LTP packet?
```

**Bug fix description**

Partially accept.

First two subcomments rejected, as decode\_phy\_packet can only be called for received packets.

Third subcomment accepted

---

**ID** 64      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D106  
**Title** data\_coming      **Status** Recommend Accept  
**C code files** ds\_arb\_functions.c      **Owner** CWS      **SCAT**  **BRAT** 488 **Recirc Clause** 17.4.1

**Problem description**

(Comment #38)  
Table 16-17, pg. 325, function data\_coming  
Data coming is also indicated by reception of DATA\_NULL (see also process\_requests, which treats DATA\_NULL the same as DATA\_PREFIX, and sets the in\_packet flag, which halts auto increment of the rx FIFO).  
Change pg. 325 line 54 from:

```
if ((portRarb[i] == DATA_PREFIX) || (portRarb[i] == SPEED)) {
to:

if ((portRarb[i] == DATA_PREFIX) || (portRarb[i] == DATA_NULL) || (portRarb[i] == SPEED)) {
```

**Bug fix description**

**ID** 65      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version**  
**Title** arb\_OK      **Status** Recommend Reject  
**C code files** ds\_arb\_functions.c      **Owner** CWS      **SCAT**  **BRAT** 489 **Recirc Clause** 17.4.1

**Problem description**

(Comment #39)  
Table 16-17, pg. 326, function arb\_OK  
If con\_mgmnt\_request is set, then initiate async arbitration, similar to the isbr flag.  
At pg. 326 line ~34, insert the following just before "else if (breq == PRIORITY\_REQ)":

```
else if (con_mgmnt_request)
own_request = async_arb_OK;
```

**Bug fix description**

Formal reject - this has been superseded by revision to split con\_mgmnt\_request into restore\_request and loop\_test\_request, and to treat these as CURRENT requests in process\_requests()

**ID** 66      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** bestReq      **Status** Recommend Accept  
**C code files** beta\_arb\_functions.c      **Owner**      **SCAT**  **BRAT** 490 **Recirc Clause** 17.4.1

**Problem description**

(Comment #40)  
Table 16-18, pg. 327, function bestReq  
Parentheses required to group elements correctly (> operator higher priority than & operator).

**Bug fix description**

Change pg. 327 lines 10-12 from:

```
*in_phase_isoch_request = (iso_cycle && (best_req.isoch & *ipm > ISOCH_NONE & *ipm));
*in_phase_async_request = (!iso_cycle && (best_req.async & *apm >= CURRENT & *apm));
to:

*in_phase_isoch_request = (iso_cycle && ((best_req.isoch & *ipm) > (ISOCH_NONE & *ipm)));
*in_phase_async_request = (!iso_cycle && ((best_req.async & *apm) >= (CURRENT & *apm)));
```

**ID** 67      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version**  
**Title** boss\_end\_packet\_actions      **Status** Recommend Accept  
**C code files** beta\_arb\_functions.c      **Owner** CWS      **SCAT**  **BRAT** 491      **Recirc Clause** 17.4.1

**Problem description**

(Comment #41)  
Table 16-18, pg. 328-329, function boss\_end\_packet\_actions  
When granting link, need to set the grant\_self flag as well as the link\_concatenation flag.  
At pg. 328 line 51 and pg. 329 line 36 (2 places), just after the "link\_concatenation = TRUE;" statement and before the "return;" statement, insert:

```
grant_self = TRUE;
```

**Bug fix description**

Note that link\_concatenation is no longer used

---

**ID** 68      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D103  
**Title** gap\_repeat\_actions      **Status** Recommend Accept in principle  
**C code files** beta\_arb\_functions.c      **Owner** CWS      **SCAT**  **BRAT** 492      **Recirc Clause** 17.4.1

**Problem description**

(Comment #42)  
Table 16-18, pg. 330, function gap\_repeat\_actions  
The token\_receive\_port variable must be reset when done.  
At pg. 330 line 15, just before the terminating }, insert:

```
token_receive_port = NPORT;
```

**Bug fix description**

Jim's fix had a danger of clobbering token\_receive\_port too soon in the case of an ASYNC\_START followed immediately by ARB\_RESET\*. So a different fix was implemented.

---

**ID** 83      **Balloter** CWS      **How submitted** Post ballot      **Fixed in version** D1005  
**Title** Bad comment in phy\_response\_actions()      **Status** Recommend Accept  
**C code files** decode\_phy\_pkt.c      **Owner**      **SCAT**  **BRAT**      **Recirc Clause** 17.4.1

**Problem description**

comment at head of routine is "overloaded to transmit LTP packet"

I don't think this is correct anymore. There is a send\_LTP() routine which calls tx\_quadlet directly.

**Bug fix description**

delete comment

---

**ID** 93      **Balloter** CWS      **How submitted** Post ballot      **Fixed in version** D106  
**Title** Can get into A1 and A2 without requesting port being set correctly      **Status** Done  
**C code files** ds\_arb\_functions.c      **Owner** CWS      **SCAT**  **BRAT**      **Recirc Clause** 17.4.1

**Problem description**

arb\_ok() can cause converted\_request to be set true. However, requesting\_port may not be set properly to point to the port which caused converted\_request to be true. Seems like whenever async\_pending or isoch\_pending is set, we also need to capture which port was preferred so that arb\_ok can set requesting\_port when converted\_request is also set.

Also, doesn't appear own\_pending, isoch\_pending, or async\_pending are ever initialized.

**Bug fix description**

own\_pending flag changed to pending\_port, and requesting port set as recommended

---

**ID** 123    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D106  
**Title**            Cycle start starvation code doesn't grant senior port as intended                    **Status** Done  
**C code files** beta\_arb\_functions.c                    **Owner**    JH            **SCAT**  **BRAT**            **Recirc Clause** 17.4.1

**Problem description**

The test\_requests() routine within beta\_arb\_functions performs the 3 step check to avoid cycle start starvation. If the conditions are met for possible starvation, any asynchronous request is stomped by setting bap to NPORT and clearing the in\_phase\_async\_request flag. However, the current code execution will land at the line stating "if (\*best\_port == NPORT) return (grant\_link);"

This is incorrect in that we were intending to grant the senior port instead.

Perhaps the test for starvation was intened to be just \_before\_ the test on "in\_phase\_isoch\_request || in\_phase\_async\_request"?

**Bug fix description**

Code was fixed as part of grander cycle start fix (SCAT #207). However, it was noted that the line of code with the comment "kill all requests except proxy\_root link requests)" was necessary. It was intended to protect the pri\_req issued by a Legacy cycle master to send the cycle start packet. However, the pri\_req isn't detected as a BOSS request in test\_requests and, consequently, won't be reported in "bap". Instead, arb\_OK() detects that breq is set to pri\_req and will service the request in A0:IDLE. So the code has been changed to simply set in\_phase\_async\_request to FALSE if the starvation conditions test true.

**ID** 134    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D106  
**Title**            Cycle Master is not receiving PH\_ISOCH\_EVEN/ODD indications                    **Status** Done  
**C code files** transmit\_functions.c                    **Owner**    PhyDogs    **SCAT**  **BRAT**            **Recirc Clause** 17.4.1

**Problem description**

In start\_tx\_packet when the cycle master is granted to send a cycle start packet, a PH\_ISOCH\_EVEN/ODD indication is \_not\_ sent to the link. I think this means that the link will not know the isoch phase, ever, unless it counts (modulo 2) after bus resets. Seems a little kludgy. And it is important for OHCI style applications to know the isoch phase so that cycle matching can be done properly. (Want an isoch stream to start up in a particular cycle ... need to know if that cycle will be odd or even so that the proper beta request is made and granted.)

But not sure of the fix ... for parallel link, PH\_ISOCH\_EVEN/ODD can not be delivered to link if the link already owns the interface ... so we would need to send it before granting the link. This is probably okay ... we don't grant until after calls to start\_tx\_packet return.

**Bug fix description**

See SCAT #207

**ID** 136    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D104  
**Title**            incorrect comment in start\_rx\_packet()                    **Status** Recommend Accept  
**C code files** receive\_functions.c                    **Owner**                    **SCAT**  **BRAT**            **Recirc Clause** 17.4.1

**Problem description**

Line in start\_rx\_packet()

```
cur_format = LEGACY;                    // assume legacy format, if see a speed signal, might learn better
```

Comment seems incorrect. I don't think it is possible to have a non Legacy format packet if DATA\_PREFIX is received before a speed code. Said differently, a leading DP is guaranteed to be legacy format packet. Comment should be removed.

**Bug fix description**

Fixed by big changes to start\_rx\_packet in C Code bug #62

**ID** 206     **Balloter**     **How submitted** Post ballot     **Fixed in version** D104  
**Title**     Speed exchange in self\_ID     **Status** Done  
**C code files** transmit\_functions     **Owner**     MT     **SCAT**  **BRAT** 0     **Recirc Clause** 17.4.1

**Problem description**

S2:S3 calls portTspeed to send raw speed signal during speed signalling ... not clear how this causes dsport\_tx to call ds\_portTspeed.

**Bug fix description**

Make clear (probably with new arb token) whether we want DP to be transmitted with the speedcode.

Note, that state machine changes are need for S2:S3 and S3:S0.

Basically, portTspeed was renamed to portTspeed\_raw. portTspeed\_raw checks if the specified port is a !Beta\_mode and, if so, schedules a change on the DS speed signal driver by using a new arb state of SPEED\_RAW which dsport\_tx.c uses to modify ds\_portTspeed. portTspeed\_general was then renamed to portTspeed to clean up things a bit. The use of portTspeed in node.c was modified to call the "new" portTspeed properly with all parameters. Eventually this call will allow a restore packet to be sent in beta format on a B bus. (Subject of another PRN.)

---

**ID** 219     **Balloter**     Biva     **How submitted** Post ballot     **Fixed in version** D105  
**Title**     isbr initiation     **Status** Done  
**C code files**     **Owner**     JH     **SCAT**  **BRAT** 0     **Recirc Clause** 17.4.1

**Problem description**

I have a small doubt regarding isbr initiation from a beta node, not sure whether it has been already addressed.

Currently we find that the isbr initiation follows the same logic used for a legacy node (c-code for arb\_OK() function P.326, L.32, D.1.01). However, it seems that it may be worthwhile to also allow boss arbitration for generating isbr. Otherwise, even for a beta only bus the node needs to wait for more than 10us before arbitrating for isbr which is not required. (I am considering the cases where SBR is not immediately initiated and has to arbitrate following the quiet windows).

I think it can be handled similar to "con\_mgmt\_request" by replacing L.18-20 P.333 of Draft1.01 as below:

```
if (current_request || con_mgmt_request || (isbr && !isbr_OK))  
    // connection management request and request for short bus reset look  
like current requests  
    own_req.async = CURRENT;
```

**Bug fix description**

Agreed - see #126 for precise resolution

---

**ID** 241    **Balloter**    CWS    **How submitted** Post ballot    **Fixed in version** D107  
**Title**        DATA\_NULL handling    **Status**    Done  
**C code files** ds\_arb\_functions.c    **Owner**        JH    **SCAT**  **BRAT**    0    **Recirc Clause** 17.4.1

**Problem description**

DATA\_NULL doesn't seem to cause nodes to exit A0:Idle. Also, DATA\_NULL followed by an arb or an IDLE state doesn't seem to allow receive actions to exit.

**Bug fix description**

Receive actions fixed. Fixes summarized below:

While fixing, it was observed that an arbitrated short bus reset wasn't be handled properly (caused ARB\_CONTEXT to be sent instead). The actions were rewritten to allow BUS\_RESETS to complete immediately, IDLE to complete immediately provided that the beta ports hadn't been put into the DATA context yet, otherwise ARB\_CONTEXT is required.

Also, with the exception of BUS\_RESET, an unexpected end of actual data bits is patched with DATA\_END and no dribble bits.

stop\_tx\_packet is called for all ending states to help manage DS\_stuck. Additionally, DS\_stuck is now cleared in R0 to deal with the case of a long bus reset forcing a transition to R0 from any state.

To track whether Beta ports had entered the packet context yet, the format\_known variable in start\_rx\_packet was renamed to format\_committed and made global.

Also, observed that if a DP DN IDLE sequence was received, the DN may not be repeated for a minimum symbol time causing problems with the sudden IDLE transition on ports that didn't see the DN. Probably an error case that can't happen, but wasn't obvious. Could have effects like reset the known format and cutting short min\_data\_prefix. (Something like the FIFO is running behind and all three DP, DN, and IDLE symbols are present when start\_rx\_packet is called. As written, we'd zip through without much concern of min prefix timings, etc. Definitely not intended.)

Seemed safer to consider the DATA\_NULL following any DATA\_PREFIX as a bona fide ending symbol. So if DATA\_NULL is discovered after the format has been committed, start\_rx\_packet returns (after min Legacy timings were met) and the DATA\_NULL is treated as the ending state, causing a call to stop\_tx\_packet for the appropriate time guaranteeing that the DATA\_NULL is repeated to set context.

A similar failure mode may have resulted if a beta format packet was launched and then terminated with no data (link canceled or something). Could get Sb DP DP DP DP ... DN DN IDLE.

---

**ID** 255    **Balloter**    CWS    **How submitted** Post ballot    **Fixed in version**  
**Title**        Boss\_end\_packet\_actions violates MIN\_IDLE\_TIME?    **Status**    Done  
**C code files**    **Owner**        JH    **SCAT**  **BRAT**    0    **Recirc Clause** 17.4.1

**Problem description**

boss\_end\_packet\_actions makes many calls to stop\_tx\_packet with a wait time of MIN\_IDLE\_TIME. This doesn't seem long enough in the case that the receive port was a D/S port (ack packet from a junior port or isoch packet from a junior port allows calls to boss\_end\_packet\_actions) or in the case that the current packet format is Legacy format.

If the receive port is a D/S port, then we need to make sure that we allow the DATA\_END to conclude and MIN\_IDLE\_TIME to elapse before driving the subsequent DATA\_PREFIX. If we only wait 40 ns in stop\_tx\_packet, seems likely we could clobber the DATA\_END or IDLE.

And when repeating a Legacy format packet, only putting MIN\_IDLE\_TIME of DP before calling start\_tx\_packet again doesn't seem sufficient to meet MIN\_PACKET\_SEPARATION of 340 ns.

For P1394a, we used DATA\_END time when calling stop\_tx\_packet for a fly-by concatenation. I don't know why we thought MIN\_IDLE\_TIME was enough for boss\_end\_packet\_actions.

**Bug fix description**

Corrected in various places. Use DATA\_END\_TIME as safe time when receiving, otherwise use CONCATENATION\_PREFIX\_TIME

---

<b>ID</b> 265	<b>Balloter</b>	<b>How submitted</b> Post ballot	<b>Fixed in version</b>
<b>Title</b>	Packet descriptions		<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> MT	<b>SCAT</b> <input checked="" type="checkbox"/> <b>BRAT</b> 0 <b>Recirc Clause</b> 17.4.1

***Problem description***

need to fix the packet descriptions such that the elasticity time after speed code doesn't need to be quantized to packet symbol rates as it currently says

***Bug fix description***

To clean things up, all wait\_times in the C code for absolute time will be replaced by s400 symbol time counts. Also, all originated packet formats will spell out the possible generated formats, considering quantization effects at S200 and S100. For example, DE may appear for 3 or 4 S100 symbol times on an S100 port, and 6 or 7 on an S200 port.

<b>ID</b> 269	<b>Balloter</b> CWS	<b>How submitted</b> Post ballot	<b>Fixed in version</b> D107
<b>Title</b>	How do we prevent cycle start starvation from phy initiated current requests		<b>Status</b> Done
<b>C code files</b>		<b>Owner</b> MT	<b>SCAT</b> <input type="checkbox"/> <b>BRAT</b> 0 <b>Recirc Clause</b> 17.4.1

***Problem description***

restore\_request, loop\_test\_requests, and isbr are requests initiated by a PHY which are converted to current requests (as if they came from an attached link). Currently, it seems these requests will be serviced almost immediately in A0:IDLE if OK\_to\_grant is set. Can this cause a problem at the cycle\_master which has a Legacy Link? After receiving a packet, it takes some time for the pri\_req to come across LREQ. If we don't wait for that pri\_req (intended for cycle start packet), can't we run off and service these PHY requests, effectively starving the cycle start packet?

Perhaps we have to make sure that at a cycle master with a legacy link that OK\_to\_grant isn't set when we enter A0:IDLE in the async phase?

***Bug fix description***

Initial proposed fix was to ensure that in proxy root with a Legacy link and not in the isoch interval, wait in IDLE for the minimum possible subaction gap (gap-count = 5) (enough time to allow the cycle start request to be issued by the link) - 1.070 usec. This is because there's nothing in 1394a which requires a link to issue cycle starts with any greater urgency than this. Only applies when connecting a Legacy link through a Legacy PHY-Link interface.

Need SCAT updated to have CM\_MIN\_IDLE\_TIME defined and added to Clause 13 constants.

Notes on fix: Realized that simply extending the minimum time in idle isn't a good fix. If a child device believes that a cycle start isn't expected, it may issue a Legacy request before the 1 us elapses. In P1394a, this request would be immediately granted so there is no reason in P1394b to do differently. Also, no need to wait the full 1us if we already received a breq for the link. Finally, I was slightly worried that if someone did set the gap count to < 5, then gap count consistency would be blown by requiring the root to stay in A0 for 1 us no matter what. So fix was to remember if the previous packet marked the end of a subaction. If it did, then the legacy cycle master will set defered\_grant rather than OK\_to\_grant immediately in boss\_end\_packet\_actions. The in A0:Idle, OK\_to\_grant will be set at the first of the link making a breq, the 1 us timer popping, or the subaction gap + arb delay timer popping. By delaying OK\_to\_grant, no beta requests will be granted ... only legacy requests. So cycle start starvation is prevented as in 1394a.

**ID** 273    **Balloter** CWS                      **How submitted** Post ballot                      **Fixed in version** D107  
**Title**            Shouldn't advance bus interval if all the facts aren't known                      **Status** Done  
**C code files**    **Owner**    MT            **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.1

**Problem description**

In a B Bus, it looks like we might advance the interval too soon in some cases. Consider a junior node which receives a GRANT from a senior node, uses it to transmit a very short packet (an ack) , and then evaluates outstanding requests. While GRANTing the junior port, the senior port is in A2:Grant and will not exit until the junior responds with DATA\_NULL. So it is possible that by the time the junior node has finished transmission, it may not have any valid request information from the senior port (stalling seeing inbound GRANT). Consequently, it may attempt to incorrectly advance the interval.

Another scenario is when a junior grants a senior for an active request, and the senior no longer has an active request because it has been withdrawn, etc. The grant (isoch or async) coming from the junior port is blocking any other requests from the junior's juniors, so the senior should not attempt to advance the interval. The original fix for this was to prefer junior ports over senior ports for granting, and to refrain from advancing the interval until if BOSSship had been achieved via a grant from senior. This "fix" is in the 1.06 c code, but doesn't really help in the former case of short packet. (After transmitting the short packet, the PHY thinks BOSSship came from !senior\_port and may advance the phase, even though "GRANT" is still coming in from the senior port.

The C code almost has a fix ... it marks inbound requests on a port as invalid (A\_NONE, L\_NONE) when it receives the start of any packet or other events such as ARB\_CONTEXT, unexpected IDLE, and BUS\_RESET. The presence of invalid requests on active ports could be used to defer the advancement of the interval. The current C code falls short of doing that.

It might still be desirable to grant juniors over seniors from an efficiency point of view ... feels that way but not really sure.

**Bug fix description**

Agreed not to advance the phase if A\_NONE or L\_NONE are being received (as appropriate). Agreed to leave the "prefer junior" prioritisation in. C code modified. Found a number of small bugs with reintroducing arb reset tokens for error recovery. Fixed at the same time. Idle actions is a bit more filled out now.

Text needs updating

---

**ID** 275    **Balloter** CWS                      **How submitted** Post ballot                      **Fixed in version**  
**Title**            Advancing fairness intervals                      **Status** Recommend Reject  
**C code files**    **Owner**    PhyDogs    **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.1

**Problem description**

For consistency, can async phase advance in gap\_repeat\_actions, similar to isoch phases?

**Bug fix description**

After working on SCAT #218, this isn't worth persuing. Sometimes an arb reset gap token is issued to advance the phase, and sometimes it is issued to restate the current phase in case of recovery from a lost token. So we no longer blindly advance the phase with each arb reset token. Consequently, gap\_repeat\_actions() would need a parameter to know whether to advance or not ... doesn't seem worth it and it isn't currently broken.

---

**ID** 278    **Balloter** CWS                      **How submitted** Post ballot                      **Fixed in version** D107  
**Title**            Should arb\_OK() return false if DS\_stuck?                      **Status** Done  
**C code files**    **Owner**    JH            **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.1

**Problem description**

**Bug fix description**

Yes



**ID** 307 **Balloter** CWS **How submitted** Post ballot **Fixed in version** D107  
**Title** Incorrect loop bounds in start\_tx\_packet **Status** Done  
**C code files** **Owner** CWS **SCAT**  **BRAT** 0 **Recirc Clause** 17.4.1

**Problem description**

Includes following line:

```
for (i = 0; i < 3; i++) // send four for robustness
```

Less than comparison means loop only executes 3 times, not four as indicated in comment.

**Bug fix description**

change 3 to 4

---

**ID** 308 **Balloter** CWS **How submitted** Post ballot **Fixed in version** D107  
**Title** Consistency of remote confirmation packet **Status** Done  
**C code files** **Owner** CWS **SCAT**  **BRAT** 0 **Recirc Clause** 17.4.1

**Problem description**

As currently written, the command packet addressed to a nephew to initiate a standby includes a port field which is ignored. The port\_num is replaced with the number of the one active port (or NPORT if there is no active port). But before the replacement, the port field of the confirmation packet is formed from the original port\_num. Then, after the replacement of port\_num, the fault, connected, bias, etc. bits are filled in the confirmation packet (possible from port number NPORT - which is broken!!). So the port\_num in the confirmation packet is broken, but even when not broken, inconsistent, possible, with the status bits returned. This seems dangerous.

**Bug fix description**

port\_num and all the port status bits are ignored in a response to a standby command, whether or not the command was accepted.

---

**ID** 311 **Balloter** CWS **How submitted** Post ballot **Fixed in version** D107  
**Title** Clearing of did\_arbrst in a B\_Bus **Status** Done  
**C code files** **Owner** JH **SCAT**  **BRAT** 0 **Recirc Clause** 17.4.1

**Problem description**

Only seems to take place in A0:IDLE. But seems like if BEPA sends and arb\_reset and then grants someone, another subsequent arb\_reset should be allowed (after the grant) without requiring the bus to go into Idle.

**Bug fix description**

**ID** 328 **Balloter** CWS **How submitted** Post ballot **Fixed in version** D1071  
**Title** received\_speed\_signal cleared to soon **Status** Done  
**C code files** **Owner** JH **SCAT**  **BRAT** 0 **Recirc Clause** 17.4.1

**Problem description**

It is cleared with the packet ending symbol, but decode\_phy\_packet doesn't try to sample it until after the packet ending symbol.

**Bug fix description**

Approach was to set received\_speed\_signal appropriately at the top of start\_rx\_packet when the packet prefix is first detected. It then stays set until the next received packet. Works fine since start\_rx\_packet is always called prior to the self\_id\_packet reception.

---

**ID** 333     **Balloter**     CWS                     **How submitted** Post ballot                     **Fixed in version** D1071  
**Title**         sent\_one in send\_control causes lockup                     **Status** Done  
**C code files**     **Owner**         JH         **SCAT**  **BRAT** 0     **Recirc Clause** 17.4.1

**Problem description**

At a senior border with no beta ports, gap\_repeat\_actions() consumes zero time because send\_control doesn't send anything. AS a result, the two fork processes in idle\_actions fight over send\_async\_start when arb\_timer == subaction gap. The second process sets it, the first process calls gap\_repeat actions and clears the flag. But because time doesn't advance, the second process sets the flag again.

**Bug fix description**

sent\_one variable and testing removed from send\_control(). This causes gap\_repeat\_actions() to wait unconditionally.

---

**ID** 334     **Balloter**     CWS                     **How submitted** Post ballot                     **Fixed in version** D1071  
**Title**         Position of PH\_DATA\_ind(PH\_DATA\_BYTE) relative to tx\_byte should be a **Status** Done  
**C code files**     **Owner**         JH         **SCAT**  **BRAT** 0     **Recirc Clause** 17.4.1

**Problem description**

Current C code has the PH\_DATA\_indication(PH\_DATA\_BYTE) given at the end of a transmitted byte rather than the beginning. I used to think this was preferred since the PH\_DATA\_START would be given before the first data byte and thus would have one symbol time to be sent on a real phy/link interface before the first data byte. But in a real implementation, the PH\_DATA\_START is communicated by the arrival of the first byte of repeat/transmit data. So in some sense, PH\_DATA\_START and PH\_DATA\_BYTE are logically coincident. Since it makes the model of the PLI easier to develop, I proposed PH\_DATA\_BYTE be placed before the tx\_byte calls.

**Bug fix description**

---

**ID** 338     **Balloter**     CWS                     **How submitted** Post ballot                     **Fixed in version** D1071  
**Title**         stop\_tx\_packet needs to drive DP on ports not being suspended, disabled, **Status** Done  
**C code files**     **Owner**         CWS         **SCAT**  **BRAT** 0     **Recirc Clause** 17.4.1

**Problem description**

stop\_tx\_packet currently schedules SUSPEND, DISABLE\_NOTIFY, or STANDBY on the appropriate ports and then waits for 4 symbol times (which could be a long time). During this time, normal active ports are sending IDLE. This risks a subaction gap. Proper operation is as per 1394a ... while sending SUSPEND, other ports should see DATA\_PREFIX (or DATA\_NULL??) in preparation for the short bus reset.

Also ... should the time spent sending the 4 tokens be overlapped with the min deletable symbol and packet prefix times, or simply tacked on to the beginning?

**Bug fix description**

DATA\_NULL is better, as some ports are speed filtered, so don't want to send DATA\_PREFIX.

Not worth the complication of saving the time in these rare circumstances

---

**ID** 345     **Balloter**     CWS                     **How submitted** Post ballot                     **Fixed in version** D1071  
**Title**         start\_rx\_packet doesn't reset arb\_timer with reception of speed code                     **Status** Done  
**C code files**     **Owner**         CWS         **SCAT**  **BRAT** 0     **Recirc Clause** 17.4.1

**Problem description**

Effect is that with a really long DATA\_NULL or DP before a speed code, we may not insert elastic symbols as required after the speed code. (Arb timer wasn't reset, and it already ticked over past deletable + nondeletable time by the time we finished repeating the speed code.)

**Bug fix description**

*ID* 349     *Balloter*    CWS                                  *How submitted* Post ballot                                  *Fixed in version* D1071  
*Title*            Check the BEPA doesn't do isbr during isoch                                  *Status*    Done  
*C code files*    *Owner*        CWS     *SCAT*  *BRAT*    0    *Recirc Clause* 17.4.1

**Problem description**

Glancing at BEPA(), looks like whenever the link is about to be granted, isbr\_OK is set to isbr. On the surface, seems like this might let an isbr be granted during an isoch interval?

Also check elsewhere ... such as in idle\_actions().

**Bug fix description**

*ID* 360     *Balloter*    CWS                                  *How submitted* Post ballot                                  *Fixed in version* 1.1  
*Title*            speed\_ok not initialized properly with Legacy format S100 packet w/o speed *Status*    Done  
*C code files*    *Owner*        JH     *SCAT*  *BRAT*    0    *Recirc Clause* 17.4.1

**Problem description**

**Bug fix description**

in start\_rx\_packet, if a DATA\_PREFIX is discovered as the first element of a packet prefix, speed\_OK is set true for all ports and cur\_speed set to S100 (for symmetry with the SPEED case).

*ID* 362     *Balloter*    CWS                                  *How submitted* Post ballot                                  *Fixed in version* 1.1  
*Title*            non\_deletable\_time uses invalid conversion function                                  *Status*    Done  
*C code files*    *Owner*        JH     *SCAT*  *BRAT*    0    *Recirc Clause* 17.4.1

**Problem description**

For a beta format packet, start\_rx\_packet has the line:  
 non\_deletable\_time = Legacy\_time(2\*symbol\_time(cur\_speed));

Problem is that symbol\_time() is only defined for S100, S200, amd S400.

**Bug fix description**

To try and reduce confusion, symbol\_time() was renamed to Legacy\_symbol\_count() since it really returns the number of integer S400 symbols in a symbol of the specified speed. It can only be called with a parameter of S100, S200, or S400. A new function, symbol\_time() returns the exact duration (in seconds) of a symbol at the specified speed (over the whoel range of speeds). For beta format packets, this function is called to calculate the non\_deletable\_time. General clean-up as well ... all of the timing functions moved to receive\_functions and then wait\_symbol\_time rewritten to make a call to symbol\_time.

*ID* 369     *Balloter*    CWS                                  *How submitted* Post ballot                                  *Fixed in version* 1.2  
*Title*            gap\_repeat\_actions inverts async phase after sending arb reset token                                  *Status*    Done  
*C code files*    *Owner*        JH     *SCAT*  *BRAT*    0    *Recirc Clause* 17.4.1

**Problem description**

**Bug fix description**

**ID** 371     **Balloter** CWS                    **How submitted** Post ballot                    **Fixed in version**  
**Title** gap\_repeat\_actions inverts async phase                    **Status** Not yet done  
**C code files**    **Owner** JH     **SCAT**  **BRAT** 0     **Recirc Clause** 17.4.1

**Problem description**

gap\_repeat\_actions (both before and after PRN #818) is attempting to invert the asynchronous phase when sending and arb reset. However, in many cases, the phase has already been set or inverted prior to the gap\_repeat\_actions() call. For example, when gap\_token() finds an ARB\_RESET\_\*, it forces the async phase to the that specified by the ARB\_RESET\_\* and then calls gap\_repeat\_actions(). So we don't want to invert the phase we just set. Similarly, in idle\_action(), tokens are reintroduced at times when an error has been detected. Sometimes the token is re-introduced with a phase changes, and sometimes without a phase change. Both cases rely on gap\_repeat\_actions(), so best not to invert the phase automatically there.

**Bug fix description**

As a rule, the phase is always set just before arb\_reset is set. And in gap\_repeat\_actions(), the phase is not updated.

---

**ID** 13     **Balloter** CWS                    **How submitted** Ballot comment                    **Fixed in version** D1005  
**Title** use of accelerating                    **Status** Recommend Accept  
**C code files** process\_req.c                    **Owner**                    **SCAT**  **BRAT** 493     **Recirc Clause** 17.4.2

**Problem description**

accelerating variable is not set on legacy ISOCH\_REQ

**Bug fix description**

```
set it
case PH_ISOCH_REQ:
    accelerating = TRUE;
    breq = ISOCH_REQ;
    req_speed = phyArbReq.speed;
    break;
```

---

**ID** 69     **Balloter** Jim Skidmore                    **How submitted** Ballot comment                    **Fixed in version** D106  
**Title** process\_requests                    **Status** Recommend Accept  
**C code files** process\_req.c                    **Owner** CWS     **SCAT**  **BRAT** 494     **Recirc Clause** 17.4.2

**Problem description**

(Comment #43)  
Table 16-19, pg. 334-335, function process\_requests

Typo in comment, line 52, change from:  
// flag to ignore the test of the packet  
to:  
// flag to ignore the rest of the packet

**Bug fix description**

Typo comment accepted  
Filtering code, and DN terminating a packet code, added  
Final comment already dealt with in SCAT #76 BRAT#504

**ID** 78      **Balloter** CWS                      **How submitted** Ballot comment                      **Fixed in version**  
**Title** PHY Cancellation of "Late" Link Requests                      **Status** Done  
**C code files** various                      **Owner** DW      **SCAT**  **BRAT** 346 **Recirc Clause** 17.4.2

**Problem description**

Setup: At link in the even phase, link issues a next\_odd (Req #1) and sometime later issues a current (Req #2). Subsequently, the link receives from the PHY an ARB\_RST\_ODD followed by a GRANT. Assuming Req #1 and Req #2 have the same speed and format, the link can not determine whether the GRANT was for Req #1 or Req #2.

Scenario A: Phy issued ARB\_RST\_ODD and GRANT after receiving Req #1 but before Req #2. Consequently, when Req #2 arrives, it is queued by the PHY.

Scenario B: Phy issued ARB\_RST\_ODD before receiving Req #1. Req #1 and Req #2 both received by PHY before it issues a GRANT. In this case, Req #2 has updated and replaced Req #1. After issuing the GRANT, no requests are pending in the PHY.

From the link's perspective, scenario A and scenario B look identical in terms of order of arriving indications from the PHY. But in scenario A, a request is still pending in the PHY and the link should not reissue, and in Scenario B no requests are pending and the link does need to re-issue.

**Bug fix description**

Proposed fix is that after sending a GRANT to the link, the PHY discards all LREQs for the same "pipe" (asynch or isoch) as the GRANT until the link acknowledges the GRANT by taking possession of the PHY/Link interface. If so, then Scenario A would end up with no request pending in the PHY since it ignored Req #2 from the link.

---

**ID** 125      **Balloter** CWS                      **How submitted** Post ballot                      **Fixed in version** D103  
**Title** Clearing of isoch\_pending, asynch\_pending, and own\_pending                      **Status** Recommend Accept  
**C code files** process\_req.c                      **Owner**                      **SCAT**  **BRAT**      **Recirc Clause** 17.4.2

**Problem description**

Within process\_req.c, isoch/asynch/own\_pending are set only if we are the senior border and are processing our senior port. Seems like it would be possible for x\_pending to be set and then, after a bus reset, for it to stay set even though we may no longer be a senior border. Doesn't seem obviously correct ... it may allow arb\_OK to return true when we weren't expecting it to.

Perhaps x\_pending should be cleared in this code in this fashion:

```
if !senior_border || proxy root
  x_pending = FALSE;
else if i == senior_port
  x_pending set as before
```

**Bug fix description**

Added an else clause to clear all of the x\_pending requests.

**ID** 129    **Balloter** Jim Skidmore    **How submitted** Ballot comment    **Fixed in version** D105  
**Title** process\_requests    **Status** Done  
**C code files** process\_req.c    **Owner** JH    **SCAT**  **BRAT** 494    **Recirc Clause** 17.4.2

**Problem description**

(Comment #43) (cont)

The in\_packet flag should only be set after we've passed tree-ID because it's possible to get data-prefix arb state on DS ports during tree-ID, but it doesn't indicate the start of a packet. Also, DATA\_NULL can indicate the end of a packet as well as the start so set packet\_ending flag if DATA\_NULL recieved if in\_packet already set.

Change pg. 334 lines 54-55 from:

```
if (!in_packet[i]) in_packet[i] = TRUE; // throttle the FIFO  
else next_arb[i] = TRUE; // signal that this is the latest
```

to:

```
if (!in_packet[i] && PHY_state >= S0)  
    in_packet[i] = TRUE; // throttle the FIFO  
else {  
    next_arb[i] = TRUE; // signal that this is the latest  
    if (portCurrent'arb == DATA_NULL)  
        packet_ending[i] = TRUE;  
}
```

The received\_speed\_signal must be maintained til end of packet so that decode\_phy\_packet can use it.

At pg. 335 line 16 and line 37 (2 places), delete:  
received\_speed\_signal = FALSE;

**Bug fix description**

Agreed -

---

**ID** 233    **Balloter** CWS                    **How submitted** Post ballot                    **Fixed in version** D1041  
**Title**        grant\_received considered harmful                    **Status** Done  
**C code files** process\_req.c                    **Owner**        JH        **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.2

**Problem description**

In process\_requests, when we receive a GRANT outside of a packet, we set a flag called grant\_received. This flag is tested in idle\_actions() to figure out if we've been granted by our senior port, I believe. The fact that we set a flag rather than just look at portRarb(senior\_port) == GRANT suggests that we thought the GRANT arb state would be "gone" by the time we looked for it.

Well, I don't see how that can happen anymore. I believe the only time we can get an isolated GRANT (not attached to the end of a packet) is if the grant is traveling down (away from proxy root). And for a grant to head down, a PHY must be in A2:GRANT. And in A2:GRANT, the granted port is sent a continuous string of grants until the requests is explicitly canceled, the port goes inactive, or a packet transmission begins. So I think an out of packet GRANT indication persists long enough for it to be seen in IDLE.

Perhaps grant\_received was needed back in the days when beta grants were going to be short.

But a DS port can report false RX\_GRANT at times. For example, if a PHY is in A2:GRANT and generating a TX\_GRANT on a child port which is received RX\_REQUEST, that child port will currently see an RX\_GRANT and want to push it in the fifo. At the top of process\_requests, grant\_received gets falsely set and stays set allowing a non proxy-root to grant itself to soon.

**Bug fix description**

So I want to remove the grant\_received flag and replace tests of it in A0:IDLE with the portRarb tests.

Sure, we could add some more filtering to make sure a bogus RX\_GRANT doesn't go into the FIFO or get processed by process\_requests, but the whole thing goes away if we don't allow receipt of GRANT to have "side effects" other than to update portRarb. I can't see why GRANT should have side effects, so I think the clean solution is to remove grant\_received.

grant\_received was removed and everywhere it was being tested was replaced with !proxy\_root and portRarb(senior\_port) == GRANT

**ID** 239    **Balloter** CWS                    **How submitted** Post ballot                    **Fixed in version**  
**Title**        FIFO contents when port goes inactive                    **Status** Recommend Reject  
**C code files**                    **Owner**        CWS        **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.2

**Problem description**

What happens to FIFO when a port goes inactive? Will process requests go to quickly after !active? Is it possible for the FIFO to still contain data (maybe a confirmation packet) which should be forwarded on before killing the port? And what about error cases when we lose a port in the middle of receiving a packet. Does a synthesized ending arb state get pushed into the FIFO always, or does the reading end have to time out after a long while? How do we get the FIFO clearly flushed before it starts up again?

**Bug fix description**

There seem to be two almost diametrically opposed issues here. One is that the FIFO will get flushed too quickly, and valuable information may be lost, the second is that the FIFO will be left with stale information.

An arb\_state of IDLE is always pushed into the FIFO when the port goes inactive. Process\_requests is always eager to dequeue a FIFO if the port is !(active || restore). So it will quickly find the IDLE and dequeue it. So the FIFO will always be emptied quickly, and there's no danger of stale information being left in it.

I don't think that the first issue is valid either, but on a rather diferent argument. The only place where packet information is processed is in active. So the problem might occur if a peer port transmitted a packet, and then immediately turned the transmitting port off. In suspend, the suspend initiator sends the PHY response packet followed by a TX\_suspend and then waits for the incoming port to lose sync before stopping the outgoing port. The suspend target will only leave the active state when it sees the RX\_SUSPEND coming in. This handshake ensures that the TX\_SUSPEND is seen and acted on. A similar handshake is used for standby. All other cases are port failures of one sort or another, and so information has already probably been lost.

**ID** 243    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D105  
**Title**            Auto advancing through received tokens                    **Status** Done  
**C code files** process\_req.c                    **Owner**            JH            **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.2

**Problem description**

While in RX:Receive, and a token is received (AST, ARBRST, CST), we set a flag. But the token doesn't appear to be removed from the FIFO as I think was intended.

**Bug fix description**

C code fix (to advance past these arb states)

---

**ID** 254    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version**  
**Title**            Detection of speed signal in S2                    **Status** Recommend Reject  
**C code files**                    **Owner**            CWS            **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.2

**Problem description**

S2 is included in the state list for speed filtering. So while in S2, I don't think we can set received\_speed\_signal to true. However, we test received\_speed\_signal during S2 in decode\_phy\_packet to figure out senior border position, etc. How, in S2, can we find out if a self\_ID packet was delivered with a speed code on it or not?

**Bug fix description**

Qualification is modified by !Beta\_mode, so there is no problem

---

**ID** 309    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D107  
**Title**            PH\_RESET not complete                    **Status** Done  
**C code files**                    **Owner**            CWS            **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.2

**Problem description**

The C code for PH\_RESET in capture\_link\_requests() is not complete. Ideally, it would document the initialization process: cancel\_requests(), do a self\_identify, and indicated DP until the PHY/Bus is at a safe point. The last point is a bit of a challenge and would require some new C code to handle.

To date, we've dodge this bullet by only documenting in C code PHY/Link interface initialiations which were initiated by the PHY (and thus in sync with arb): namely; a bus reset or a restore.

So solution could be to either invent the new C code to determine when it is safe to release the bus, or add weasel comments to PH\_RESET, or simply remove PH\_RESET since it is redundant with the LPS reset mechanism.

**Bug fix description**

Code added, plus some weasel words

---

**ID** 329    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D1071  
**Title**            Underrun doesn't free FIFO to auto advance                    **Status** Done  
**C code files**                    **Owner**            JH            **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.2

**Problem description**

If an underrun is detected in receive\_actions (while in a packet), process\_requests isn't freed to auto flush the FIFO. So it is possible for data to stay blocked in the FIFO and for the arb machine to hang since no new arb states come in the stuck port. Proper operation is to inform process requests that the packet is over, and that things should be auto flushed until the next start of packet. This can be accomplished by getting process\_requests() in\_packet flag set false. One brute force method of doing this is to set packet\_ending true on underrun, and then set advance\_OK true. process\_requests will wake up, see packet\_ending, and clear in\_packet. Other alternative it to just clear in\_packet directly.

**Bug fix description**

Fix in receive\_actions which has us setting packet\_ending if receive\_actions detects and underflow, and then freeing the FIFO to move ahead as normal (as if we had called portR\_next\_arb to find the end of packet).

---

**ID** 336    **Balloter** CWS    **How submitted** Post ballot    **Fixed in version** D1071  
**Title** clear Link\_CS\_indications on PHY/Link reset    **Status** Done  
**C code files**    **Owner** CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.2

**Problem description**

Clear on PH\_RESET and PH\_LPS\_INACTIVE and remove comments in power reset

**Bug fix description**

---

**ID** 85    **Balloter** CWS    **How submitted** Post ballot    **Fixed in version** D1005  
**Title** grant\_self and send\_null\_packet not initialized    **Status** Recommend Accept  
**C code files** reset.c    **Owner**    **SCAT**  **BRAT**    **Recirc Clause** 17.4.3

**Problem description**

grant\_self is first tested on A0:TX but has not yet been assigned. send\_null\_packet is first tested at the top of transmit\_actions, but may not have been assigned yet (first visit to transmit\_actions).

**Bug fix description**

Recommended fix is to add initialization of both of these values in reset\_start\_actions to FALSE.

---

**ID** 94    **Balloter** CWS    **How submitted** Post ballot    **Fixed in version** D1005  
**Title** Race condition for resumption\_done    **Status** Recommend Accept  
**C code files** reset.c    **Owner**    **SCAT**  **BRAT**    **Recirc Clause** 17.4.3

**Problem description**

Each port in resume\_actions attempts to synchronize to the start of a new bus reset by making sure that bus\_initialize\_active is false before setting ibr, isbr, and resumption\_done. After setting these, the port will wait for bus\_initialize\_active to go true.

**Bug fix description**

reset\_start\_actions was rearranged such that bus\_initialize\_active is set immediately upon entry. A wait could be thought of as being inserted at that point to allow enough time for isbr, ibr, and resumption\_done signals to propagate from a port which just missed the rising edge of resumption\_done. The C code doesn't contain the wait and some handwaving argument about single threading within routines can be given.

---

**ID** 132    **Balloter** CWS    **How submitted** Post ballot    **Fixed in version** D1006  
**Title** Wrong port speed for reset\_start\_actions()    **Status** Recommend Accept  
**C code files** reset.c    **Owner**    **SCAT**  **BRAT**    **Recirc Clause** 17.4.3

**Problem description**

reset\_start\_actions() schedules a bus reset on all active ports by calling portTarb on all ports. portTarb sets the portT speed to DEFAULT for all active ports. Since reset\_start\_actions() must also schedule a reset on resuming and attaching ports, it sets portT "by hand" for those ports. However, it currently is using a speed of S100 rather than DEFAULT.

**Bug fix description**

For consistency with the active ports, the speed for resuming and attaching ports should be DEFAULT as well.

---

**ID** 208    **Balloter** CWS    **How submitted** Post ballot    **Fixed in version** D105  
**Title** Max\_Legacy\_path\_speed not initialised    **Status** Done  
**C code files** reset.c    **Owner** CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.3

**Problem description**

Need to be initialised in R0

**Bug fix description**

Do all the initialisation in R0, remove partial initialisation from self\_ID\_transmit\_actions()

**ID** 209    **Balloter** CWS                    **How submitted** Post ballot                    **Fixed in version** D103  
**Title**        Gap\_count\_reset\_disable is not initialised                    **Status** Recommend Accept  
**C code files** reset.c                                    **Owner**        CWS            **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.3

**Problem description**

should be set to FALSE in arb\_power\_reset

**Bug fix description**

---

**ID** 282    **Balloter** DW                                    **How submitted** Post ballot                    **Fixed in version** D107  
**Title**        Link-on behavior on powerup                                    **Status** Done  
**C code files**    **Owner**        DW            **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.3

**Problem description**

see email  
can have a node which the PHY is primed to generate an interrupt if something happens, and then go into low power. Then if the PHY power cycles, these setting are lost. So good to assert LinkOn (provided that power class permits this) to wake up the link.

**Bug fix description**

On power up of a PHY, send Link On if the power class is 0-4, for a min of 166 usecs (16384 PClk cycles). Max is implementation dependent. (CWS, Ch 14 currently has a max of 500ns . . .)  
Put a PH indication in the power reset code (with a cross-reference to the spec for full details) (done)

---

**ID** 70      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** Child ports in Tree\_ID      **Status** Recommend Accept  
**C code files** tree\_id.c      **Owner**      **SCAT**  **BRAT** 496 **Recirc Clause** 17.4.4

**Problem description**

(Comment #10)

Table 16-21, pg. 339, function tree\_ID\_start\_actions

Child ports are not always marked as child ports. Consider the case of a node receiving PARENT\_NOTIFY on all active ports (the node should become root). As written, the code will not mark the last active port as a child, causing an unnecessary root-contention cycle. The code that checks the child count should be moved outside of the for loop. I suggest the following.

**Bug fix description**

Change pg. 339 lines 11-24 from:

```
do {  
...  
} while (!(reset_detected() || ibr || isbr || arb_timer == CONFIG_TIMEOUT));
```

to:

```
do {  
  children = 0; // Count the kids afresh on each loop  
  for (i = 0; i < NPORT; i = i + 1)  
    if (!active[i] || portRarb[i] == PARENT_NOTIFY) {  
      child[i] = TRUE; // Child if disabled, disconnected, suspended,  
      children++; // or if other PHY asks us to be parent.  
    }  
  if (children == NPORT - 1 && (!force_root || arb_timer >= FORCE_ROOT_TIMEOUT))  
    return; // Only one port left as the parent.  
  else if (children == NPORT)  
    return; // We are the root.  
} while (!(reset_detected() || ibr || isbr || arb_timer == CONFIG_TIMEOUT));
```

---

**ID** 71      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** lowest\_unidentified\_child      **Status** Recommend Accept  
**C code files** self\_id.c      **Owner**      **SCAT**  **BRAT** 497 **Recirc Clause** 17.4.5

**Problem description**

(Comment #07)  
Table 16-22, pg. 340, function self\_ID\_start\_actions  
The lowest\_unidentified\_child variable is not computed correctly (its always comes out as the lowest numbered child port).

**Bug fix description**

Change pg. 340 lines 13-20 from:

```
for (i = 0; i < NPORT; i++) {  
    if (child_ID_complete[i]) // Tell identified children to prepare to receive data  
        portTarb(i, DATA_PREFIX);  
    else  
        portTarb(i, IDLE); // Allow parent to finish  
}  
for (i = 0; i < NPORT; i++)  
    if (child[i] && active[i]) { // If active child  
        if (all_child_ports_identified)  
            lowest_unidentified_child = i;  
        all_child_ports_identified = FALSE;  
    }  
}
```

to:

```
for (i = 0; i < NPORT; i = i + 1)  
    if (child_ID_complete[i]) // Tell identified children to prepare to receive data.  
        portTarb(i, DATA_PREFIX);  
    else {  
        portTarb(i, IDLE); // Allow parent to finish  
        if (child[i] && active[i]) { // If active child  
            if (all_child_ports_identified)  
                lowest_unidentified_child = i;  
            all_child_ports_identified = FALSE;  
        }  
    }  
}
```

Agreed ... i = i + 1 replaced with i++.

---

**ID** 72      **Balloter** Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** senior\_border starting assumption      **Status** Recommend Reject  
**C code files** self\_id.c      **Owner**      **SCAT**  **BRAT** 498 **Recirc Clause** 17.4.5

**Problem description**

(Comment #08)  
Table 16-22, pg. 342, function self\_ID\_transmit\_actions  
If node is a border node, the starting assumption is that it's the senior\_border and that there's NO senior\_port.  
Change pg. 342 line 11 from:

```
senior_port = parent_port; // working assumption, may be changed later
```

to:

```
senior_port = NPORT; // working assumption, may be changed later
```

**Bug fix description**

Further issue shown up by Biva - see C code bug #113 for complete solution

Tree ID establishes parent\_port and root, and we presume senior\_port and proxy\_root settings are identical. Self ID establishes a true proxy\_root and, if different from root, senior\_ports are adjusted to always point to the proxy\_root. Once in normal arbitration, root and parent\_port are completely ignored. All arbitration is based on senior\_port and proxy\_root.

---

**ID** 73      **Balloter** Clay E Hudgins      **How submitted** Ballot comment      **Fixed in version** D1005  
**Title** casual comment      **Status** Recommend Accept  
**C code files** self\_id.c      **Owner**      **SCAT**  **BRAT** 499 **Recirc Clause** 17.4.5

**Problem description**

p. 343, line 5-6. "I'm boss" seems too casual.

**Bug fix description**

delete comment

---

**ID** 82      **Balloter** CWS      **How submitted** Post ballot      **Fixed in version** D1005  
**Title** proxy\_root not initialized for an isolated node      **Status** Recommend Accept  
**C code files** self\_id.c      **Owner**      **SCAT**  **BRAT**      **Recirc Clause** 17.4.5

**Problem description**

Current spec assigns proxy\_root within decode\_phy\_packet(). However, an isolated node doesn't call decode\_phy\_packet since it never receives any self-ID's from other nodes.

**Bug fix description**

Suggested fix is to copy line

```
proxy_root = (B_bus && root) || (senior_border && (root || Beta_mode[parent_port]));
```

from decode\_phy\_packet and place it within self\_ID\_transmit\_actions() immediately before the "if (root and B\_Bus)" test.

Note that the assignment for proxy\_root is required in both places. In some cases, we never get to decode\_phy\_packet(). In others, we decode\_phy\_packet after self\_ID\_transmit\_actions and we need to revise our first proxy\_root guess.

---

**ID** 103    **Balloter** CWS                      **How submitted** Post ballot                      **Fixed in version** D1005  
**Title**        Uncle self-ID    **Status** Recommend Accept  
**C code files** self\_id.c                                      **Owner**                                      **SCAT**  **BRAT**                      **Recirc Clause** 17.4.5

**Problem description**

Self\_ID transmit actions are wrong - they send the wrong value for a port in standby on an an uncle

**Bug fix description**

modify line 30 from

```
else if (!active[port_number])
```

to

```
else if (!active[port_number] && !proxy[port_number])
```

---

**ID** 112    **Balloter** Biva                                      **How submitted** Post ballot                      **Fixed in version** D1006  
**Title**        self\_ID and proxy    **Status** Recommend Accept in principl  
**C code files** self\_id.c                                      **Owner**                                      **SCAT**  **BRAT** <sup>^</sup>                      **Recirc Clause** 17.4.5

**Problem description**

P.340, L.18-20 (self\_ID\_start\_actions()) should include child\_ID\_complete[i] and proxy[i] flags too. Hence the condition should look like

```
if (!child_ID_complete[i]) {
    if (child[i] && (active[i] || proxy[i]))
        // or ((child && active) || proxy) ??
        lowest_unidentified_child = i;
    all_child_ports_identified = FALSE;
}
```

**Bug fix description**

[JH] Fix above doesn't appear to be complete. Skid's #7 (C code bug #71) dealt with what is documented above, but I expect that Biva is pointing out an additional bug related to proxy ports. Awaiting the complete fix before implementing ... Biva's little questioning comment is interesting (and, as it turns out, benign).

At this point, not all combinations of Child/Active/Proxy are possible. In particular, if Active is FALSE, then Child is TRUE. Equally, Active and Proxy cannot both be TRUE.

So the possible combinations are:-

Child	Active	Proxy	Set lowest_unidentified_child?
F	T	F	N
T	F	F	N
T	F	T	Y
T	T	F	Y

So Biva's fix is the "obvious" one, and his alternative is IMHO, less obviously correct (although it still happens to work)  
 Bug fixed (format slightly different due to also fixing a bug from Skid)

---

**ID** 113      *Balloter*    Biva      *How submitted* Post ballot      *Fixed in version*

*Title*      senior\_port in self\_ID      *Status* Done

*C code files* self\_id.c      *Owner*      *SCAT*  *BRAT*      *Recirc Clause* 17.4.5

*Problem description*

In self\_ID\_transmit\_actions() (P.342, L.11) the "senior\_port = parent\_port;" should, in my opinion, be replaced with  
senior\_port = (root || Beta\_mode[parent\_port]) ? NPORT : parent\_port;  
// We can neither set it blindly to NPORT, nor to parent\_port.

*Bug fix description*

[JH] Similar bug reported as Skid #8 (C code bug #72) which blindly sets senior\_port to NPORT. senior\_port is then updated later by decode\_phy\_packet if necessary. Basically, senior\_port points toward proxy\_root the same way parent\_port points to root. senior\_port == NPORT only at proxy\_root.

The solution takes the approach: at the end of tree-id, parent\_port == senior\_port and root == proxy\_root, so the two "trees" are coincident. During self\_id, if a border is advertised, proxy\_root and senior\_port are updated. Whenever senior\_port is assigned, so is proxy\_root to enforce that one implies the other. In fact, proxy\_root == TRUE iff senior\_port == NPORT.

---

**ID** 130      *Balloter*    CWS      *How submitted* Post ballot      *Fixed in version* D1006

*Title*      grant\_received not being cleared in self-id      *Status* Recommend Accept

*C code files* self\_id.c      *Owner*      *SCAT*  *BRAT*      *Recirc Clause* 17.4.5

*Problem description*

S0:S1 relies on receiving a GRANT or SELF\_ID\_GRANT from the parent\_port. The receipt of a grant causes process\_requests.c to set grant\_received. Consequently, when a non-root node completes self-id and heads off into A0, grant\_received may be stale and set to TRUE, causing undesirable visits to TX (possibly after a quick visit to RX to receive more self-ID packets).

*Bug fix description*

clear grant\_received at top of self\_ID\_grant\_actions()

---

**ID** 223      *Balloter*    DW      *How submitted* Post ballot      *Fixed in version* D104

*Title*      register 0 in self\_ID      *Status* Done

*C code files* self\_id.c      *Owner*    JH      *SCAT*  *BRAT*    0      *Recirc Clause* 17.4.5

*Problem description*

move register 0 transfer so that it is earlier than it is now in self\_ID transactions so that it is coincident with data\_end indications. (Phydogs 8/18)

*Bug fix description*

See also Scat #237

---

**ID** 226    **Balloter** Jim Skidmore    **How submitted** Post ballot    **Fixed in version** D103  
**Title** C code tidbit in self\_ID\_grant\_actions()    **Status** Done  
**C code files** self\_id.c    **Owner** JH    **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.5

**Problem description**

In self\_id\_grant\_actions, braces and additional test of active flag are needed to get correct arbitration signal sent on ports which are not the lowest\_unidentified\_child port.

**Bug fix description**

Change from:

```
for (i = 0; i < NPORT; i++) {  
    if (!all_child_ports_identified && (i == lowest_unidentified_child))  
  
        if (!proxy[i]) portTarb(i, GRANT); // Send grant to lowest unidentified child (if any)  
        else portTarb(i, DATA_PREFIX); // Otherwise, tell others to prepare for packet  
}
```

to:

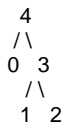
```
for (i = 0; i < NPORT; i++) {  
    if (!all_child_ports_identified && (i == lowest_unidentified_child))  
{  
        if (!proxy[i]) portTarb(i, GRANT); // Send grant to lowest unidentified child (if any)  
        } else portTarb(i, DATA_PREFIX); // Otherwise, tell others to prepare for packet  
}
```

**ID** 325    **Balloter** CWS    **How submitted** Post ballot    **Fixed in version** D1071  
**Title** S0 needs to wait MIN\_IDLE\_TIME?    **Status** Done  
**C code files**    **Owner** CWS    **SCAT**  **BRAT** 0    **Recirc Clause** 17.4.5

**Problem description**

It seems that S0 needs to wait MIN\_IDLE\_TIME before sending DP to identified children.

Here's a scenario



The situation occurs when node 1 has just finished transmitting its self-ID, and occurs on the 4 - 0 connection. Node 1 gives an ident done, but this gets translated into IDLE on the 3 - 4 connection. Node 4 is in self\_ID receive, and on receiving the idle from 3, repeats this to 0, but then goes immediately into S0, and immediately starts to generate DP on 4 - 0, as 0 has already identified. So poor 0 does not see a full MIN\_IDLE\_TIME.

I have several problems though.

Is this going to affect anything else - if we entered S0 because we saw DATA\_PREFIX (S2:S0), will we "get behind" by not responding and taking S0:S2 quickly enough? Or is that S2:S0 DP test there because other nodes are not guaranteeing min idle time after the previous data\_end?????? (I vaguely remember Lou Fasano from IBM insisting on this test when we were doing 1394a). If someone else is not respecting min idle time, then we can't just invent it!!!

Not trivial . . .

**Bug fix description**

Fix is to wait at the end of S2 to repeat IDLE (or until one of the other S2 exit conditions) for MIN\_IDLE\_TIME

**ID** 327     **Balloter**     CWS                     **How submitted** Post ballot                     **Fixed in version** D1071  
**Title**             Speed negotiation of DS ports                     **Status** Done  
**C code files**     **Owner**     MT     **SCAT**  **BRAT** 0     **Recirc Clause** 17.4.5

**Problem description**

Currently, the C code and self\_ID state machine says that DS ports send PHY\_SPEED to each other during self\_ID to determine the operating speed of a DS\_mode connection.

With PHY\_SPEED often exceeding S400, this seems a little optimistic.

This raises two questions:-

- 1) What speed should be used
- 2) What speed constraints (if any) do we want to put on ports capable of operating in DS mode.

To discuss the second question first.

2.1) There is a requirement currently in the spec that a bi-lingual port shall be capable of supporting S100-S400 when operating in DS mode.

2.2) The spec is silent on ports which operate in DS mode only. I believe that the intention was to require S100-S400 as per 2.1 - but I may be wrong.

There is a port field "max operating speed", but that applies strictly to Beta mode operation. Indeed, it seems to me that in full generality a port has a max Beta mode operating speed and a max DS mode operating speed, which are independent of each other.

OPTION A) If I am right concerning 2.2 above, then the fix is easy - the speed code which is send during self\_ID on ports operating in DS mode is S400.

But this seems to me to be a little harsh in the case of a product which is a 2-port DS to UTP repeater - there is no point of running the DS port above S100 - and that can save all the analog speed stuff. Indeed, even if the "DS" port is a bi-lingual port, the same holds true.

Likewise, if the product is a UTP/POF to DS repeater, there is no point in running the DS port above S100 if the UTP/POF is configured for UTP, or S200 if it is configured for POF.

OPTION B) So it seems that we probably should have a "max DS speed" field in the port register map and use this value at self\_ID.

**Bug fix description**

Implementation dependent array of DS port speeds

No constraints on implementations of DS ports.

---

**ID** 347     **Balloter**     CWS                     **How submitted** Post ballot                     **Fixed in version** D1071  
**Title**             Self\_ID as soon as possible                     **Status** Done  
**C code files**     **Owner**     CWS     **SCAT**  **BRAT** 0     **Recirc Clause** 17.4.5

**Problem description**

The reason we want to do this is to make sure that the link has some extra time to send it's first async request before self\_id concludes. In a B bus, self\_ID concludes quickly and can give both an ASYNC\_START and an ARB\_RST\_\* before the first async request comes over to the PHY assuming we wait until after the self\_ID packets are delivered to launch the async request. The ARB\_RST\_\* is wasteful and slightly unfair if the PHY in question didn't finish up it's async interval prior the the bus reset which invoked the self\_id phase.

**Bug fix description**

**ID** 15      **Balloter** CWS                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**        reset of did\_arbrst    **Status** Recommend Accept  
**C code files** idle.c    **Owner**    **SCAT**  **BRAT** 500 **Recirc Clause** 17.5

**Problem description**

need to clear this on exit even in iso\_cycle, to be consistent with the way that 1394a behaves (any time that there are no requests to process after the bus has been busy causes a gap)

**Bug fix description**

remove the test on iso\_cycle  
did\_arbrst = FALSE;                      // only relevant to a B\_bus

---

**ID** 17      **Balloter** CWS                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**        Isoch concatenated packets    **Status** Recommend Accept  
**C code files** transmit\_actions.c    **Owner**    **SCAT**  **BRAT** 503 **Recirc Clause** 17.5

**Problem description**

Hold protocol is not used for concatenated iso packets from beta link

**Bug fix description**

Remove C code for this  
} else if (data\_to\_transmit.reqType == PH\_REQ\_DATA\_PREFIX) {  
    // concatenated packets from Legacy link  
    end\_of\_packet = link\_concatenation = TRUE; // End of packet indicator  
    stop\_tx\_packet(DATA\_NULL, tx\_speed, CONCATENATION\_PREFIX\_TIME, tx\_format, NPORT);  
    // MIN\_PACKET\_SEPARATION  
    req\_speed = data\_to\_transmit.speed;

---

**ID** 74      **Balloter** Jim Skidmore                      **How submitted** Ballot comment                      **Fixed in version** D1005  
**Title**        idle\_actions parens    **Status** Recommend Accept  
**C code files** idle.c    **Owner**    **SCAT**  **BRAT** 501 **Recirc Clause** 17.5

**Problem description**

(Comment #44)  
Table 16-23, function idle\_actions, pg. 346  
Parentheses required to group elements correctly (== operator is higher priority than ? operator).

**Bug fix description**

Change pg. 346 lines 10-11 from:

```
if (!OK_to_grant && proxy_root &&  
    (arb_timer == STORES_GAP_COUNT ? subaction_gap: BOSS_RESTART_TIME)) {
```

to:

```
if (!OK_to_grant && proxy_root &&  
    (arb_timer == (STORES_GAP_COUNT ? subaction_gap: BOSS_RESTART_TIME))) {
```

---

<b>ID</b> 75	<b>Balloter</b> Jim Skidmore	<b>How submitted</b> Ballot comment	<b>Fixed in version</b>
<b>Title</b>	transmit_actions	<b>Status</b>	Recommend Accept in principle
<b>C code files</b>	transmit_actions.c	<b>Owner</b> MT	<b>SCAT</b> <input type="checkbox"/> <b>BRAT</b> 502 <b>Recirc Clause</b> 17.5

**Problem description**

(Comment #09)

1) Table 16-25, pg. 347-348, function transmit\_actions

The immediate\_request variable is used later in the function and should not be set FALSE until then. Delete pg. 347 line 38:

```
immediate_request = FALSE;
```

2) The code does not correctly reset an immediate request from a legacy link. Move pg. 348 line 7

```
immediate_request = FALSE;
```

to immediately before the while statement at pg. 348 line 15.

3) The while loop which processes the data\_to\_transmit from the link does not quantize/stretch the data-prefix symbols when PH\_REQ\_HOLD is indicated by the link (all symbols transmitted in a packet from the speed-signal sequence to the terminating ending symbols must be stretched or padded according to the packet speed). I suggest the following. Change pg. 348 lines ~16-21 from:

```
do { // Wait for data or release from the link
    wait_event(PH_BYTE_CLOCK); // intended to synchronize with clock
    PH_CLOCK_indication();
    data_to_transmit = waitPH_DATA_request();
} while (data_to_transmit.reqType == PH_REQ_HOLD);
// Hold only valid before data starts
if (data_to_transmit.reqType == PH_REQ_DATA) {
```

to:

```
// Wait for data or release from the link
waitPH_DATA_request(data_to_transmit);
if (data_to_transmit.DreqType == PH_REQ_HOLD) { // Hold only valid before data starts
    PH_CLOCK_indication();
    wait_next_symbol(tx_speed);
} else if (data_to_transmit.DreqType == PH_REQ_DATA) {
```

NOTE: An alternative is to change the portTarb function so that the packet speed is maintained (once determined) instead of changed to DEFAULT so that the tx\_character function does the character stretching/padding automatically.

**Bug fix description**

1) [JH] Agreed in principle ... see next comment.

2) [JH] Agreed in principle. I like the pairing of clearing requests with indications to the link, however. So ultimately, BREQ and immediate\_request are no longer cleared on page 347 ~38. Instead, when the legacy link is granted, both breq and immediate\_request are cleared. The clearing of immediate\_request at pg 348 line 15 stays as is as a consequence.

3) [JH] Needs further discussion, but preliminary thought is to disagree. Although the packet specifications may need clarification to allow this, there should be no reason to require a rigid number of symbols between the minimum amount of speed signaling and DP and the start of data. Should be okay to have this optional "HOLD" period be purely elastic. It does make a worse min/max PHY\_DELAY jitter, but can improve latency in some cases. Certainly the PHY/Link interface is not required to insert a symbol's worth of hold cycles. For a legacy interface, holds come in 20 ns granularity. So pli block would have to buffer internally until the next clock\_indication arrived. Not conclusive, but suggestive. PHYDogs agreed to reject C code fix, but to change wait timings to units of S400. Mike has to specify more carefully the packet transmit formats.

---

**ID** 76      **Balloter**    Jim Skidmore      **How submitted** Ballot comment      **Fixed in version** D106  
**Title**      receive\_speed\_signal      **Status**    Recommend Accept in principl  
**C code files** receive\_actions.c      **Owner**      CWS      **SCAT**  **BRAT** <sup>^</sup> 504 **Recirc Clause** 17.5

**Problem description**

(Comment #45)  
Table 16-26, pg. 349-351, function receive\_actions  
The receive\_speed\_signal needs to be cleared at end of processing.  
At pg. 350 line 52, just before the "return;" statement and at pg. 351 line 17, just before the "}" (2 places), insert:

```
received_speed_signal = FALSE;
```

**Bug fix description**

In fact the fix is better made in process\_requests. Inspection of that routine unearthed that the flag was getting cleared too quickly anyway, the flag must not be cleared until processing has passed the end of the packet.

---

**ID** 81      **Balloter**    CWS      **How submitted** Post ballot      **Fixed in version** D1005  
**Title**      idle\_bus() is incomplete ... missing ALL:R0x terms      **Status**    Recommend Accept  
**C code files** idle.c      **Owner**      **SCAT**  **BRAT**      **Recirc Clause** 17.5

**Problem description**

idle\_bus() must return false if any of the exit conditions from state A0 are true. The current idle\_bus() code includes all explicit A0:xx transitions, but is missing the three implicit ones into R0.

**Bug fix description**

Suggestion is to add:  
return (!((power\_reset) or  
          (reset\_detected()) or  
          (ibr) or  
          ((proxy\_root ...

---

**ID** 109      **Balloter**    Biva      **How submitted** Post ballot      **Fixed in version** D1005  
**Title**      extra wait in receive\_actions      **Status**    Recommend Accept  
**C code files** receive\_actions.c      **Owner**      **SCAT**  **BRAT**      **Recirc Clause** 17.5

**Problem description**

In Draft 1.01 P.349, L.52 the statement "wait\_next\_symbol(tx\_speed);" should be removed, since it causes 2 PBT wait in the do-while loop per byte. Already one PBT duration is taken up by the tx\_byte() function in Line 44.

**Bug fix description**

---

**ID** 116      **Balloter**    CWS      **How submitted** Post ballot      **Fixed in version** D107  
**Title**      receive\_actions() fails to conclude null beta packets properly      **Status**    Done  
**C code files** receive\_actions.c      **Owner**      JH      **SCAT**  **BRAT**      **Recirc Clause** 17.5

**Problem description**

if receive\_actions() repeats a null packet, it always waits DATA\_END\_TIME. This is not correct if the null packet was of BETA format. (Such as a DATA\_NULL concluded with GRANT.)

Also, start\_rx\_packet() appears to be borken in that it won't return if a DATA\_NULL/GRANT sequence is received.

**Bug fix description**

The first noted problem of DATA\_END on null packets was fixed as a consequence of SCAT #59. The second noted problem was addressed by SCAT #62 which reworked start\_rx\_packet and used any arb state other than SPEED, DATA\_PREFIX, DATA\_NULL, or CYCL\_START\_TOKEN\_E/O to end the prefix and return control.

---

*ID* 118    *Balloter*    CWS                      *How submitted* Post ballot                      *Fixed in version* D107  
*Title*            Speed filtering of Link?    *Status* Done  
*C code files* receive\_actions.c                                      *Owner*            DW            *SCAT*  *BRAT*            *Recirc Clause* 17.5

***Problem description***

When attached to a legacy link, a PHY needs to speed filter beta packets from being delivered across the interface. Particularly in receive\_actions, data bytes should not be delivered and, when the end of a packet is reached, the link interface should stay "locked up". As such, fly\_by\_permitted can not be called in such a situation. Said differently, I think fly\_by\_permitted should only be called if the current packet format is legacy.

Similarly, when a beta parallel link is attached, packets >S800 must not be pushed across the interface.

***Bug fix description***

The C code description will filter format ... that is, we won't let beta format go to a legacy link, won't allow fly-by, etc. Fix for this checked in.

Speed filtering, however, is a property of the phy/link interface. So we need to have the parallel PHY/Link interface chapter specify that for packets greater than S800, DATA ON will be driven.

*ID* 120    *Balloter*    CWS                      *How submitted* Post ballot                      *Fixed in version* D1005  
*Title*            Current/Next format and speed for S100 Downshift test not maintained prop    *Status* Recommend Accept  
*C code files* various    *Owner*    *SCAT*  *BRAT*            *Recirc Clause* 17.5

***Problem description***

fly\_by\_permitted(), test\_requests(), and transmit\_actions() all implement the "no downshift to S100" rule. As such, they need access to the last packet speed and format transmitted/received as well as the about to be transmitted packet format and speed.

These format and speed indications are not being maintained consistently for all transmits and all recives. Additionally, the speed of the next beta request was being tested in test\_requests before it was actually set by transmit\_actions.

***Bug fix description***

Basic approach was to collapse tx\_speed and rx\_speed into cur\_speed. Likewise, tx\_format and format into cur\_format. These current packet speed and types are updated anytime we transmit or repeat a packet with the possible exception of repeating the restore packet to the nephew's link. Additionally, cur\_speed and cur\_format are reset whenever we visit A0:IDLE so that the downshift tests don't accidentally trigger on the first visit to TX.

Also, when performing the test in test\_requests or transmit\_actions, the next format and speed are calculated and stored in local variables next\_speed and next\_format so that the test can be easily done.

*ID* 126    *Balloter*    CWS                      *How submitted* Post ballot                      *Fixed in version* D105  
*Title*            Don't want B-only PHY's to need arb\_timers for collisions or isbr    *Status* Done  
*C code files* various    *Owner*            JH            *SCAT*  *BRAT*            *Recirc Clause* 17.5

***Problem description***

The current code uses arb\_OK within ds\_arb\_functions to arbitrate for resolving collisions and scheduling an isbr. However, arb\_ok implies an arbitration timer (sensitive to gaps). For B-only PHY's we didn't want to require such a timer.

Since resolve\_collision\_request doesn't require any arbitration time, it isn't too difficult. But for isbr, we have to wait until we know we're in the async phase. Wouldn't it be better to mark the isbr as a "current\_request" and let it be handled like other beta/BOSS requests? Any reason it must be done as a Legacy style request to the parent??

***Bug fix description***

All agreed, C code modified accordingly

**ID** 228    **Balloter** Jim Skidmore    **How submitted** Post ballot    **Fixed in version** D106  
**Title** grant\_received\_from\_senior in calls to boss\_end\_packet\_actions()    **Status** Done  
**C code files**    **Owner** JH    **SCAT**  **BRAT** 0    **Recirc Clause** 17.5

**Problem description**

boss\_end\_packet\_actions is always called with the grant\_received\_from\_senior parameter always set to FALSE. Is this an oversight?

**Bug fix description**

```
case GRANT:
case DATA_END:
...
else {
  if (rx_arb_state == GRANT) {
    boss_end_packet_actions(receive_port == senior_port);
  } else if ((iso_cycle || (byte_count == 1)) && (receive_port != senior_port)) {
    boss_end_packet_actions(FALSE);
  } else {
    stop_tx_packet(DATA_END, cur_speed, DATA_END_TIME, cur_format, NPORT);
  }
}
break;
```

Fix above seems incomplete. Fix was instead taken from e-mail entitled "Question about boss\_end\_packet\_actions, gant\_received\_from\_senior parameter".

---

**ID** 238    **Balloter** CWS    **How submitted** Post ballot    **Fixed in version** D105  
**Title** Problem with sending standby control symbol    **Status** Done  
**C code files** transmit\_actions.c    **Owner** PhyDogs    **SCAT**  **BRAT** 0    **Recirc Clause** 17.5

**Problem description**

The C code causes the link to be granted! Don't think that was intended. Also the C code for sending the control symbols in start\_tx\_packet() is broken - the symbols will be overwritten by a Speed symbol or a DATA\_PREFIX.

**Bug fix description**

Its sent by the nephew node, so there are no other ports to receive DP. Issue of granting the link fixed as a side-effect of fixing SCAT #122. Issue of control symbols being overwritten fixed.

---

**ID** 274    **Balloter** CWS    **How submitted** Post ballot    **Fixed in version**  
**Title** Should DATA\_NULL at end of receive packet set concatenated\_packet?    **Status** Done  
**C code files**    **Owner** PhyDogs    **SCAT**  **BRAT** 0    **Recirc Clause** 17.5

**Problem description**

In receive actions, a packet which ends with DATA\_PREFIX sets concatenated\_packet which is the only test for the RX:RX transition. However, a packet which ends with DATA\_NULL doesn't set concatenated\_packet. Seems like this means the RX:A0 transition could happen, causing the repeated DATA\_NULL to be replaced with IDLE. Was intent to take RX:RX?

**Bug fix description**

Fix to SCAT #241 caught this as well. receive\_actions now treats DATA\_PREFIX and DATA\_NULL identically at the end of a packet.

---

*ID* 276     *Balloter*    CWS                      *How submitted* Post ballot                      *Fixed in version*  
*Title*        Clean up TX:Transmit                      *Status*    Not yet done  
*C code files*                                      *Owner*        JH        *SCAT*  *BRAT*    0    *Recirc Clause* 17.5

*Problem description*

Code should now be able to track what type of grant was used to get into TX. Consequently, this may allow some cleanup of checking which requests are valid.

Also, maintenance of grant\_to\_give might be consolidated in stop\_tx\_packet and at the top of IDLE, etc.

*Bug fix description*

---

*ID* 277     *Balloter*    CWS                      *How submitted* Post ballot                      *Fixed in version*  
*Title*        Deadlock on asynchronous fairness?                      *Status*    Done  
*C code files*                                      *Owner*        PhyDogs    *SCAT*  *BRAT*    0    *Recirc Clause* 17.5

*Problem description*

Is it possible to get into a fairness interval deadlock? A node misses a phase change token and therefore is forwarding the wrong phase information. How does this timeout and async get reintroduced?

*Bug fix description*

Looks like a duplicate bug. This was dealt with in SCAT #218 by having a timeout in idle actions at senior\_borders and proxy\_root to re-introduce arb reset tokens whenever the bus failed to move along.

---

*ID* 279     *Balloter*    CWS                      *How submitted* Post ballot                      *Fixed in version* D107  
*Title*        Okay if tokens take the first symbol position in the 2 symbol packet prefix?    *Status*    Done  
*C code files*                                      *Owner*        CWS        *SCAT*  *BRAT*    0    *Recirc Clause* 17.5

*Problem description*

AS currently written, I believe that the cycle start token, async start token, and maybe the arb\_reset token repeated in start\_rx\_packet could become one of the starting symbols on an S100 legacy packet. So may see IDLE | CST| DP | Packet. Don't think this is "broken", but it probably isn't allowed by the packet format description. So either need to fix the description, or fix the C code to disallow.

*Bug fix description*

Modify text if necessary to ensure that this is not disallowed (at a repeater or receiver).

No action necessary, decided that these tokens are not part of the packet prefix.

This is not correct. The updated text says:

"At a repeating node, the value of p shall be such that: the time, before quantization effects are taken into account, for the DATA\_PREFIX symbols shall be at least MIN\_DATA\_PREFIX."

But as this PRN is trying to point out, start\_rx\_packet doesn't guarantee this. start\_rx\_packet only guarantees that the token plus the DP be at least MIN\_DATA\_PREFIX.

C code fixed.

---

**ID** 288    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D1071  
**Title**            Optimisation of ACK packets                    **Status** Done  
**C code files**    **Owner**            JH            **SCAT**  **BRAT**    0    **Recirc Clause** 17.5

**Problem description**

Possible future optimization ... don't insert deletables for ack packet. Problem ... originator knows, but repeater doesn't. So if repeater tries to insert cans and musts ... could fall behind. See email discussion, including discussion on must/delete algorithm.

**Bug fix description**

After discussion, no agreed method which solves all issues. Code improvements in the must/delete algorithm:-

Now require this routine to exit the instant we cross the must\_delete point. (As written, if we started out just shy of that limit, we'd end up waiting a whole symbol time even if we immediately crossed the limit.) To encourage low latency, wait using the fastest time base available to the arb state machine so that the delay from crossing the threshold and recognizing it is minimized (waiting in granularity of rx\_speed symbols is too coarse).

More on deletable symbols . . . there is a bigger problem of intent. Currently, the can/must algorithm is implemented in the wait\_fifo\_fill\_time routine.

Consider an originator and a repeater which are 100% in frequency lock to make things simple. At S100 beta format, the originator would send four 80 ns symbols prior to first data. The first two, speed and DP, are the min required by the beta format. (The non-deletable stuff.) The last two are the can and must delete symbols inserted by an originator. So from the start of the speed symbol, we get 320 ns to the start of first data.

Now consider the repeater. When it first sees the speed symbol, it will call start\_rx\_packet which, as a minimum, will send out the speed symbol and the first DP. It will then wait for first data (which comes two symbols times later). So start\_rx\_packet consumes 320 ns of time before wait\_fifo\_fill\_time() is called. Inside of wait\_fifo\_fill\_time() the can symbols are inserted, causing up to an additional 80 ns of delay. So the well intended 320 ns prefix at the originator was forced to become 400ns at the first repeater, 480 at the 2nd, etc., etc. This 80ns growth is pretty bad for bounding PHY\_DELAY, etc.

The other issue with the current code is that if wait\_fifo\_fill\_time() is called just as fifo\_backlog is just shy of watermark, then control falls immediately into the final do/while loop. And in a single tick of the clock, fifo\_backlog can exceed the watermark and we begin to repeat. No can symbol was generated as intended. (Only times cans aren't generated is when we are in the must delete region).

Our real intent at a repeater is to meet the following requirements:

After sending non-deletable stuff, data repeat begins immediately if we ever enter the "must-delete" region.

Provided we never cross into the must-delete region, send 20ns or one packet symbol (whichever is larger) of deletable stuff; i.e., the "can" symbol before data.

From the first data indication at the receiver, repeating shall begin no sooner than 20 ns to prevent underflow. Please note that this 20ns is not dependent on packet speed.

An optimized implementation would allow requirements 2 and 3 to overlap in time. In other words, if data first arrived while we were transmitting the can symbol and we spent at least 20 ns repeating the can symbol, then #3 has already been met and doesn't need additional enforcement. And as another optimization, we shouldn't bother sending deletable symbols if we aren't going to send data.

The general approach is to send "most" of the can symbol just after the non-deletable stuff is sent. By "most", I mean send DP for a duration of time 20ns less than the necessary can symbol duration. Then, start looking for data (which may or may not have just arrived). When data is observed, wait an additional 20 ns for FIFO centering, just like in P1394a (except it seemed to wait for 40 ns to make up for the dribble bit pipeline I guess). So provided we haven't crossed into must-delete at any point, we've waited at least a can symbol's worth of time while overlapping the 20ns centering. Of course if we cross into must at any point while sending the can or the centering time, we exit fast.

---

**ID** 295    **Balloter**    CWS                    **How submitted** Post ballot                    **Fixed in version** D107  
**Title**            Nephew doesn't properly received restore packet                    **Status** Done  
**C code files**    idle.c    **Owner**            CWS            **SCAT**  **BRAT**    0    **Recirc Clause** 17.5

**Problem description**

Nephew can't get into transmit\_actions so can't call con\_mgmt\_granted(). Consequently, won't get to bit of code which calls rx\_PHY\_packet and received the restore packet.

**Bug fix description**

new flag - immediate\_restore\_request

*ID* 299   *Balloter*   Biva   *How submitted* Post ballot   *Fixed in version* D1071  
*Title*   Clearing of send\_async\_start\_token in idle\_actions()   *Status* Done  
*C code files*   *Owner* MT   *SCAT*  *BRAT* 0   *Recirc Clause* 17.5

***Problem description***

This one is related to the following statements at the start of idle\_actions:

```
-----  
// an inappropriate gap count setting may result in a gap token having been received  
// during the last packet. The best way to deal with this is to pretend the node didn't see the gap.  
send_async_start_token = arb_reset = FALSE;  
-----
```

We are facing a scenario in which the async\_start sent by the cycle master immediately at the end of the cycle start packet for a null iso cycle in a betaOnly bus is not being repeated (or acted upon - phase change) by the peer phys. The cycle master sends DN following the cycle start packet followed by ASYNC\_START token. The peer does packet end actions (concurrently sees ASYNC\_START received and sets the flag send\_async\_start\_token). Thereafter, it enters idle\_actions() where the flag gets cleared and it does not perform the gap\_repeat\_actions().

I feel this is a valid situation which might happen and probably we need to put these clearing statements under some condition allowing the flags to remain set if asserted near the end of the previous packet.

***Bug fix description***

Qualify the test to only delete tokens received on ports other than the receive port

JH:-

I think the current intent is for start\_rx\_packet() to repeat gap tokens received on the receive port at the end of "packets". In the case Biva lists specifically, I think the intention was that the DN following the cycle start packet causes a receiver to take the RX:RX transition and call start\_rx\_packet() again. And when the gap token is received, it gets repeated from start\_rx\_packet. So for the scenario outlined, I don't think the C code would ever end up in idle\_actions with send\_async\_start\_token or arb\_reset set following a legitimate packet receive sequence. Said differently, if the token\_receive\_port == receive\_port, I think start\_rx\_packet() would have repeated any needed token already. As a consequence, I conclude that the C code fix is a harmless no-op relative to 1.06.

But this did cause me to think about the original reason behind having this line of code in idle\_actions(). Only thing I could come up with is when the senior border in a cloud got a gap count setting which was too low and, as a result, launched a gap token while a far away node was just starting to send an ack packet. In this situation, the send\_async\_start\_token flag could be set at some intermediate node while that node was in RX for the ack packet. And upon returning to IDLE, we didn't want to send the now stale gap indication. (I'm not sure how fatal it would have been ... but that is what we decided.) The collision, as Colin correctly observes, occurs when the token\_receive\_port != receive\_port.

But the fix in 1.06 as well as the fix below don't really protect us completely from this collision. For example, the intermediate node could be in start\_rx\_packet() about to repeat an ack packet from a non-senior port when all of a sudden a gap token shows up on the senior port. As start\_rx\_packet() is currently written, the intermediate node will attempt to repeat the token even though it didn't arrive from the receive port.

So I wonder if we need to add code to ignore gap tokens from other than the receive port when in start\_rx\_packet(), or whether we just admit that we can never completely squelch a gap token issued because of an incorrect gap count and therefore strip out the attempt at the top of idle\_actions().

I don't really have a preference ... I just feel that we haven't really peeled this onion sufficiently.

What to do?  
Jerry

P.S. It just occurred to me that if we want to do our best to suppress gap tokens, then we might be able to take advantage of the following assertions:

- 1) In A0:Idle, gap tokens only arrive from the senior port.
- 2) In A1:Legacy Request, gap tokens only arrive from the senior port.
- 3) In RX: Receive, gap tokens only arrive from the receive port. Furthermore, in a hybrid bus the receive port will be the senior port.

Fix is to take out all of the special processing in process\_requests for ASYNC\_START and ARB\_RESET\_\*. Treat them just like any other arbitration symbol. Push them into portRarb when they get to the top of the FIFO, and advance them (or not) automatically just like all other symbols. That is, if we aren't in the middle of receiving a packet on this port, the tokens get advanced automatically. If we are in receive on this port, they can only be advanced by receive\_actions().

Then, in A0, A1, and RX, we deal with them if we see them just like all other arb symbols. Example:

In A0:Idle we're looking to the senior port for a GRANT anyway ... if we see a gap token we can process it then. So where idle\_actions is testing for send\_async\_start, etc ... we just do a portRarb[senior\_port] == ASYNC\_START, etc. In Idle, the senior port isn't in receive, so the tokens will be automatically advancing for us. As long as we see it sometime during the 80 ns, no problem.

In A1:Legacy Request, we're looking to the senior port for a GRANT as well. Where the A1:A1 transition tests for send\_async\_start\_token, we could test on portRarb[senior\_port] == ASYNC\_START, etc. And again, the senior\_port isn't in a packet receive mode, and will be auto advancing then. So we don't have to do anything special to dequeue the symbol.

Finally, in RX, calls to portR\_next\_arb might reveal the token and pop it for us automatically.

I think this approach means that token\_receive\_port, etc. can be deleted. No squelching is required because the tokens are ignored when not expected.

Need, in A0 and A1, to only send the ASYNC\_START only once. (Might still be at the top of portRarb after we've repeated the token.) Another flag is used ... but then we have to worry about clearing it at the proper points. Cleared at the top of A0. Then once we saw and repeated a token in gap\_repeat\_actions, we'd set the flag. So if we sent the token in A0, we won't accidentally send it again immediately in A0 or right as we make it to A1 and it hasn't disappeared from the parent port yet, for example.

Why we didn't do this originally? I think it had to do with the idea, a long time ago, that any and all borders could send us the token while we were in the midst of repeating packets, etc. So we needed background processing to see it, remember, and remind us to send one later. But now with the senior border approach, legitimate tokens are only expected at times when we can handle them immediately.

Details:-

- boolean function gap\_token (int port) invented to check a specific port - implements filtering by returning FALSE and does not set the "send" flags
- use of "send" flags to cause sending of tokens retained
- filtering to ensure sending only once in A0 used for arb reset as well
- need to clear the 'sent' flag in RX as well

A1:A1 transition now has the qualification  
gap\_token(senior\_port);

---

<i>ID</i>	302	<i>Balloter</i>	CWS	<i>How submitted</i>	Post ballot	<i>Fixed in version</i>	D1071
<i>Title</i>	Gap Tokens should not be used to send indications to a Legacy Link				<i>Status</i>	Done	
<i>C code files</i>		<i>Owner</i>	CWS	<i>SCAT</i>	<input type="checkbox"/>	<i>BRAT</i>	0
<i>Recirc Clause</i>	17.5						

**Problem description**

1.06 proposes that border nodes which are `_not_` senior borders should ignore their internal gap timers and only use the tokens received on the bus to decide when gaps had happened, etc.

This may be ill-advised, to a degree, if a Legacy link is attached. A border with a legacy link probably needs to detect gaps using the old timing methods and deliver them to the link at that instant. If it waits until an AST comes, it may be too late since the AST can be embedded in a bunch of DATA\_NULLs which have tied up the PHY/Link interface.

So with a Legacy link, timers are what causes the indications to the link, and the tokens are never delivered to the legacy link.

Question on the table is whether the PHY should use the timers or the tokens (or both?) to end the iso interval or set the arb\_enable flag.

**Bug fix description**

Idle actions with a Legacy\_Link notifies the link immediately if one of the gap timers expires. If senior border, then also sets the appropriate flag to invoke gap\_repeat\_actions.

Gap\_repeat\_actions issues (repeats) tokens, tells beta links and advances the phase/ends iso cycle etc.

Care with the case of senior\_border with a Beta\_link

---

**ID** 332     *Balloter*     CWS     *How submitted* Post ballot     *Fixed in version*  
*Title*     grant\_only\_at\_async\_start not being cleared     *Status* Done  
*C code files*     *Owner* JH     *SCAT*  *BRAT* 0     *Recirc Clause* 17.5

*Problem description*

In a hybrid bus, idle\_actions sets grant\_only\_at\_async\_start and ok\_to\_grant at subaction\_gap + arb\_delay to give the proxy\_root one and only one chance to make a request before the second quiet window begins. (The top half of idle\_actions uses grant\_only\_at\_async\_start to clear OK\_to\_grant.)

Problem is that grant\_only\_at\_async\_start is not subsequently cleared at the end of the 2nd quiet window, so OK\_to\_grant gets continuously cleared.

*Bug fix description*

Ideally, I wanted to clear grant\_only\_at\_async\_start in the top half of idle actions just after it tested true. That way, the top half would set it FALSE and the bottom half only TRUE. But grant\_only\_at\_async\_start was also being used in the bottom half to make sure that the OK\_to\_grant was only being set once the arb\_timer hit the == point. So it was important not to clear grant\_only\_at\_async\_start until some time had elapsed. Consequently, it is cleared in the lower half when the second quiet window expires (arb\_reset\_gap + arb\_delay).

**ID** 346     *Balloter*     CWS     *How submitted* Post ballot     *Fixed in version* D1071  
*Title*     Gap\_repeat\_actions takes time and cannot be called on a state transition     *Status* Done  
*C code files*     *Owner* MT     *SCAT*  *BRAT* 0     *Recirc Clause* 17.5

*Problem description*

Needs to be moved inside A1 with suitable qualification

*Bug fix description*

**ID** 353     *Balloter*     CWS     *How submitted* Post ballot     *Fixed in version* D1071  
*Title*     senior\_border doesn't recover from ack\_missing after beta packet     *Status* Done  
*C code files* idle.c     *Owner* CWS     *SCAT*  *BRAT* 0     *Recirc Clause* 17.5

*Problem description*

Scenario: in a junior beta cloud, a node originates a beta format directed asynchronous packet. Assertion is that senior border is driving DP into parent legacy cloud.

Originated packet concludes with DE (not the end of a subaction) and the senior\_border is implicitly granted. But the senior port of the senior border stays in DP.

How is the ack missing detected, and how is the senior DS cloud released? Can't see how this would work with current C code. Seems like the senior border which times out the ack\_missing needs to know to send a null packet and free up the senior cloud.

*Bug fix description*

extra test put into A0 - send a subaction gap and schedule a null packet

**ID** 359    **Balloter**    CWS    **How submitted** Post ballot    **Fixed in version** 1.1  
**Title**    Timing guards in idle\_actions.c preventing legacy link notification    **Status** Done  
**C code files**    **Owner**    JH    **SCAT**  **BRAT**    0    **Recirc Clause** 17.5

**Problem description**

The 2nd fork body in idle\_actions uses !send\_async\_start\_token and !arb\_reset to guard executing (arb\_timer == blah) sections multiple times before the arb\_timer advances. Problem is that send\_async\_start\_token and arb\_reset can be set by other processes and, under some circumstances, cause the code to be blocked before it is executed even once.

For example, a node which is not senior border but with a legacy link may receive async\_start from it's senior port, causing send\_async\_start to be set. Shortly thereafter, the local arb\_timer == subaction\_gap. This would normally cause an indication to be sent to the Legacy link, but can't execute in this example because send\_async\_start was already set.

**Bug fix description**

Added two new timing guards: reached\_subaction and reached\_arb\_reset to act as edge detects. And added comments to all timing blocks in the 2nd half of idle\_actions to clarify which signals were also providing a guard function.

---

**ID** 361    **Balloter**    CWS    **How submitted** Post ballot    **Fixed in version** 1.1  
**Title**    Shouldn't call gap\_token(senior\_port) at the proxy\_root    **Status** Done  
**C code files**    **Owner**    JH    **SCAT**  **BRAT**    0    **Recirc Clause** 17.5

**Problem description**

Can't call gap\_token(senior\_port) at the proxy\_root since senior\_port will be out of range.

**Bug fix description**

In idle\_actions(), call to gap\_token(senior\_port) is qualified by !proxy\_root

---

**ID** 366    **Balloter**    CWS    **How submitted**    **Fixed in version**  
**Title**    request cancellation race condition    **Status** Not yet done  
**C code files**    **Owner**    JH    **SCAT**  **BRAT**    0    **Recirc Clause** 17.5

**Problem description**

There is a race condition in the current C code (carried over from 1394a) where the PHY issues a PH\_LOST at the start of transmission of a PHY packet, but the implementation is that the link only interprets the "lost" status on receiving the 9th bit. Even without this, there is a race condition of the link issuing a request simultaneously with the PHY canceling, with the result that the PHY accepts the request but the link thinks it is canceled. The link may make a further request (for different speed and format), but is subsequently granted for the original request (speed and format).

**Bug fix description**

rework C code to avoid the first mismatch of services with implementation, and change the meaning of PH\_LOST such that it is issued whenever queued asynchronous requests would be cleared ... even if no requests were yet queued. The C code tells the link interface when a cancellation period begins, whether or not an actual cancellation occurs.

---

**ID** 370    **Balloter**    CWS    **How submitted** Post ballot    **Fixed in version**  
**Title**    transmit\_actions calls tx\_byte in a for loop    **Status** Not yet done  
**C code files**    **Owner**    JH    **SCAT**  **BRAT**    0    **Recirc Clause** 17.5

**Problem description**

When transmitting data from the link, transmit\_actions calls tx\_byte inside of a NPORT loop. The loop index isn't used for anything, and the multiple calls to tx\_byte causes bytes to be repeated and the PH\_CLOCK.indication timing to be incorrect for any PHY with NPORT != 1.

**Bug fix description**

---