MTTFPA with stateless PCS decoder

Adee Ran, Cisco

Overview

- 800 Gb/s Ethernet has a PCS defined in Clause 172 of IEEE Std 802.3df-2024
- 1.6 Tb/s Ethernet has a PCS defined in Clause 175 of P802.3dj
- These PCSs have common elements, including
  - Two flows based on the clause 119 PCS , each of which includes two FEC decoders
  - Separate scrambling on each flow, with a self-synchronizing descrambler (defined in clause 49)
  - Stateless block decoder, defined in clause 172 as an alternative to the earlier decoder of clause 119 (originally also from clause 49).
- The combination of the elements above creates a potential for unmarked errors when the FEC decoder encounters an uncorrectable codeword, which can cause false packet acceptance (FPA)
  - This was not recognized during development of 802.3df.
  - The probability of this scenario is low, but not enough to be negligible.
- This presentation explains the mechanism for FPA, analyzes the probability, and suggests a possible change, in preparation for a future comment.

Data path and error marking in the 800GBASE-R and 1.6TBASE-R PCSs

800GBASE-R PCS

Contains two flows of 400GBASE-R, each with its own scrambler

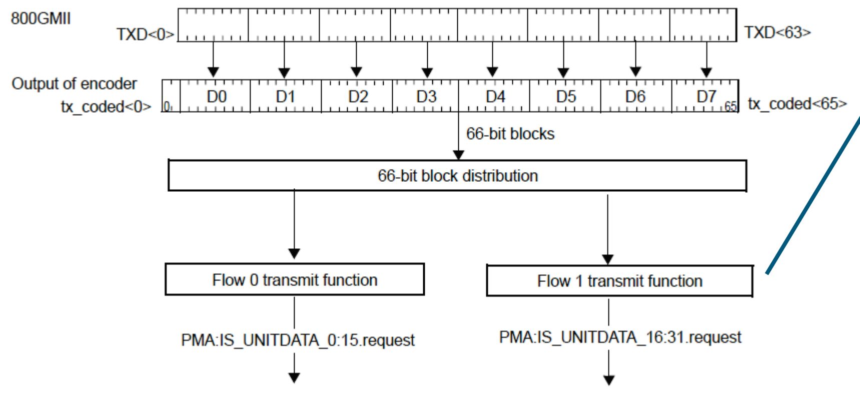


Figure 172–4—800GBASE-R PCS transmit bit ordering and distribution

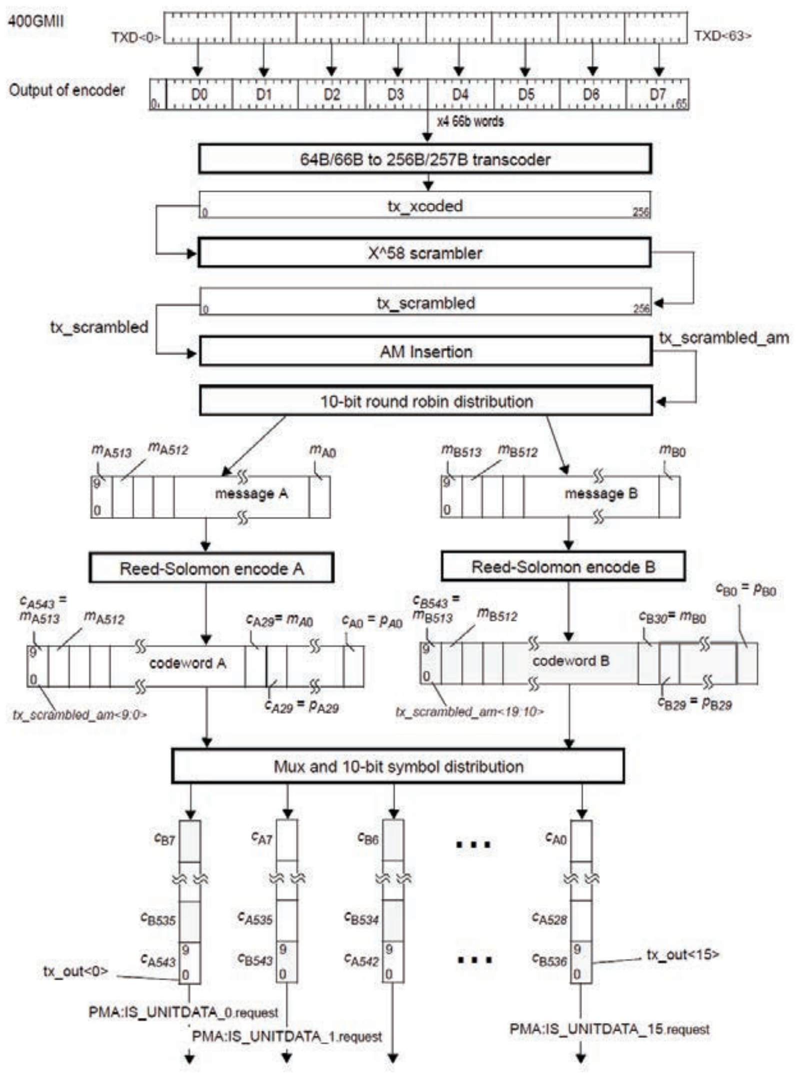
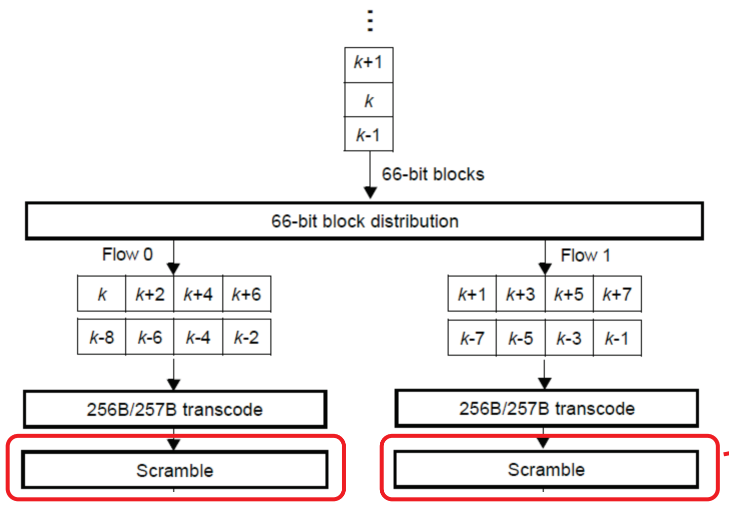


Figure 119–11—400GBASE-R Transmit bit ordering and distribution



Even and odd blocks go through separate (de)scramblers

Top part of Figure 172–3—800GBASE-R PCS alignment marker insertion

1.6TBASE-R PCS

Contains two flows of 800G, each with its own (de)scrambler.
The block distribution before the scramblers is different from that of 800GBASE-R.

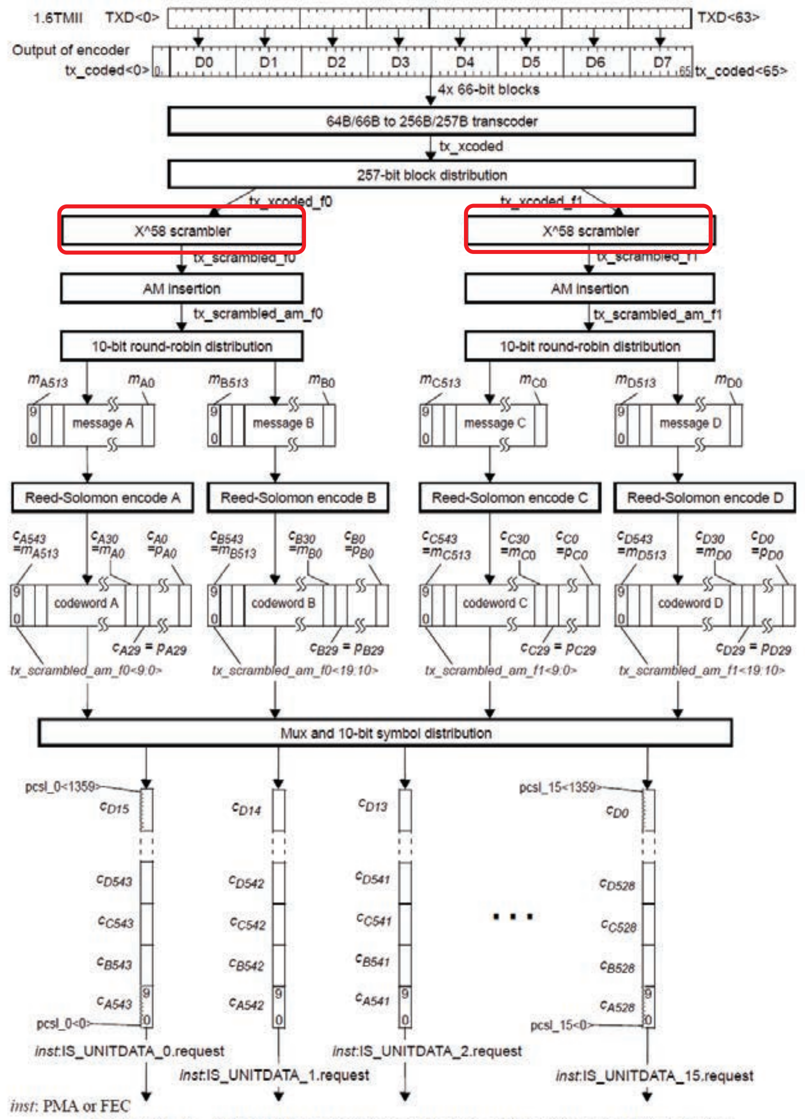


Figure 175–7—1.6TBASE-R PCS transmit data distribution and bit ordering

Block distribution – 800GBASE-R

The tables below are the result of Figure 172–3, Figure 172–4, and Figure 119–11 shown on slide 4

257b block #	Scrambler flow #	Codeword mapping	66b block sequential numbers			
0	0	A0, B0	0	2	4	6
1	1	C0, D0	1	3	5	7
2	0	A0, B0	8	10	12	14
3	1	C0, D0	9	11	13	15
…	…	…	…	…	…	…
78	0	A0, B0	312	314	316	318
79	1	C0, D0	313	315	317	319
80	0	A1, B1	320	322	324	326
81	1	C1, D1	321	323	325	327

66b/64b stateless decoder
Block sequential numbers

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
…	…	…	…
312	313	314	315
316	317	318	319
320	321	322	323
324	325	326	327

The PCS re-orders the blocks from the 257b reverse transcoding to create the order shown on the right.

Block distribution – 1.6TBASE-R

The tables below are the result of Figure 175–7 shown on slide 5

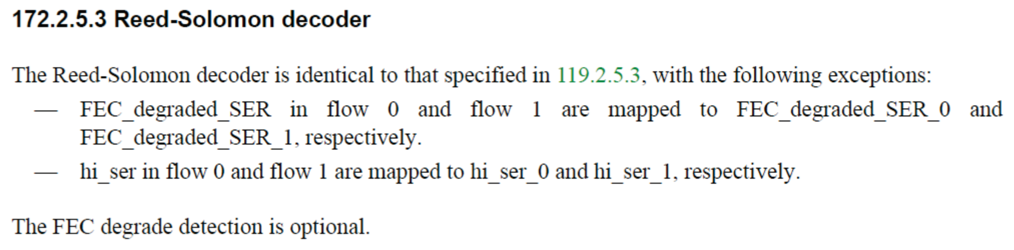
257b block #	Scrambler flow #	Codeword mapping	66b block sequential numbers			
0	0	A0, B0	0	1	2	3
1	1	C0, D0	4	5	6	7
2	0	A0, B0	8	9	10	11
3	1	C0, D0	12	13	14	15
…	…	…	…	…	…	…
78	0	A0, B0	312	313	314	315
79	1	C0, D0	316	317	318	319
80	0	A1, B1	320	321	322	323
81	1	C1, D1	324	325	326	327

66b/64b stateless decoder
Block sequential numbers

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
…	…	…	…
312	313	314	315
316	317	318	319
320	321	322	323
324	325	326	327

The PCS re-orders the blocks from the 257b reverse transcoding to create the order shown on the right.

Error marking in 800GBASE-R PCS

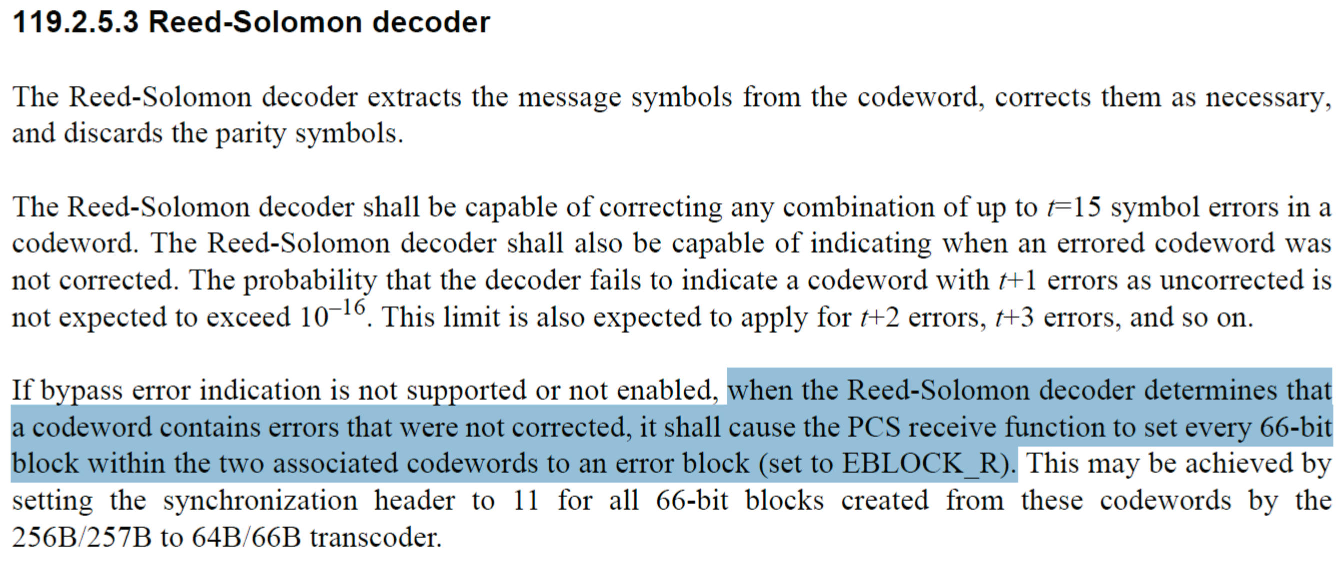


172.2.5.3 Reed-Solomon decoder

The Reed-Solomon decoder is identical to that specified in 119.2.5.3, with the following exceptions:

- FEC_degraded_SER in flow 0 and flow 1 are mapped to FEC_degraded_SER_0 and FEC_degraded_SER_1, respectively.
- hi_ser in flow 0 and flow 1 are mapped to hi_ser_0 and hi_ser_1, respectively.

The FEC degrade detection is optional.



119.2.5.3 Reed-Solomon decoder

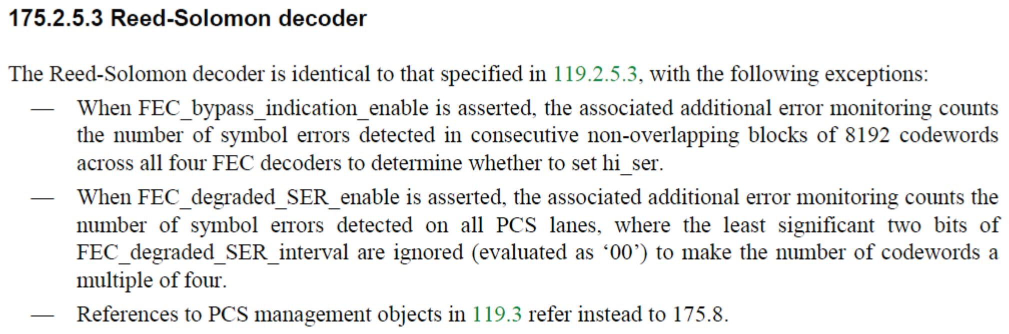
The Reed-Solomon decoder extracts the message symbols from the codeword, corrects them as necessary, and discards the parity symbols.

The Reed-Solomon decoder shall be capable of correcting any combination of up to t=15 symbol errors in a codeword. The Reed-Solomon decoder shall also be capable of indicating when an errored codeword was not corrected. The probability that the decoder fails to indicate a codeword with t+1 errors as uncorrected is not expected to exceed 10^{-16}. This limit is also expected to apply for t+2 errors, t+3 errors, and so on.

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.

This marking occurs separately in each of the two 400G flows (A/B and C/D)

Error marking in 1.6TBASE-R PCS



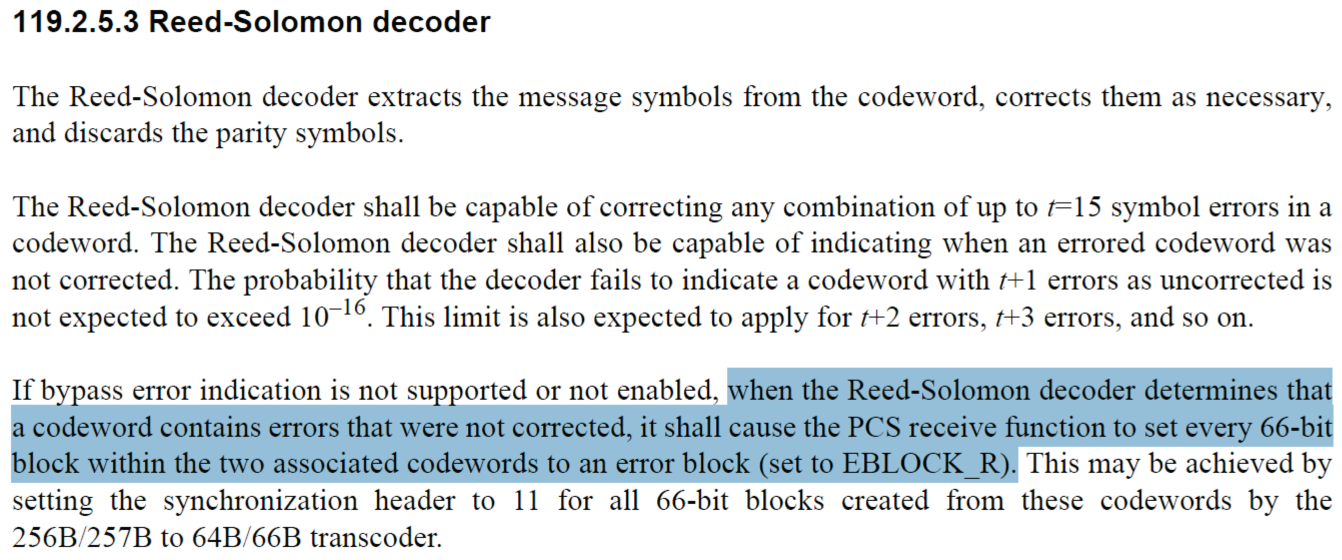
175.2.5.3 Reed-Solomon decoder

The Reed-Solomon decoder is identical to that specified in 119.2.5.3, with the following exceptions:

— When FEC_bypass_indication_enable is asserted, the associated additional error monitoring counts the number of symbol errors detected in consecutive non-overlapping blocks of 8192 codewords across all four FEC decoders to determine whether to set hi_ser.

— When FEC_degraded_SER_enable is asserted, the associated additional error monitoring counts the number of symbol errors detected on all PCS lanes, where the least significant two bits of FEC_degraded_SER_interval are ignored (evaluated as ‘00’) to make the number of codewords a multiple of four.

— References to PCS management objects in 119.3 refer instead to 175.8.



119.2.5.3 Reed-Solomon decoder

The Reed-Solomon decoder extracts the message symbols from the codeword, corrects them as necessary, and discards the parity symbols.

The Reed-Solomon decoder shall be capable of correcting any combination of up to t=15 symbol errors in a codeword. The Reed-Solomon decoder shall also be capable of indicating when an errored codeword was not corrected. The probability that the decoder fails to indicate a codeword with t+1 errors as uncorrected is not expected to exceed 10^{-16}. This limit is also expected to apply for t+2 errors, t+3 errors, and so on.

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.

In 1.6TBASE-R, error marking is the same as in 800GBASE-R

How unmarked errors can occur

Descrambler error multiplication

119.2.5.6 Descrambler

The descrambler shall process rx_scrambled<256:0> to reverse the effect of the scrambler using the polynomial given in Equation (49-1).

$$G(x) = 1 + x^{39} + x^{58} \qquad (49\text{–}1)$$

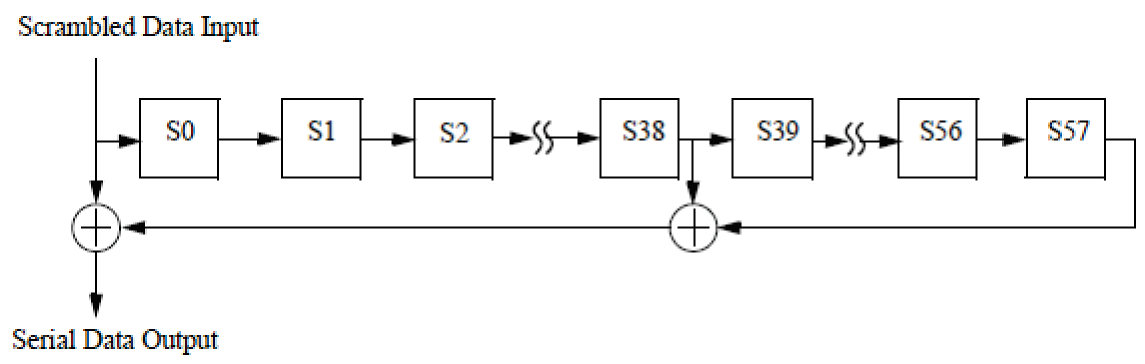
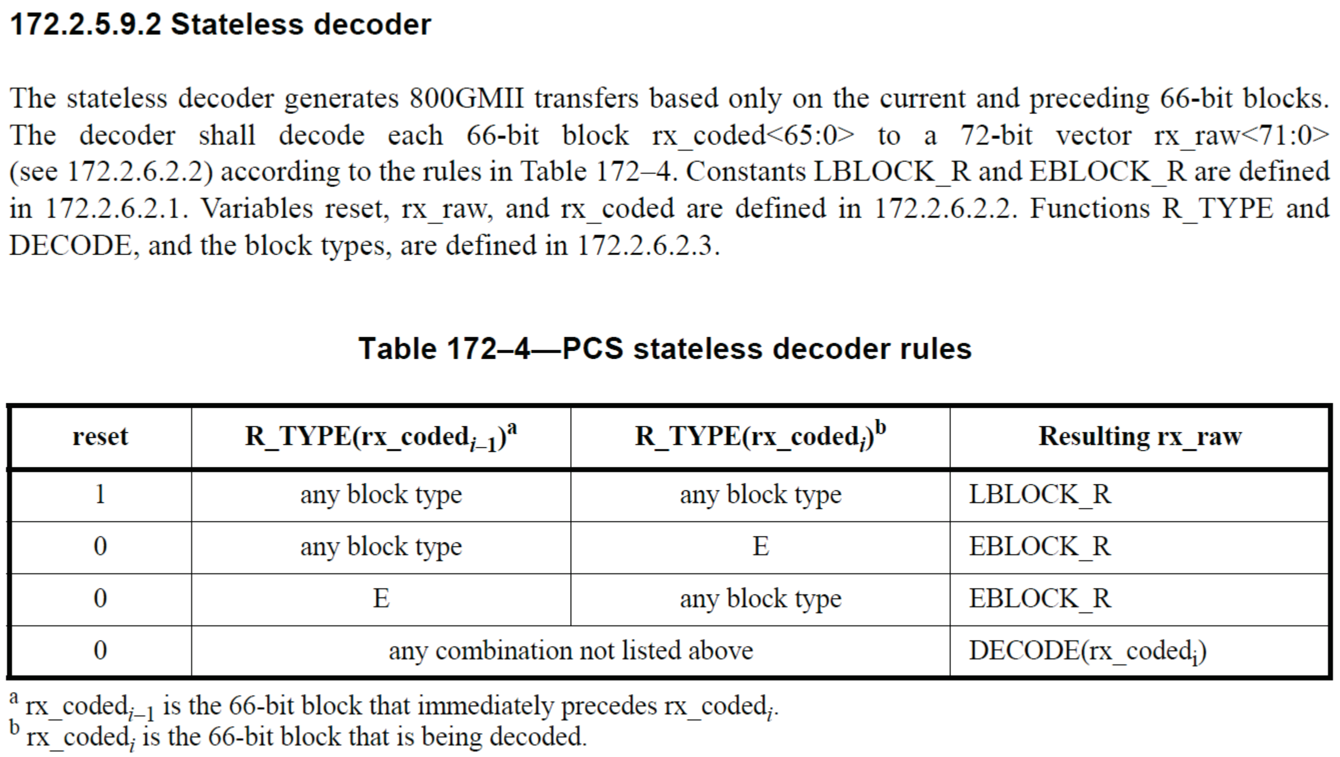


Figure 49–10—Descrambler

Each input bit error creates 2 additional bit errors, at the 39th and 58th subsequent bits

Stateless decoder



172.2.5.9.2 Stateless decoder

The stateless decoder generates 800GMII transfers based only on the current and preceding 66-bit blocks. The decoder shall decode each 66-bit block rx_coded<65:0> to a 72-bit vector rx_raw<71:0> (see 172.2.6.2.2) according to the rules in Table 172–4. Constants LBLOCK_R and EBLOCK_R are defined in 172.2.6.2.1. Variables reset, rx_raw, and rx_coded are defined in 172.2.6.2.2. Functions R_TYPE and DECODE, and the block types, are defined in 172.2.6.2.3.

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.

Unlike the state-diagram decoder used in previous PCS specifications, this decoder does not check the sequence of blocks. It only creates and additional error block after every input block identified as an error.

Unmarked error scenario – 800GBASE-R

Problem: the descrambler multiplication is not covered by the 66b/64b stateless decoder

If codeword D0 is uncorrectable, it will cause the whole content of C0 and D0 to be marked as bad.

257b block #	Scrambler flow #	Codeword mapping	66b block sequential numbers			
0	0	A0, B0	0	2	4	6
1	1	C0, D0	1	3	5	7
2	0	A0, B0	8	10	12	14
3	1	C0, D0	9	11	13	15
…	…	…	…	…	…	…
78	0	A0, B0	312	314	316	318
79	1	C0, D0	313	315	317	319
80	0	A1, B1	320	322	324	326
81	1	C1, D1	321	323	325	327

66b/64b stateless decoder
Block sequential numbers

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
…	…	…	…
312	313	314	315
316	317	318	319
320	321	322	323
324	325	326	327

🟧 Original error
🟨 Stateless decoder error extension

If D0 had bit errors in the last 57 bits, the error multiplication of the descrambler will cause an unmarked bit error in the subsequent 257b block (bits from C1), which is not marked as error by the stateless decoder.
Similarly, uncorrectable B0 (in 66B block 318) can cause errors in A1 (in 66B block 320), which is not marked as an error by the stateless decoder.
However, uncorrectable A0 and C0 can't cause the same effect in B1 and D1.

Unmarked error scenario – 1.6TBASE-R

Problem: the descrambler multiplication is not covered by the 66b/64b stateless decoder

If codeword D0 is uncorrectable, it will cause the whole content of C0 and D0 to be marked as bad.

257b block #	Scrambler flow #	Codeword mapping	66b block sequential numbers			
0	0	A0, B0	0	1	2	3
1	1	C0, D0	4	5	6	7
2	0	A0, B0	8	9	10	11
3	1	C0, D0	12	13	14	15
...	...	...	...	...	...	...
78	0	A0, B0	312	313	314	315
79	1	C0, D0	316	317	318	319
80	0	A1, B1	320	321	322	323
81	1	C1, D1	324	325	326	327

66b/64b stateless decoder
Block sequential numbers

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
...	...	...	...
312	313	314	315
316	317	318	319
320	321	322	323
324	325	326	327

🟧 Original error
🟨 Stateless decoder error extension

If D0 had bit errors in the last 58 bits, the error multiplication of the descrambler will cause an unmarked bit error in the subsequent 257b block (bits from C1), which is not marked as error by the stateless decoder. In this case, a similar effect can happen in all codewords.

Effects of unmarked errors

Possible hazards of errors after FEC decoding

- If the descrambler-generated bit errors fall within packet data – it will be captured by the MAC
  - The scrambler can generate 1 or 2 bit errors following the original one
  - CRC32 will detect any 2-bit error in a packet
  - This may be considered “unclean” but there is no risk of data corruption (assuming the Ethernet MAC is used)
- But if bit errors fall within control blocks, packets can be merged or truncated, and data can potentially be corrupted
  - Specifically, the 4-bit hamming distance encoding of the “Block Type Field” in the first control block is lost in the transcoding
- Also, an error in the first bit of the 257b block can turn a data block into a control block
  - The state-diagram decoder offers sufficient me protection by requiring an IPG before a “start” block
  - For this analysis we assume the use of the stateless decoder defined in 802.3df (172.2.5.9.2), which has looser requirements

Unmarked error hazards

Bit 0 (block type) is the most important:

- If this bit is flipped from 0 to 1, it causes a control/data block to become a data block
  - In normal traffic this could cause packet merging
  - But right after error blocks, this isn't possible
  - No hazard in this case
- If this bit is flipped from 1 to 0, it causes a data block to become a control/data block
  - Then bits 1-4 will be interpreted as the data/control type of the 4 66-bit blocks; 0 means control
  - If the next 4 bits are 0111, data corruption is possible
- If this bit was originally 1 and is unchanged, there is no FPA hazard (the whole block is data, protected by CRC32)
- If this bit was originally 0 and is unchanged, a FPA event seems impossible
  - Strange events like two consecutive "start" blocks can occur, with low probability
  - Won't be analyzed here

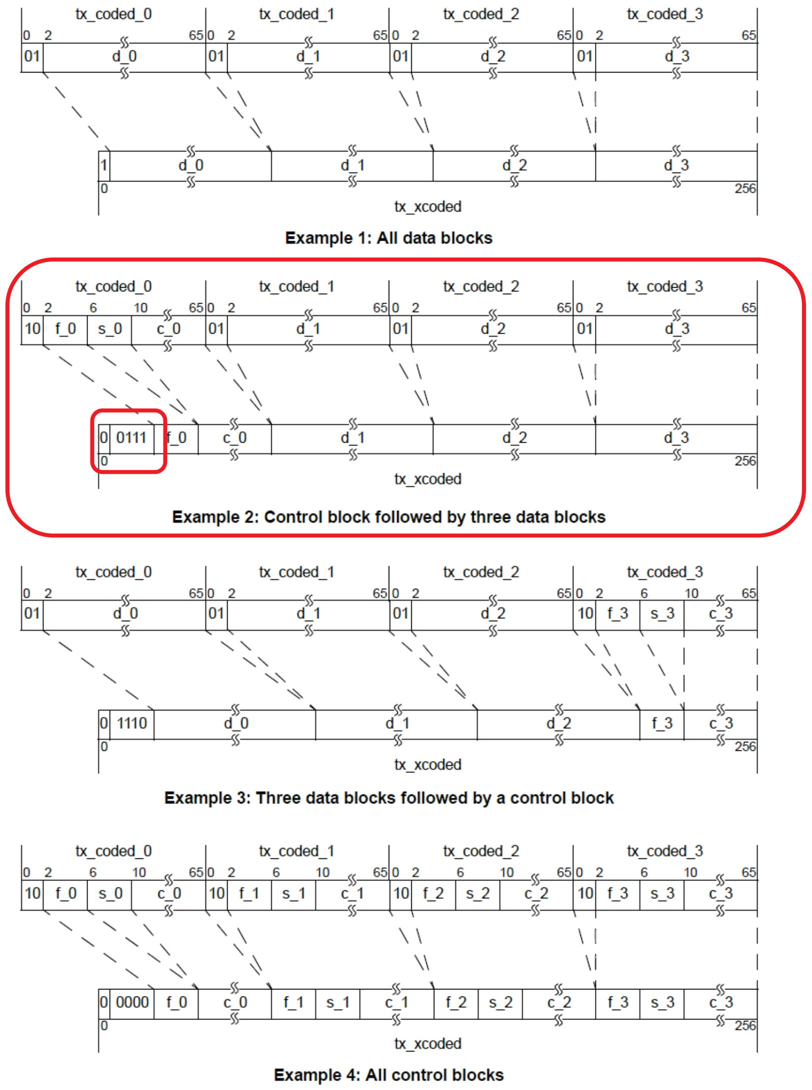

tx_coded_0
tx_coded_1
tx_coded_2
tx_coded_3
01
d_0
d_1
d_2
d_3
tx_xcoded
Example 1: All data blocks
10
f_0
s_0
c_0
0111
Example 2: Control block followed by three data blocks
1110
f_3
s_3
c_3
Example 3: Three data blocks followed by a control block
0000
f_1
s_1
c_1
f_2
s_2
c_2
Example 4: All control blocks

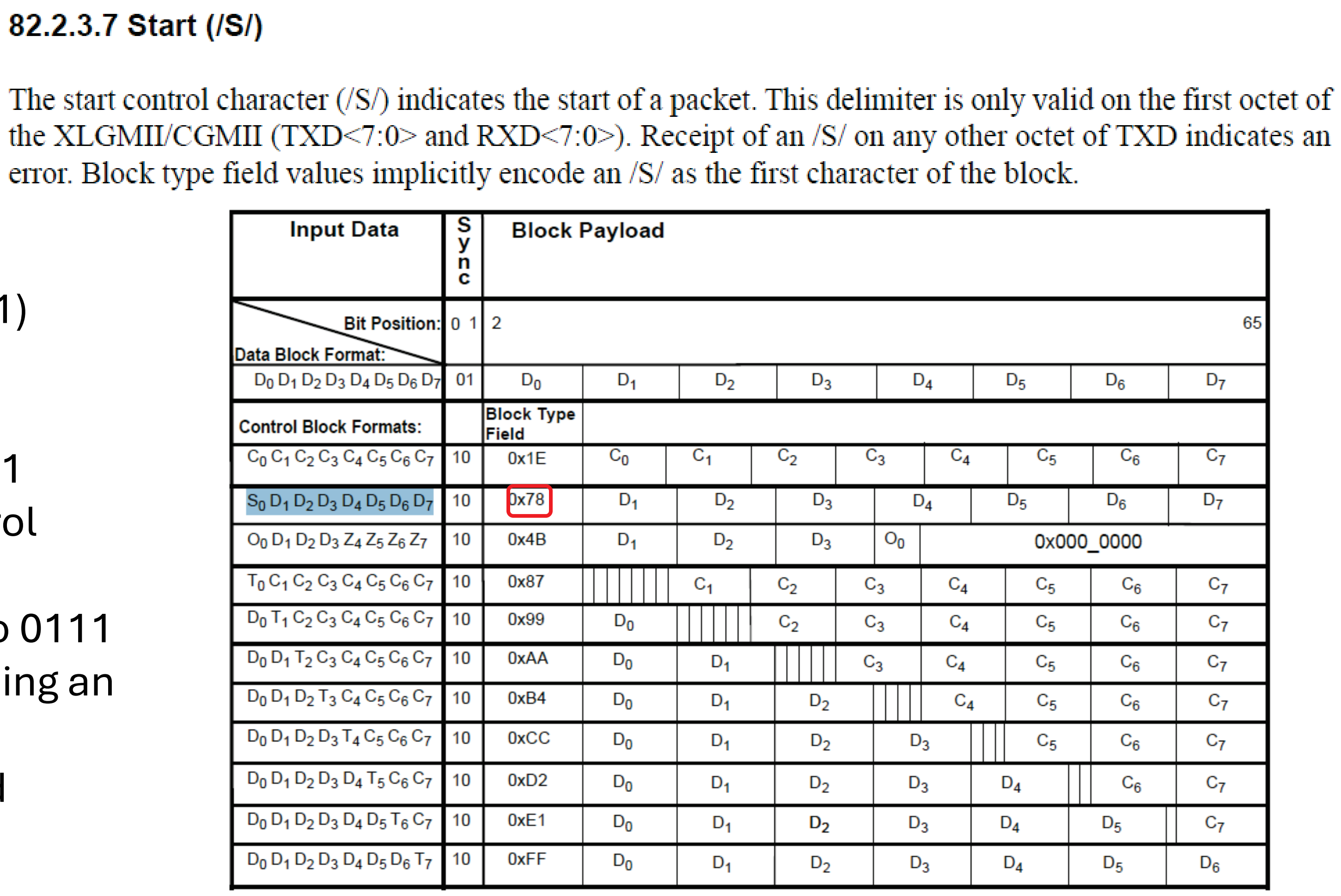
Figure 119–3—Examples of the construction of tx_xcoded

How a bogus start-of-packet can be created

It requires:

- A 257b block consisting of only data (bit 0 is 1)
- Bit 0 of the 257b block is flipped from 1 to 0 (indicating some control/data block)
- Bits 1-4 of the 257b block were originally 0111 (interpreted as if the first 66b block is a control block)
- Bits 5-8 of the 257b block were originally also 0111 (interpreted as if the "f_0" field is 0x7, encoding an /S/ as the first character)

In total, this event requires a specific bit flip and specific 8 bits being 0x77 in the data stream.



82.2.3.7 Start (/S/)

The start control character (/S/) indicates the start of a packet. This delimiter is only valid on the first octet of the XLGMII/CGMII (TXD<7:0> and RXD<7:0>). Receipt of an /S/ on any other octet of TXD indicates an error. Block type field values implicitly encode an /S/ as the first character of the block.

Input Data	Sync	Block Payload								
Bit Position: / Data Block Format:	0 1	2								65
$D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7$	01	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
Control Block Formats:		Block Type Field								
$C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7$	10	0x1E	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
$S_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7$	10	0x78	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
$O_0 D_1 D_2 D_3 Z_4 Z_5 Z_6 Z_7$	10	0x4B	D_1	D_2	D_3	O_0	0x000_0000			
$T_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7$	10	0x87		C_1	C_2	C_3	C_4	C_5	C_6	C_7
$D_0 T_1 C_2 C_3 C_4 C_5 C_6 C_7$	10	0x99	D_0		C_2	C_3	C_4	C_5	C_6	C_7
$D_0 D_1 T_2 C_3 C_4 C_5 C_6 C_7$	10	0xAA	D_0	D_1		C_3	C_4	C_5	C_6	C_7
$D_0 D_1 D_2 T_3 C_4 C_5 C_6 C_7$	10	0xB4	D_0	D_1	D_2		C_4	C_5	C_6	C_7
$D_0 D_1 D_2 D_3 T_4 C_5 C_6 C_7$	10	0xCC	D_0	D_1	D_2	D_3		C_5	C_6	C_7
$D_0 D_1 D_2 D_3 D_4 T_5 C_6 C_7$	10	0xD2	D_0	D_1	D_2	D_3	D_4		C_6	C_7
$D_0 D_1 D_2 D_3 D_4 D_5 T_6 C_7$	10	0xE1	D_0	D_1	D_2	D_3	D_4	D_5		C_7
$D_0 D_1 D_2 D_3 D_4 D_5 D_6 T_7$	10	0xFF	D_0	D_1	D_2	D_3	D_4	D_5	D_6	

Figure 82–5—64B/66B block formats

The effect of a bogus start-of-packet

- With a stateless decoder, a bogus start-of-packet in the middle of the data stream creates a “front-truncated” packet (the beginning is lost).
- The CRC32 does not have a guaranteed performance for a truncated packet; it has a probability of $2^{-32} \approx$ 2e-10 to match.
- If the CRC matches, the MAC falsely accepts a corrupted packet (FPA).
- There may be other safety nets (addresses, EtherType, etc. are usually checked), but we will only look at CRC, as typically done in MTTFPA calculations.

What is the probability of a codeword error leading to an FPA event?

- The codeword error ratio required by Ethernet is <1.45e-11 (174A.5)
    - This does not include xMII extenders which can slightly increase the total ratio
- Given an uncorrectable codeword, there are at least 16 bit errors in the 5140-bit payload; Consider 2 extreme scenarios for the number of bit errors N:
    A. N=16 (minimum, best case)
    B. N=2570 (major burst with BER=0.5, worst case)
- Assume the location of the errors within the codeword is distributed uniformly
- With descrambler multiplication, there are 2 specific locations where errors can cause a flip of bit 0 of the next codeword
    - The probability of an error happening in one location is 16/5140 ≈ 3e-3 (best case) or 0.5 (worst case)
    - The probability of an error in either location is ~6e-3 (best case) or 0.75 (worst case)
- The probability that bit 0 of a codeword was 1 (all-data block) depends on the traffic pattern
    - With large packets and high link utilization, it is close to 1, so assume it is 1
- The probability that bits 1-8 are 0x77, assuming random data, is 1/256 ≈ 4e-3
- The probability of CRC32 escape is ~2e-10
- Thus, for 1.6GBASE-R:
    - In scenario A, an FPA will happen in at most 1.45e-11*6e-3*4e-3*2e-10 of the codewords; calculation yields 8e-26
    - In scenario A, an FPA will happen in at most 1.45e-11*0.75*4e-3*2e-10 of the codewords; calculation yields 1e-23
- In 800G the problem can only occur with uncorrectable codewords B or D, so there is an additional factor of 0.5

Worst case MTTFPA

Data rate	Codeword time	MTTFPA, scenario A	MTTFPA, scenario B
800GBASE-R	6.4 ns	5 billion years	40 million years
1.6TBASE-R	3.2 ns	1.2 billion years	10 million years

Possible improvement

Preferable change: the stateless decoder

- Implementations can use the state-diagram decoder defined in Figure 119–15 of Clause 119. However, this is a more complex decoder and it does not enable advanced features (such as UEC control ordered sets) which will likely be required by future applications.
- Instead, the stateless decoder can be changed to mark more than one additional block as error (increased “block memory”).
  - For 800GBASE-R, extending the error marking to the second block mitigates the MTTFPA issue.
  - For 1.6TBASE-R, the marking needs to be extended to the fifth block.
  - The effect is shown in the subsequent slides.
- With this change, there is still a possibility for corruption of 1 or 2 bits in the first packet after the bad set of codewords, which would always be caught by the CRC32.
  - This is not considered a problem.

Unmarked error scenario – 800GBASE-R

The stateless decoder should mark two subsequent blocks as errors instead of one

If codeword D0 is uncorrectable, it will cause the whole content of C0 and D0 to be marked as bad.

257b block #	Scrambler flow #	Codeword mapping	66b block sequential numbers			
0	0	A0, B0	0	2	4	6
1	1	C0, D0	1	3	5	7
2	0	A0, B0	8	10	12	14
3	1	C0, D0	9	11	13	15
…	…	…	…	…	…	…
78	0	A0, B0	312	314	316	318
79	1	C0, D0	313	315	317	319
80	0	A1, B1	320	322	324	326
81	1	C1, D1	321	323	325	327

66b/64b stateless decoder
Block sequential numbers

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
…	…	…	…
312	313	314	315
316	317	318	319
320	321	322	323
324	325	326	327

Original error
Stateless decoder error extension

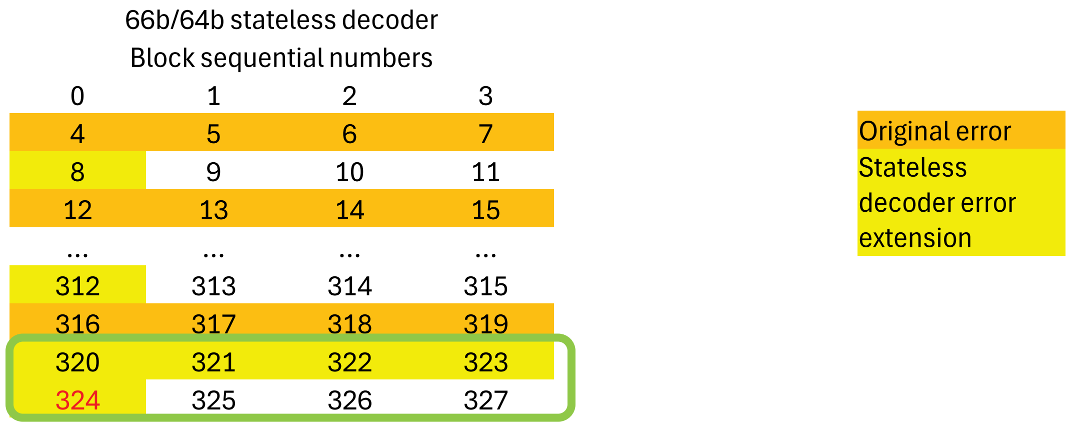
The longer error extension causes the single block that can contain descrambler-multiplied bit errors to be discarded. (the subsequent block is more than 58 bits away, beyond the range of descrambler errors)

Unmarked error scenario – 1.6TBASE-R

The stateless decoder should mark five subsequent blocks as errors instead of one

If codeword D0 is uncorrectable, it will cause the whole content of C0 and D0 to be marked as bad.

257b block #	Scrambler flow #	Codeword mapping	66b block sequential numbers			
0	0	A0, B0	0	1	2	3
1	1	C0, D0	4	5	6	7
2	0	A0, B0	8	9	10	11
3	1	C0, D0	12	13	14	15
…	…	…	…	…	…	…
78	0	A0, B0	312	313	314	315
79	1	C0, D0	316	317	318	319
80	0	A1, B1	320	321	322	323
81	1	C1, D1	324	325	326	327



66b/64b stateless decoder Block sequential numbers			
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
…	…	…	…
312	313	314	315
316	317	318	319
320	321	322	323
324	325	326	327

🟧	Original error
🟨	Stateless decoder error extension

Due to the sequence of blocks, the stateless decoder must have a longer “memory” than in 800GBASE-R to discard the block that can contain descrambler-multiplied bit errors.

That’s all

Questions?