Frame Check Sequence (FCS) Calculation Explained

NTCIP and AB 3418 protocol developers typically use the inverse polynomial method to calculate the Frame Check Sequence (FCS). This method checks against the "magic number" of 0xf0b8 to ensure error-free transmission. ISO 3309 specifies the "magic number" of 0x1d0f, which is the reverse bits of 0xf0b8. The inverse polynomial method corrects for the fact that each byte's bits are reversed at transmission time when transmitted over serial lines, even though the method checks against a frame ordered most significant bit to least significant.

FCS Calculation at the Transmitter

The method to calculate an FCS at the transmitter is as follows:

1. Invert the first 16 bits of the frame to be transmitted, 1's become 0's and 0's become 1's.
2. Add 16 zeros to the end of the frame.
3. Divide the frame by the generator polynomial. This step involves XORing the remainder repeatedly with the 17 bit generator polynomial until the remainder is 16 bits long.
4. The FCS is the 1's compliment of the remainder, R(x).

Our test frame to be transmitted is 0x72 0xd3 0x4f

```
   7    2    d    3    4    f
0111 0010 1101 0011 0100 1111
```

Invert the first 16 bits.

```
   8    d    2    c    4    f
1000 1101 0010 1100 0100 1111
```

Reverse the bit order in each byte and shift 16 bits, padding the remaining 16 bits with zeros.

```
   b    1    3    6    f    2    0    0    0    0
1011 0001 0011 0100 1111 0010 0000 0000 0000 0000
```

Perform the division by XORing with the generator polynomial P(x) x^16 + x^12 + x^5 + 1
(binary 10001000000100001).

```
10110001001101001111001000000000000000
10001000000100001  <- P(x)
  11100100100100011
  10001000000100001  <- P(x)
  11011001000000101
   10001000000100001  <- P(x)
    10100010001001000
    10001000000100001  <- P(x)
      10101000110100101
      10001000000100001  <- P(x)
        10000011000010000
        10001000000100001  <- P(x)
            10110001100010000
              10001000000100001  <- P(x)
               11100110011000100
               10001000000100001  <- P(x)
               11011100111001010
               10001000000100001  <- P(x)
                10101001111010110
                10001000000100001  <- P(x)
                  10000111111011100
                  10001000000100001  <- P(x)
                      11111111111010000
                      10001000000100001  <- P(x)
                       11101111111100010
```

```
                        10001000000100001  <- P(x)
                         1100111111000011  <- R(x)

R(x) = 11001111 11000011 = 0xcfc3
R(x) bar = 00110000 00111100 = 0x303c
```

The value 0x303c is the FCS that is transmitted across the wire, which for serial communication is least significant bit first. The bitwise reverse of R(x) bar is 0000 1100 0011 1100 or 0x0c3c, which is the FCS from the frame's perspective.

Transmitted Frame (bytewise)

```
72 d3 4f 0c 3c
```

Transmitted Frame (bitwise)

```
b1 36 f2 30 3c
```

FCS Verification at the Receiver

The method to calculate an FCS at the receiver is the same as the transmitter except for not taking the 1's compliment of the remainder. A new FCS is calculated on the entire frame, including the old FCS, and checked against a value of 0x1d0f. (This differs from typical Cyclic Redundancy Check (CRC) algorithms which checks the frame, minus the transmitted CRC, and makes sure the new CRC equals old CRC.) The steps to calculate the FCS at the receiver are as follows:

1. Invert the first 16 bits of the frame received, 1's become 0's and 0's become 1's.
2. Add 16 zeros to the end of the frame.
3. Divide the frame by the generator polynomial.
4. The remainder should be 0x1dof, indicating the frame is free of transmission errors.

The frame received is 0x72 0xd3 0x4f 0x0c 0x3c

```
   7    2    d    3    4    f    0    c    3    c
0111 0010 1101 0011 0100 1111 0000 1100 0011 1100
```

Invert the first 16 bits.

```
   8    d    2    c    4    f    0    c    3    c
1000 1101 0010 1100 0100 1111 0000 1100 0011 1100
```

Reverse the bit order in each byte and shift 16 bits, padding the remaining 16 bits with zeros.

```
   b    1    3    6    f    2    3    0    3    c    0    0    0    0
1011 0001 0011 0100 1111 0010 0011 0000 0011 1100 0000 0000 0000 0000
```

Perform the division by XORing with the generator polynomial P(x) x^16 + x^12 + x^5 + 1 (0x10001000000100001).

```
10110001001101001111001000110000001111000000000000000000
10001000000100001  <- P(x)
 11100100100100011
 10001000000100001  <- P(x)
  11011001000000101
  10001000000100001  <- P(x)
   10100010001001000
   10001000000100001  <- P(x)
    10101000110100101
    10001000000100001  <- P(x)
     10000011000010000
     10001000000100001  <- P(x)
         10110001100010110
         10001000000100001  <- P(x)
          11100110011011100
          10001000000100001  <- P(x)
           11011100111111010
           10001000000100001  <- P(x)
            10101001110110110
            10001000000100001  <- P(x)
             10000111001011101
             10001000000100001  <- P(x)
                 11110011111001110
                 10001000000100001  <- P(x)
                  11110111111011110
                  10001000000100001  <- P(x)
                   11111111111111110
                   10001000000100001  <- P(x)
                    11101111110111110
                    10001000000100001  <- P(x)
                     11001111100111110
                     10001000000100001  <- P(x)
                      10001111000111110
                      10001000000100001  <- P(x)
                           11100001111100000
                           10001000000100001  <- P(x)
                            11010011110000010
                            10001000000100001  <- P(x)
                             10110111101000110
                             10001000000100001  <- P(x)
                               11111110110011100
                               10001000000100001  <- P(x)
                                11101101101111010
                                10001000000100001  <- P(x)
                                 11001011010110110
                                 10001000000100001  <- P(x)
                                  10000110100101110
                                  10001000000100001  <- P(x)
                                   00011101000011111  <- R(x)

                                    0001 1101 0000 1111
                                       1    d    0    f
```

The frame at the receiving end can as well be checked by inverting the received FCS and checking for a remainder of zero.

```
   b    1    3    6    f    2    c    f    c    3
1011 0001 0011 0100 1111 0010 1100 1111 1100 0011
```

```
   b   1   3   6   f   2   c   f   c   3
1011000100110100111100101100111111000011
10001000000100001  <- P(x)
 11100100100100011
 10001000000100001  <- P(x)
  11011001000000101
  10001000000100001  <- P(x)
   10100010001001000
   10001000000100001  <- P(x)
    10101000110100101
    10001000000100001  <- P(x)
     10000011000010001
     10001000000100001  <- P(x)
         10110001100001001
         10001000000100001  <- P(x)
           11100110010100011
           10001000000100001  <- P(x)
            11011100100000101
            10001000000100001  <- P(x)
             10101001001001001
             10001000000100001  <- P(x)
               10000100110100010
               10001000000100001  <- P(x)
                   11001100000110001
                   10001000000100001  <- P(x)
                    10001000000100001
                    10001000000100001  <- P(x)
                    00000000000000000  <- R(x)
```