Clause 184 functions (supporting comments 243-247, 249, 250, 252)

Tom Huber, Nokia

Supporters

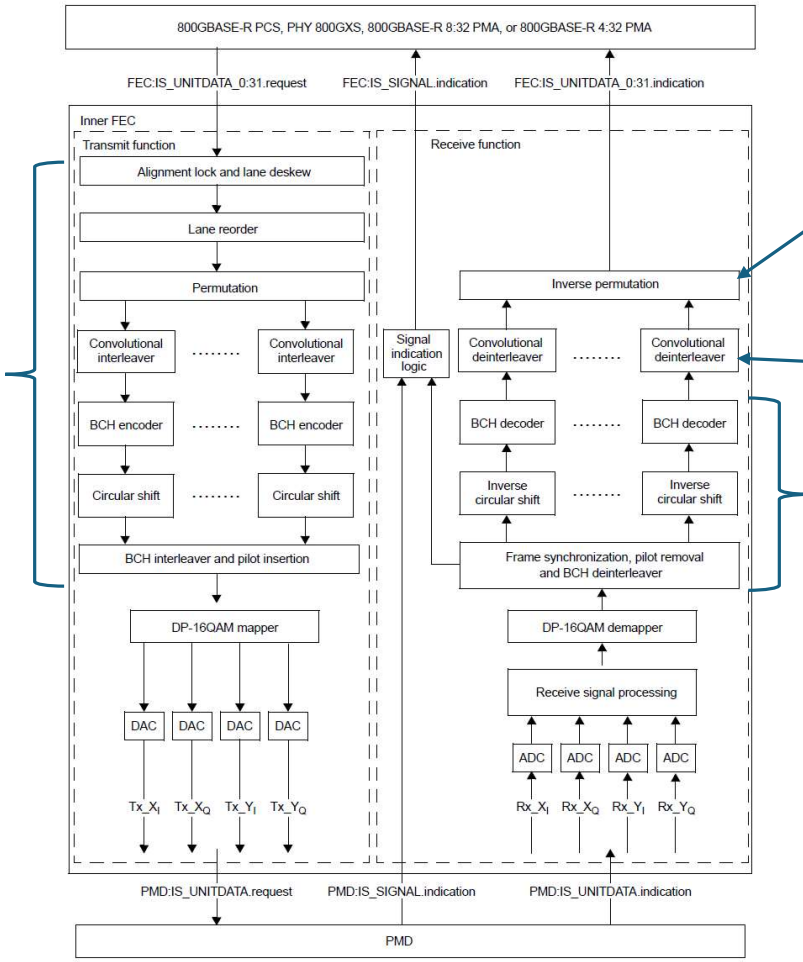
- Steve Gorshe, Microchip
- Matt Brown, Alphawave
- Arnon Loewenthal, Alphawave
- Gary Nicholl, Cisco
- Eugene Opsasnick, Broadcom
- Leon Bruckman, Nvidia

General problem statement

- Clause 184 defines the inner FEC and PMA for 800GBASE-LR1 using pseudocode to describe the processes in 184.4 and 184.5
- Multiple comments were submitted against D1.0 with the intention to simplify the pseudocode; these were rejected on the basis that:
  - The text was technically correct, unambiguous, and came from the baseline slides
  - It would be better to have a more complete proposal before making any changes (in particular wrt how changes to one subclause might impact a subsequent subclause)
- This presentation provides a more complete proposal in support of comments 243-247, 249, 250, and 252

Structure of the inner FEC functions

Each of these functions is specified with pseudocode, with the output of each function providing the input to the next one



This function is specified as the inverse of the transmit process

This function is specified with pseudocode

These functions are specified as the inverse of the transmit process

Clarifying the pseudocode fragments

- The pseudocode fragments in clause 184.4.[1-7] and 184.5.8 are unnecessarily complex, which hinders understanding for readers that don't already understand the processing that is occurring
  - The complexity largely comes from the inclusion of extra iterators for lanes (when functions are performed per-lane) and/or bits within the symbols that are being manipulated
  - In addition, pseudocode for the PCS lane alignment and reordering is introduced, even though these processes are well-specified already with state diagrams
- It would also be beneficial to have English descriptions of what some of these functions are doing at the beginning of each subclause so the reader can better understand the detailed manipulations that are being specified via the pseudocode

Lane alignment and reordering (Subclauses 184.4.1-2, comments 243-245)

- Clauses 184.4.1 and 184.4.2 provide pseudocode that is intended to explain how to lock to the AMs and reorder the PCS lanes; there is no need for this level of specification in clause 184
  - These functions are essentially unchanged since 802.3ba, other than the number of PCS lanes and the values of the AMs
  - The functions are specified using state diagrams in clause 119 (referenced by clause 172)
  - The functions are used not only in the x00GBASE-R PCS (x=1,2,4,8), but also in the x00GXS (x=2,4,8)
- Aligning to AMs inherently aligns to 10-bit symbols because the RS FEC frame is based on 10-bit symbols and is delimited by the AMs

PCS receive example (clause 119, to which clause 172 refers)

- **119.2.5.1 Alignment lock and deskew**

The receive PCS forms *n* separate bit streams by concatenating the bits from each of the *n* PMA:IS_UNITDATA_*i*.indication primitives in the order they are received (where *n* = 8 for a 200GBASER PCS and *n* = 16 for a 400GBASE-R PCS). It obtains lock to the alignment markers as specified by the alignment marker lock state diagram shown in Figure 119–12. Note that alignment marker lock is achieved before FEC codewords are processed and therefore the alignment markers are processed in a high error probability environment.

After alignment marker lock is achieved on each of the *n* lanes (bit streams), all inter-lane Skew is removed as specified by the PCS synchronization state diagram shown in Figure 119–13. The PCS receive function shall support a maximum Skew of 180 ns, and maximum Skew Variation of 4 ns, between PCS lanes.

Not required!

- **119.2.5.2 Lane reorder and de-interleave**

PCS lanes can be received on different lanes of the service interface from which they were originally transmitted. The PCS receive function shall order the PCS lanes according to the PCS lane number. The PCS lane number is defined by the unique portion (UM0 to UM5) of the alignment marker that is mapped to each PCS lane (see 119.2.4.4).

PHY_XS examples (clauses 118 and 171)

- **118.1.2 200GXS/400GXS Sublayer**

The 200GXS, if implemented, shall be identical in function to the 200GBASE-R PCS in Clause 119 with the addition of the functions defined in 118.2. A single device may be configured as either a 200GXS or the 200GBASE-R PCS and may be managed through different optional management registers.

The 400GXS, if implemented, shall be identical in function to the 400GBASE-R PCS in Clause 119 with the addition of the functions defined in 118.2. A single device may be configured as either a 400GXS or the 400GBASE-R PCS and may be managed through different optional management registers.

- **171.3 PHY 800GXS**

The PHY 800GXS shall be identical in function to an 800GBASE-R PCS (see Clause 172) with the following exceptions:
— The PCS is inverted with the transmit function used for the receive direction and vice versa.

Pseudocode from 184.4.1-2

- Alignment lock and deskew (184.4.1)

For each *I*
  For *m* = 0 to 31
    pcsli[*m*, *i*] = FEC:IS_UNITDATA_*m*.request(tx_symbol)
  End for
End for

RS-FEC symbol alignment shall be achieved on the 32 pcsli[*m*, *i*] lanes (*m* = 0 to 31) as follows:
— *j* mod 10 = 0 when pcsli[*m*, *j*] corresponds to the first bit of an RS-FEC symbol

While this correctly describes alignment to RS-FEC symbols (which is what was intended – full deskew is not required), it isn't adding any new information beyond what is in the state diagram in figure 119-12

- Reordering (184.4.2)

The 32 pcsli[*m*] lanes (*m* = 0 to 31) are rearranged to 32 pcsla[*q*] lanes (*q* = 0 to 31) where *q* corresponds to PCSL *q*

This is more complex than what is in clause 119 and isn't really adding any additional information

Proposal for subclauses 184.4.1-2

- The main value of the pseudocode in these clauses is that it ultimately defines the vector pcsla[], which is the input to the next function
  - pcsla[] can be defined directly and more clearly without pseudocode
- The alignment lock and lane reordering functions are clearly defined in 172.2.5.1-2 (which point to 119.2.5.1-2)
- Suggested changes:
  - Remove ‘deskew’ from the title and text of 184.4.1 (since full deskew is not required)
  - Replace the contents of 184.4.1 and 184.4.2 with references to 172.2.5.x (or 119.2.5.x)
  - Add a definition of pcsla[q,i] in 184.4.3 that is based on 10-bit RS FEC symbols:
    - The vector pcsla[q,i] represents the PCS lanes, aligned to 10-bit RS FEC symbols, where the index *q* indicates the PCS lane number (0 to 31) and the index *i* represents the sequence of 10-bit RS FEC symbols.

Lane permutation (Subclause 184.4.3, comments 245-246)

- The 800GBASE-R PCS has two flows, with two FEC encoders each
  - PCS lanes 0-15 come from flow0, lanes 16-31 from flow1
  - Within each PCS lane, the FEC symbols from the two encoders for that flow alternate
    - Lanes in flow0 have symbols A, B, A, B; those in flow1 have C, D, C, D
- The purpose of the lane permutation function is to create a set of 32 output lanes (inner FEC flows) that all have the symbol pattern A, B, C, D
- This is accomplished by swapping the assignment of PCS lanes of flow0 and flow1 to the corresponding sets of 16 output lanes every two symbols
  - In other words, output lane 0 takes two symbols from PCS lane 0, then two from PCS lane 16, output lane 1 takes two symbols from PCS lane 1, then two from PS lane 17, etc.

Pseudocode from 184.4.3

```
For each i
    For q = 0 to 31
        For j = 0 to 9
            permo[q, 10i + j] = pcsla[(q + 16 × floor (i/2)) mod 32, 10i + j]
        End for
    End for
End for
```

Symbol iterator

PCS lane iterator

Bits within a symbol iterator

The permutation function does not change bit positions within the symbols, so it is simpler to describe the operation on symbols (i.e., replace $10i+j$ with i and eliminate the j loop).

Proposal for subclause 184.4.3 (1)

- Introductory text:

  This function rearranges the RS FEC symbols of the PCS lanes to create 32 output inner FEC lanes such that each group of four consecutive symbols on each output lane contains one symbol from each of the four RS FEC encoders in the 800GBASE-R PCS

- Pseudocode

Define pcsla[*q*, *i*] to be the 10-bit symbol in PCS lane *q* at time *i* (after lane alignment and reordering)

Define permo[*q*, *i*] to be the 10-bit symbol in output lane *q* at time *i* at the output of the permutation function

The permutation function is defined by the following pseudocode:

For each *i*

  For each *q* = 0 to 31

    permo[*q* ,*i*] = pcsla[(*q* + 16×floor(*i*/2)) mod 32, *i*]

  End for

End for

Proposal for subclause 184.4.3 (2)

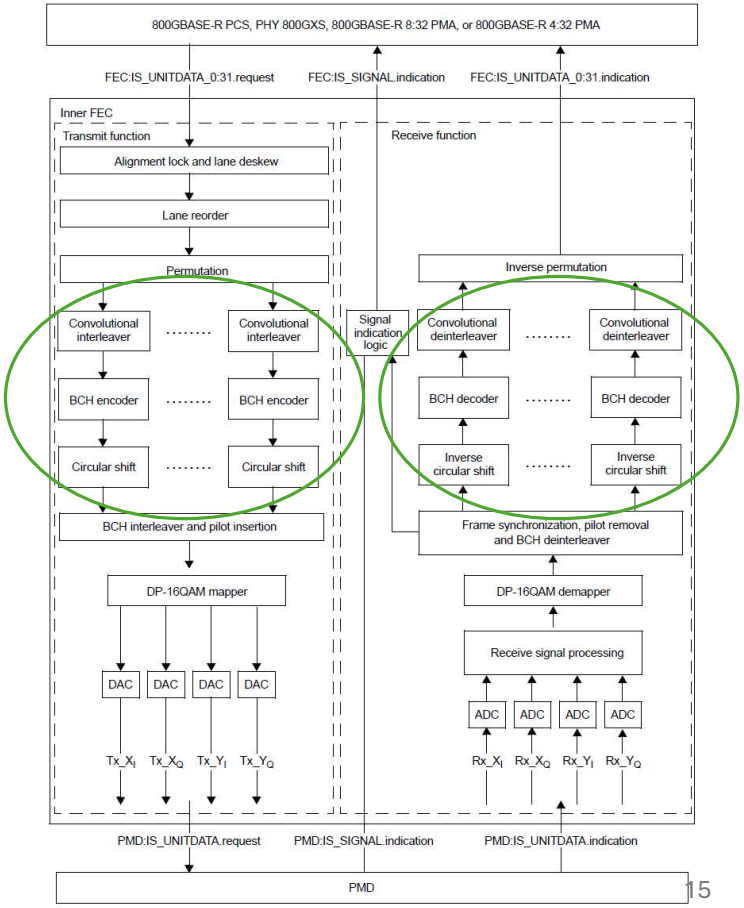
	RS-FEC In			
pcsl	0	1	2	3
Lane 0	A0	B8	A16	B24
Lane 1	B0	A8	B16	A24
Lane 2	A1	B9	A17	B25
Lane 3	B1	A9	B17	A25
Lane 4	A2	B10	A18	B26
Lane 5	B2	A10	B18	A26
Lane 6	A3	B11	A19	B27
Lane 7	B3	A11	B19	A27
Lane 8	A4	B12	A20	B28
Lane 9	B4	A12	B20	A28
Lane 10	A5	B13	A21	B29
Lane 11	B5	A13	B21	A29
Lane 12	A6	B14	A22	B30
Lane 13	B6	A14	B22	A30
Lane 14	A7	B15	A23	B31
Lane 15	B7	A15	B23	A31
Lane 16	C0	D8	C16	D24
Lane 17	D0	C8	D16	C24
Lane 18	C1	D9	C17	D25
Lane 19	D1	C9	D17	C25
Lane 20	C2	D10	C18	D26
Lane 21	D2	C10	D18	C26
Lane 22	C3	D11	C19	D27
Lane 23	D3	C11	D19	C27
Lane 24	C4	D12	C20	D28
Lane 25	D4	C12	D20	C28
Lane 26	C5	D13	C21	D29
Lane 27	D5	C13	D21	C29
Lane 28	C6	D14	C22	D30
Lane 29	D6	C14	D22	C30
Lane 30	C7	D15	C23	D31
Lane 31	D7	C15	D23	C31

	RS-FEC Out			
pcsl	0	1	2	3
Lane 0	A0	B8	C16	D24
Lane 1	B0	A8	D16	C24
Lane 2	A1	B9	C17	D25
Lane 3	B1	A9	D17	C25
Lane 4	A2	B10	C18	D26
Lane 5	B2	A10	D18	C26
Lane 6	A3	B11	C19	D27
Lane 7	B3	A11	D19	C27
Lane 8	A4	B12	C20	D28
Lane 9	B4	A12	D20	C28
Lane 10	A5	B13	C21	D29
Lane 11	B5	A13	D21	C29
Lane 12	A6	B14	C22	D30
Lane 13	B6	A14	D22	C30
Lane 14	A7	B15	C23	D31
Lane 15	B7	A15	D23	C31
Lane 16	C0	D8	A16	B24
Lane 17	D0	C8	B16	A24
Lane 18	C1	D9	A17	B25
Lane 19	D1	C9	B17	A25
Lane 20	C2	D10	A18	B26
Lane 21	D2	C10	B18	A26
Lane 22	C3	D11	A19	B27
Lane 23	D3	C11	B19	A27
Lane 24	C4	D12	A20	B28
Lane 25	D4	C12	B20	A28
Lane 26	C5	D13	A21	B29
Lane 27	D5	C13	B21	A29
Lane 28	C6	D14	A22	B30
Lane 29	D6	C14	B22	A30
Lane 30	C7	D15	A23	B31
Lane 31	D7	C15	B23	A31

- Modify figure 184-3 to more clearly shows the detail of the 40-bit symbols on which the lane permutation is operating and how the function is producing groups of 4 symbols in each lane that come from four different FEC codewords.

Remaining functions are per-lane

- Per figure 184-2, the functions between lane permutation and interleaving into the BCH FEC frame are performed separately on each lane
- Corresponding pseudocode should describe the processing for a single lane, not the set of 32 lanes

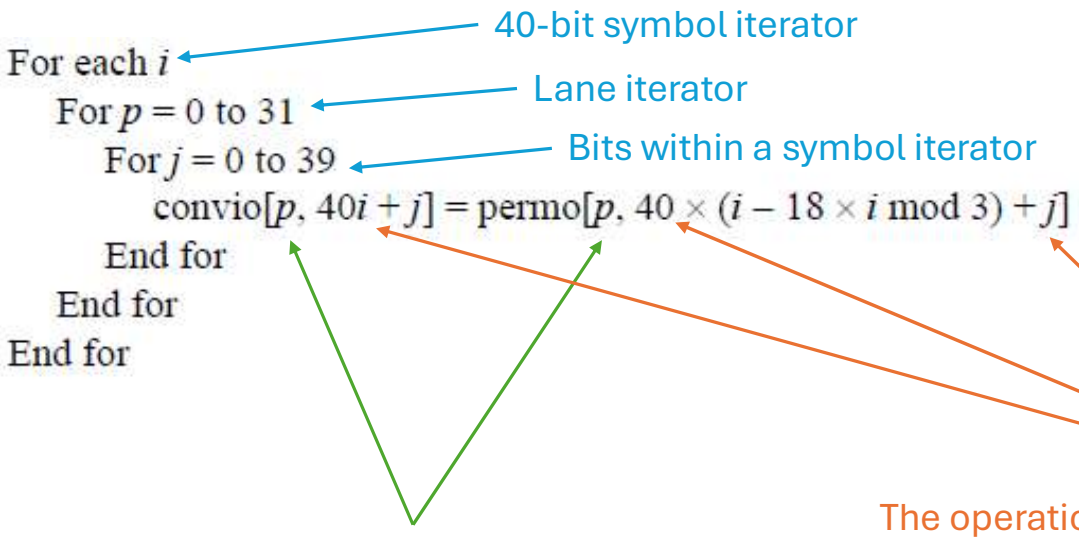


Convolutional interleaver (Subclause 184.4.4, comment 247)

- The purpose of the convolutional interleaver is to rearrange the time order of the RS FEC symbols for each lane such that each BCH FEC symbol (which has a payload of 11 RS FEC symbols) has no more than one RS FEC symbol from any RS FEC codeword
- The (preceding) lane permutation function has created lanes where each group of 4 consecutive RS FEC (10-bit) symbols comes from four separate RS FEC codewords, so the convolutional interleaver operates on 40-bit symbols
- The output of the convolutional interleaver reorders the 40-bit symbols of a lane such that consecutive symbols in the output stream were separated by 17 symbols in the input stream
  - RS FEC codewords are 5440 bits (13.6 40-bit symbols)

Pesudocode from 184.4.4

This function operates independently on each lane, on 40-bit symbols (i.e. groups of four 10-bit RS FEC symbols). It rearranges the symbols such that the BCH FEC codewords (which are 110 bits) contain no more than one symbol from any RS FEC codeword.



The operation does not move symbols between lanes, so it can be specified more simply as operating on an individual lane, in which case the index p and associated for loop can be removed

The operation does not change bit positions within the 40-bit symbols; it simply changes the position of the symbols. The multiplier 40, index j and associated for loop can be removed so the operation is described based on 40-bit symbols.

Proposal for subclause 184.4.4

- Introductory text is fine as written (from the start of the clause through the first paragraph after the figure)
- Replace the text on page 479, starting at the second paragraph below figure 184-4:

The following is performed individually on each of the *q* lanes of permo, operating on 40-bit symbols (consisting of four RS-FEC symbols) *j*:

For each *j*

convio[*j*] = permo[*j* – 18 × *j* mod 3]

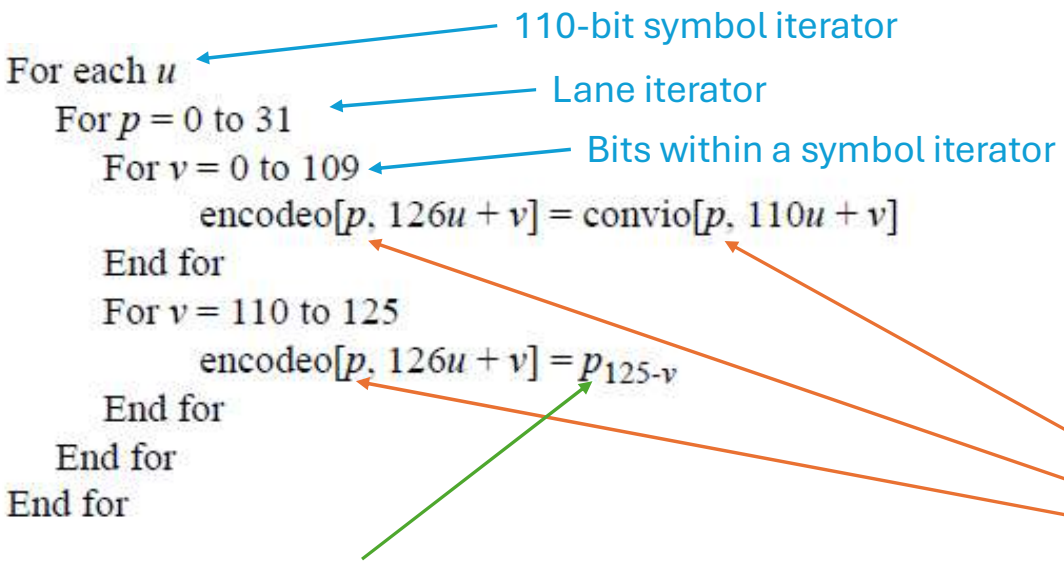
End for

Note: convio[*j*] is undefined when the index to permo is negative.

BCH encoder (Subclause 184.4.5, comment 249)

- The BCH encoder adds the inner FEC code
- It operates on 110-bit symbols (11 RS-FEC symbols) and adds 16 parity bits to create 126-bit symbols
  - This can be described more clearly in pseudocod using the ‘range of bits’ notation rather than a bit-level iterator
- Since the function operates on each lane individually, there is no need to include a per-lane index in the description of the encoder

Pseudocode from 184.4.5



This function operates independently on each lane, on 110-bit symbols (i.e. groups of eleven 10-bit symbols). It computes the 16 parity bits and adds them to the end of the 110 input bits to create the 126-bit BCH codeword.

The operation could be described more simply as appending 16 bits to each group of 110 bits rather than copying 110 bits and then adding 16 bits, like what is done in clause 91.

This *p* is not referring to the lane number iterator, but to the parity bits computed by the FEC.

The operation does not change lane numbers, so the algorithm can be specified more simply as operating on an individual lane and the index *p* and associated for loop can be removed

Proposal for subclause 184.4.5

On page 480:

- Delete the dashed items defining the indexes *q*, *i*, and *j*
- Delete the text describing how *u* and *v* are related to *i* and *j*
- Remove the references to *q* in the description of the message polynomial *m(x)*
- Define parity[15:0] to be the coefficients of the computed parity polynomial
- Replace the pseudocode with this:

For each *u*

encodeo[126*u*:126*u*+109] = convio[110*u*:110*u*+109]

encodeo[126*u*+110:126*u*+125] = parity[15:0]

End for

(i.e., after each 110 bits, add the 16 computed parity bits for those 110 bits)

184.4.6 - Circular shift (Subclause 184.4.6, comment 250)

- The circular shift rotates the bits within the 110 payload bits of the BCH codewords to further improve burst tolerance (leaving the 16 parity bits unchanged)
- The amount of shift depends on the lane number

Pseudocode from 184.4.6

```
For each i
   For p = 0 to 31
      For j = 0 to 109
            circo[p, j] = encodeo[p, ((j – 20p) mod 110)]
      End for
      For j = 110 to 125
            circo[p, j] = encodeo[p, j]
      End for
   End for
End for
```

126-bit symbol iterator

Lane iterator

Bits within a symbol iterator

This function operates independently on each lane, on the first 110 bits of each BCH FEC codeword, reorganizing the bits. The amount of shifting is different for each lane, so the index *p* is needed, but there is no need to include an iterator since the function is applied to each lane

Since bits 110-125 do not change, there is no need to explain that; the second for *j* loop can be removed.

The function operates on each lane or the *p* loop (the value of *p* is still important, however, since the shift does depend on the lane number)

Proposal for subclause 184.4.6

Replace the entire clause with this:

The circular shift function is applied to each lane. It rearranges the 110 payload bits of each BCH FEC codeword to further increase robustness to burst errors. Consider each 126-bit BCH FEC codeword as a vector of bits, encodeo[j], and apply the following process:

For j = 0 to 109

  circo[j] = encodeo[(j – 20q) mod 110]

End for

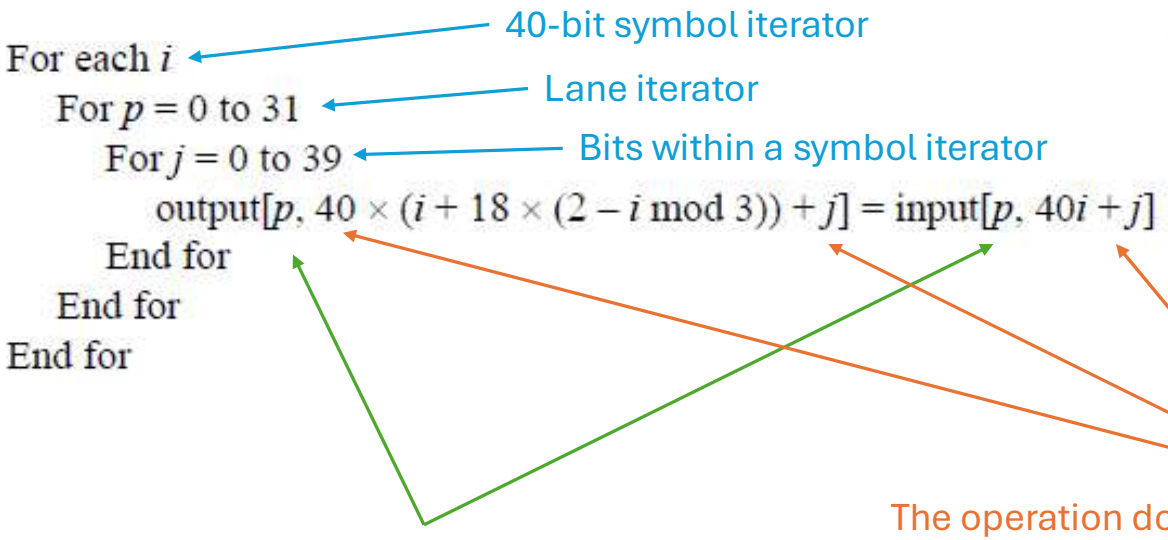
Where q corresponds to the lane number (0 to 31)

Convolutional de-interleaver (Subclause 184.5.8, comment 252)

- The purpose of the convolutional de-interleaver is to undo the manipulation performed by the convolutional interleaver

Pseudocode from 184.5.8

This function operates independently on each lane, on 40-bit symbols (i.e. groups of four 10-bit RS FEC symbols). It restores the original order of 40-bit symbols prior to convolutional interleaving in the transmitter.



The operation does not move symbols between lanes, so it can be specified more simply as operating on an individual lane, in which case the index *p* and associated for loop can be removed

The operation does not change bit positions within the 40-bit symbols; it simply changes the position of the symbols. The multiplier 40, index *j* and associated for loop can be removed so the operation is described based on 40-bit symbols.

Proposal for subclause 184.5.8

- Introductory text is fine as written (up through the first paragraph after the figure)
- Replace the text on page 490 with this:

The following is performed individually on each of the *q* lanes:

Denote the input and output of the convolutional de-interleaver as input[*j*] and output[*j*], where the index *j* identifies 40-bit symbols.

For each *j*

output[$j + 18 \times (2 - j \bmod 3)$] = input[*j*]

End for

Note: output[*j*] is undefined when the index is negative.