802.3dj D2.0 Comment Resolution Logic Track

Gary Nicholl (Cisco), Logic Track Lead Editor
Eugene Opsasnick (Broadcom), Logic Editor
Matt Brown (Alphawave Semi), 802.3dj Chief Editor
Xiang He (Huawei), Logic Editor
K. Shrikhande (Marvell), Logic Editor
<others>

Introduction

- This slide package was assembled by the 802.3dj editorial team to provide background and detailed resolutions to aid in comment resolution.
- Specifically, these slides are for the logic track comments

Inner FEC bin counters

Comment #68, [561, 282, 283], 570

Comment #68, [561, 282, 283], 570 – FEC bin counters (Background)

The following counters are currently defined for RS-FEC:

FEC_cw_counter
FEC_corrected_cw_counter
FEC_uncorrected_cw_counter
FEC_symbol_error_counter_<0:n>   [Note: per PCS lane]
FEC_codeword_error_bin_<1:15>

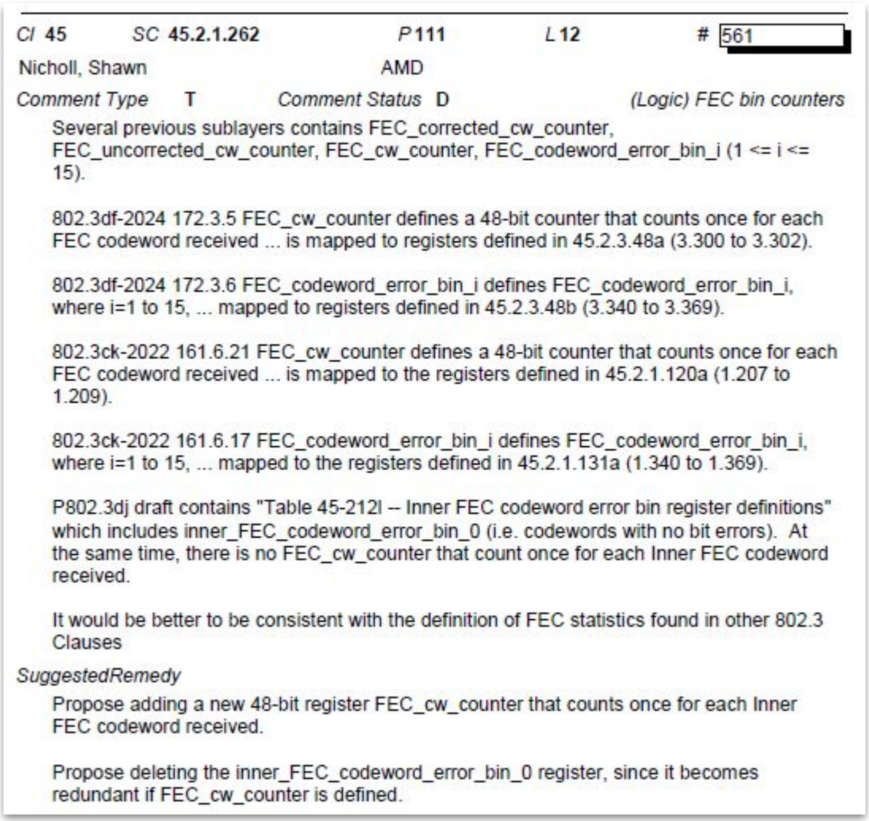
- Consistent set of counters for all RS-FEC (Clauses 119, 161, 172, 176) and using a common set of MDIO registers (45)
- FEC bin counters initially added in Clauses 161 (3ck) and 172 (3df) without a bin_0 counter.
- FEC bin counters being added in Clauses 119 and 175 as part of 3dj are following the same approach for consistency, and to reuse existing MDIO registers (i.e. no bin_0)
- If desired the bin_0 count can be derived from (total_cw - corrected_cw - uncorrected_cw).

The following counters are currently defined for Inner-FEC:

Inner_FEC_corrected_cw_counter
Inner_FEC_uncorrected_cw_counter
Inner_FEC_codeword_error_bin_<0:n>
Inner_FEC_total_bits_counter
Inner _FEC_corrected_bits_counter

- New set of counters being added for new InnerFEC (Clauses 177 and 184) being added in 3dj.
- No legacy to be concerned about. New set of MDIO registers for InnerFEC (separate from MDIO registers for RS-FEC)
- In this case a bin_0 counter was included as a convenience for the user
- Observation in comment: No “Inner_FEC_cw_counter” currently defined. Should there be?

Comment # [561, 282, 283] – FEC bin counters



Cl **45** *SC* **45.2.1.262** *P* **111** *L* **12** # 561

Nicholl, Shawn AMD

Comment Type **T** *Comment Status* **D** *(Logic) FEC bin counters*

Several previous sublayers contains FEC_corrected_cw_counter, FEC_uncorrected_cw_counter, FEC_cw_counter, FEC_codeword_error_bin_i (1 <= i <= 15).

802.3df-2024 172.3.5 FEC_cw_counter defines a 48-bit counter that counts once for each FEC codeword received ... is mapped to registers defined in 45.2.3.48a (3.300 to 3.302).

802.3df-2024 172.3.6 FEC_codeword_error_bin_i defines FEC_codeword_error_bin_i, where i=1 to 15, ... mapped to registers defined in 45.2.3.48b (3.340 to 3.369).

802.3ck-2022 161.6.21 FEC_cw_counter defines a 48-bit counter that counts once for each FEC codeword received ... is mapped to the registers defined in 45.2.1.120a (1.207 to 1.209).

802.3ck-2022 161.6.17 FEC_codeword_error_bin_i defines FEC_codeword_error_bin_i, where i=1 to 15, ... mapped to the registers defined in 45.2.1.131a (1.340 to 1.369).

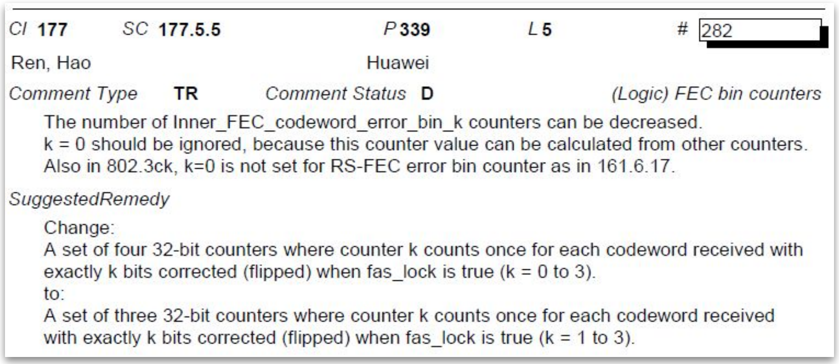
P802.3dj draft contains "Table 45-212l -- Inner FEC codeword error bin register definitions" which includes inner_FEC_codeword_error_bin_0 (i.e. codewords with no bit errors). At the same time, there is no FEC_cw_counter that count once for each Inner FEC codeword received.

It would be better to be consistent with the definition of FEC statistics found in other 802.3 Clauses

SuggestedRemedy

Propose adding a new 48-bit register FEC_cw_counter that counts once for each Inner FEC codeword received.

Propose deleting the inner_FEC_codeword_error_bin_0 register, since it becomes redundant if FEC_cw_counter is defined.



Cl **177** *SC* **177.5.5** *P* **339** *L* **5** # 282

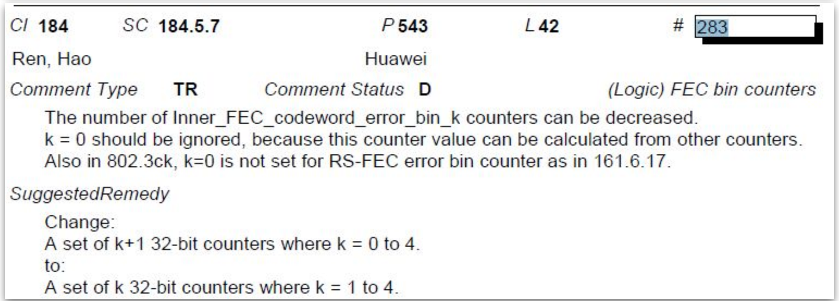
Ren, Hao Huawei

Comment Type **TR** *Comment Status* **D** *(Logic) FEC bin counters*

The number of Inner_FEC_codeword_error_bin_k counters can be decreased.
k = 0 should be ignored, because this counter value can be calculated from other counters.
Also in 802.3ck, k=0 is not set for RS-FEC error bin counter as in 161.6.17.

SuggestedRemedy

Change:
A set of four 32-bit counters where counter k counts once for each codeword received with exactly k bits corrected (flipped) when fas_lock is true (k = 0 to 3).
to:
A set of three 32-bit counters where counter k counts once for each codeword received with exactly k bits corrected (flipped) when fas_lock is true (k = 1 to 3).



Cl **184** *SC* **184.5.7** *P* **543** *L* **42** # 283

Ren, Hao Huawei

Comment Type **TR** *Comment Status* **D** *(Logic) FEC bin counters*

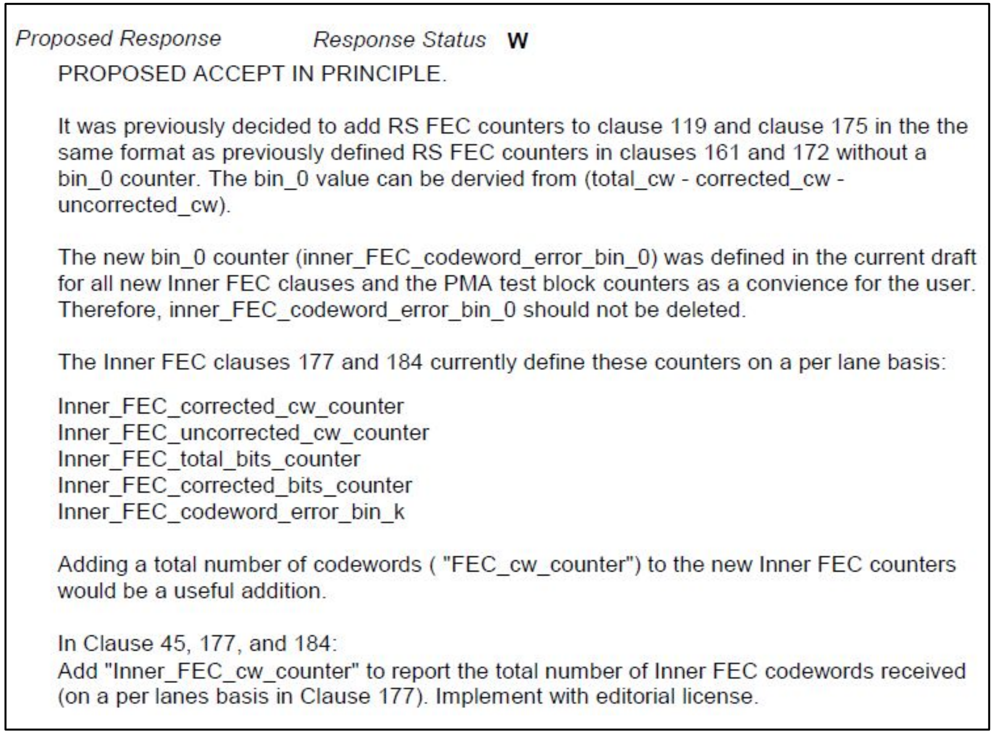
The number of Inner_FEC_codeword_error_bin_k counters can be decreased.
k = 0 should be ignored, because this counter value can be calculated from other counters.
Also in 802.3ck, k=0 is not set for RS-FEC error bin counter as in 161.6.17.

SuggestedRemedy

Change:
A set of k+1 32-bit counters where k = 0 to 4.
to:
A set of k 32-bit counters where k = 1 to 4.

- The three comments suggested to remove bin_0 because it is redundant (can be derived from existing counters)
  Inner_FEC_codeword_error_bin_0 = Inner_FEC_total_bits_counter/128 - Inner_FEC_corrected_cw_counter - Inner_FEC_uncorrected_cw_counter

Comment # [<u>561</u>, 282, 283] – Proposed response



Proposed Response *Response Status* **W**

PROPOSED ACCEPT IN PRINCIPLE.

It was previously decided to add RS FEC counters to clause 119 and clause 175 in the the same format as previously defined RS FEC counters in clauses 161 and 172 without a bin_0 counter. The bin_0 value can be dervied from (total_cw - corrected_cw - uncorrected_cw).

The new bin_0 counter (inner_FEC_codeword_error_bin_0) was defined in the current draft for all new Inner FEC clauses and the PMA test block counters as a convience for the user. Therefore, inner_FEC_codeword_error_bin_0 should not be deleted.

The Inner FEC clauses 177 and 184 currently define these counters on a per lane basis:

Inner_FEC_corrected_cw_counter
Inner_FEC_uncorrected_cw_counter
Inner_FEC_total_bits_counter
Inner_FEC_corrected_bits_counter
Inner_FEC_codeword_error_bin_k

Adding a total number of codewords ("FEC_cw_counter") to the new Inner FEC counters would be a useful addition.

In Clause 45, 177, and 184:
Add "Inner_FEC_cw_counter" to report the total number of Inner FEC codewords received (on a per lanes basis in Clause 177). Implement with editorial license.

Proposed response:

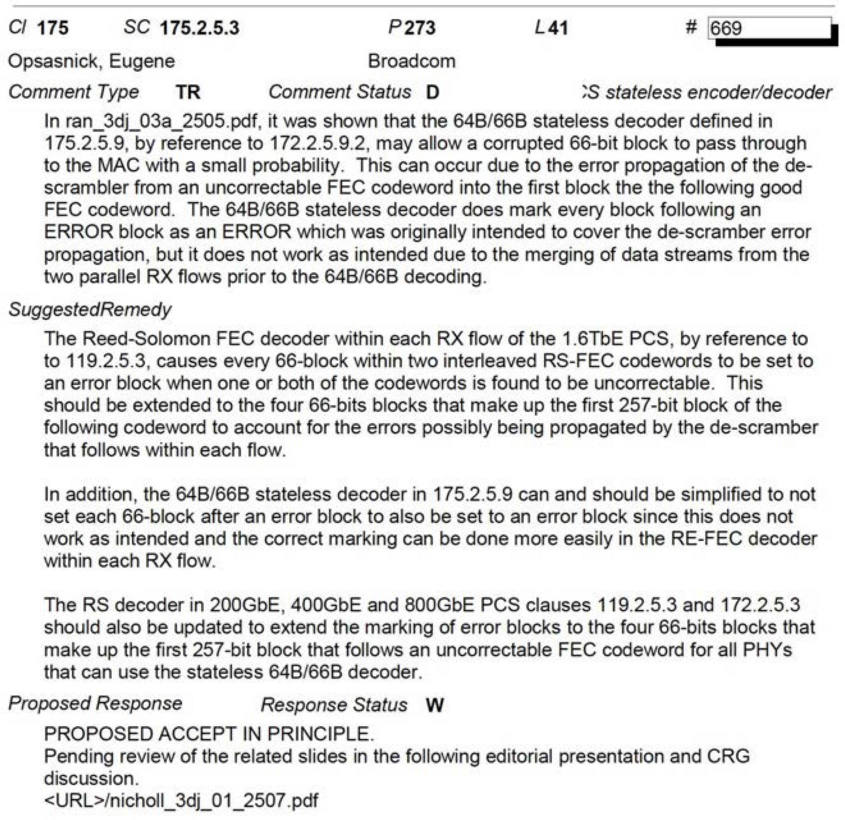
- Keep bin_0 counter for Inner FEC as currently defined in Clause 177 and 184 (no change to draft)
- Add a counter for total number of codewords
  - Inner_FEC_cw_counter

Pending CRG discussion and potential straw poll(s) if necessary.

PCS stateless encoder/decoder

Comments [669, 432, 433, 670, 331, 431, 584, 676, 339]

Comment #669, #432, #433 (Handling scrambler error extension)


Cl **175** *SC* **175.2.5.3** *P* **273** *L* **41** # 669

Opsasnick, Eugene Broadcom

Comment Type **TR** *Comment Status* **D** *S stateless encoder/decoder*

In ran_3dj_03a_2505.pdf, it was shown that the 64B/66B stateless decoder defined in 175.2.5.9, by reference to 172.2.5.9.2, may allow a corrupted 66-bit block to pass through to the MAC with a small probability. This can occur due to the error propagation of the de-scrambler from an uncorrectable FEC codeword into the first block the the following good FEC codeword. The 64B/66B stateless decoder does mark every block following an ERROR block as an ERROR which was originally intended to cover the de-scramber error propagation, but it does not work as intended due to the merging of data streams from the two parallel RX flows prior to the 64B/66B decoding.

SuggestedRemedy

The Reed-Solomon FEC decoder within each RX flow of the 1.6TbE PCS, by reference to to 119.2.5.3, causes every 66-block within two interleaved RS-FEC codewords to be set to an error block when one or both of the codewords is found to be uncorrectable. This should be extended to the four 66-bits blocks that make up the first 257-bit block of the following codeword to account for the errors possibly being propagated by the de-scrambler that follows within each flow.

In addition, the 64B/66B stateless decoder in 175.2.5.9 can and should be simplified to not set each 66-block after an error block to also be set to an error block since this does not work as intended and the correct marking can be done more easily in the RE-FEC decoder within each RX flow.

The RS decoder in 200GbE, 400GbE and 800GbE PCS clauses 119.2.5.3 and 172.2.5.3 should also be updated to extend the marking of error blocks to the four 66-bits blocks that make up the first 257-bit block that follows an uncorrectable FEC codeword for all PHYs that can use the stateless 64B/66B decoder.

Proposed Response *Response Status* **W**

PROPOSED ACCEPT IN PRINCIPLE.
Pending review of the related slides in the following editorial presentation and CRG discussion.
<URL>/nicholl_3dj_01_2507.pdf

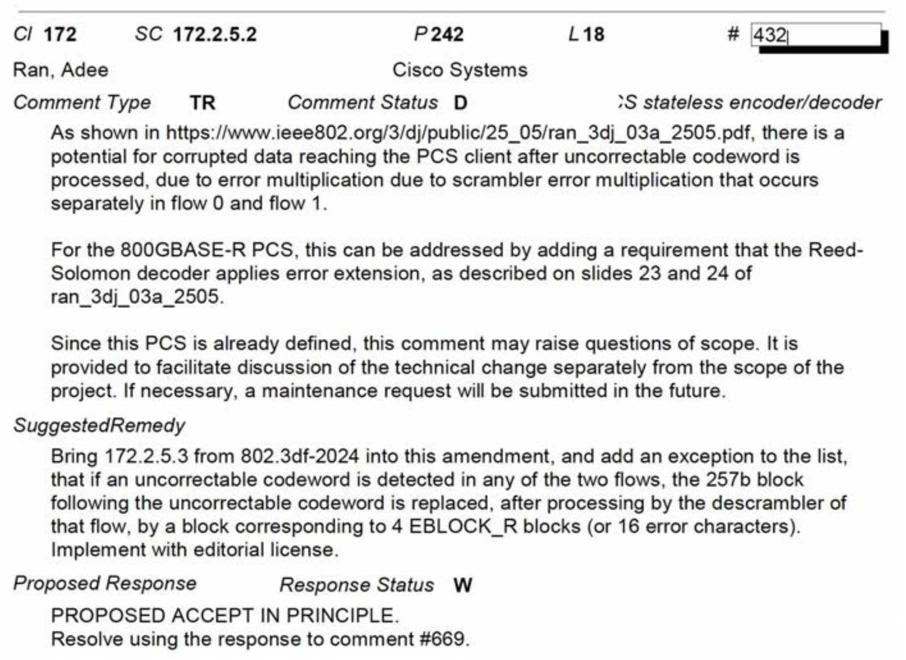
What's the issue?

-> The 64b/66b stateless decoder was defined to invalidate (substitute an EBLOCK_R) in the first block after an uncorrectable RS-FEC codeword to cover the case of the descrambler "error extension". But it has two issues as defined in 802.3df, clause 172.

1. The descrambler works on 257-bit blocks before inverse transcoding, so we really need to invalidate all four 64b blocks that make up the first 257b block.
2. It does not account for the interleave of blocks from the two flows of the 800G/1.6T PCS RX function.

#1 needs to be fixed in Clauses 119, 172 and 175.
#2 needs to be fixed in Clauses 172 and 175.

Comment #669, #432, #433 (Handling scrambler error extension 2)



Cl **172** *SC* **172.2.5.2** *P* **242** *L* **18** # 432

Ran, Adee Cisco Systems

Comment Type **TR** *Comment Status* **D** *S stateless encoder/decoder*

As shown in https://www.ieee802.org/3/dj/public/25_05/ran_3dj_03a_2505.pdf, there is a potential for corrupted data reaching the PCS client after uncorrectable codeword is processed, due to error multiplication due to scrambler error multiplication that occurs separately in flow 0 and flow 1.

For the 800GBASE-R PCS, this can be addressed by adding a requirement that the Reed-Solomon decoder applies error extension, as described on slides 23 and 24 of ran_3dj_03a_2505.

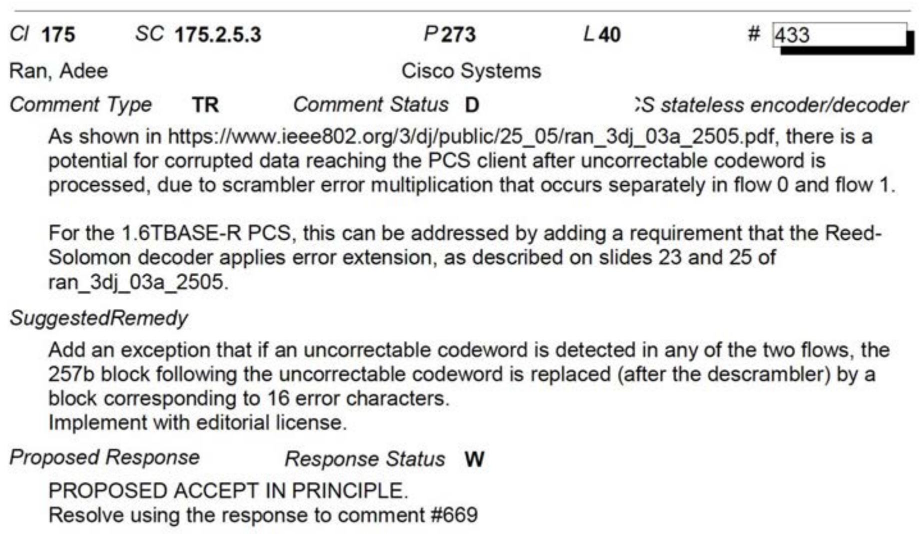
Since this PCS is already defined, this comment may raise questions of scope. It is provided to facilitate discussion of the technical change separately from the scope of the project. If necessary, a maintenance request will be submitted in the future.

SuggestedRemedy

Bring 172.2.5.3 from 802.3df-2024 into this amendment, and add an exception to the list, that if an uncorrectable codeword is detected in any of the two flows, the 257b block following the uncorrectable codeword is replaced, after processing by the descrambler of that flow, by a block corresponding to 4 EBLOCK_R blocks (or 16 error characters). Implement with editorial license.

Proposed Response *Response Status* **W**

PROPOSED ACCEPT IN PRINCIPLE.
Resolve using the response to comment #669.



Cl **175** *SC* **175.2.5.3** *P* **273** *L* **40** # 433

Ran, Adee Cisco Systems

Comment Type **TR** *Comment Status* **D** *S stateless encoder/decoder*

As shown in https://www.ieee802.org/3/dj/public/25_05/ran_3dj_03a_2505.pdf, there is a potential for corrupted data reaching the PCS client after uncorrectable codeword is processed, due to scrambler error multiplication that occurs separately in flow 0 and flow 1.

For the 1.6TBASE-R PCS, this can be addressed by adding a requirement that the Reed-Solomon decoder applies error extension, as described on slides 23 and 25 of ran_3dj_03a_2505.

SuggestedRemedy

Add an exception that if an uncorrectable codeword is detected in any of the two flows, the 257b block following the uncorrectable codeword is replaced (after the descrambler) by a block corresponding to 16 error characters.
Implement with editorial license.

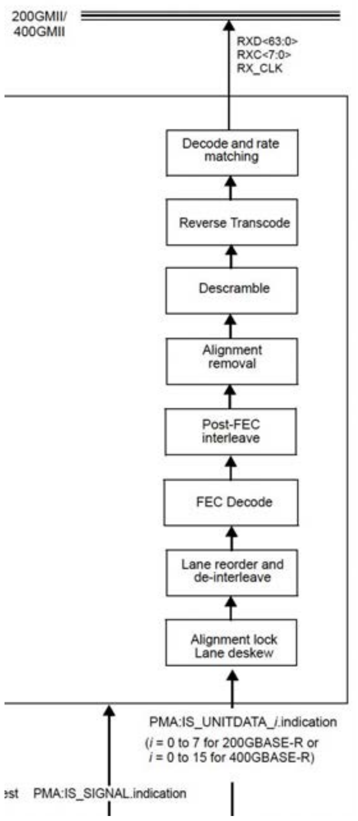
Proposed Response *Response Status* **W**

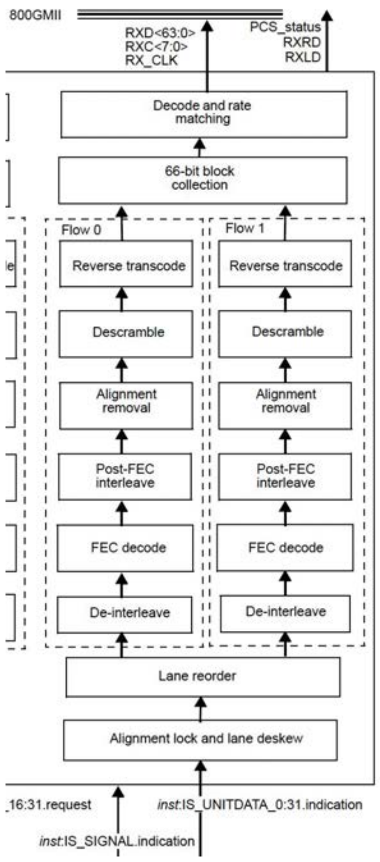
PROPOSED ACCEPT IN PRINCIPLE.
Resolve using the response to comment #669

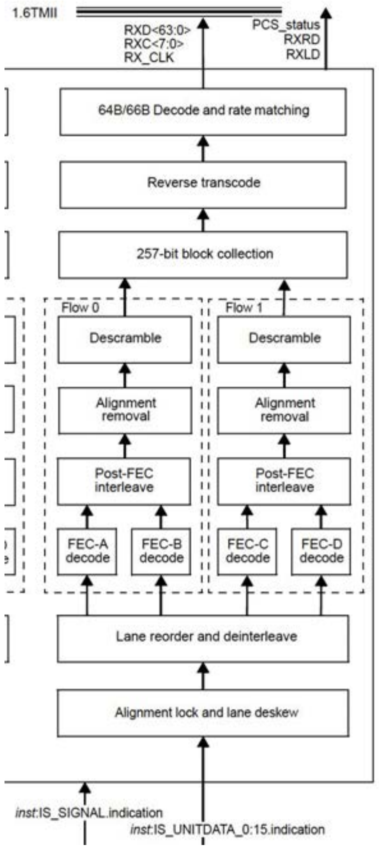
Comment #432 points out the same issues for Clause 172 (800GbE PCS).

Comment #433 points out the same issues for Clause 175 (1.6TbE PCS).

Comment #669, #432, #433 (Handling scrambler error extension - 3)





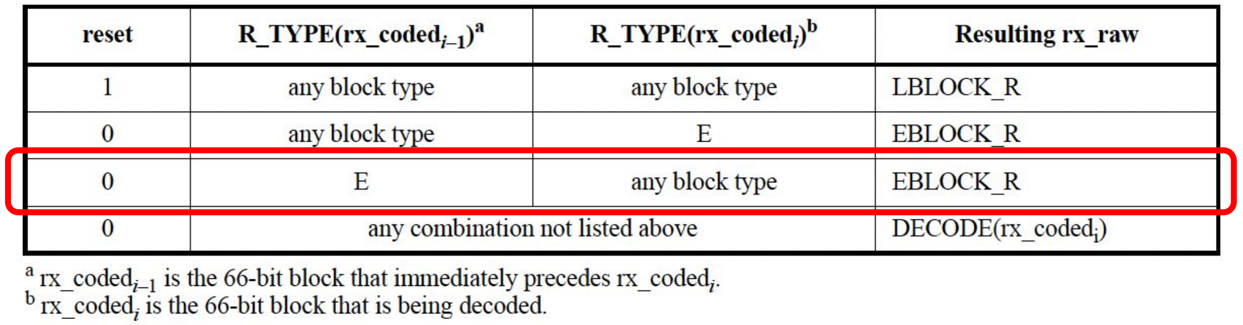


Note:

- 200G/400G have a single RX flow. 800G/1.6T PCS RX perform RS-FEC decode and Descramble separately within each of their two flows.
- Descramble is performed on the 257b blocks (before Reverse transcode).
- 64B/66B decode is performed after merging of the two flows into a single data flow (800G & 1.6T)
- 800G PCS RX flows merge on 66b block boundaries and 1.6T PCS RX merges 257b blocks.

Comment #669, #432, #433 (Handling scrambler error extension - 4)

Table 172–4—PCS stateless decoder rules

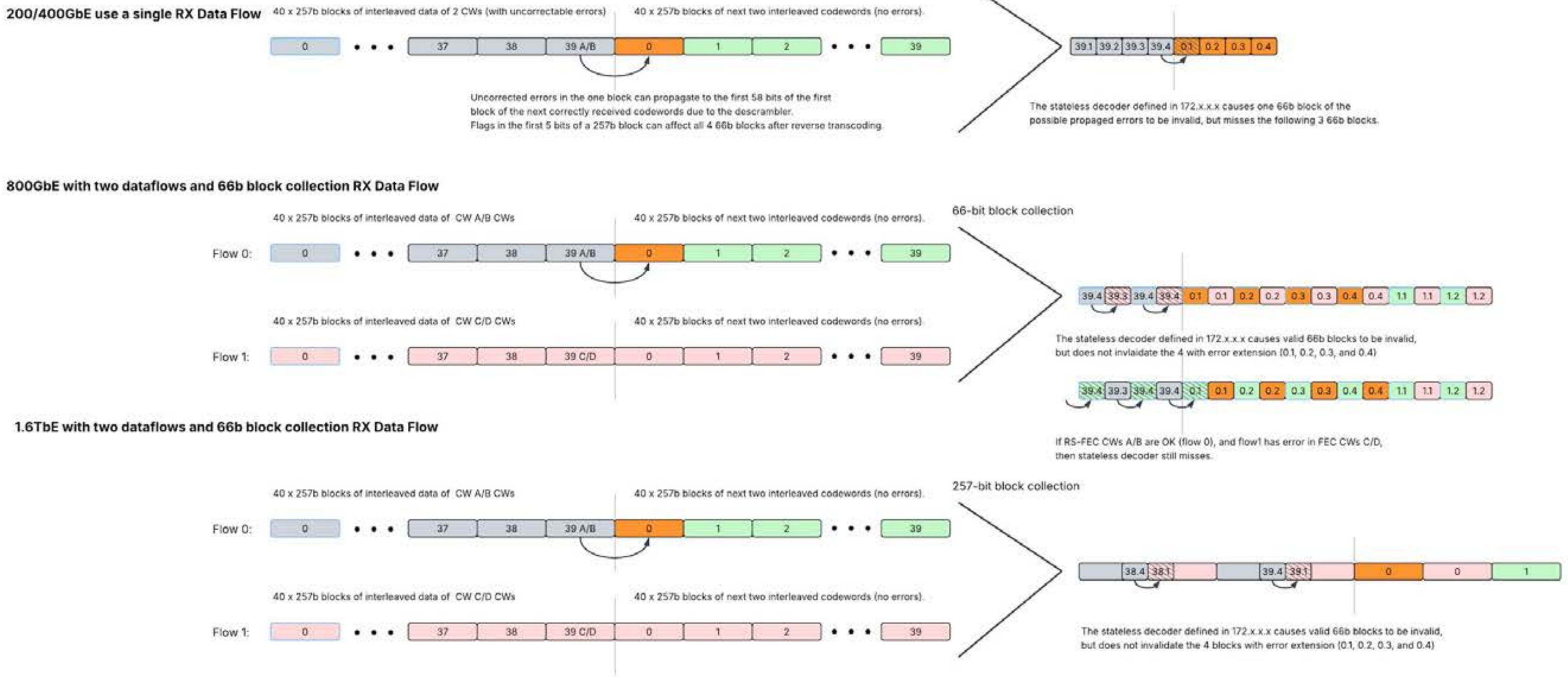


reset	R_TYPE(rx_coded$_{i-1}$)[a]	R_TYPE(rx_coded$_i$)[b]	Resulting rx_raw
1	any block type	any block type	LBLOCK_R
0	any block type	E	EBLOCK_R
0	E	any block type	EBLOCK_R
0	any combination not listed above		DECODE(rx_coded$_i$)

[a] rx_coded$_{i-1}$ is the 66-bit block that immediately precedes rx_coded$_i$.
[b] rx_coded$_i$ is the 66-bit block that is being decoded.

The rule in this row is intended to invalidate (set to EBLOCK_R) the block following any block marked as an error by the RS-FEC decoder due to “descrambler error extension”.

Comment #669, #432, #433 (Scrambler error extension example)


200/400GbE use a single RX Data Flow
40 x 257b blocks of interleaved data of 2 CWs (with uncorrectable errors)
40 x 257b blocks of next two interleaved codewords (no errors).
0
37
38
39 A/B
0
1
2
39
39.1
39.2
39.3
39.4
0.1
0.2
0.3
0.4
800GbE with two dataflows and 66b block collection RX Data Flow
40 x 257b blocks of interleaved data of CW A/B CWs
40 x 257b blocks of next two interleaved codewords (no errors).
Flow 0:
40 x 257b blocks of interleaved data of CW C/D CWs
Flow 1:
39 C/D
66-bit block collection
39.4
39.3
39.4
39.4
0.1
0.1
0.2
0.2
0.3
0.3
0.4
0.4
1.1
1.1
1.2
1.2
1.6TbE with two dataflows and 66b block collection RX Data Flow
257-bit block collection
38.4
38.1
39.4
39.1
0
0
1

Uncorrected errors in the one block can propagate to the first 58 bits of the first block of the next correctly received codewords due to the descrambler.
Flags in the first 5 bits of a 257b block can affect all 4 66b blocks after reverse transcoding

The stateless decoder defined in 172.x.x.x causes one 66b block of the possible propaged errors to be invalid, but misses the following 3 66b blocks.

The stateless decoder defined in 172.x.x.x causes valid 66b blocks to be invalid, but does not invlaidate the 4 with error extension (0.1, 0.2, 0.3, and 0.4)

If RS-FEC CWs A/B are OK (flow 0), and flow1 has error in FEC CWs C/D, then stateless decoder still misses.

The stateless decoder defined in 172.x.x.x causes valid 66b blocks to be invalid, but does not invalidate the 4 blocks with error extension (0.1, 0.2, 0.3, and 0.4)

🟩 = 257b block of an valid pair of RS-FEC codewords A & B
🟥 = 257b block of a valid pair of RS-FEC codewords C & D
⬜ = 257b block of an invalid pair of RS-FEC codewords
🟧 = 257b block affected by descrambler error propagation
▨ = 66b block marked invalid by stateless 64b/66b decode

Comment #669, #432, #433 (Handling scrambler error extension - 6)

Fix approach:

The RS-FEC decode subclause in Clause 119 currently requires that all 66-bit blocks of two interleaved codewords are to be marked as ERROR blocks if either of the codewords is uncorrectable. Add a statement in the FEC decode section that the four 66-bit blocks (that make up the following 257-bit block) following an uncorrectable codeword must also be marked as ERROR blocks due to scrambler error extension.

- ***Require*** the full fix for all new PHYs that may use the stateless decoder.
  - Applies to all 200GE/400GE/1.6TE PHYs
    - All 1.6TE PHYs are new in .3dj.
    - The stateless decoder option for 200/400GE PHYs is new in .3dj.
  - Modify (simplify) the stateless decoder definition to remove useless ERROR marking
  - Add the new text to Clause 119, and add references to it from Clause 175.
- ***Optional*** (should) do “full fix” in Clause 172 for 800GE PHYs, but may keep the current definition of the stateless decoder (in 802.3df) due to project scope.

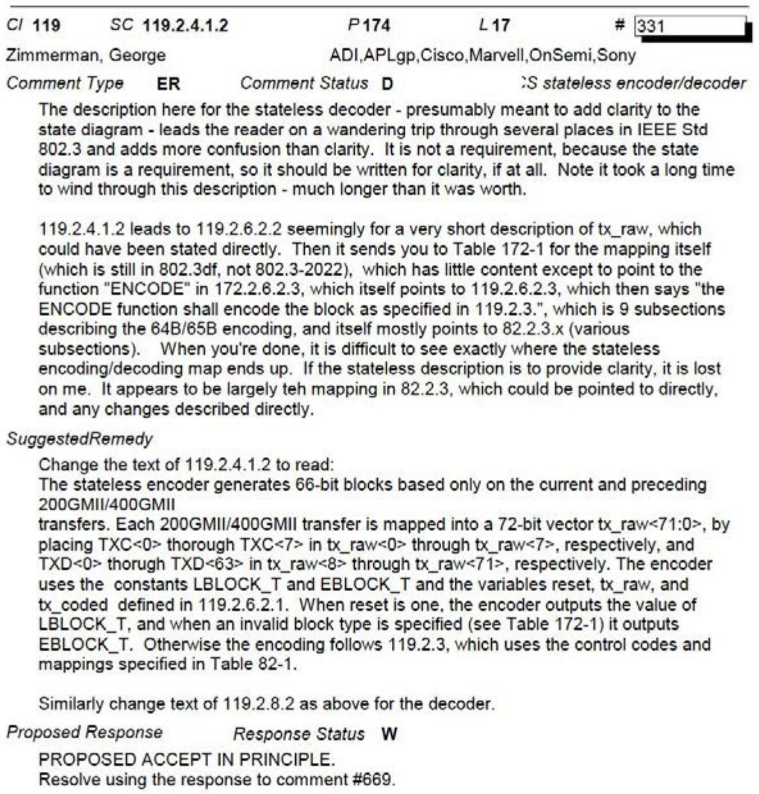
Comment #669, #432, #433 (Handling scrambler error extension - Changes 1)

Update the 3rd paragraph of **119.2.5.3 “Reed-Solomon decoder”** with an added sentence:

> If bypass error indication is not supported or not enabled, when the Reed-Solomon decoder determines that a codeword contains errors that were not corrected, it shall cause the PCS receive function to set every 66-bit block within the two associated codewords to an error block (set to EBLOCK_R). This may be achieved by setting the synchronization header to 11 for all 66-bit blocks created from these codewords by the 256B/257B to 64B/66B transcoder. When the stateless 64B/66B decoder is used as defined in 119.2.5.8.2, then the first four 66-bit blocks following the uncorrected codewords shall also be set to an error block.

Note: The above also applies to 172.2.5.3 and 175.2.5.3 by cross-reference to 119.2.5.3.

Comments #331 (Excessive cross-references)



Cl **119**	*SC* **119.2.4.1.2**	*P* **174**	*L* **17**	*#* 331

Zimmerman, George ADI,APLgp,Cisco,Marvell,OnSemi,Sony

Comment Type **ER** *Comment Status* **D** *:S stateless encoder/decoder*

The description here for the stateless decoder - presumably meant to add clarity to the state diagram - leads the reader on a wandering trip through several places in IEEE Std 802.3 and adds more confusion than clarity. It is not a requirement, because the state diagram is a requirement, so it should be written for clarity, if at all. Note it took a long time to wind through this description - much longer than it was worth.

119.2.4.1.2 leads to 119.2.6.2.2 seemingly for a very short description of tx_raw, which could have been stated directly. Then it sends you to Table 172-1 for the mapping itself (which is still in 802.3df, not 802.3-2022), which has little content except to point to the function "ENCODE" in 172.2.6.2.3, which itself points to 119.2.6.2.3, which then says "the ENCODE function shall encode the block as specified in 119.2.3.", which is 9 subsections describing the 64B/65B encoding, and itself mostly points to 82.2.3.x (various subsections). When you're done, it is difficult to see exactly where the stateless encoding/decoding map ends up. If the stateless description is to provide clarity, it is lost on me. It appears to be largely teh mapping in 82.2.3, which could be pointed to directly, and any changes described directly.

SuggestedRemedy

Change the text of 119.2.4.1.2 to read:
The stateless encoder generates 66-bit blocks based only on the current and preceding 200GMII/400GMII
transfers. Each 200GMII/400GMII transfer is mapped into a 72-bit vector tx_raw<71:0>, by placing TXC<0> thorough TXC<7> in tx_raw<0> through tx_raw<7>, respectively, and TXD<0> thorugh TXD<63> in tx_raw<8> through tx_raw<71>, respectively. The encoder uses the constants LBLOCK_T and EBLOCK_T and the variables reset, tx_raw, and tx_coded defined in 119.2.6.2.1. When reset is one, the encoder outputs the value of LBLOCK_T, and when an invalid block type is specified (see Table 172-1) it outputs EBLOCK_T. Otherwise the encoding follows 119.2.3, which uses the control codes and mappings specified in Table 82-1.

Similarly change text of 119.2.8.2 as above for the decoder.

Proposed Response *Response Status* **W**

PROPOSED ACCEPT IN PRINCIPLE.
Resolve using the response to comment #669.

Comment #331 points out that the stateless encoder (and stateless decoder) have an excessive amount of nested cross-references that can be simplified.

This can be done as part of the new simplified stateless encoder and decoder.

Comment #669, #432, #433 (Handling scrambler error extension - Changes 2)

Change **119.2.5.8.2 “Stateless decoder”** as follows:

The stateless decoder generates 200GMII/400GMII transfers based only on the current ~~and preceding~~ 66-bit block~~s~~ and PCS reset. ~~The decoder shall decode each 66-bit block rx_coded<65:0> to a 72-bits vector rx_raw<71:0> (see 119.2.6.2.2) according to the rules in Table 172-4. Constants LBLOCK_R and EBLOCK_R are defined in 119.2.6.2.1. Variables reset, rx_raw, and rx_coded are defined in 119.2.6.2.2. Functions R_TYPE and DECODE, and the block types are defined in 119.2.6.2.3.~~ When PCS reset is asserted, RXD<63:0> and RXC<7:0> are set to the constant LBLOCK_R (see 119.2.6.2.1). When PCS reset is not asserted, RXD<63:0> and RXC<7:0> are decoded from rx_coded<65:0> as defined in 119.2.3.

Comment #669, #432, #433 (Handling scrambler error extension - Changes 3)

Update **175.2.5.9 “64B/66B decoder”** as follows:

The receive PCS decodes 66-bit blocks to produce RXD<63:0> and RXC<7:0> for transmission to the 1.6TMII. One 1.6TMII transfer is decoded from each 66-bit block. The receive PCS may use either the state-diagram decoder defined in Figure 119-15 or the stateless decoder defined in ~~172.2.5.9.2~~ 119.2.5.8.2.

Comment #669, #432, #433 (Handling scrambler error extension - Changes 4)

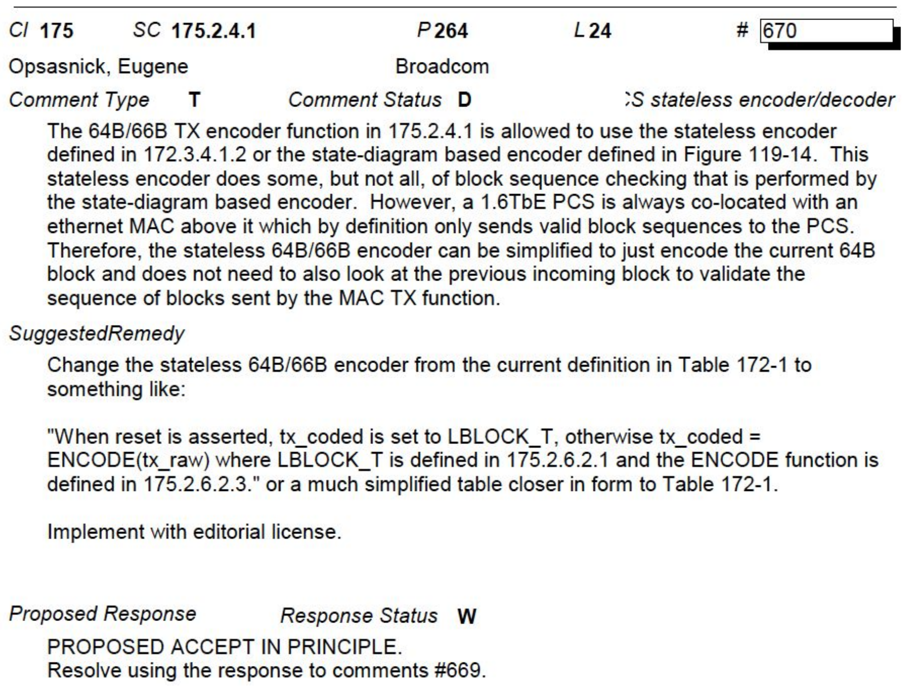
Making changes to Clause 172 is complicated by the current stateless decoder definition in 802.3df-2024 and scope of the 802.3dj project.

The editors propose wording to the effect of: the new Clause 119 stateless encoder ***should*** be used, but the current stateless decoder ***may*** still be used.

Change **172.2.5.9 "64B/66B decoder"** as follows:

> The receive PCS decodes 66-bit blocks to produce RXD<63:0> and RXC<7:0> for transmission to the 800GMII. One 800GMII transfer is decoded from each 66-bit block. The receive PCS shall use one of ~~the~~ two decoding methods, ~~that are defined in 172.2.5.9.1 and 172.2.5.9.2~~ a stateful method as defined in 172.2.5.9.1 or a stateless method. If using a stateless method, then the stateless decoder as defined in 119.2.5.8.2 should be used while the stateless decoder as defined in 172.2.5.9.2 may be used.

Comment #670 (Simpler Stateless Encoder)



Cl **175** *SC* **175.2.4.1** *P* **264** *L* **24** # 670

Opsasnick, Eugene Broadcom

Comment Type **T** *Comment Status* **D** *S stateless encoder/decoder*

The 64B/66B TX encoder function in 175.2.4.1 is allowed to use the stateless encoder defined in 172.3.4.1.2 or the state-diagram based encoder defined in Figure 119-14. This stateless encoder does some, but not all, of block sequence checking that is performed by the state-diagram based encoder. However, a 1.6TbE PCS is always co-located with an ethernet MAC above it which by definition only sends valid block sequences to the PCS. Therefore, the stateless 64B/66B encoder can be simplified to just encode the current 64B block and does not need to also look at the previous incoming block to validate the sequence of blocks sent by the MAC TX function.

SuggestedRemedy

Change the stateless 64B/66B encoder from the current definition in Table 172-1 to something like:

"When reset is asserted, tx_coded is set to LBLOCK_T, otherwise tx_coded = ENCODE(tx_raw) where LBLOCK_T is defined in 175.2.6.2.1 and the ENCODE function is defined in 175.2.6.2.3." or a much simplified table closer in form to Table 172-1.

Implement with editorial license.

Proposed Response *Response Status* **W**

PROPOSED ACCEPT IN PRINCIPLE.
Resolve using the response to comments #669.

#670 suggests simplifying the stateless encoder in a similar way to the stateless decoder.

Following the approach used for changing the stateless decoder:
- The new definition should be added to Clause 119.
- Clause 175 is updated to refer to Clause 119.
- Clause 172 can refer to Clause 119 as an option to the encoder already defined in Clause 172.

Comment #670 (Simpler Stateless Encoder) - 2

Change **119.2.4.1.2 “Stateless encoder”** as follows:

The stateless encoder generates generates 66-bit blocks based only on the current ~~and preceding~~ 200GMII/400GMII transfer~~s~~ and PCS reset. ~~Each 200GMII/400GMII transfer is mapped into a 72-bit vector tx_raw<71:0> (see 119.2.6.2.2). The encoder shall encode each tx_raw<71:0> to a 66-bit block tx_coded<65:0> according to the rules in Table 172-1. Constants LBLOCK_T and EBLOCK_T are defined in 119.2.6.2.1. Variables reset, tx_raw, and tx_coded are defined in 119.2.6.2.1. Functions T_TYPE and ENCODE, and the block types are defined in 119.2.6.2.3.~~ When PCS reset is asserted, tx_coded<65:0> is set to the constant LBLOCK_T (see 119.2.6.2.1). When PCS reset is not asserted, tx_coded<65:0> is encoded from the TXD<63:0> and TXC<7:0> signals as defined in 119.2.3.

Comment #670 (Simpler Stateless Encoder) - 3

Update **175.2.4.1 “64B/66B encoder”** as follows:

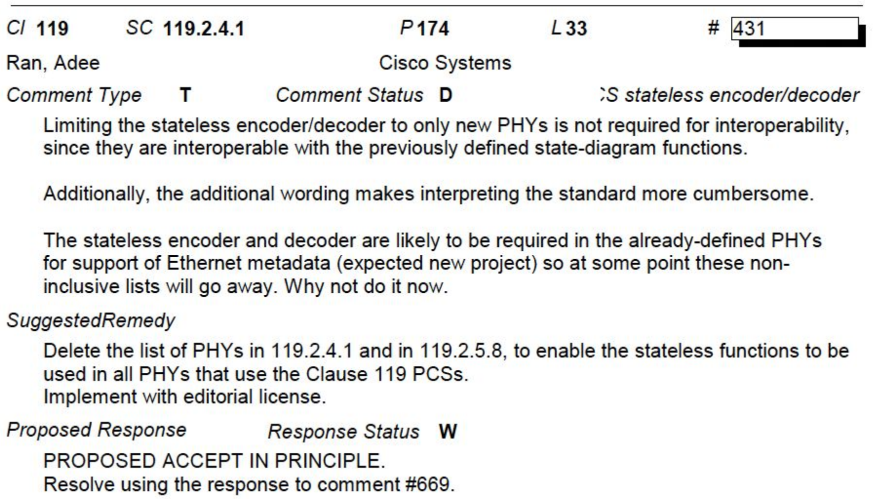
The transmit PCS may use either the state-diagram encoder defined by Figure 119-14 or the stateless encoder defined in ~~172.2.4.1.2~~ 119.2.4.1.2.

Comment #670 (Simpler Stateless Encoder) - 4

Change the text in **172.2.4.1 “64B/66B encoder”** as follows:

> The transmit PCS generates 66-bit blocks based on the TXD<63:0> and TXC<7:0> signals received from the 800GMII. Each 800GMII transfer is encoded into one 66-bit block. The contents of each 66-bit block are contained in a vector tx_coded<65:0> with tx_coded<1:0> containing the sync header and the remainder of the bits the payload. The transmit PCS shall use ~~the encoding method defined in either 172.2.4.1.1 or 172.2.4.1.2.~~ one of two encoding methods, a stateful method as defined in 172.2.4.1.1 or a stateless method. If using a stateless method, then the stateless encoder as defined in 119.2.5.1.2 should be used while the stateless encoder as defined in 172.2.4.1.2 may be used.

Comments #431, 584, 676 (Allowing stateless encoder/decoder for all 200G/400G PHYs)



Cl **119** *SC* **119.2.4.1** *P* **174** *L* **33** # 431

Ran, Adee Cisco Systems

Comment Type **T** *Comment Status* **D** *:S stateless encoder/decoder*

Limiting the stateless encoder/decoder to only new PHYs is not required for interoperability, since they are interoperable with the previously defined state-diagram functions.

Additionally, the additional wording makes interpreting the standard more cumbersome.

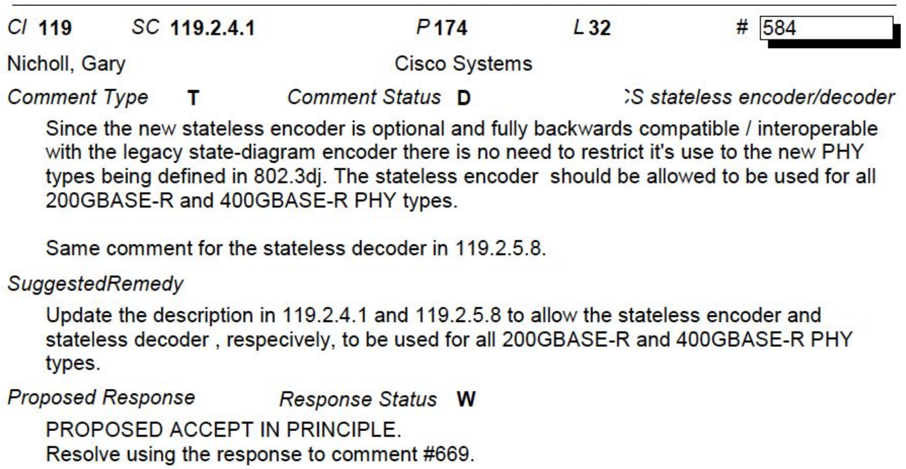
The stateless encoder and decoder are likely to be required in the already-defined PHYs for support of Ethernet metadata (expected new project) so at some point these non-inclusive lists will go away. Why not do it now.

SuggestedRemedy

Delete the list of PHYs in 119.2.4.1 and in 119.2.5.8, to enable the stateless functions to be used in all PHYs that use the Clause 119 PCSs.
Implement with editorial license.

Proposed Response *Response Status* **W**

PROPOSED ACCEPT IN PRINCIPLE.
Resolve using the response to comment #669.



Cl **119** *SC* **119.2.4.1** *P* **174** *L* **32** # 584

Nicholl, Gary Cisco Systems

Comment Type **T** *Comment Status* **D** *:S stateless encoder/decoder*

Since the new stateless encoder is optional and fully backwards compatible / interoperable with the legacy state-diagram encoder there is no need to restrict it's use to the new PHY types being defined in 802.3dj. The stateless encoder should be allowed to be used for all 200GBASE-R and 400GBASE-R PHY types.

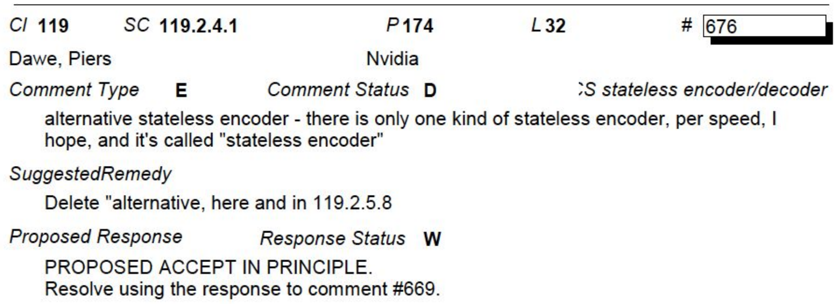
Same comment for the stateless decoder in 119.2.5.8.

SuggestedRemedy

Update the description in 119.2.4.1 and 119.2.5.8 to allow the stateless encoder and stateless decoder , respecively, to be used for all 200GBASE-R and 400GBASE-R PHY types.

Proposed Response *Response Status* **W**

PROPOSED ACCEPT IN PRINCIPLE.
Resolve using the response to comment #669.



Cl **119** *SC* **119.2.4.1** *P* **174** *L* **32** # 676

Dawe, Piers Nvidia

Comment Type **E** *Comment Status* **D** *:S stateless encoder/decoder*

alternative stateless encoder - there is only one kind of stateless encoder, per speed, I hope, and it's called "stateless encoder"

SuggestedRemedy

Delete "alternative, here and in 119.2.5.8

Proposed Response *Response Status* **W**

PROPOSED ACCEPT IN PRINCIPLE.
Resolve using the response to comment #669.

#431 and #584 make the same proposed change to "clean up" 119.2.4.1 and 119.2.5.8.

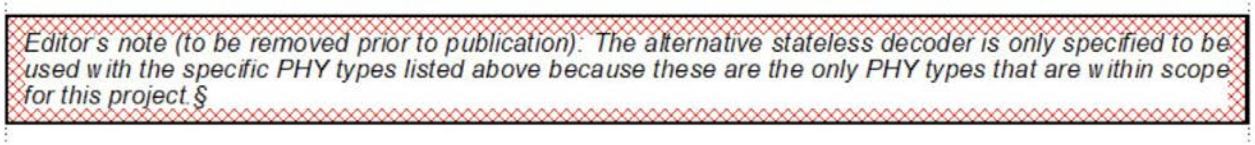
#676 proposes a specific wording change in the same section that should also happen (remove "alternative").

Comments #431, 584, 676 (Allowing stateless encoder/decoder for all 200G/400G PHYs) - 3

Change the text in **119.2.4.1** as follows:

The transmit PCS generate 66-bit blocks using the state-diagram encoder defined in 119.2.4.1.1 ~~for any 200GBASE-R or 400GBASE-R PHY type,~~ or using the ~~alternative~~ stateless encoder defined in 119.2.4.1.2 ~~for the following PHY types:~~

- ~~200GBASE-KR1~~
- ~~200GBASE-CR1~~
- ~~200GBASE-DR1~~
- ~~200GBASE-DR1-2~~
- ~~400GBASE-KR2~~
- ~~400GBASE-CR2~~
- ~~400GBASE-DR2~~
- ~~400GBASE-DR2-2~~

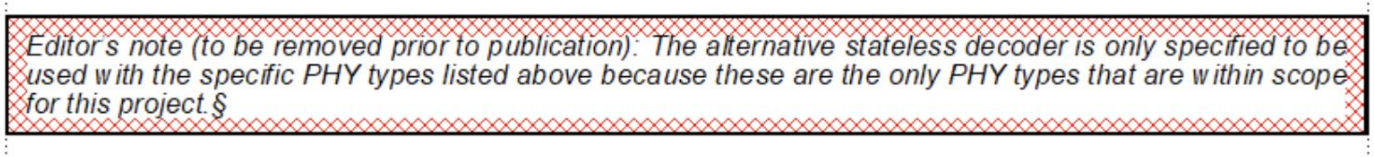

Editor's note (to be removed prior to publication): The alternative stateless decoder is only specified to be used with the specific PHY types listed above because these are the only PHY types that are within scope for this project.§

Comments #431, 584, 676
(Allowing stateless encoder/decoder for all 200G/400G PHYs) - 2

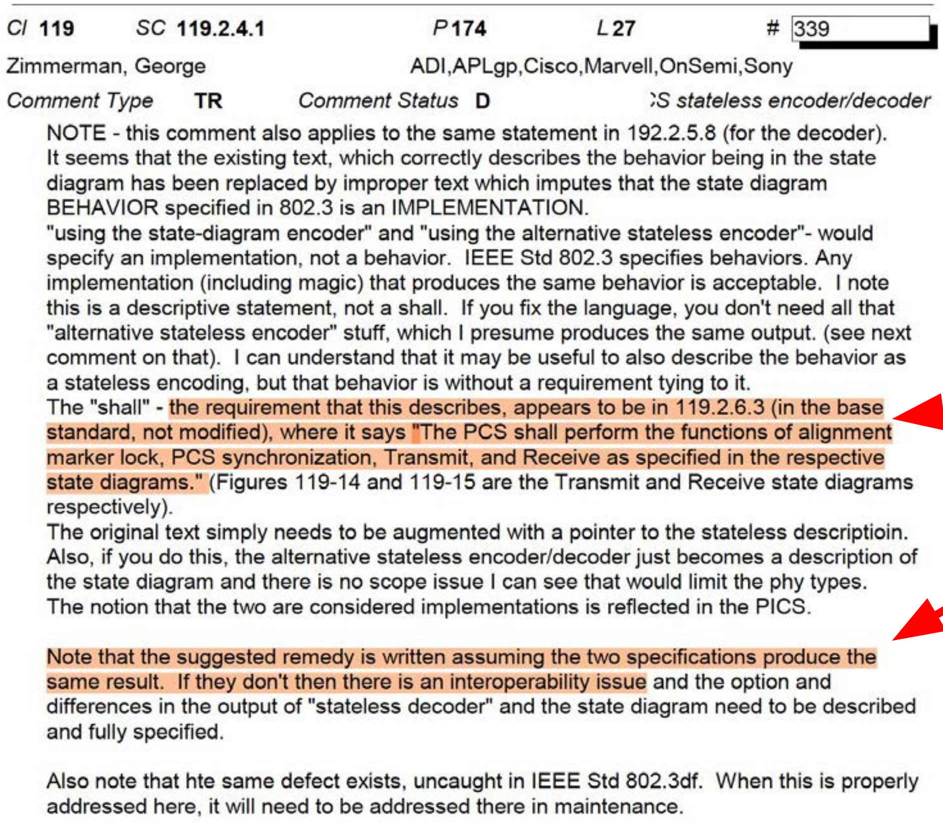
Change the text in **119.2.5.8** as follows:

The receive PCS decode 66-bit blocks using the state-diagram decoder defined in 119.2.5.8.1 ~~for any 200GBASE-R or 400GBASE-R PHY types,~~ or using the ~~alternative~~ stateless decoder defined in 119.2.5.8.2. ~~for the following PHY types:~~

- ~~200GBASE-KR1~~
- ~~200GBASE-CR1~~
- ~~200GBASE-DR1~~
- ~~200GBASE-DR1-2~~
- ~~400GBASE-KR1~~
- ~~400GBASE-CR1~~
- ~~400GBASE-DR1~~
- ~~400GBASE-DR2~~


Editor's note (to be removed prior to publication): The alternative stateless decoder is only specified to be used with the specific PHY types listed above because these are the only PHY types that are within scope for this project.§

Comment #339 (Stateless and State Diagram encoder/decoder equivalence)



Cl **119** *SC* **119.2.4.1** *P* **174** *L* **27** # 339

Zimmerman, George ADI,APLgp,Cisco,Marvell,OnSemi,Sony

Comment Type **TR** *Comment Status* **D** *:S stateless encoder/decoder*

NOTE - this comment also applies to the same statement in 192.2.5.8 (for the decoder). It seems that the existing text, which correctly describes the behavior being in the state diagram has been replaced by improper text which imputes that the state diagram BEHAVIOR specified in 802.3 is an IMPLEMENTATION.
"using the state-diagram encoder" and "using the alternative stateless encoder"- would specify an implementation, not a behavior. IEEE Std 802.3 specifies behaviors. Any implementation (including magic) that produces the same behavior is acceptable. I note this is a descriptive statement, not a shall. If you fix the language, you don't need all that "alternative stateless encoder" stuff, which I presume produces the same output. (see next comment on that). I can understand that it may be useful to also describe the behavior as a stateless encoding, but that behavior is without a requirement tying to it.
The "shall" - the requirement that this describes, appears to be in 119.2.6.3 (in the base standard, not modified), where it says "The PCS shall perform the functions of alignment marker lock, PCS synchronization, Transmit, and Receive as specified in the respective state diagrams." (Figures 119-14 and 119-15 are the Transmit and Receive state diagrams respectively).
The original text simply needs to be augmented with a pointer to the stateless descriptioin. Also, if you do this, the alternative stateless encoder/decoder just becomes a description of the state diagram and there is no scope issue I can see that would limit the phy types. The notion that the two are considered implementations is reflected in the PICS.

Note that the suggested remedy is written assuming the two specifications produce the same result. If they don't then there is an interoperability issue and the option and differences in the output of "stateless decoder" and the state diagram need to be described and fully specified.

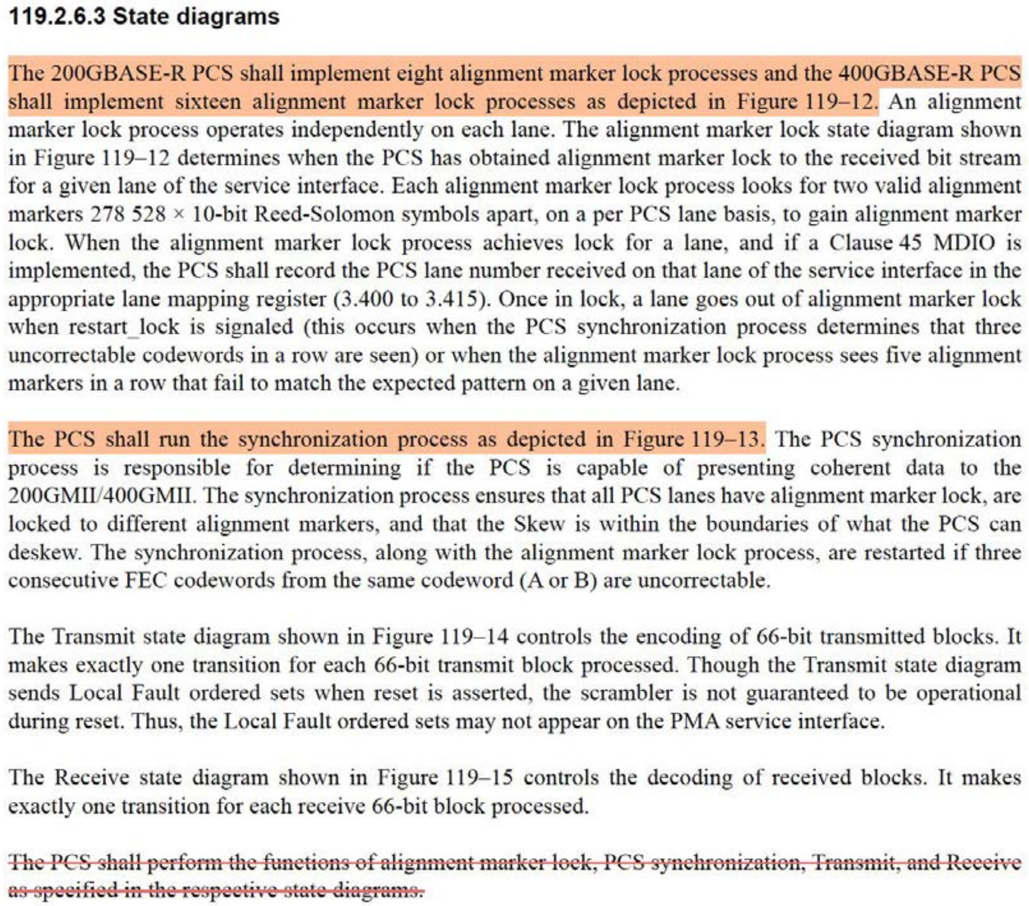
Also note that hte same defect exists, uncaught in IEEE Std 802.3df. When this is properly addressed here, it will need to be addressed there in maintenance.

Comment #339 assumes the stateless encoder/decoder is an equivalent description of the state-diagram-based encoder/decoder. They are in fact different and produce different output sequences.

But the comment does point out some text that needs to be updated,

The suggested remedy is not useful since it assumes the stateless and stateful descriptions are equivalent.

Comment #339 - Changes



119.2.6.3 State diagrams

The 200GBASE-R PCS shall implement eight alignment marker lock processes and the 400GBASE-R PCS shall implement sixteen alignment marker lock processes as depicted in Figure 119–12. An alignment marker lock process operates independently on each lane. The alignment marker lock state diagram shown in Figure 119–12 determines when the PCS has obtained alignment marker lock to the received bit stream for a given lane of the service interface. Each alignment marker lock process looks for two valid alignment markers 278 528 × 10-bit Reed-Solomon symbols apart, on a per PCS lane basis, to gain alignment marker lock. When the alignment marker lock process achieves lock for a lane, and if a Clause 45 MDIO is implemented, the PCS shall record the PCS lane number received on that lane of the service interface in the appropriate lane mapping register (3.400 to 3.415). Once in lock, a lane goes out of alignment marker lock when restart_lock is signaled (this occurs when the PCS synchronization process determines that three uncorrectable codewords in a row are seen) or when the alignment marker lock process sees five alignment markers in a row that fail to match the expected pattern on a given lane.

The PCS shall run the synchronization process as depicted in Figure 119–13. The PCS synchronization process is responsible for determining if the PCS is capable of presenting coherent data to the 200GMII/400GMII. The synchronization process ensures that all PCS lanes have alignment marker lock, are locked to different alignment markers, and that the Skew is within the boundaries of what the PCS can deskew. The synchronization process, along with the alignment marker lock process, are restarted if three consecutive FEC codewords from the same codeword (A or B) are uncorrectable.

The Transmit state diagram shown in Figure 119–14 controls the encoding of 66-bit transmitted blocks. It makes exactly one transition for each 66-bit transmit block processed. Though the Transmit state diagram sends Local Fault ordered sets when reset is asserted, the scrambler is not guaranteed to be operational during reset. Thus, the Local Fault ordered sets may not appear on the PMA service interface.

The Receive state diagram shown in Figure 119–15 controls the decoding of received blocks. It makes exactly one transition for each receive 66-bit block processed.

~~The PCS shall perform the functions of alignment marker lock, PCS synchronization, Transmit, and Receive as specified in the respective state diagrams.~~

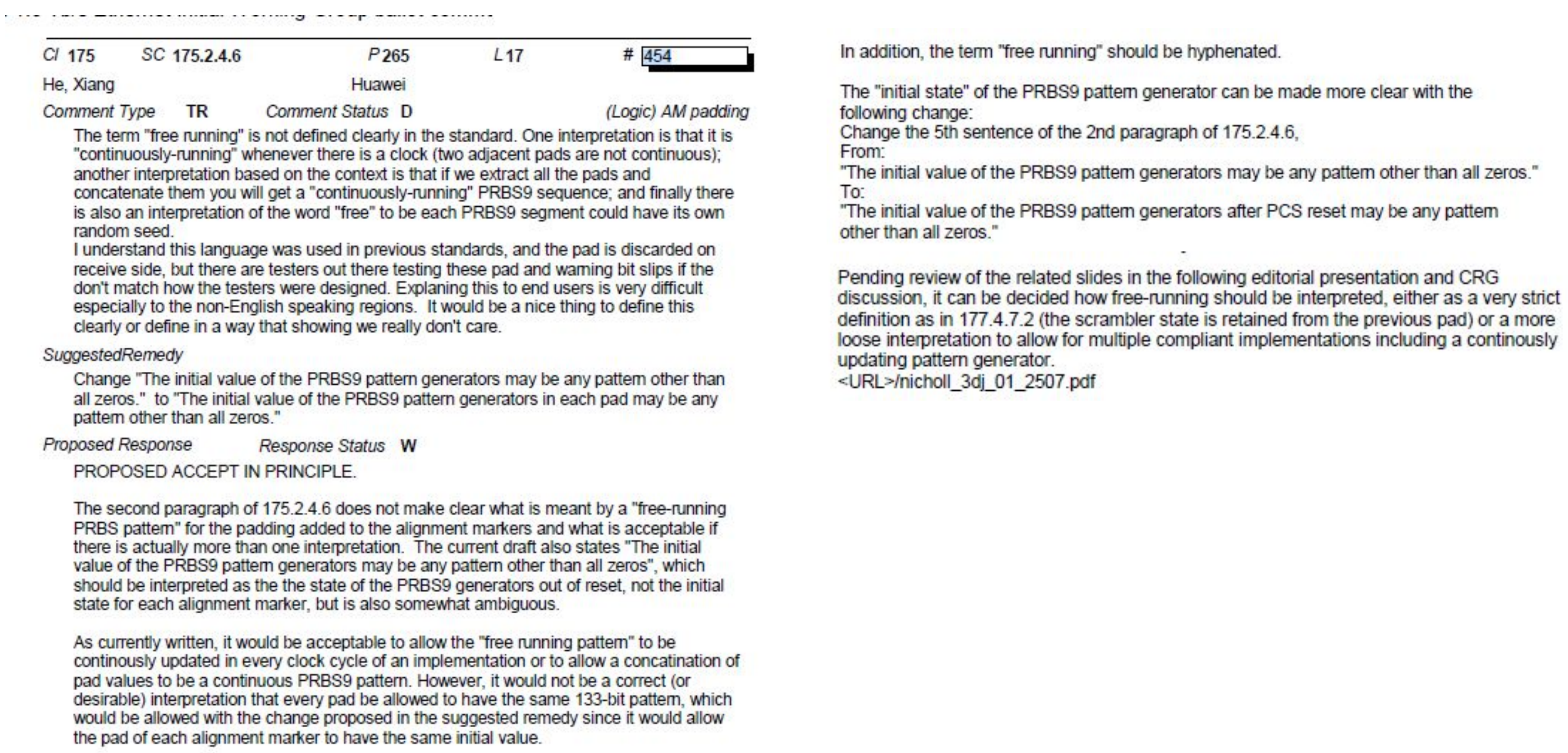
The first sentence of the first two paragraphs in 119.2.6.3 contain the same “shall” requirement as is stated in the last sentence of the subclause. The last sentence is now incorrect in requiring the state-machine encoder/decoder since the stateless encoder/decoder are now an acceptable alternative as described in changes to 119.2.4.1 and 119.2.5.8.

→ Delete the last sentence of 119.2.6.3. (This was missed in the changes to the previous drafts.)

Alignment Marker Padding

Comment #454

Comment #454



Cl 175 *SC* 175.2.4.6 *P* 265 *L* 17 # 454

He, Xiang Huawei

Comment Type TR *Comment Status* D *(Logic) AM padding*

The term "free running" is not defined clearly in the standard. One interpretation is that it is "continuously-running" whenever there is a clock (two adjacent pads are not continuous); another interpretation based on the context is that if we extract all the pads and concatenate them you will get a "continuously-running" PRBS9 sequence; and finally there is also an interpretation of the word "free" to be each PRBS9 segment could have its own random seed.
I understand this language was used in previous standards, and the pad is discarded on receive side, but there are testers out there testing these pad and warning bit slips if the don't match how the testers were designed. Explaning this to end users is very difficult especially to the non-English speaking regions. It would be a nice thing to define this clearly or define in a way that showing we really don't care.

SuggestedRemedy

Change "The initial value of the PRBS9 pattern generators may be any pattern other than all zeros." to "The initial value of the PRBS9 pattern generators in each pad may be any pattern other than all zeros."

Proposed Response *Response Status* W

PROPOSED ACCEPT IN PRINCIPLE.

The second paragraph of 175.2.4.6 does not make clear what is meant by a "free-running PRBS pattern" for the padding added to the alignment markers and what is acceptable if there is actually more than one interpretation. The current draft also states "The initial value of the PRBS9 pattern generators may be any pattern other than all zeros", which should be interpreted as the the state of the PRBS9 generators out of reset, not the initial state for each alignment marker, but is also somewhat ambiguous.

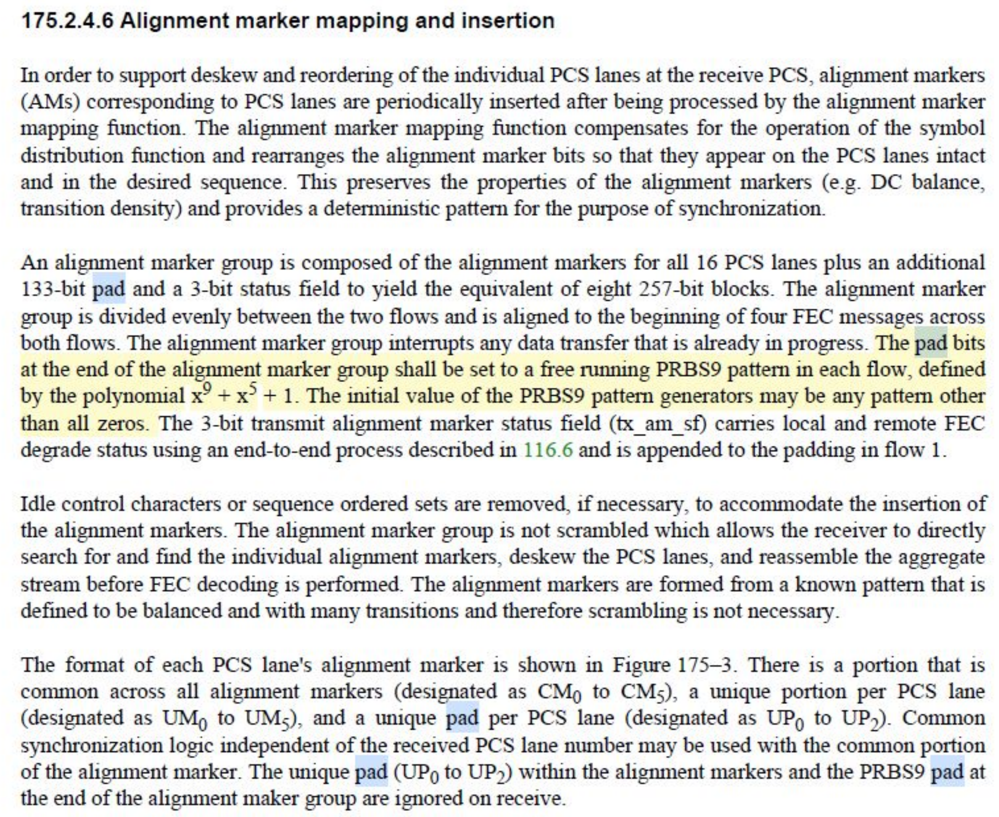
As currently written, it would be acceptable to allow the "free running pattern" to be continously updated in every clock cycle of an implementation or to allow a concatination of pad values to be a continuous PRBS9 pattern. However, it would not be a correct (or desirable) interpretation that every pad be allowed to have the same 133-bit pattern, which would be allowed with the change proposed in the suggested remedy since it would allow the pad of each alignment marker to have the same initial value.

In addition, the term "free running" should be hyphenated.

The "initial state" of the PRBS9 pattern generator can be made more clear with the following change:
Change the 5th sentence of the 2nd paragraph of 175.2.4.6,
From:
"The initial value of the PRBS9 pattern generators may be any pattern other than all zeros."
To:
"The initial value of the PRBS9 pattern generators after PCS reset may be any pattern other than all zeros."

Pending review of the related slides in the following editorial presentation and CRG discussion, it can be decided how free-running should be interpreted, either as a very strict definition as in 177.4.7.2 (the scrambler state is retained from the previous pad) or a more loose interpretation to allow for multiple compliant implementations including a continously updating pattern generator.
<URL>/nicholl_3dj_01_2507.pdf

Comment #454 - Current text in 175.2.4.6



175.2.4.6 Alignment marker mapping and insertion

In order to support deskew and reordering of the individual PCS lanes at the receive PCS, alignment markers (AMs) corresponding to PCS lanes are periodically inserted after being processed by the alignment marker mapping function. The alignment marker mapping function compensates for the operation of the symbol distribution function and rearranges the alignment marker bits so that they appear on the PCS lanes intact and in the desired sequence. This preserves the properties of the alignment markers (e.g. DC balance, transition density) and provides a deterministic pattern for the purpose of synchronization.

An alignment marker group is composed of the alignment markers for all 16 PCS lanes plus an additional 133-bit pad and a 3-bit status field to yield the equivalent of eight 257-bit blocks. The alignment marker group is divided evenly between the two flows and is aligned to the beginning of four FEC messages across both flows. The alignment marker group interrupts any data transfer that is already in progress. The pad bits at the end of the alignment marker group shall be set to a free running PRBS9 pattern in each flow, defined by the polynomial $x^9 + x^5 + 1$. The initial value of the PRBS9 pattern generators may be any pattern other than all zeros. The 3-bit transmit alignment marker status field (tx_am_sf) carries local and remote FEC degrade status using an end-to-end process described in 116.6 and is appended to the padding in flow 1.

Idle control characters or sequence ordered sets are removed, if necessary, to accommodate the insertion of the alignment markers. The alignment marker group is not scrambled which allows the receiver to directly search for and find the individual alignment markers, deskew the PCS lanes, and reassemble the aggregate stream before FEC decoding is performed. The alignment markers are formed from a known pattern that is defined to be balanced and with many transitions and therefore scrambling is not necessary.

The format of each PCS lane's alignment marker is shown in Figure 175–3. There is a portion that is common across all alignment markers (designated as CM_0 to CM_5), a unique portion per PCS lane (designated as UM_0 to UM_5), and a unique pad per PCS lane (designated as UP_0 to UP_2). Common synchronization logic independent of the received PCS lane number may be used with the common portion of the alignment marker. The unique pad (UP_0 to UP_2) within the alignment markers and the PRBS9 pad at the end of the alignment maker group are ignored on receive.

The comment is requesting an update to the text highlighted in yellow to more clearly define what is meant by a “free-running PRBS9 pattern generator” in the context of the “133-bit pad” that is inserted at the end of the alignment marker group.

The two valid options are:

1. The PRBS9 pattern generator increments for every bit of the 133-bit pad, with the state of the pattern generator being maintained between 133-bit pads. The initial value of the PRBS9 pattern generators after PCS reset may be any pattern other than all zeros.

2. The PRBS9 pattern generator may either increment every bit of the 133-bit pad, with the state of the pattern generator being maintained between 133-bit pads, or it may increment with every transmitted bit, with the current state of the pattern generator during every 133-bit pad being used for the the pad value.

   The initial value of the PRBS9 pattern generators after PCS reset may be any pattern other than all zeros.

Comment #454 - Option #1

Update the text in the 2nd paragraph of 175.2.4.6 as follows:

An alignment marker group is composed of the alignment markers for all 16 PCS lanes plus an additional 133-bit pad and a 3-bit status field to yield the equivalent of eight 257-bit blocks. The alignment marker group is divided evenly between the two flows and is aligned to the beginning of four FEC messages across both flows. The alignment marker group interrupts any data transfer that is already in progress. The pad bits at the end of the alignment marker group shall be set to a free-running PRBS9 pattern in each flow, defined by the polynomial
$x9 + x5 + 1$. The initial value of the PRBS9 pattern generators after PCS reset is deasserted may be any pattern other than all zeros, and the pattern generator state is retained from the previous pad. The 3-bit transmit alignment marker status field (tx_am_sf) carries local and remote FEC degrade status using an end-to-end process described in 116.6 and is appended to the padding in flow 1.

Comment #454 - Option #2

Update the text in the 2nd paragraph of 175.2.4.6 as follows:

An alignment marker group is composed of the alignment markers for all 16 PCS lanes plus an additional 133-bit pad and a 3-bit status field to yield the equivalent of eight 257-bit blocks. The alignment marker group is divided evenly between the two flows and is aligned to the beginning of four FEC messages across both flows. The alignment marker group interrupts any data transfer that is already in progress. The pad bits at the end of the alignment marker group shall be set to a free-running PRBS9 pattern in each flow, defined by the polynomial
x9 + x5 + 1. The initial value of the PRBS9 pattern generators after PCS reset is deasserted may be any pattern other than all zeros. The pattern generator state may be retained from the previous pad, or it may advance continuously with the pad set to the current state at the time of pad insertion. The 3-bit transmit alignment marker status field (tx_am_sf) carries local and remote FEC degrade status using an end-to-end process described in 116.6 and is appended to the padding in flow 1.

Comment #454 - Option #3

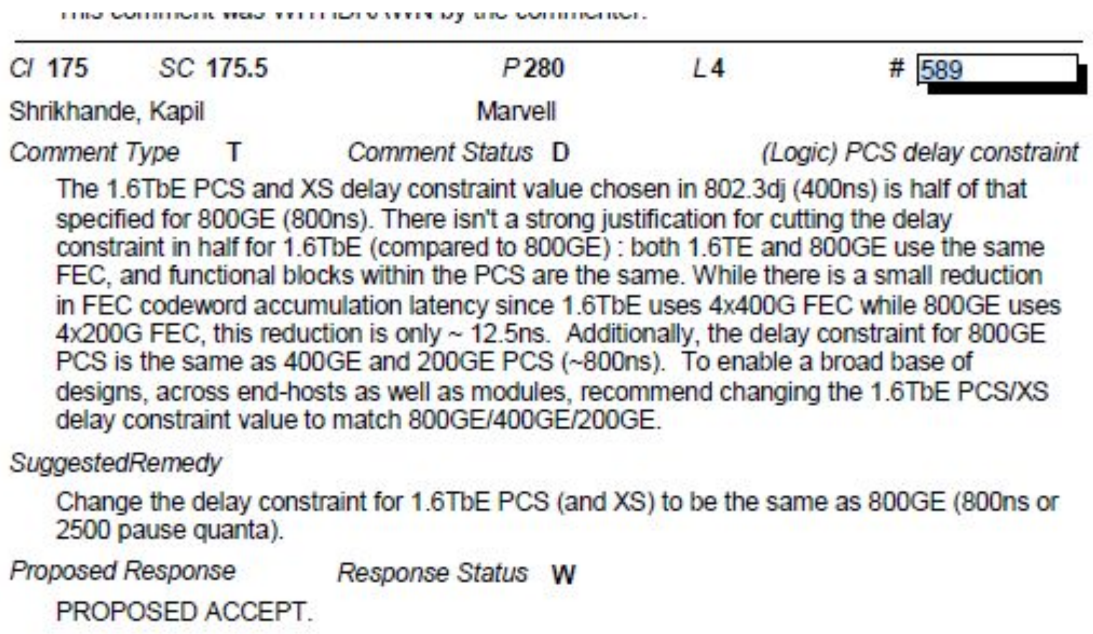
Update the text in the 2nd paragraph of 175.2.4.6 as follows:

An alignment marker group is composed of the alignment markers for all 16 PCS lanes plus an additional 133-bit pad and a 3-bit status field to yield the equivalent of eight 257-bit blocks. The alignment marker group is divided evenly between the two flows and is aligned to the beginning of four FEC messages across both flows. The alignment marker group interrupts any data transfer that is already in progress. The pad bits at the end of the alignment marker group shall be set to a free-running PRBS9 pattern in each flow, defined by the polynomial x9 + x5 + 1. The initial value of the PRBS9 pattern generators after PCS reset is deasserted may be any pattern other than all zeros. The 3-bit transmit alignment marker status field (tx_am_sf) carries local and remote FEC degrade status using an end-to-end process described in 116.6 and is appended to the padding in flow 1.

PCS Delay Constraint

Comment #589

Comment #589



This comment was WITHDRAWN by the commenter.

Cl 175	*SC* 175.5	*P* 280	*L* 4	# 589

Shrikhande, Kapil Marvell

Comment Type T *Comment Status* D *(Logic) PCS delay constraint*

The 1.6TbE PCS and XS delay constraint value chosen in 802.3dj (400ns) is half of that specified for 800GE (800ns). There isn't a strong justification for cutting the delay constraint in half for 1.6TbE (compared to 800GE) : both 1.6TE and 800GE use the same FEC, and functional blocks within the PCS are the same. While there is a small reduction in FEC codeword accumulation latency since 1.6TbE uses 4x400G FEC while 800GE uses 4x200G FEC, this reduction is only ~ 12.5ns. Additionally, the delay constraint for 800GE PCS is the same as 400GE and 200GE PCS (~800ns). To enable a broad base of designs, across end-hosts as well as modules, recommend changing the 1.6TbE PCS/XS delay constraint value to match 800GE/400GE/200GE.

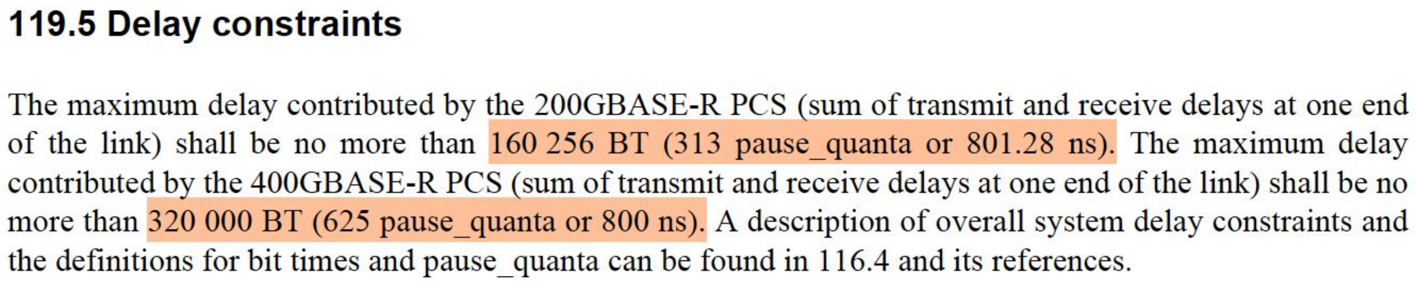
SuggestedRemedy

Change the delay constraint for 1.6TbE PCS (and XS) to be the same as 800GE (800ns or 2500 pause quanta).

Proposed Response *Response Status* W

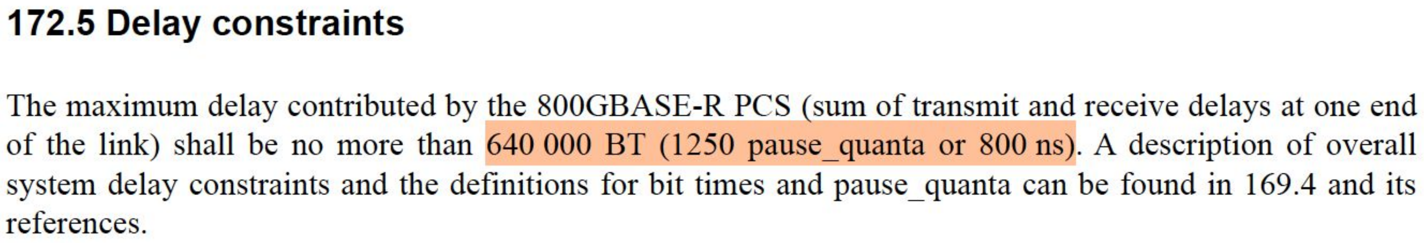
PROPOSED ACCEPT.

Comment #589



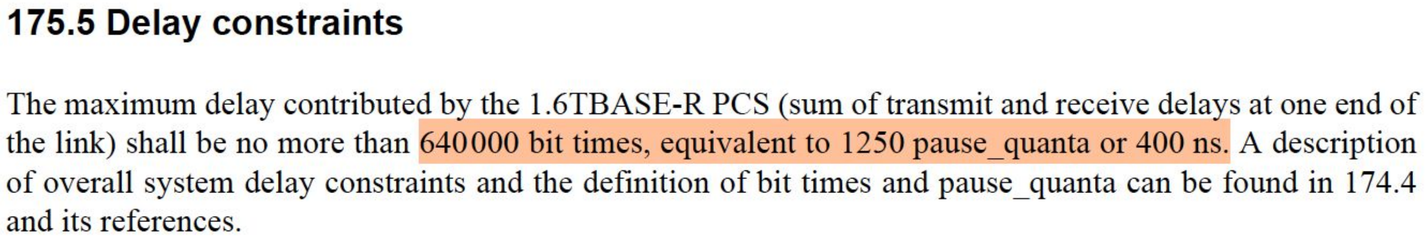
119.5 Delay constraints

The maximum delay contributed by the 200GBASE-R PCS (sum of transmit and receive delays at one end of the link) shall be no more than 160 256 BT (313 pause_quanta or 801.28 ns). The maximum delay contributed by the 400GBASE-R PCS (sum of transmit and receive delays at one end of the link) shall be no more than 320 000 BT (625 pause_quanta or 800 ns). A description of overall system delay constraints and the definitions for bit times and pause_quanta can be found in 116.4 and its references.



172.5 Delay constraints

The maximum delay contributed by the 800GBASE-R PCS (sum of transmit and receive delays at one end of the link) shall be no more than 640 000 BT (1250 pause_quanta or 800 ns). A description of overall system delay constraints and the definitions for bit times and pause_quanta can be found in 169.4 and its references.



175.5 Delay constraints

The maximum delay contributed by the 1.6TBASE-R PCS (sum of transmit and receive delays at one end of the link) shall be no more than 640000 bit times, equivalent to 1250 pause_quanta or 400 ns. A description of overall system delay constraints and the definition of bit times and pause_quanta can be found in 174.4 and its references.

Comment #589, con’t

Change text in 175.5 as follows:

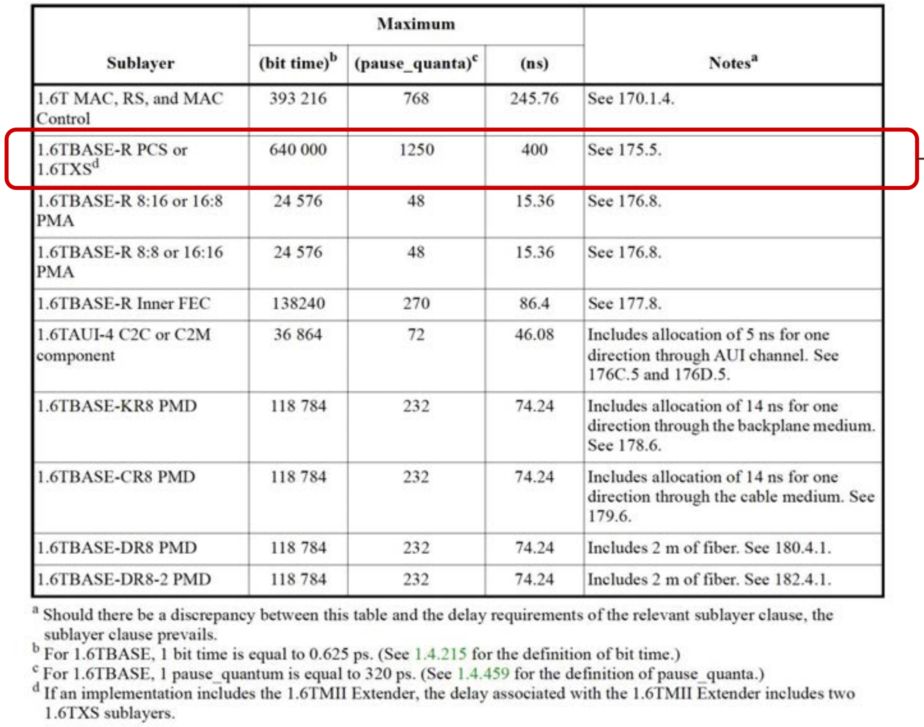
175.5 Delay constraints

The maximum delay contributed by the 1.6TBASE-R PCS (sum of transmit and receive delays at one end of the link) shall be no more than ~~640 000 bit times, equivalent to 1250 pause_quanta or 400 ns~~ 1 280 000 bit times, equivalent to 2500 pause_quanta or 800 ns. A description of overall system delay constraints and the definition of bit times and pause_quanta can be found in 174.4 and its references.

Comment #589, cntd

Current Table 174-4 in D2.0

Table 174–4—Sublayer delay constraints



Sublayer	Maximum (bit time)[b]	Maximum (pause_quanta)[c]	Maximum (ns)	Notes[a]
1.6T MAC, RS, and MAC Control	393 216	768	245.76	See 170.1.4.
1.6TBASE-R PCS or 1.6TXS[d]	640 000	1250	400	See 175.5.
1.6TBASE-R 8:16 or 16:8 PMA	24 576	48	15.36	See 176.8.
1.6TBASE-R 8:8 or 16:16 PMA	24 576	48	15.36	See 176.8.
1.6TBASE-R Inner FEC	138240	270	86.4	See 177.8.
1.6TAUI-4 C2C or C2M component	36 864	72	46.08	Includes allocation of 5 ns for one direction through AUI channel. See 176C.5 and 176D.5.
1.6TBASE-KR8 PMD	118 784	232	74.24	Includes allocation of 14 ns for one direction through the backplane medium. See 178.6.
1.6TBASE-CR8 PMD	118 784	232	74.24	Includes allocation of 14 ns for one direction through the cable medium. See 179.6.
1.6TBASE-DR8 PMD	118 784	232	74.24	Includes 2 m of fiber. See 180.4.1.
1.6TBASE-DR8-2 PMD	118 784	232	74.24	Includes 2 m of fiber. See 182.4.1.

[a] Should there be a discrepancy between this table and the delay requirements of the relevant sublayer clause, the sublayer clause prevails.
[b] For 1.6TBASE, 1 bit time is equal to 0.625 ps. (See 1.4.215 for the definition of bit time.)
[c] For 1.6TBASE, 1 pause_quantum is equal to 320 ps. (See 1.4.459 for the definition of pause_quanta.)
[d] If an implementation includes the 1.6TMII Extender, the delay associated with the 1.6TMII Extender includes two 1.6TXS sublayers.

Proposed changes to Table 174-4

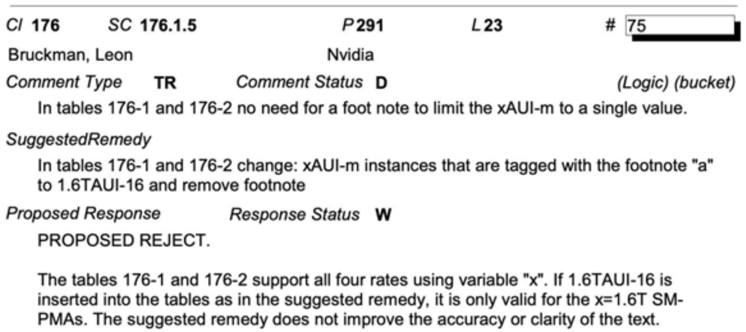
Change 1.6TBASE-R PCS or 1.6TXS delay constraint values in the table as follows:

From 640 000 to 1 280 000 bit times
From 1250 to 2500 pause_quanta
From 400 to 800 ns

PMA layering tables

Comment #75

Comment # 75 (Cl176)



Cl **176**	*SC* **176.1.5**	*P* **291**	*L* **23**	*#* 75

Bruckman, Leon — Nvidia

Comment Type **TR** *Comment Status* **D** *(Logic) (bucket)*

In tables 176-1 and 176-2 no need for a foot note to limit the xAUI-m to a single value.

SuggestedRemedy

In tables 176-1 and 176-2 change: xAUI-m instances that are tagged with the footnote "a" to 1.6TAUI-16 and remove footnote

Proposed Response *Response Status* **W**

PROPOSED REJECT.

The tables 176-1 and 176-2 support all four rates using variable "x". If 1.6TAUI-16 is inserted into the tables as in the suggested remedy, it is only valid for the x=1.6T SM-PMAs. The suggested remedy does not improve the accuracy or clarity of the text.

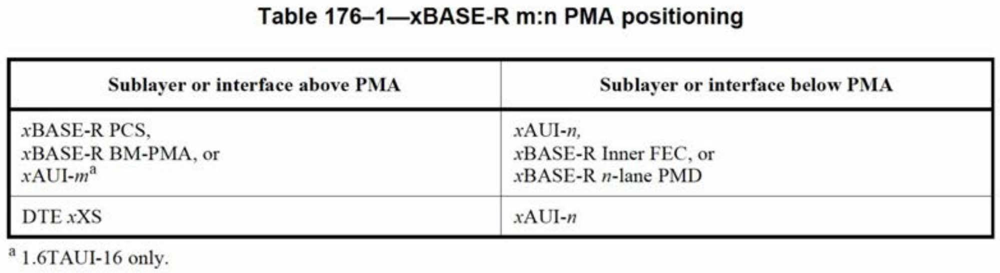


Table 176–1—xBASE-R m:n PMA positioning

Sublayer or interface above PMA	Sublayer or interface below PMA
*x*BASE-R PCS, *x*BASE-R BM-PMA, or *x*AUI-*m*[a]	*x*AUI-*n*, *x*BASE-R Inner FEC, or *x*BASE-R *n*-lane PMD
DTE *x*XS	*x*AUI-*n*

[a] 1.6TAUI-16 only.

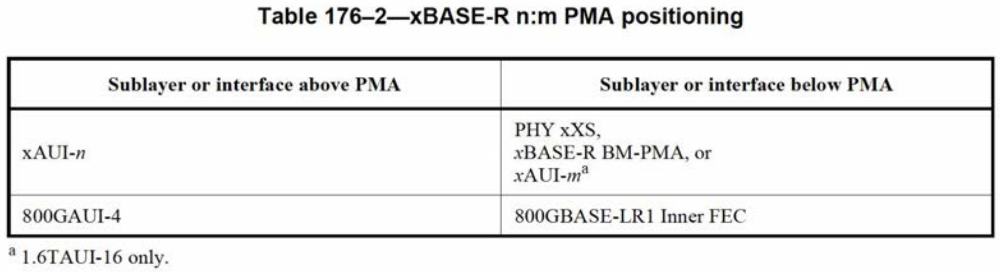


Table 176–2—xBASE-R n:m PMA positioning

Sublayer or interface above PMA	Sublayer or interface below PMA
xAUI-*n*	PHY xXS, *x*BASE-R BM-PMA, or *x*AUI-*m*[a]
800GAUI-4	800GBASE-LR1 Inner FEC

[a] 1.6TAUI-16 only.

Comment # 75 (Cl176): possible change to the table 176-1

Table 176-1 in 802.3dj D2.0

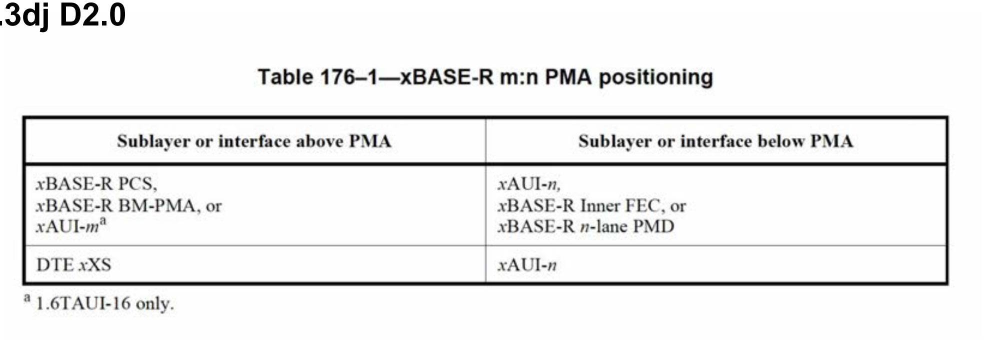


Table 176–1—xBASE-R m:n PMA positioning

Sublayer or interface above PMA	Sublayer or interface below PMA
*x*BASE-R PCS, *x*BASE-R BM-PMA, or *x*AUI-*m*[a]	*x*AUI-*n*, *x*BASE-R Inner FEC, or *x*BASE-R *n*-lane PMD
DTE *x*XS	*x*AUI-*n*

[a] 1.6TAUI-16 only.

Table 176-1 with a separate row for the 1.6TAUI-16 case

Sublayer or interface above PMA	Sublayer or interface below PMA
xBASE-R PCS, or xBASE-R BM-PMA	xAUI-n, xBASE-R Inner FEC, or xBASE-R n-lane PMD
DTE xXS	xAUI-n
1.6TAUI-16	1.6TAUI-8, 1.6TBASE-R Inner FEC, or 1.6TBASE-R 8-lane PMD

Comment # 75 (Cl176): possible change to the table 176-2

Table 176-2 in 802.3dj D2.0

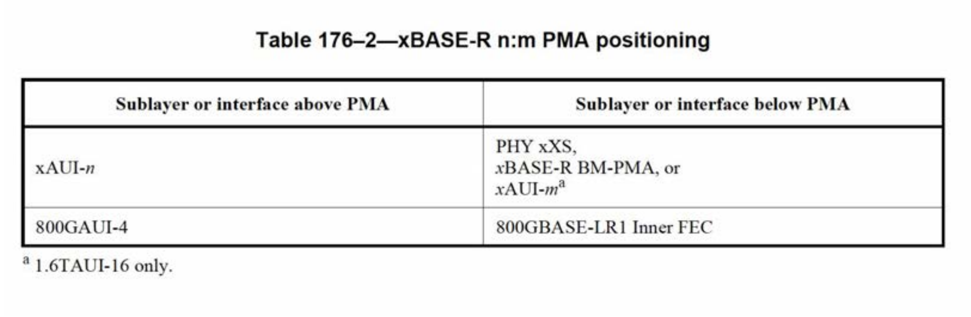


Table 176–2—xBASE-R n:m PMA positioning

Sublayer or interface above PMA	Sublayer or interface below PMA
xAUI-*n*	PHY xXS, *x*BASE-R BM-PMA, or *x*AUI-*m*[a]
800GAUI-4	800GBASE-LR1 Inner FEC

[a] 1.6TAUI-16 only.

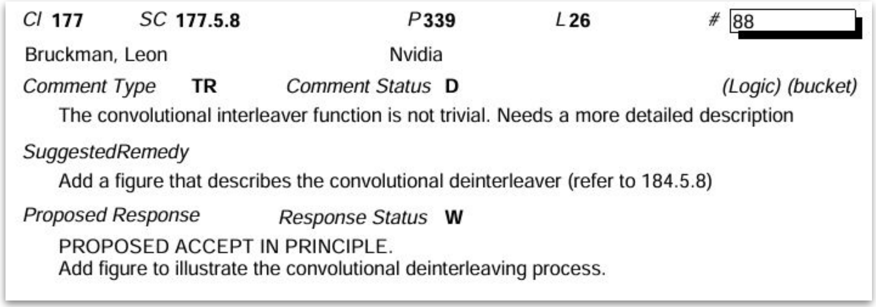
Table 176-2 with a separate row for the 1.6TAUI-16 case

Sublayer or interface above PMA	Sublayer or interface below PMA
xAUI-n	PHY xXS, or xBASE-R BM-PMA
800GAUI-4	800GBASE-LR1 Inner FEC
1.6TAUI-8	1.6TAUI-16

Convolutional Deinterleaver

Comment #88

Comment #88 – convolutional deinterleaver



Cl **177** *SC* **177.5.8** *P* **339** *L* **26** *#* 88

Bruckman, Leon Nvidia

Comment Type **TR** *Comment Status* **D** *(Logic) (bucket)*

The convolutional interleaver function is not trivial. Needs a more detailed description

SuggestedRemedy

Add a figure that describes the convolutional deinterleaver (refer to 184.5.8)

Proposed Response *Response Status* **W**

PROPOSED ACCEPT IN PRINCIPLE.
Add figure to illustrate the convolutional deinterleaving process.

Bucket pull:
The proposed response is "PROPOSED ACCEPT IN PRINCIPLE. Add figure to illustrate the convolutional deinterleaving process". There isn't enough guidance for the figure; Preferably, the commenter (or the editor, if he volunteers) should implement and present the suggested figure, otherwise the CRG does not know what it is supposed to accept. Until then, the proposed response should be REJECT: the SR does not include sufficient detail to implement.

Proposed change:
Insert the following figure in "177.5.8 Convolutional deinterleaver"

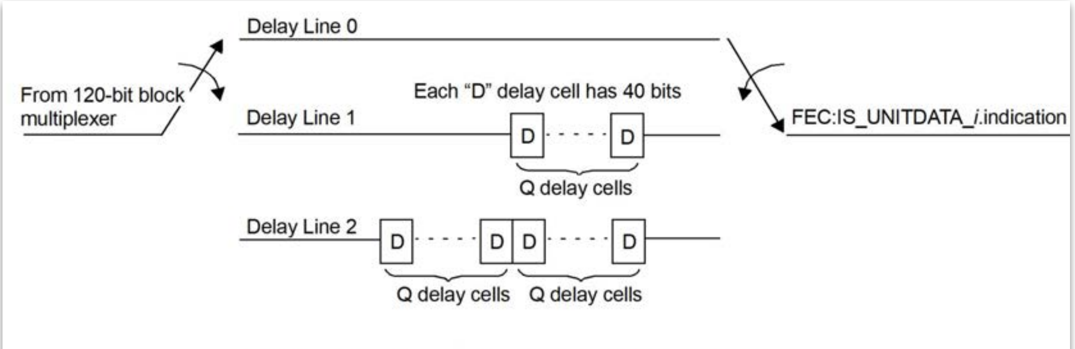
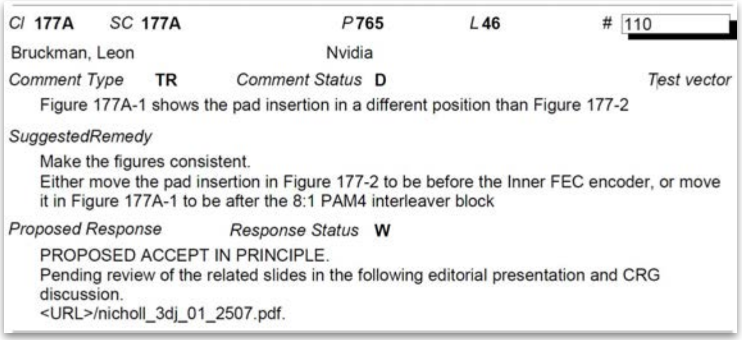

Delay Line 0
From 120-bit block multiplexer
Each "D" delay cell has 40 bits
Delay Line 1
D
D
FEC:IS_UNITDATA_*i*.indication
Q delay cells
Delay Line 2
D
D
D
D
Q delay cells
Q delay cells

Figure 177–10—Convolutional deinterleaver§

Clause 177 test vectors and pad insertion

Comment #110

Comment #110 – pad insertion location


Cl 177A *SC* 177A *P* 765 *L* 46 # 110

Bruckman, Leon Nvidia

Comment Type TR *Comment Status* D *Test vector*

Figure 177A-1 shows the pad insertion in a different position than Figure 177-2

SuggestedRemedy

Make the figures consistent.
Either move the pad insertion in Figure 177-2 to be before the Inner FEC encoder, or move it in Figure 177A-1 to be after the 8:1 PAM4 interleaver block

Proposed Response *Response Status* W

PROPOSED ACCEPT IN PRINCIPLE.
Pending review of the related slides in the following editorial presentation and CRG discussion.
<URL>/nicholl_3dj_01_2507.pdf.

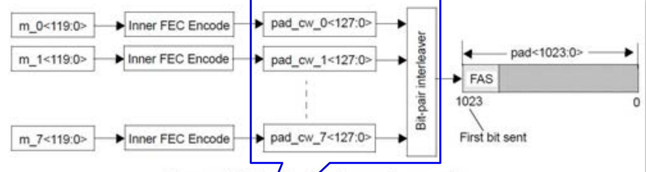


Figure 177–8—Pad codewords mapping

In Figure 177-8 and 177A-1, it seems like "pad insertion" is **before** the "8:1 bit-pair interleaver".

In Figure 177-2, "pad insertion" is **after** "8:1 bit-pair interleaver".

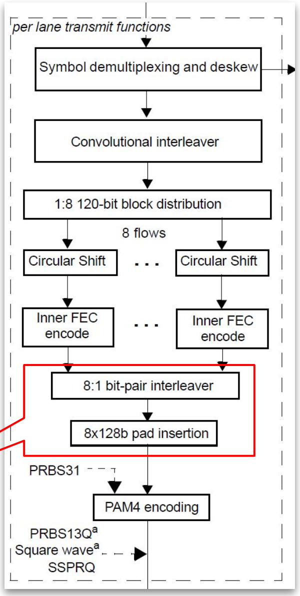


Figure 177-2

The test vectors are generated as shown in Figure 177A–1, where six test points (TP1 to TP6) are defined. TP1 and TP6 are required for interoperability, and TP2 to TP5 are given as a reference for debug. The pad insertion process is equivalent to the process shown in Figure 177–2, where the 120 bits of the payload of each pad codeword is defined in 177.4.7, and all pad payload will go through the Inner FEC encode and 8:1 PAM4 interleaver.

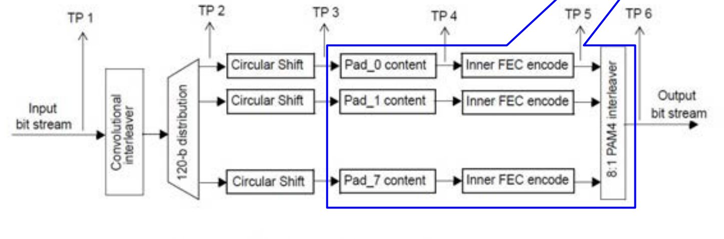


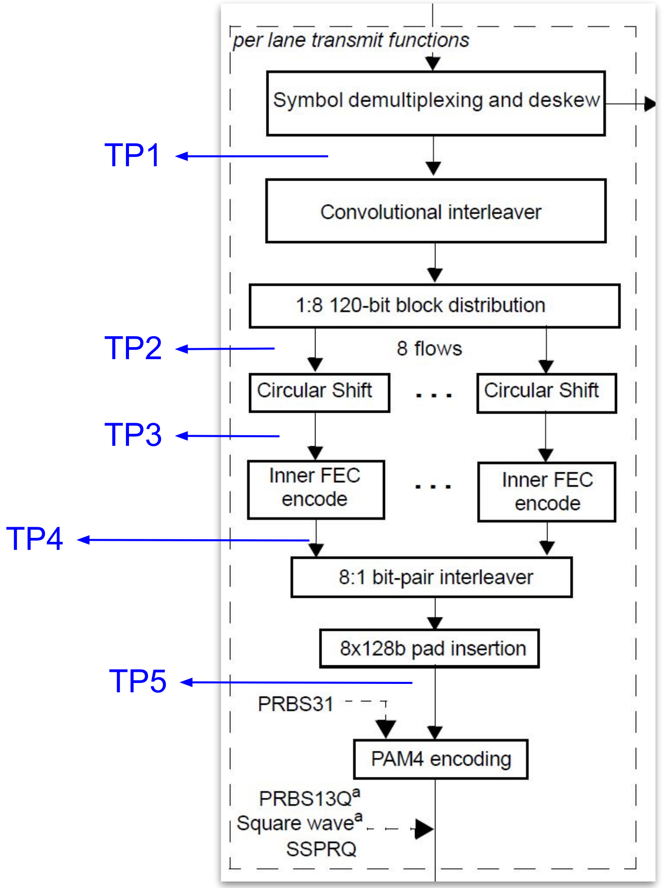
Figure 177A–1—Test points for test vectors

- In Figure 177-2, pad is inserted as a 1024-bit block (8x128b).
- Figure 177-8 describes how is this 1024-bit pad is generated, just like normal payload.
  - 8 "pad codewords" encoded with the same Inner FEC are 8:1 bit-pair interleaved.
  - This allows both transmit and receive to use the exact same processing flow for both, without any special handling.
- Figure 177A-1 is an informative equivalent way of implementing the above process, inserted the pad message bits before Inner FEC encode. The output is exactly the same as Figure 177-2.

Comment #110 – proposed solution

Simple solution:

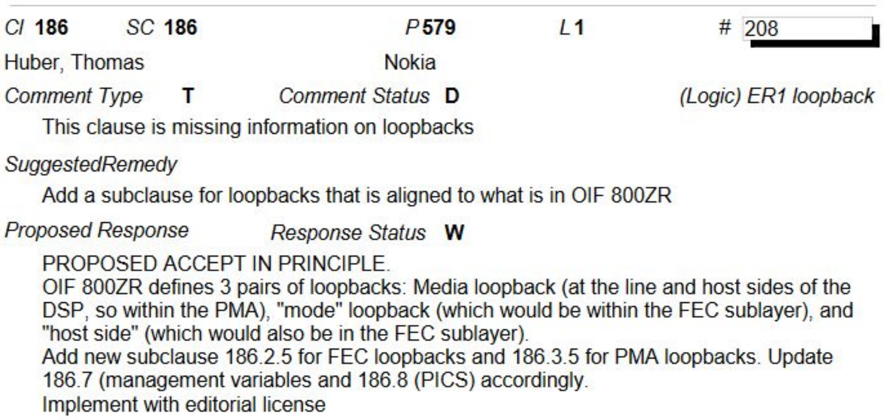
- Replace Figure 177A-1 with the transmit functions in Figure 177-2 shown to the right with the new test points.



ER1 loopbacks

Comment #208

Comment #208 - ER1 loopbacks (1)



CI **186** *SC* **186** *P* **579** *L* **1** # 208

Huber, Thomas Nokia

Comment Type **T** *Comment Status* **D** *(Logic) ER1 loopback*

This clause is missing information on loopbacks

SuggestedRemedy

Add a subclause for loopbacks that is aligned to what is in OIF 800ZR

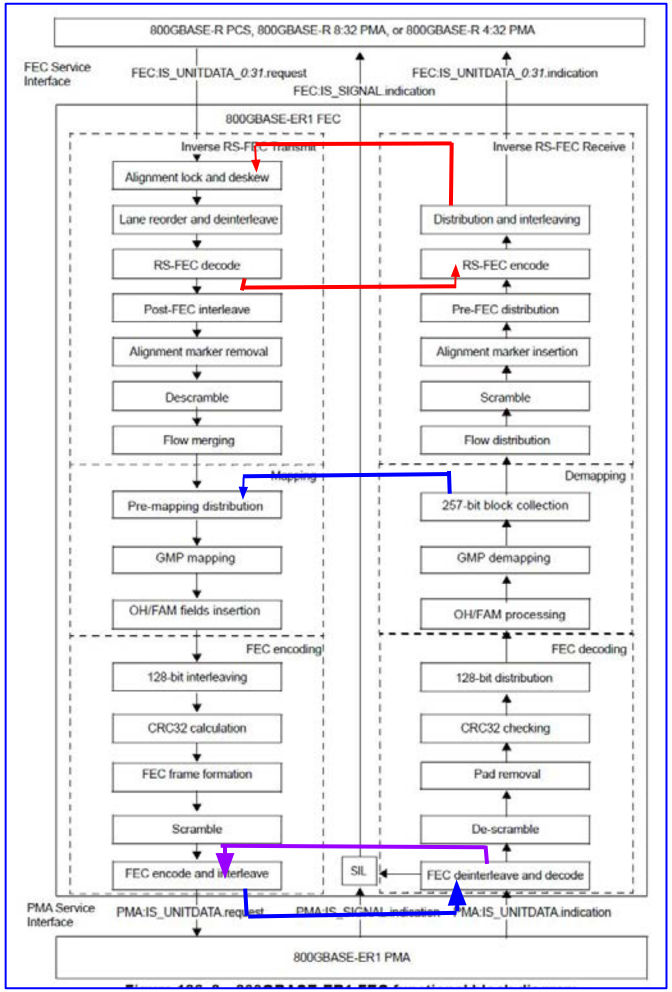
Proposed Response *Response Status* **W**

PROPOSED ACCEPT IN PRINCIPLE.
OIF 800ZR defines 3 pairs of loopbacks: Media loopback (at the line and host sides of the DSP, so within the PMA), "mode" loopback (which would be within the FEC sublayer), and "host side" (which would also be in the FEC sublayer).
Add new subclause 186.2.5 for FEC loopbacks and 186.3.5 for PMA loopbacks. Update 186.7 (management variables and 186.8 (PICS) accordingly.
Implement with editorial license

The figure shows the FEC sublayer

Red lines indicate 'host side' loopbacks
Blue lines indicate 'modem' loopbacks
Purple line indicates receive direction 'media' loopback

Insert new subclause 186.2.5 to describe these loopbacks, add new management variables to control these loopbacks (one to indicate whether each loopback is supported, one to indicate whether it is activated), and update PICS (more detail on next slide)



Comment #208 - ER1 loopbacks (2)

The figure shows the PMA sublayer

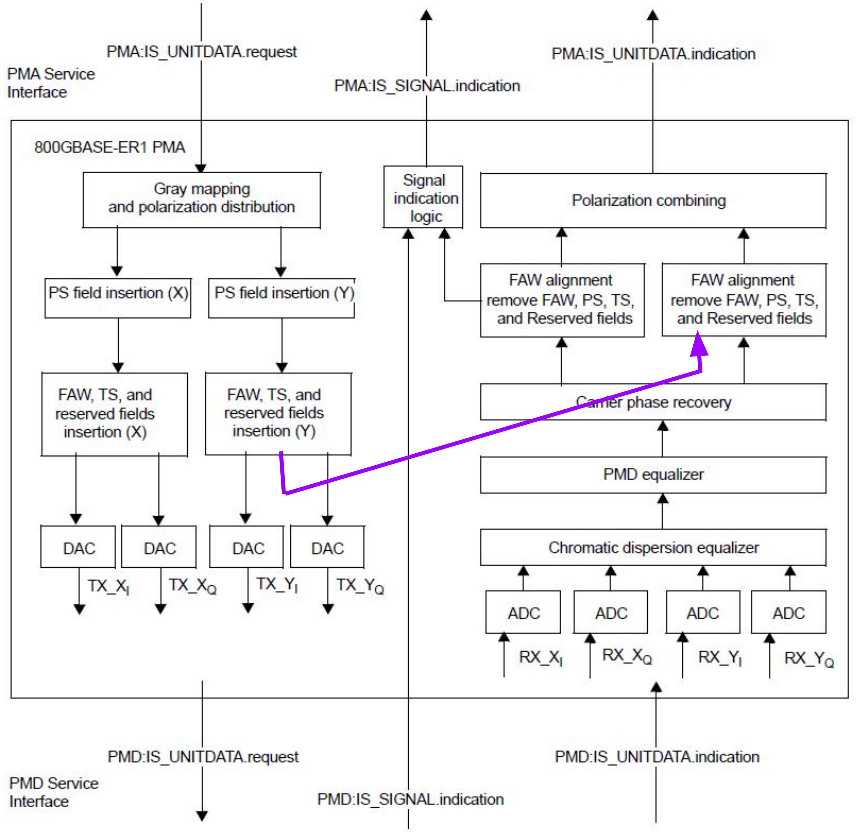
Purple line indicates transmit direction 'media' loopback (only one polarity shown to avoid clutter)

Insert new subclause 186.3.5 to describe this loopback, add new management variables to control this loopback, and update PICS

WRT the PICS:
OIF requires at least one of the following pairs to be supported:

- Modem TX and modem RX
- Modem TX and host RX
- Media TX and modem RX
- Host TX and host RX

The first, second, and fourth options are entirely in the FEC sublayer, so easy to describe in the PICS for the FEC sublayer as a set of 3 options, at least one of which must be done. The third option is spread across the FEC and PMA sublayers, which is awkward, so it is proposed to not include it.


PMA:IS_UNITDATA.request
PMA Service Interface
PMA:IS_SIGNAL.indication
PMA:IS_UNITDATA.indication
800GBASE-ER1 PMA
Gray mapping and polarization distribution
Signal indication logic
Polarization combining
PS field insertion (X)
PS field insertion (Y)
FAW alignment remove FAW, PS, TS, and Reserved fields
FAW alignment remove FAW, PS, TS, and Reserved fields
FAW, TS, and reserved fields insertion (X)
FAW, TS, and reserved fields insertion (Y)
Carrier phase recovery
PMD equalizer
DAC
DAC
DAC
DAC
TX_X$_I$
TX_X$_Q$
TX_Y$_I$
TX_Y$_Q$
Chromatic dispersion equalizer
ADC
ADC
ADC
ADC
RX_X$_I$
RX_X$_Q$
RX_Y$_I$
RX_Y$_Q$
PMD:IS_UNITDATA.request
PMD Service Interface
PMD:IS_SIGNAL.indication
PMD:IS_UNITDATA.indication