8.10.1 Galois field

- Field: field is a set of elements and operations of addition and multiplication. The operations must follows rules below
  - Closed: Closure impliles that the sum and product of any two elements in the field are also elements of the field
  - Commutative (ab = ba and a+b = b+a)
  - Associative (a(bc) = (ab)c, and a + (b + c) = (a + b) + c)
  - Distributive law relates multiplication and addition: a(b + c) = ab + ac.
  - Has additive and multiplicative identities (0 and 1) such that a + 0 = a and 1a = a for any element in the field.
  - Elements of a field must have additive and multiplicative inverses. The additive inverse of a is an element b such that a+b = 0 and the multiplicative inverse of a is an element c such that ac = 1.
  - E.g:
    - set of real numbers and addition, multiplication creates field.

8.10.1 Galois field

- Finite field:
  - Denoted by Zp that contains
    - The set of integers {0, 1, .., p−1}
    - Modulo p arithmetic.
    - p is a prime number
- Galois field: GF (p^n) contains
  - p is prime number
  - n is arbitrary positive integer
  - Each element is denoted by polynomial *a1*x^0+ *a2*x^1+….+ *aN*x^{N-1} where the coefficients ai take on values in the set {0, 1, ..., p − 1}.
  - To add two polynomials, for each power of x present in the summands, just add the corresponding coefficients modulo p
  - a(x)= *a1*x^0+ *a2*x^1+….+ *aN*x^{N-1}; b(x)= *b1*x^0+ *b2*x^1+….+ *bN*x^{N-1}
  - c(x) = a(x)+b(x) = (a1⨁b1) + (a2⨁b2)x + …+ (*aN* ⨁ *bN)*x^{N-1}
  - ai⨁ bi = ai+bi if ai+bi <p

    = ai +bi –p if ai+bi >=p
  - Multiplication of two polynomials is done by multilication in modulo x^{N-1} where x^{N-1} is modulo polynomial a(x) x b(x) modulo (x^{N-1}) = reminder of ((a(x) x b(x))/ x^{N-1})

8.10.2 Definition

- *Cyclic code uses Galois Field GF(p^N)*
- *Codeword a is considered as polynomials*
  - *E.g. a = {a_1,a_2,…,a_N} is considered as a(x)= a_1x^0+ a_2x^1+….+ a_Nx^{N-1}*
- *Multiplication is calculated in modulo x^N −1*
- *Multiple with x is equivalence to right shift its coefficients*

  xa(x) = a_N+ a_0x^1+ a_1x^2+….+ a_{N-1}+ $a_N(x^N$-1)

  xa(x) modulo $(x^N$-1) = a_N+ a_0x^1+ a_1x^2+….+ a_{N-1}
- Cyclic code *is a linear code with the property that any cyclic shift of a code word is also a code word*
- *A cyclic code has a unique non-zero polynomial of minimal degree*
  - *This polynomial is called generator polynomial with degree r:*

    g(x) = g_o + g_1x +…+g_rx^r
  - *g(x) is the generator polynomial of a cyclic code if and only if it is a factor of $(x^N$-1)*
  - *The remainder of division between arbitrary codeword and g(x) =0*
    - *If c(x) is codeword then c(x) = m(x) g(x)*

8.10.2. Definition(Cont.)

- Generator matrix:

$$G = \begin{bmatrix} g_0 & g_1 & \dots & g_r & 0 & 0 & \dots & 0 \\ 0 & g_0 & \dots & g_{r-1} & g_r & 0 & \dots & 0 \\ & & & \vdots & & & & \\ 0 & \dots & 0 & g_0 & g_1 & \dots & g_r & 0 \\ 0 & \dots & 0 & 0 & g_0 & \dots & g_{r-1} & g_r \end{bmatrix}$$

  - G is a cyclic matrix (each row is obtained by shifting the previous row one column to the right).

8.10.2. Definition(Cont.)

- Since g(x) is the factor of $(x^N$*-1), that*

  $(x^N$-1) = g(x) h(x)

  Where h(x) is called check parity matrix

    - *If c(x) is codeword then c(x) h(x) = m(x) g(x) h(x) modulo $(x^N$-1) = 0*

  - h(x) = $h_0 + h_0$x +…+ $h_k x^{k-1}$

- Check parity matrix H: is a cyclic matrix (each row is obtained by shifting the previous row one column to the right).
  - First row is h(x)

8.10.3.Encoding and decoding

- Encoding process is multiple generator polynomial g(x) with carrying information (message) polynomial m(x)
    - c(x) = m(x) g(x)
- Decoding process:
    - Syndrome S is remainder of division between received polynomial r(x) and g(x)
        - S = r(x) mod g(x) modulo (x^N*-1)*
        - *If S = 0* → *codeword*
        - *If S* ≠ 0 → S = e(x) mod g(x) modulo (x^N*-1)*
            - *Can find error polynomial e(x) from S*

8.10.3.Encoding and decoding (Cont.)

- If generator matrix G is transformed into canonical form, codeword is in systematic form
  - c(x) = m(x) + d(x) x^k

  Where d(x) is a polynomial has degree of n-k-1
- Since c(x) mod g(x) modulo(x^N*-1) = 0,*

  then d(x) = m(x) x^{N-k} *mod* g(x) modulo(x^N*-1)*

8.10.4. Cyclic Redundancy Check Codes

- Is cyclic systematic code
- Used for send or store the information
- Codeword c(x) = m(x) – crc
    - crc = m(x) mod g(x) modulo $(x^N - 1)$
- Decoding
    - Let r(x) = m'(x) – crc' where m'(x) = m(x) + e_1(x); crc'=crc+ e_2(x)
        - e_1(x) first L symbol of e(x)
        - e_2(x) remaining N-L symbols of e(x)
    - S= m'(x) mod g(x) modulo $(x^N - 1)$ – crc'
        - S= 0 → no error
        - S≠ 0 → S = e_1(x) mod g(x) modulo $(x^N - 1)$ - e_2(x)
            - Calculate e_1(x), e_2(x) from S

Example

- Let

$$M(x) = \ a_m.x^m + a_{m-1}.x^{m-1} + a_{m-2}.x^{m-2} + \cdots + a_1.x^1 + a_0$$

$$G(x) = \ a_n.x^n + a_{n-1}.x^{n-1} + a_{n-2}.x^{n-2} + \cdots + a_1.x^1 + a_0$$

- Codeword: $T(x) = \ a_n.x^n.M(x)$

→ left shift n bit of M(x)

- CRC string R(X):

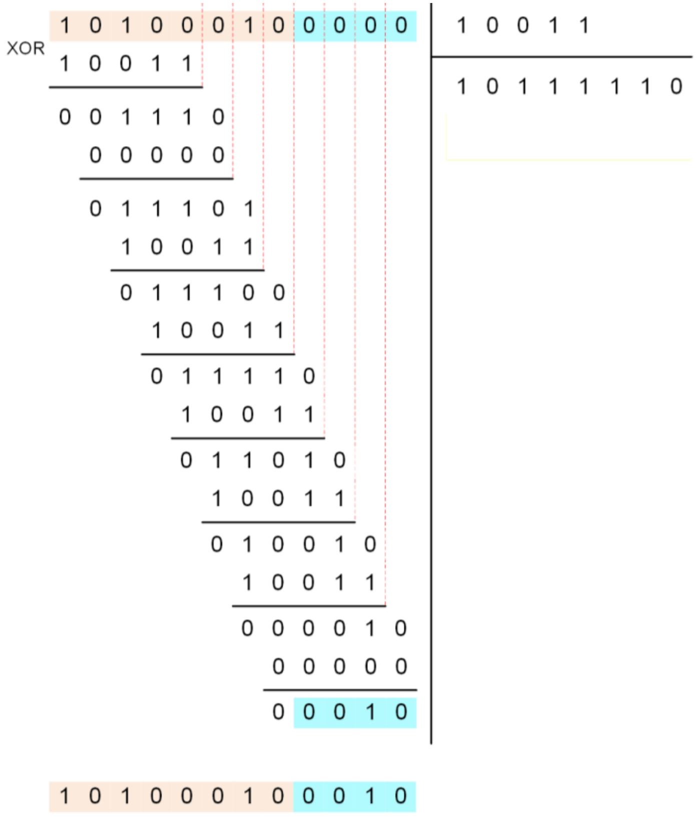
$$R(x) = T(x)\ \%\ G(x)$$

Example

- CRC-4 → G(X): x^4 + x + 1 (10011)
- Input: M(X) = x^7 + x^5 + x (1010_0010)
- To extent to T(X): M(X) wi ifted 4 positions:

  →1010_0010_0000

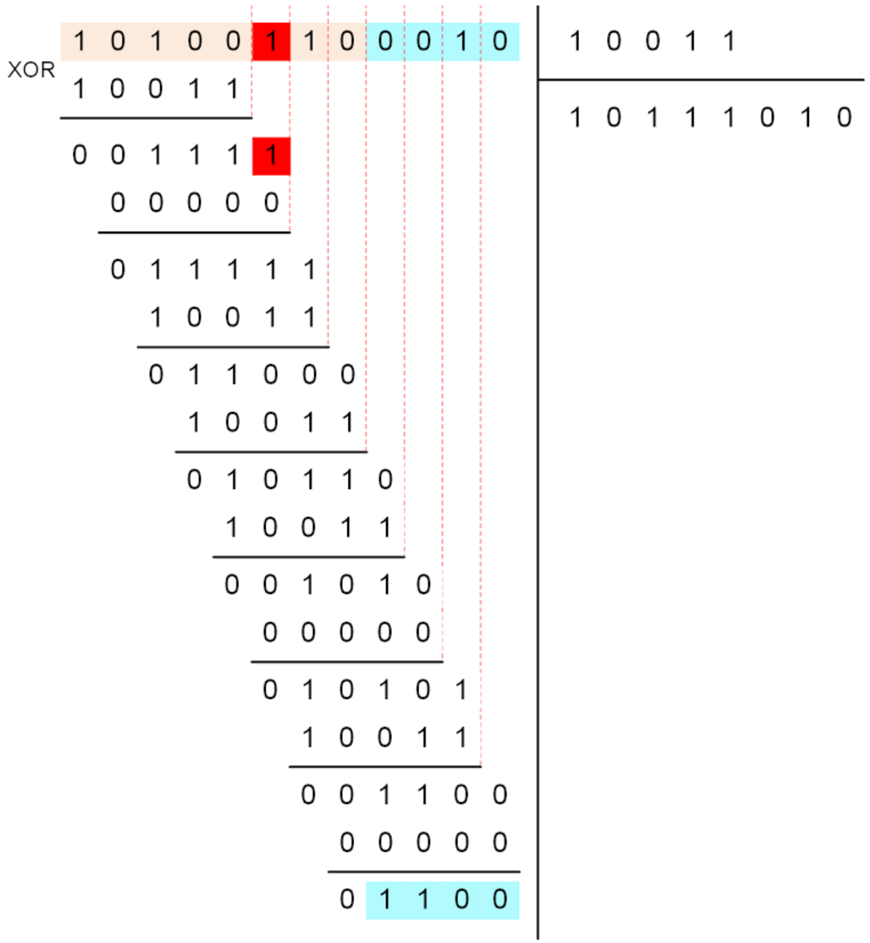
Calculate codeword



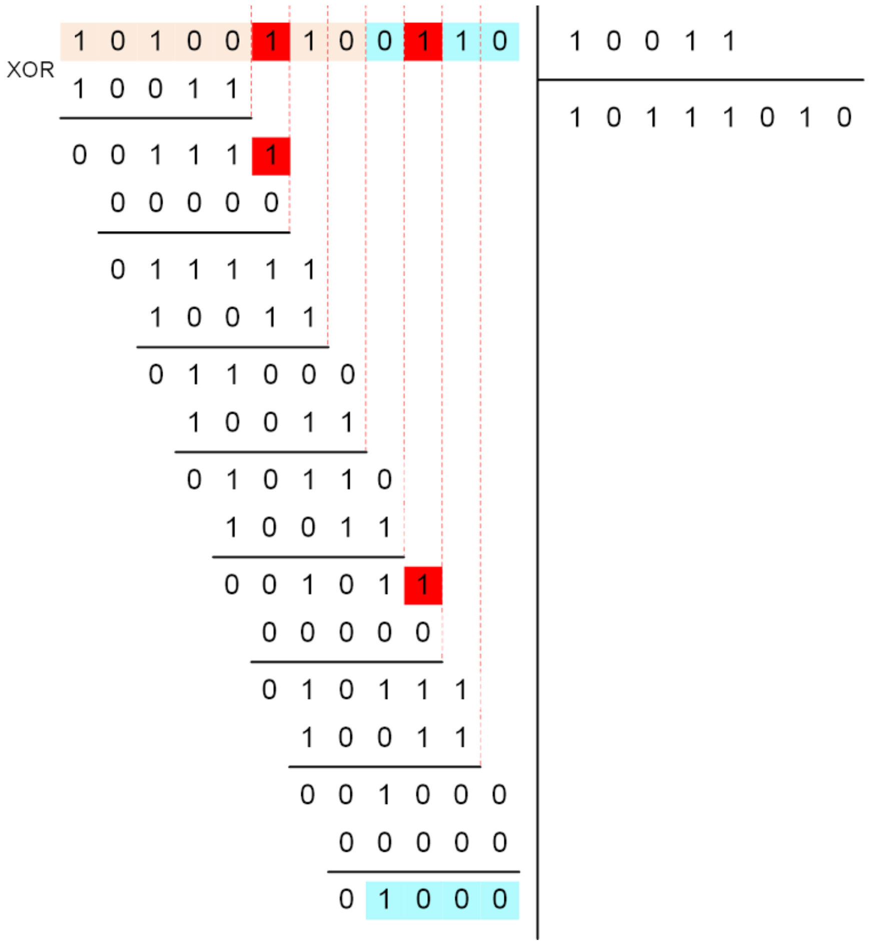
```
    1 0 1 0 0 0 1 0 0 0 0 0 | 1 0 0 1 1
XOR 1 0 0 1 1               |----------------
    ---------               | 1 0 1 1 1 1 1 0
    0 0 1 1 1 0             |
      0 0 0 0 0             |
      ---------             |
      0 1 1 1 0 1           |
        1 0 0 1 1           |
        ---------           |
        0 1 1 1 0 0         |
          1 0 0 1 1         |
          ---------         |
          0 1 1 1 1 0       |
            1 0 0 1 1       |
            ---------       |
            0 1 1 0 1 0     |
              1 0 0 1 1     |
              ---------     |
              0 1 0 0 1 0   |
                1 0 0 1 1   |
                ---------   |
                0 0 0 0 1 0 |
                  0 0 0 0 0 |
                  --------- |
                  0 0 0 1 0 |

    1 0 1 0 0 0 1 0 0 0 1 0
```

Test: errors

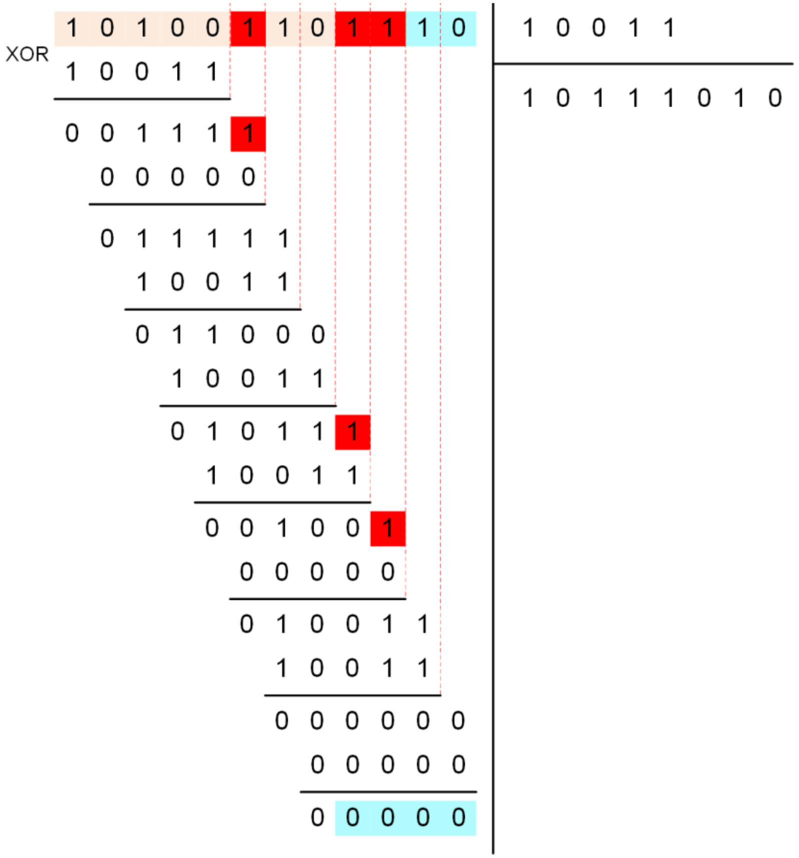


```
    1 0 1 0 0 1 1 0 0 0 1 0 | 1 0 0 1 1
XOR 1 0 0 1 1               |----------------
    ---------               | 1 0 1 1 1 0 1 0
    0 0 1 1 1 1
      0 0 0 0 0
      ---------
      0 1 1 1 1 1
        1 0 0 1 1
        ---------
        0 1 1 0 0 0
          1 0 0 1 1
          ---------
          0 1 0 1 1 0
            1 0 0 1 1
            ---------
            0 0 1 0 1 0
              0 0 0 0 0
              ---------
              0 1 0 1 0 1
                1 0 0 1 1
                ---------
                0 0 1 1 0 0
                  0 0 0 0 0
                  ---------
                  0 1 1 0 0
```



```
    1 0 1 0 0 1 1 0 0 1 1 0 | 1 0 0 1 1
XOR 1 0 0 1 1               |----------------
    ---------               | 1 0 1 1 1 0 1 0
    0 0 1 1 1 1
      0 0 0 0 0
      ---------
      0 1 1 1 1 1
        1 0 0 1 1
        ---------
        0 1 1 0 0 0
          1 0 0 1 1
          ---------
          0 1 0 1 1 0
            1 0 0 1 1
            ---------
            0 0 1 0 1 1
              0 0 0 0 0
              ---------
              0 1 0 1 1 1
                1 0 0 1 1
                ---------
                0 0 1 0 0 0
                  0 0 0 0 0
                  ---------
                  0 1 0 0 0
```

CRC does not work



```
    1 0 1 0 0 1 1 0 1 1 1 0  | 1 0 0 1 1
XOR 1 0 0 1 1                |----------
    ---------                | 1 0 1 1 1 0 1 0
    0 0 1 1 1 1              |
      0 0 0 0 0              |
      ---------              |
      0 1 1 1 1 1            |
        1 0 0 1 1            |
        ---------            |
        0 1 1 0 0 0          |
          1 0 0 1 1          |
          ---------          |
          0 1 0 1 1 1        |
            1 0 0 1 1        |
            ---------        |
            0 0 1 0 0 1      |
              0 0 0 0 0      |
              ---------      |
              0 1 0 0 1 1    |
                1 0 0 1 1    |
                ---------    |
                0 0 0 0 0 0  |
                  0 0 0 0 0  |
                  ---------  |
                  0 0 0 0 0  |
```