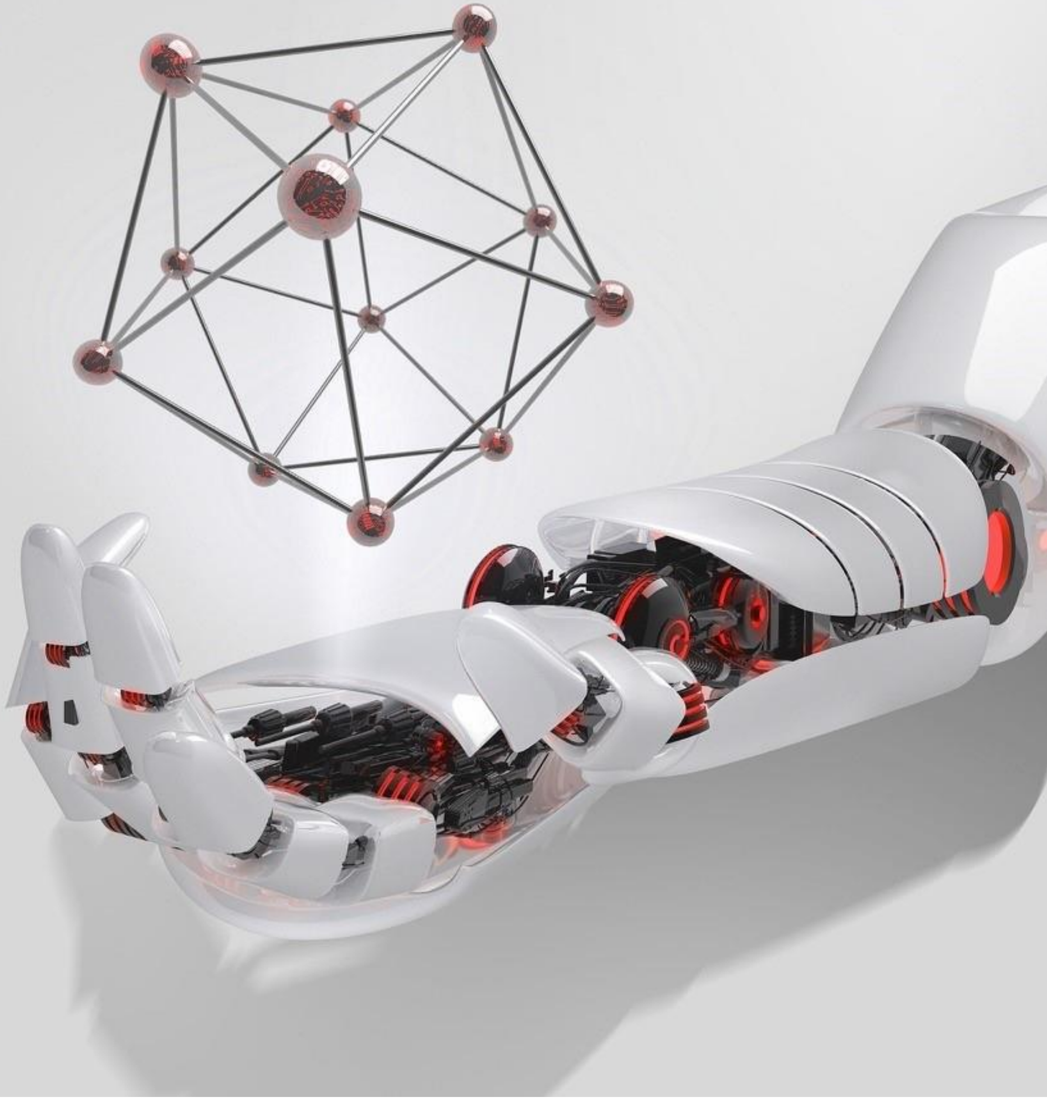
Constructing a BCH/Hamming Inner Code for 200 Gb/s per lane PMDs

Xiang He, Xinyuan Wang, Hao Ren, Nianqi Tang

Huawei Technologies



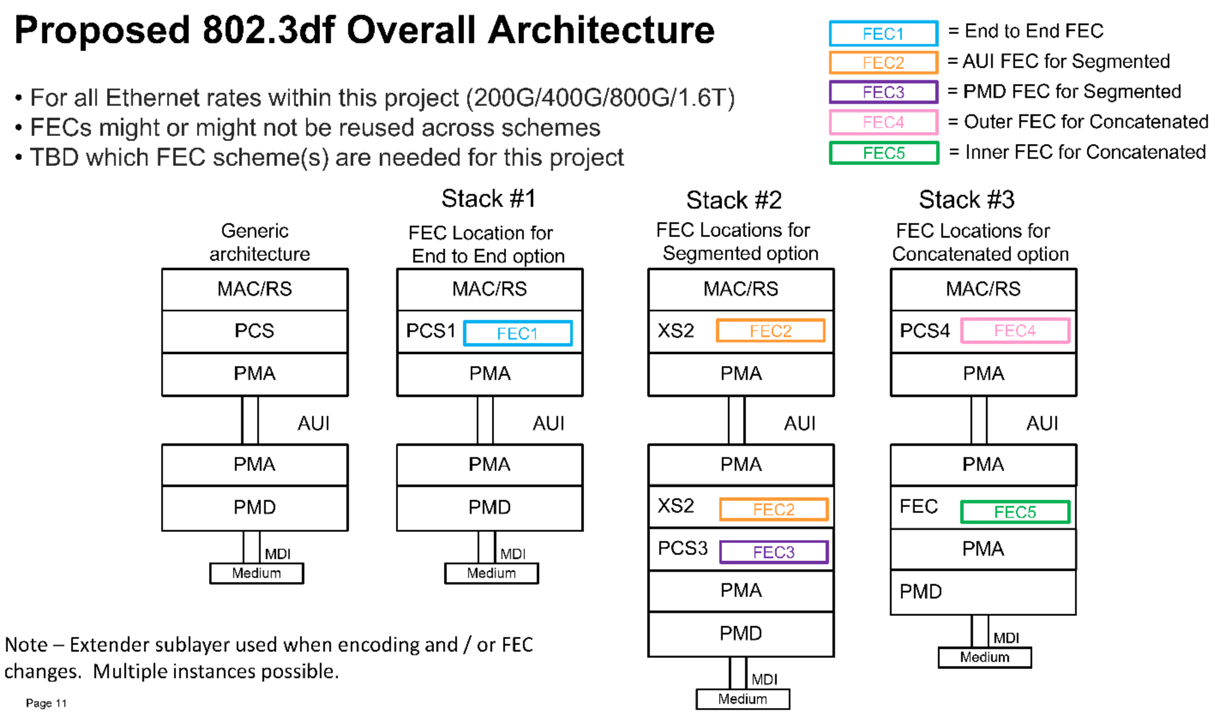
HUAWEI

Background: Previous FEC Related Discussions

- In welch_3df_01b_220602, discussions were focused on feasibility and availability of FEC capability to enable 200 Gb/s per lane with IM-DD PAM4.
  - ~2 – 2.4E-3 is the possible pre-FEC BER range based on data in this group.
- wang_3df_01a_220609 and zhang_3df_01_220609 proposed 800 Gb/s single carrier coherent based on DP-QAM modulation for 800GBASE-LR1.
  - Analysis was based on ~4 – 4.5E-3 pre-FEC BER range.
- For 200 Gb/s per lane AUI, ~1E-5 pre-FEC BER is under discussion in the Task Force.
- Concatenated code with BCH/Hamming as inner code based on soft decision implementation is interested in the Task Force.

Background: Adopted Logic Layer Baseline

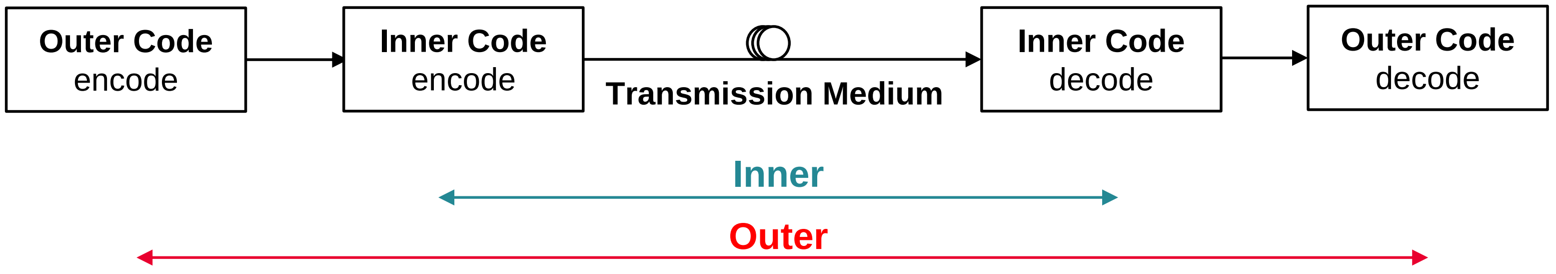
- Concatenated code is one of the FEC schemes adopted in logic architecture baseline in May.



- Motivation: Introduce general methods on how to construct a suitable inner code to support 200 Gb/s per lane PMDs of 800G/1.6TbE, and new 200/400GbE.

General Approach for Concatenated Codes

- Basic concepts:
  - Outer code: the first encoded FEC code. It appears on the “outer” side of the whole link.
  - Inner code: the later encoded FEC code. It appears on the “inner” side of the whole link.

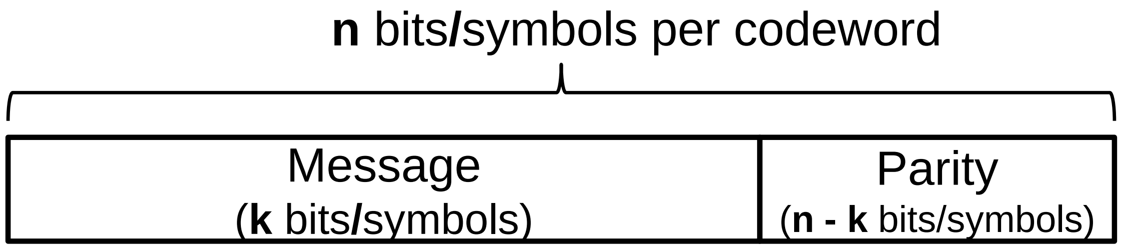


- For 802.3dj, we assume RS(544,514) as the outer code.
  - To construct an inner code working with RS(544,514) code and can be adapted to the PCS/PMA architecture.
  - BCH code with t=1 and Hamming code are under discussion as moderate ASIC implementation cost and reasonable FEC performance as shown in he_3df_01a_220308, bliss_3df_01a_220517, patra_3df_01a_2207, ghiasi_3df_02a_2207, etc.

HUAWEI

Brief Introduction of BCH Code

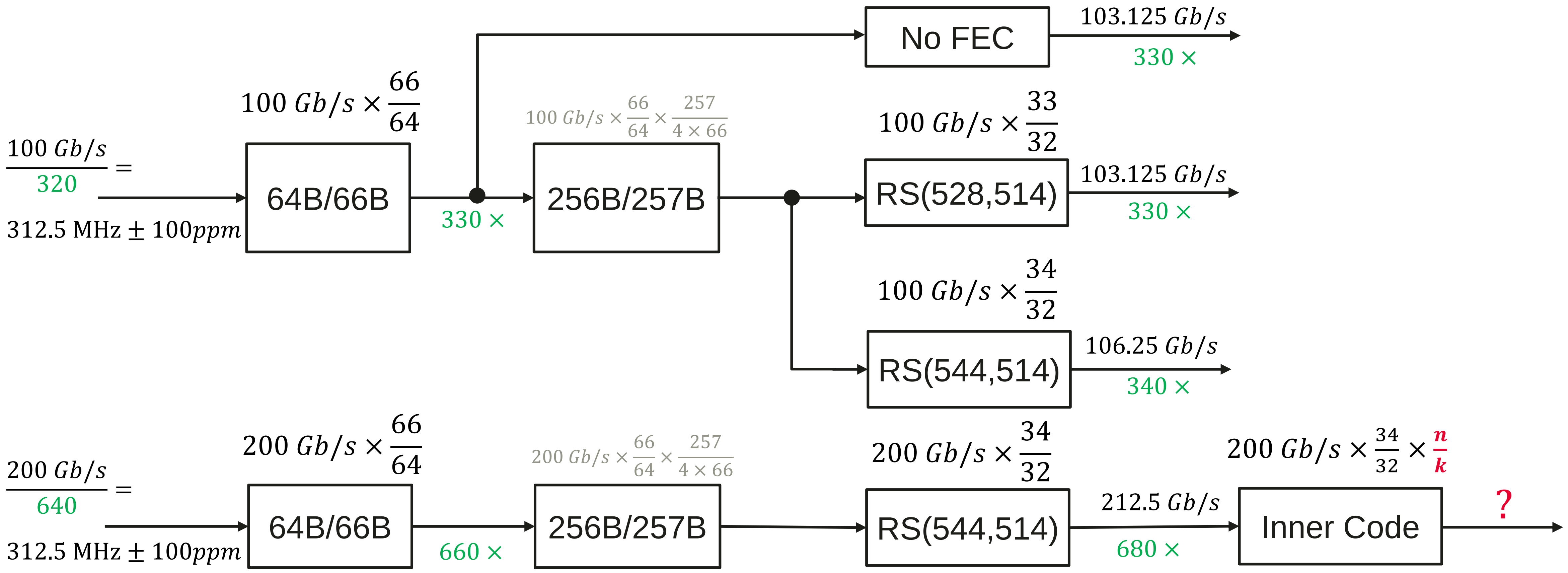
- Bose–Chaudhuri–Hocquenghem (BCH) codes are a class of cyclic error-correcting codes that are constructed using polynomials over a finite field (Galois Field).
- As the inner code of the concatenated architecture in this project, we only consider binary BCH codes to work together with the non-binary BCH outer code, such as RS(544,514).
- For any positive integers $\mathrm{m} \geq 2$ and $t < 2^{m-1}$, there exists a binary BCH code with the following parameters:
  - Codeword length: $\mathrm{n} = 2^m - 1$
  - Number of message bits: k
  - Number of corrected bits: t
  - Galois field index, GF(2^m): m
  - Number of parity bits: $\mathrm{n} - \mathrm{k} \leq mt$
  - Minimum hamming distance: $d \geq 2t + 1$



- BCH(n,k) code can be ***extended*** to eBCH(n+1,k) by adding an additional parity bit.
- BCH(n,k) code can be ***shortened*** to BCH(n-c,k-c) by removing the c bits of padded 0s.

Derive Overhead of Inner Code from Bit Rate in Ethernet

- Choosing the inner code (n, k) with the right n/k ratio could enable simpler 200 Gb/s per lane based PLL design, similar as 66/64(or 33/32) and 34/32(or 17/16) overhead ratio we chose before.



- BCH(144,136), BCH(126,119) will both give a 225 Gb/s per lane rate.

Step-by-Step Construction of a BCH Code

- The following steps describe how to construct a **narrow-sense binary primitive** BCH code.

1. Determine the desired codeword length and error correction capability.
   - Finding the suitable $\boldsymbol{n}(=2^m\text{-}1)$ and $\boldsymbol{t}$ (or d equivalent, $\boldsymbol{d} = 2\boldsymbol{t} + 1$).
2. Find a degree $\boldsymbol{m}$ primitive polynomial $\boldsymbol{p(x)}$ to construct the Galois Field $\mathrm{GF}(2^m)$.
   - A primitive element $\boldsymbol{\alpha}$ is the root of $\boldsymbol{p(x)}$.
   - Every non-zero element in $\mathrm{GF}(2^m)$ can be expressed by $\boldsymbol{\alpha^j}$ with integer $\boldsymbol{j}$.
3. Find the minimal polynomials $\boldsymbol{f(x)}$ for each primitive element $\boldsymbol{\alpha^i}$ $(i = 1,2,3,\dots,2t)$ in $\mathrm{GF}(2^m)$.
   - The minimal polynomial $f_i(x) = (x - c_1)(x - c_2)\cdots(x - c_l), l \le m$, $\{c_l\}$: conjugate root of $f_i(x)$ including $\boldsymbol{\alpha^i}$.
4. Get the generating polynomial $\boldsymbol{g}(\boldsymbol{x}) = \mathrm{LCM}[f_1(x), f_2(x), \dots, f_{2t}(x)]$
   - The generating polynomial is similar as the one we use to define the encode process in previous standard specification, like RS(544,514) code in CL119.2.4.4.
   - $g(x)$ can be expressed as: $\sum_{i=0}^{p} g_i x^i = g_p x^p + g_{p-1} x^{p-1} + \cdots + g_1 x + g_0$, which can be implemented using linear feedback shift register form and be illustrated as such. The number of parity bits is *p*. The coefficient g_i takes value 0 or 1.
5. Encode the $\boldsymbol{k}$ message bits with the generating polynomial $\boldsymbol{g(x)}$ to get $\boldsymbol{n}$ bits codeword.

Step-by-Step Construction of a Binary BCH Code (t = 1)

- The following steps describe how to construct a **narrow-sense binary primitive** BCH code with t = 1.

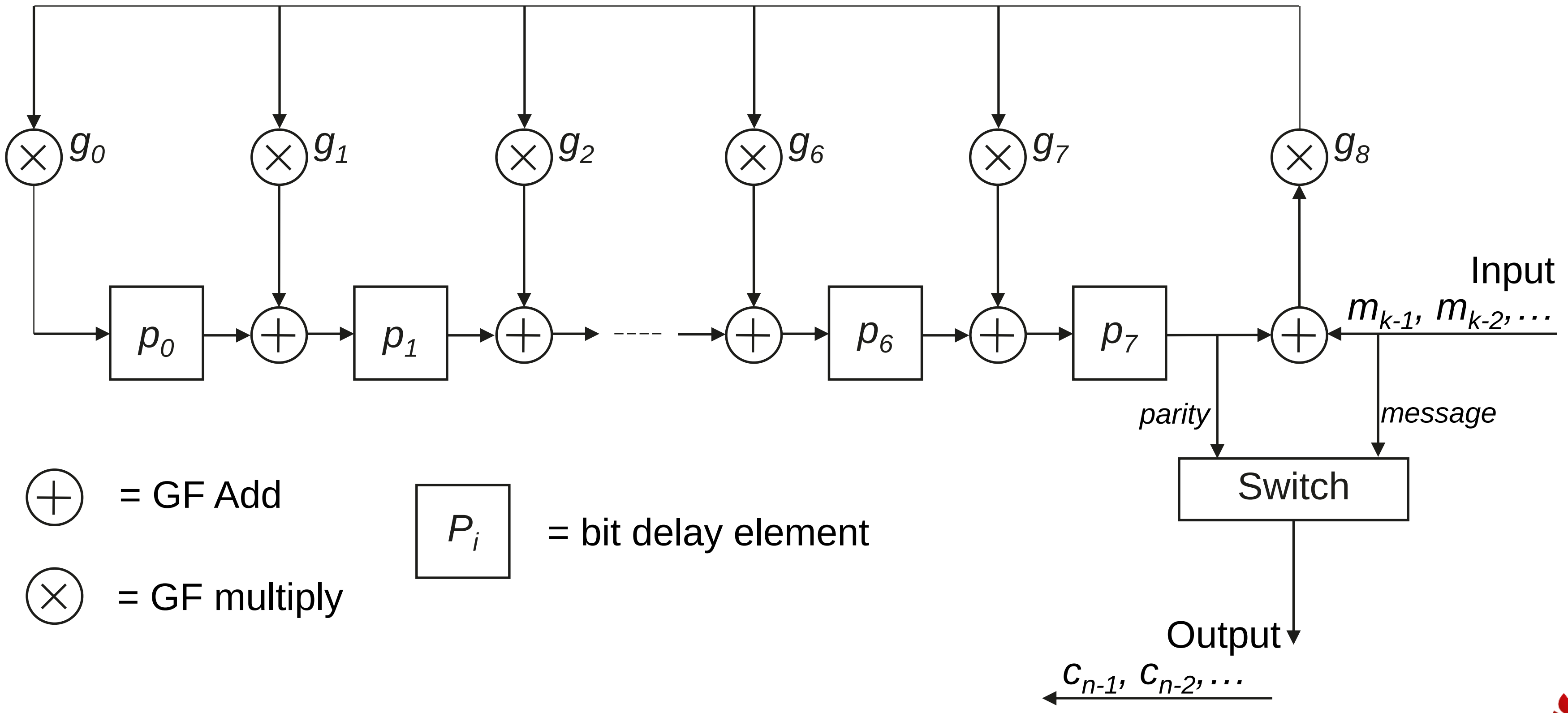
1. Determine the desired codeword length and error correction capability.
   - For t = 1, we can simply use $\mathrm{m} = \lceil \log_2(n+1) \rceil$.
2. Find the degree ***m*** primitive polynomial to construct the Galois Field $\mathrm{GF}(2^{\mathrm{m}})$.
   - Primitive polynomials have been well studied in academic research.
   - Many polynomials can be found online such as: https://www.partow.net/programming/polynomials/index.html#deg08
3. The generating polynomial is the same as the primitive polynomial for t=1 primitive BCH.

Why and How We Choose BCH(144,136) (and other codes)

- Various codes with different lengths and t values are tried in a concatenated scheme.
  - Codeword length between 100 ~ 1000 bits were studied (m = 7:10). Some examples were shown in he_b400g_01_210426 and he_3df_01a_220308.
  - Shorter codeword length had better performance when using soft-decision and limited LRPs.
  - t = 1 codes have lower latency and lower complexity.
  - Overall shorter BCH codes with t = 1 works better with the RS(544,514) outer code.
- Shorten the m = 8 primitive BCH(255,247), by prefixing to the message bits a sequence of 0s.
  - E.g., we can use primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$ (“implicit + 1” notation 0x8E) to construct the code.
  - There are many other primitive polynomials with degree of 8: 0x95, 0xAF, 0xB1, 0xB2, 0xB4, 0xE1, 0xF3, ...
  - The zero prefix sequence is not transmitted and is only used to calculate the parity of the primitive code.
- Apply the overhead ratio n/k = 36/34 (18/17), we can have BCH(144,136).
- eBCH(76,68) used in he_3df_01a_220308 can be constructed in a similar way, with m = 7 and 1-bit extension, and its overhead ratio n/k = 38/34 (19/17).
  - Example polynomial: $x^7 + x^3 + 1$
  - The same code length and overhead can also be a non-extended version of BCH(76,68) with m = 8.

BCH(144,136) Encoder Functional Model

- For $g(x) = x^8 + x^4 + x^3 + x^2 + 1$ of BCH(144,136), the 9 coefficient are 100011101, MSB on the left.
- The parity calculation shall produce the same result as the shift register implementation below in the similar form as in 119.2.4.4 for RS encoder and 115.2.3.3, 115.2.4.3.2 for BCH encoder.
- The outputs of the delay elements are initialized to zero prior to the computation of the parity for a given message.



Relationship between BCH and Hamming Code

- BCH code is a generalization of the Hamming code for multiple-error correction.
- The single-error-correcting binary BCH code of length 2^{r-1} is a Hamming code.
  - Codeword length: $n = 2^r - 1,\ r \geq 2;$
  - Message length: $k = 2^r - r - 1$
  - Hamming Distance: $d = 3;$
- Hamming code can be constructed with parity-check matrix:

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits (Message bit:d1,d2,d3,,,)		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit (p1,p2,p3,,,) Coverage	p1	✓		✓		✓		✓		✓		✓		✓		✓		✓		✓		
	p2		✓	✓			✓	✓			✓	✓			✓	✓			✓	✓		...
	p4				✓	✓	✓	✓					✓	✓	✓	✓					✓	
	p8								✓	✓	✓	✓	✓	✓	✓	✓						
	p16																✓	✓	✓	✓	✓	

Refer to: https://en.wikipedia.org/wiki/Hamming_code

Example: BCH(144,136) and Hamming(144,136)

- Construct the BCH code with a polynomial.
- Once the generator polynomial is selected and its LFSR is given, this code is uniquely defined.
  - Calling it BCH or Hamming does not matter.
  - Functional test and verification can be conducted easily with generator polynomial and LFSR.
  - This is the method IEEE 802.3bj/bs/bv used to define the RS and BCH FEC.

Generator polynomial $\boldsymbol{g}(\boldsymbol{x}) = x^8 + x^4 + x^3 + x^2 + 1$

⇩

Generator matrix $\boldsymbol{G} = [\ \boldsymbol{I}_{136} \mid \boldsymbol{P}^T\ ]$

⇩

Parity-check matrix $\boldsymbol{H} = [\ \boldsymbol{P} \mid \boldsymbol{I}_8\ ]$

$\boldsymbol{I}_n$ is an $n \times n$ Identity matrix.

Example: Double-Extended Hamming(128,119)

- Definition used in ITU-T G.709.3 Annex D, and IEEE P802.3cw.

- Parity-check column vectors $\boldsymbol{g}(\boldsymbol{i}) = \begin{bmatrix} s_{0,i} \\ s_{1,i} \\ \vdots \\ s_{6,i} \\ s_{7,i} \\ 1 \end{bmatrix},$

$$where\ i = 64s_{6,i} + 32s_{5,i} + \cdots + 2s_{1,i} + s_{0,i}$$

$$s_{7,i} = (s_{0,i} \wedge s_{2,i}) \vee (\overline{s_{0,i}} \wedge \overline{s_{1,i}} \wedge \overline{s_{2,i}}) \vee (s_{0,i} \wedge s_{1,i} \wedge \overline{s_{2,i}})$$

- Parity-check matrix $\boldsymbol{H} =$ [g(0): g(62), g(64): g(94), g(96): g(110), g(112): g(118), g(120), g(122), g(124), g(63), g(95), g(111), g(119), g(121), g(123), g(125): g(127)]

$$\boldsymbol{H} = [\ \boldsymbol{P} \mid \boldsymbol{I_9}\ ]$$

- Generator matrix $\boldsymbol{G} = [\ \boldsymbol{I_{119}} \mid \boldsymbol{P^T}\ ]$

$\boldsymbol{I_n}$ is an n×n Identity matrix.

$$\boldsymbol{H} = \begin{bmatrix}
1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1& 0&0&0&0&0&0&0&0& 0&0&0&0&0&0&0&0& 0&0&0&0&0&0&0&0& 0&0&0&0&0&0&0& 0&0&0&0&0&0&0&0& 0&0&0&0&0&0&0& 0&0&0&0&0&0&0& 0&0&0& 1&0&0&0&0&0&0&0&0 \\
1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 0&0&0&0&0&0&0&0& 0&0&0&0&0&0&0&0& 0&0&0&0&0&0&0&0& 0&0&0&0&0&0&0& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1& 0&0&0&0&0&0&0&0& 0&0&0&0&0&0&0& 0&0&0&0&0&0&0& 0&0&0& 0&1&0&0&0&0&0&0&0 \\
1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 0&0&0&0&0&0&0&0& 0&0&0&0&0&0&0&0& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 0&0&0&0&0&0&0&0& 0&0&0&0&0&0&0& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1&1& 0&0&0&0&0&0&0&0& 0&0&0&0&0&0&0& 1&1&1&1&1&1&1&1& 1&1&1&1&1&1&1& 0&0&0&0&0&0&0& 0&0&0& 0&0&1&0&0&0&0&0&0 \\
1&1&1&1&1&1&1&1& 0&0&0&0&0&0&0&0& 1&1&1&1&1&1&1&1& 0&0&0&0&0&0&0&0& 1&1&1&1&1&1&1&1& 0&0&0&0&0&0&0&0& 1&1&1&1&1&1&1&1& 0&0&0&0&0&0&0& 1&1&1&1&1&1&1&1& 0&0&0&0&0&0&0&0& 1&1&1&1&1&1&1&1& 0&0&0&0&0&0&0& 1&1&1&1&1&1&1&1& 0&0&0&0&0&0&0& 1&1&1&1&1&1&1& 0&0&0& 0&0&0&1&0&0&0&0&0 \\
1&1&0&0&0&0&0&0& 1&1&0&0&0&0&0&0& 1&1&0&0&0&0&0&0& 1&1&0&0&0&0&0&0& 1&1&0&0&0&0&0&0& 1&1&0&0&0&0&0&0& 1&1&0&0&0&0&0&0& 1&1&0&0&0&0&0& 1&1&0&0&0&0&0&0& 1&1&0&0&0&0&0&0& 1&1&0&0&0&0&0&0& 1&1&0&0&0&0&0& 1&1&0&0&0&0&0&0& 1&1&0&0&0&0&0& 1&1&0&0&0&0&0& 1&0&0& 0&0&0&0&1&0&0&0&0 \\
0&0&1&1&0&0&0&0& 0&0&1&1&0&0&0&0& 0&0&1&1&0&0&0&0& 0&0&1&1&0&0&0&0& 0&0&1&1&0&0&0&0& 0&0&1&1&0&0&0&0& 0&0&1&1&0&0&0&0& 0&0&1&1&0&0&0& 0&0&1&1&0&0&0&0& 0&0&1&1&0&0&0&0& 0&0&1&1&0&0&0&0& 0&0&1&1&0&0&0& 0&0&1&1&0&0&0&0& 0&0&1&1&0&0&0& 0&0&1&1&0&0&0& 0&1&0& 0&0&0&0&0&1&0&0&0 \\
0&0&0&0&1&1&0&0& 0&0&0&0&1&1&0&0& 0&0&0&0&1&1&0&0& 0&0&0&0&1&1&0&0& 0&0&0&0&1&1&0&0& 0&0&0&0&1&1&0&0& 0&0&0&0&1&1&0&0& 0&0&0&0&1&1&0& 0&0&0&0&1&1&0&0& 0&0&0&0&1&1&0&0& 0&0&0&0&1&1&0&0& 0&0&0&0&1&1&0& 0&0&0&0&1&1&0&0& 0&0&0&0&1&1&0& 0&0&0&0&1&1&0& 0&0&1& 0&0&0&0&0&0&1&0&0 \\
1&0&1&0&1&0&1&0& 1&0&1&0&1&0&1&0& 1&0&1&0&1&0&1&0& 1&0&1&0&1&0&1&0& 1&0&1&0&1&0&1&0& 1&0&1&0&1&0&1&0& 1&0&1&0&1&0&1&0& 1&0&1&0&1&0&1& 1&0&1&0&1&0&1&0& 1&0&1&0&1&0&1&0& 1&0&1&0&1&0&1&0& 1&0&1&0&1&0&1& 1&0&1&0&1&0&1&0& 1&0&1&0&1&0&1& 1&0&1&0&1&0&1& 1&1&1& 0&0&0&0&0&0&0&1&0 \\
1&0&1&0&1&0&0&1& 0&1&0&1&0&1&1&0& 0&1&0&1&0&1&1&0& 1&0&1&0&1&0&0&1& 0&1&0&1&0&1&1&0& 1&0&1&0&1&0&0&1& 1&0&1&0&1&0&0&1& 0&1&0&1&0&1&1& 0&1&0&1&0&1&1&0& 1&0&1&0&1&0&0&1& 1&0&1&0&1&0&0&1& 0&1&0&1&0&1&1& 1&0&1&0&1&0&0&1& 0&1&0&1&0&1&1& 0&1&0&1&0&1&1& 1&1&1& 0&0&0&0&0&0&0&0&1
\end{bmatrix}$$

Summary

- Suggest to develop BCH inner code for concatenated FEC using the polynomial methodology in this contribution.
- BCH(144,136) can be a candidate inner code that well matches with RS(544,514) outer code and Ethernet rate.

Thank you.

Bring digital to every person, home and organization for a fully connected, intelligent world.