CYCLIC REDUNDANCY CHECK (CRC) ALGORITHMS IN SENSOR COMMUNICATIONS

By TakaHide Ohkami
Allegro MicroSystems

INTRODUCTION

Cyclic redundancy check (CRC) is an error-detecting scheme for digital data communications. The sender calculates the special CRC value from a message and sends it with the original message, as shown in Figure 1. The receiver checks the received CRC value for the received message to determine whether it is valid. If the value is valid, the received message is correct; otherwise, the message or CRC value has been corrupted in transmission.

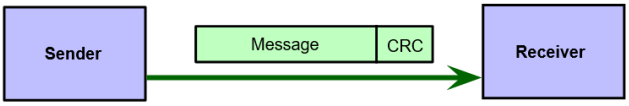

Figure 1: Message Transmission with CRC.

The recent Allegro sensor ICs use this CRC scheme to check data integrity in Manchester, SPI, and SENT communications. Some selected examples are shown in Table 1.

Table 1: Allegro Sensor ICs Using CRC in Communications

Allegro Part Number	Sensor Type	Communication with CRC		
		Manchester	SPI	SENT
A1333	Angle Sensor	✓	✓	–
A1335	Angle Sensor	✓	✓	✓
A1337	Angle Sensor	✓	✓	✓
A1339	Angle Sensor	✓	✓	–
A1346	Linear Sensor	✓	–	✓
A1363	Linear Sensor	✓	–	–
A1367	Linear Sensor	✓	–	–

Many articles, papers, and books are available on various CRC algorithms. Still, confusion may arise about CRC algorithm implementations due to the different methods used.

This application note describes the following different CRC algorithms and shows that they are conditionally equivalent:

- Basic Hand-Calculation Algorithm
- Linear Feedback Shift Register (LFSR) Algorithm 1 (Post-Multiply)
- Linear Feedback Shift Register (LFSR) Algorithm 2 (Pre-Multiply)
- Table Lookup Algorithm 1 (Large)
- Table Lookup Algorithm 2 (Small)

This application note is intended to help customers better understand the CRC algorithms and their conditional equivalence.

THEORETICAL BACKGROUND

GF(2) – Galois Field of Order 2

The CRC algorithm is based on the polynomial division over GF(2) (Galois Field of Order 2).

GF(2) is the special algebraic structure that defines the finite field with two elements (0 and 1) and two binary operations (addition and multiplication). In this structure, addition and multiplication are performed by the logical exclusive-OR and logical AND operations, respectively, as shown in Table 2.

Table 2: Addition and Multiplication over GF(2)

x	y	Addition $x + y$	Multiplication $x \times y$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Subtraction used for division is the same as addition and is performed by the logical exclusive-OR operation.

The binary operations in GF(2) are summarized in Table 3.

Table 3: GF(2) and Actual Operations.

GF(2) Operation	Actual Operation
Addition (+)	Logical Exclusive-OR
Subtraction (−)	Logical Exclusive-OR
Multiplication (×)	Logical AND

Polynomial Operations in GF(2)

A polynomial in GF(2) is a binary polynomial that has coefficients of 0's or 1's. An example polynomial is shown below.

$$1 \cdot x^5 + 0 \cdot x^4 + 1 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 = x^5 + x^3 + 1.$$

Three polynomials are defined as follows:

$$X(x) = x^3 + x^2 + 1\,,$$
$$Y(x) = x^4 + x^2 + x^1 + 1\,,$$
$$Z(x) = x^6 + x^2 + 1\,.$$

Example addition, subtraction, and multiplication operations with these polynomials are shown in Table 4.

Polynomial Division over GF(2)

The polynomial division over GF(2) is the division of a binary polynomial by another binary polynomial, which can be performed by repeating subtractions.

Let A(x) and B(x) be the dividend and divisor polynomials, respectively, as shown below:

$$A(x) = x^7 + x^5 + x^3 + 1\,,$$
$$B(x) = x^3 + x^2 + 1\,.$$

The division of A(x) by B(x) is performed as follows:

$$\begin{array}{r|l}
 & x^4 + x^3 + x^1 + 1 \\
\hline
x^3 + x^2 + 1 & x^7 + x^5 + x^3 + 1 \\
 & x^7 + x^6 + x^4 \\
\hline
 & x^6 + x^5 + x^4 + x^3 + 1 \\
 & x^6 + x^5 + x^3 \\
\hline
 & x^4 + 1 \\
 & x^4 + x^3 + x^1 \\
\hline
 & x^3 + x^1 + 1 \\
 & x^3 + x^2 + 1 \\
\hline
 & x^2 + x^1
\end{array}$$

From this division, the following is obtained:

$$A(x) = Q(x)B(x) + R(x)\,.$$

where

$$\text{Quotient}: \quad Q(x) = x^4 + x^3 + x^1 + 1\,,$$
$$\text{Remainder}: \quad R(x) = x^2 + x^1\,.$$

The remainder R(x) for some fixed divisor B(x) is the CRC value obtained by the polynomial division.

For notational convenience, use the modulo operation (mod) as follows:

$$R(x) = A(x) \bmod B(x) = Q(x)\,B(x) + R(x) \bmod B(x)\,.$$

Note that Q(x) B(x) mod B(x) = 0 because Q(x) B(x) is divisible by B(x).

Polynomial Degrees

The degree of a binary polynomial is the largest power (exponent) of variable x with a non-zero coefficient. The degree of a polynomial P (x) is denoted by deg(P(x)).

The degrees of A(x) and B(x) used in the previous division example are given by:

$$\deg(A(x)) = \deg(x^7 + x^5 + x^3 + 1) = \deg(x^7) = 7\,,$$
$$\deg(B(x)) = \deg(x^3 + x^2 + 1) = \deg(x^3) = 3\,.$$

The degree of the division remainder R(x) is always less than the degree of the divisor B(x), therefore:

$$\deg(R(x)) \leq \deg(B(x)) - 1\,.$$

The degree of A(x) × B(x) is given by:

$$\deg(A(x) \times B(x)) = \deg((x^7 + \ldots) \times (x^3 + \ldots)) = \deg(x^{10} + \ldots) = 10\,.$$

Binary String Representations of Polynomials

A binary polynomial can be represented by a binary string, such as 100111 for $x^5 + x^2 + x^1 + 1$. The length of the binary string for a polynomial is equal to the degree of the polynomial plus 1.

Because of the nature of GF(2), the polynomial division over GF(2) can be performed using the binary strings with the logical exclusive-OR operation for addition and subtraction.

The binary strings for A(x) and B(x) are given by:

$$10101001 \text{ for } A(x) = x^7 + x^5 + x^3 + 1\,,$$
$$1101 \text{ for } B(x) = x^3 + x^2 + 1\,.$$

Table 4: Example Polynomial Addition, Subtraction, and Multiplication.

Operation	Operations with $X(x)$ and $Y(x)$	Operations with $Z(x)$
Addition	$X(x) + Y(x) = x^4 + x^3 + x^1$	$Z(x) + Z(x) = 0$
Subtraction	$X(x) - Y(x) = x^4 + x^3 + x^1$	$Z(x) - Z(x) = 0$
Multiplication	$X(x) \times Y(x) = x^7 + x^6 + x^5 + x^4 + x^1 + 1$	$Z(x) \times Z(x) = x^{12} + x^4 + 1$

The division of A(x)/B(x) = 10101001/1101 is performed as shown below.

```
        11011
    ┌──────────
1101 │ 10101001
       1101
    ──────────
       01111001
       .1101
    ──────────
       .0010001
       ...1101
    ──────────
       ...01011
       ....1101
    ──────────
       ....0110
```

From this binary string division, the same quotient and remainder are obtained by the direct polynomial division:

$$\text{Quotient}: \quad Q(x) = 11011\ (x^4 + x^3 + x^1 + 1)\ ,$$
$$\text{Remainder}: \quad R(x) = 0110\ (x^2 + x^1)\ .$$

BASIC HAND-CALCULATION ALGORITHM

CRC-Related Polynomials

The CRC algorithm uses the binary polynomial division to find the remainder polynomial representing the CRC value for a given message polynomial.

Several polynomials to describe the CRC algorithm are defined in Table 5.

Table 5: CRC-Related Polynomials

Polynomial	Degree	Description
G(x)	g	Generator Polynomial
M(x)	m	Message Polynomial
C(x)	$g - 1$	CRC/Remainder Polynomial
Q(x)	$m - g$	Polynomial Quotient Polynomial
E(x)	$m + g$	Extended Message Polynomial
S(x)	$m + g$	Sender Data Polynomial

The generator polynomial G(x) determines the CRC scheme (the polynomial division scheme), which is applied to the message polynomial M(x) to generate the CRC/remainder polynomial C(x) and the quotient polynomial Q(x). Note that the number of the CRC bits is deg(G(x)) = g.

The extended message polynomial E(x) is created by multiplying the message polynomial M(x) by x^g :

$$E(x) = x^g\, M(x)\ .$$

The CRC polynomial C(x) for the message polynomial M(x) is found by performing the polynomial division E(x)/G(x), based the following equation.

$$E(x) = Q(x)\ G(x) + C(x)\ .$$

The remainder/CRC polynomial C(x) is given by:

$$C(x) = E(x) \bmod G(x) = x^g\, M(x) \bmod G(x)\ .$$

The sender data polynomial S(x) represents the actual data to send, which includes the raw message M(x) and the calculated CRC value C(x), and is defined by:

$$S(x) = E(x) + C(x) = x^g\, M(x) + C(x)\ .$$

If there is no communication error, the receiver receives S(x). In this case, the receiver finds the zero remainder in the polynomial division S(x)/G(x), as shown below:

$$\begin{aligned} S(x) \bmod G(x) &= x^g\, M(x) + C(x) \bmod G(x) \\ &= Q(x)\ G(x) + C(x) + C(x) \bmod G(x) \\ &= C(x) + C(x) \bmod G(x) \\ &= 0\ , \end{aligned}$$

because Q(x) G(x) mod G(x) = 0 and C(x) + C(x) = 0. The non-zero remainder means that the received message and/or CRC value is not valid.

Note that the extended message, sender data, and the data received by the receiver are all m + g + 1 bits long, because:

$$\begin{aligned} \deg(S(x)) &= \deg(E(x)) = \deg(x^g\, M(x)) \\ &= \deg(x^g \times (x^m + \ldots)) = g + m\ . \end{aligned}$$

CRC Calculation with Example Polynomials

Define the example generator and message polynomials $G_1(x)$ and $M_1(x)$ as follows:

$$\begin{aligned} G_1(x) &= x^4 + x^3 + 1 = 11001\ , \\ M_1(x) &= x^7 + x^5 + x^3 + x^1 + 1 = 10101011\ . \end{aligned}$$

Note that deg($G_1(x)$) = 4 and deg($M_1(x)$) = 7.

Then the extended message polynomial $E_1(x)$ is given by:

$$\begin{aligned} E_1(x) &= x^4\, M_1(x) = x^{11} + x^9 + x^7 + x^5 + x^4 \\ &= 10101011\_0000\ . \end{aligned}$$

Divide $E_1(x)$ by $G_1(x)$ to find $C_1(x)$, using the simplified binary string division:

```
E1(x)   10101011_0000
G1(x)   11001
        ──────────────
        01100011_0000
        .11001
        ──────────────
        .0000111_0000
        .....110_01
        ──────────────
        .....001_0100
        .......1_1001
        ──────────────
C1(x)   .......0_1101
```

From this division, the following 4-bit CRC value as the remainder is obtained:

$$C_1(x) = E_1(x) \bmod G_1(x) = x^3 + x^2 + 1 = 1101\ .$$

The sender data polynomial $S_1(x)$ is formed as follows:

$$S_1(x) = E_1(x) + C_1(x) = 10101011\_0000 + 1101$$
$$= 10101011\_1101 .$$

Assume that the receiver receives the correct data polynomial $S_1(x)$. Then the division of $S_1(x)$ by $G_1(x)$ produces the zero remainder as shown below.

```
S1(x)  10101011_1101
G1(x)  11001
       ----------
       01100011_1101
       .11001
       ----------
       .0000111_1101
       .....110_01
       ----------
       .....001_1001
       .......1_1001
       ----------
       .......0_0000
```

Use this example set of polynomials throughout the rest of this note. Table 6 lists these polynomials.

LINEAR FEEDBACK SHIFT REGISTER ALGORITHM 1 (POST-MULTIPLY)

Two Types of Linear Feedback Shift Register

The CRC value for a message can be computed using a special shift register called “linear feedback shift register” or “LFSR”. To accomplish this, create the LFSR for a given generator polynomial, feed the message bits to the LFSR, and find the CRC value left in the register after feeding the last bit.

There are two types of LFSR:

- Post-Multiply LFSR (PST-LFSR)
- Pre-Multiply LFSR (PRE-LFSR)

The following notation is used for the LFSR operations:

$$C = \mathrm{LFSR}(G, I, V)$$

where

$$\mathrm{LFSR} = \text{PST-LFSR or PRE-LFSR}$$

$$C = \text{Output CRC String}$$

$$G = \text{Generator String}$$

$$I = \text{Initial CRC String}$$

$$M = \text{Input String}$$

This notation indicates that the LFSR (Post-Multiply or Pre-Multiply) created for the generator string G starts with the initial CRC string I, runs for the input string M, and produces the output CRC string C for the input string.

The first algorithm is described based on the post-multiply LFSR (PST-LFSR) in this section, and the second algorithm based on the pre-multiply LFSR (PRE-LFSR) in the next section.

Post-Multiply LFSR Operations

Create the post-multiply LFSR (PST-LFSR1) for the generator polynomial G_1 = 11001. The LFSR configuration is based on the bit values of the generator string except the top one (1001), as shown in Figure 2.

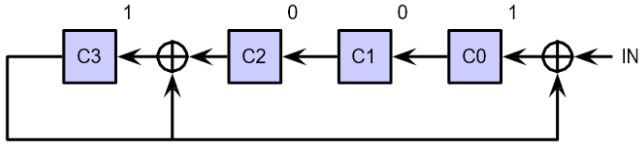

Figure 2: Post-Multiply LFSR for G_1 = 11001 (PST-LFSR1).

In Figure 2, C0, C1, C2, and C3 are the four register bits (CRC bits), ⊕ is the exclusive-OR operator placed at the register input if the corresponding bit of the generator string is 1, and IN is the input bit.

Table 7 shows the PST-LFSR1 operations.

Table 7: PRE-LFSR1 Operations

Cycle	C3	C2	C1	C0	IN
N	D	C	B	A	X
N+1	E=D⊕C	B	A	D⊕X	Y
N+2	E⊕B	A	D⊕X	E⊕Y	Z

Table 6: Example Polynomials

Polynomial	Binary String	Description
$G_1(x) = x^4 + x^3 + 1$	11001	Generator
$M_1(x) = x^7 + x^5 + x^3 + x^1 + 1$	10101011	Message
$C_1(x) = x^3 + x^2 + 1$	1101	CRC/Remainder
$E_1(x) = x^{11} + x^9 + x^7 + x^5 + x^4$	10101011_0000	Extended Message
$S_1(x) = x^{11} + x^9 + x^7 + x^5 + x^4 + x^3 + x^2 + 1$	10101011_1101	Sender Data

Run this LFSR by feeding the 12 bits of the extended message E_1 = 10101011_0000. Figure 3 shows the result.

Cycle	C3–0	IN
0	0000	10101011_0000
1	0001	0101011_0000
2	0010	101011_0000
3	0101	01011_0000
4	1010	1011_0000
5	1100	011_0000
6	0001	11_0000
7	0011	1_0000
8	0111	0000
9	1110	000
10	0101	00
11	1010	0
12	1101	

Figure 3: PST-LFSR1 Operations for E_1 = 10101011_0000.

After 12 cycles, the CRC value (remainder) is left in the C3, C2, C1, and C0 registers: C_1 = 1101. Therefore:

$$C_1 = 1101 = \text{PST-LFSR1}(G_1, 0, E_1)$$
$$= \text{PST-LFSR1}(11001, 0, 10101011\_0000)$$

Analysis of Post-Multiply LFSR Operations

The PST-LFSR algorithm is equivalent to the basic hand-calculation algorithm.

In order to demonstrate this, re-arrange the polynomial division method like the basic hand-calculation algorithm, which shifts the divisor against the dividend.

In this division method, divide the 0-padded dividend (0_0000_101010110000) by the full divisor (1_1001) and shift the extended dividend against the divisor.

Figure 4 shows this method.

Cycle	Operation	Divisor/Dividend
0	Divisor	1_1001
0	Extended Dividend	0_0000_101010110000
1	Shift	0_0001_01010110000
2	Shift	0_0010_1010110000
3	Shift	0_0101_010110000
4	Shift	0_1010_10110000
5a	Shift for Subtract	1_0101_0110000
5b	Subtract	0_1100_0110000
6a	Shift for Subtract	1_1000_110000
6b	Subtract	0_0001_110000
7	Shift	0_0011_10000
8	Shift	0_0111_0000
9	Shift	0_1110_000
10a	Shift for Subtract	1_1100_00
10b	Subtract	0_0101_00
11	Shift	0_1010_0
12a	Shift for Subtract	1_0100
12b	Subtract	0_1101

Figure 4: PST-LFSR1 Operations for Extended Dividend (0_0000_101010110000)

When C3 = 0, the next operation is the regular left shift.

When C3 = 1, the next operation is the extra left shift followed by the subtraction. The two operations for C3 = 1 are combined in one step in an PST-LFSR operation.

Software Function of Post-Multiply LFSR

It is easy to implement the PST-LFSR operation by software. Figure 5 shows an example C-like function code to compute the CRC value for the message bits using the integers as binary strings.

```
crc_pst_lfsr (gen, genlen, ini, msg, msglen)
{
   // gen       = generator bits
   // genlen    = # generator bits
   // ini       = initial CRC bits
   // msg       = message bits
   // msglen    = # message bits
   crc = ini;                    // initial value
   xms = msg << genlen;          // extended message
   xml = msglen + genlen; // extended message  length
   s = genlen - 1;     // shift count for top crc bit
   p = xml - 1;        // shift count for message bit
   for (i = 0; i < xml; i ++) {
      b = (xms >> p) & 0x1;      // message bit
      t = (crc >> s) & 0x1;      // top crc bit
      crc = (crc << 1) | b;      // for top crc bit = 0
      if (t == 1) crc ^= gen;    // for top crc bit = 1
      p -= 1;
   }  // for i
   msk = (1 << genlen) - 1;  //  crc mask
   crc &= msk;
   return crc;
}
```

Figure 5: An Example PST-LFSR Function Code.

If this function is executed for gen = 1001, genlen = 4, ini = 0, msg = 10101011, and msglen = 8, the following result is obtained:

$$1101 = \text{crc\_pst\_lfsr}(1001, 4, 0, 10101011, 8)$$

LINEAR FEEDBACK SHIFT REGISTER ALGORITHM 2 (PRE-MULTIPLY)

Pre-Multiply LFSR

The pre-multiply LFSR (PRE-LFSR) is the second type of LFSR, which updates the CRC value for each new message bit.

Let the generator polynomial G(x) of degree 4 be:

$$G(x) = x^4 + G_3\,x^3 + G_2\,x^2 + G_1\,x^1 + G_0\,x^0$$

where G_i = 0 or 1 for i = 0, 1, 2, 3.

Assume that the 4-bit CRC value $R_n(x)$ for the n-bit message $M_n(x)$ is:

$$R_n(x) = C_3\,x^3 + C_2\,x^2 + C_1\,x^1 + C_0\,x^0.$$

The extended message $E_n(x)$ for $M_n(x)$ is given by:

$$E_n(x) = x^4\,M_n(x) = Q_n(x)\,G(x) + R_n(x)$$

where $Q_n(x)$ is the quotient of the polynomial division $E_n(x)/G(x)$.

Consider the (n+1)-th bit B following $M_n(x)$. The (n+1)-bit message polynomial $M_{n+1}(x)$ is given by:

$$M_{n+1}(x) = M_n(x)x + B\,.$$

The new extended message $E_{n+1}(x)$ is:

$$\begin{aligned} E_{n+1}(x) &= x^4\, M_{n+1}(x) = x^4\, M_n(x)x + Bx^4 \\ &= E_n(x)x + Bx^4\,. \end{aligned}$$

With $E_n(x) = Q_n(x)\, G(x) + R_n(x)$:

$$\begin{aligned} E_{n+1}(x) &= (Q_n(x)\, G(x) + R_n(x))x + Bx^4 \\ &= Q_n(x)\, G(x)x + R_n(x)x + Bx^4\,. \end{aligned}$$

Let $R_{n+1}(x)$ be the new CRC value (remainder) for $E_{n+1}(x)$. Then:

$$\begin{aligned} R_{n+1}(x) &= E_{n+1}(x) \bmod G(x) \\ &= R_n(x)x + Bx^4 \bmod G(x)\,. \end{aligned}$$

Since:

$$\begin{aligned} x^4 \bmod G(x) &= G(x) + G_3\, x^3 + G_2\, x^2 + G_1\, x^1 + G_0\, x^0 \bmod G(x) \\ &= G_3\, x^3 + G_2\, x^2 + G_1\, x^1 + G_0\, x^0\,, \end{aligned}$$

the following is obtained:

$$\begin{aligned} R_{n+1}(x) &= R_n\,(x)x + Bx^4 \bmod G(x) \\ &= (C_3\, x^3 + C_2\, x^2 + C_1\, x^1 + C_0\, x^0)x + Bx^4 \bmod G(x) \\ &= (C_3 + B)x^4 + C_2\, x^3 + C_1\, x^2 + C_0\, x^1 \bmod G(x) \\ &= (C_3 + B)(G_3\, x^3 + G_2\, x^2 + G_1\, x^1 + G_0\, x^0) + C_2\, x^3 \\ &\quad + C_1\, x^2 + C_0\, x^1 \bmod G(x) \\ &= K_3\, x^3 + K_2\, x^2 + K_1\, x^1 + K_0\, x^0\,, \end{aligned}$$

where

$$K_3 = (C_3 + B)G_3 + C_2\,,$$

$$K_2 = (C_3 + B)G_2 + C_1\,,$$

$$K_1 = (C_3 + B)G_1 + C_0\,,$$

$$K_0 = (C_3 + B)G_0\,.$$

For $G(x) = G_1(x) = x^4 + x^3 + 1 = 11001$ ($G_3 = 1$, $G_2 = 0$, $G_1 = 0$, $G_0 = 1$):

$$K_3 = C_3 + C_2 + B\,,$$

$$K_2 = C_1\,,$$

$$K_1 = C_0\,,$$

$$K_0 = C_3 + B\,.$$

The coefficients K_3, K_2, K_1, and K_0 determine the new CRC value with a new input bit B.

Therefore, for a given generator polynomial G(x), the new CRC value $R_{n+1}(x)$ can be calculated from the previous CRC value $R_n(x)$ calculated for the n-bit message $M_n(x)$ when a new bit B follows the message.

Pre-Multiply LFSR Operations

Create the pre-multiply LFSR (PRE-LFSR1) for $G_1(x)$ = 1101. The LFSR configuration is based on the CRC coefficients K_3, K_2, K_1, and K0, and is shown in Figure 6.

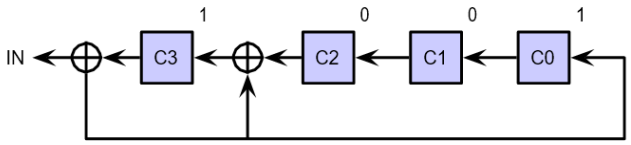

Figure 6: Pre-Multiply LFSR for G1 = 1101 (PRE-LFSR1).

Table 8 shows the PRE-LFSR operations.

Table 8: PRE-LFSR1 Operations

Cycle	IN	C3	C2	C1	C0
N	X	D	C	B	A
N+1	Y	E=X⊕D⊕C	B	A	X⊕D
N+2	Z	Y⊕E⊕B	A	X⊕D	Y⊕E

Run this LFSR by feeding the 8 bits of the message M_1 = 10101011. After 8 cycles, the CRC value (1101) is obtained as shown in Figure 7.

Cycle	C3–0	IN
0	0000	10101011
1	1001	0101011
2	1011	101011
3	0110	01011
4	1100	1011
5	1000	011
6	1001	11
7	0010	1
8	1101	

Figure 7: PRE-LFSR1 Operations for M_1 = 10101011.

After 8 cycles, the CRC value is left in the C_3, C_2, C_1, and C_0 registers: C_1 = 1101. Therefore:

$$\begin{aligned} C_1 = 1101 &= \text{PRE-LFSR1}(G_1, 0, M_1) \\ &= \text{PRE-LFSR1}(11001, 0, 10101011). \end{aligned}$$

Software Function of Pre-Multiply LFSR

This LFSR operation can also be implemented by software. Figure 8 shows an example C-like function for the pre-multiply LFSR.

```
crc_pre_l fsr (gen, genlen, ini, msg, msglen)
{
   // gen    = generator bits
   // genlen = # generator bits
   // ini    = initial CRC bits
   // msg    = message bits
   // msglen = # message bits
   crc = ini;        // initial value
   s = genlen - 1;  // shift count for top crc bit
   p = msglen - 1;  // shift count for message bit
   for (i = 0; i < msglen; i++) {
      b = (msg >> p) & 0x1;
```

```
      t = (crc >> s) & 0x1; t ^= b;
      crc = crc << 1;
      if (t == 1) crc ^= gen; p -= 1;
   }  // for i
   msk = (1 << genlen) - 1;  //  crc mask
   crc &= msk;
   return crc;
}
```

Figure 8: An Example PRE-LFSR Function Code.

If this function is executed for gen = 1001, genlen = 4, ini = 0, msg = 10101011, and msglen = 8, the following result is obtained:

$$1101 = \text{crc\_pre\_lfsr}(1001, 4, 0, 10101011, 8)\ .$$

LFSR OPERATIONS WITH NON-ZERO INITIAL VALUES

Non-Zero Initial Values

The post- and pre-multiply LFSR operations with the zero initial value (CRC) have been described previously. For the example polynomials, the following operations have been shown:

$$C_1 = \text{PST-LFSR1}(G_1, 0, E_1)\ ,$$
$$C_1 = \text{PRE-LFSR1}(G_1, 0, M_1)\ ,$$

where

$$G_1 = 11001\ ,$$
$$M_1 = 10101011\ ,$$
$$E_1 = 10101011\_0000\ ,$$
$$C_1 = 1101.$$

Often, a non-zero initial value I is used to compute the CRC for the input message.

For any non-zero initial value I, the initialization bit sequence (InitSeq) can be found to create I for a PST-/PRE-LFSR.

If an LFSR starts with a non-zero initial value I and produces CRC = C for a message M , then the LFSR starts with a zero initial value and produces the same CRC for the InitSeq for I followed by the message M. This is illustrated in Figure 9.

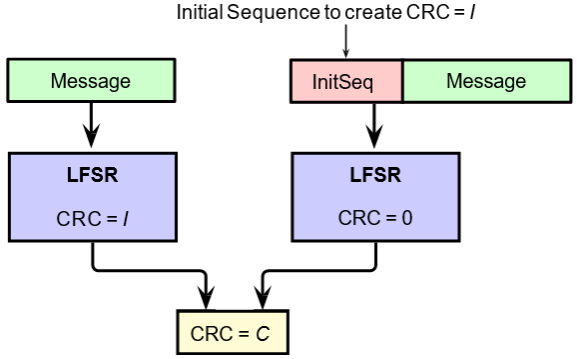

Figure 9: LFSR with Non-Zero Initial Value.

Non-Zero Initial Values in Post-Multiply LFSR

The structure of the general 4-bit PST-LFSR is shown in Figure 10. From this structure, an input bit from IN is shifted left while C3 = 0. Therefore, any 4-bit value can be shifted into the CRC registers.

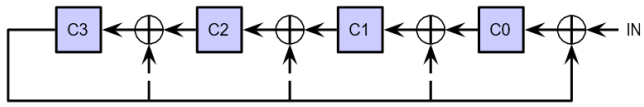

Figure 10: General 4-Bit Post-Multiply LFSR Structure.

Let the 4-bit PST-LFSR1 be the post-multiply LFSR created for G_1 = 11001. Run this LFSR with the initial value 1011 for E_1 = 10101011_0000. The result is 1001 as shown in Figure 11.

Cycle	C3-0	IN
0	1011	10101011_0000
1	1110	0101011_0000
2	0101	101011_0000
3	1011	01011_0000
4	1111	1011_0000
5	0110	011_0000
6	1100	11_0000
7	0000	1_0000
8	0001	0000
9	0010	000
10	0100	00
11	1000	0
12	1001	

Figure 11: PST-LFSR Operations with Initial Value 1011.

For the PST-LFSR, the initialization bit sequence for the initial value of 1011 is the same value as the initial value (1011).

If the same PST-LFSR1 is run with the zero initial value for the initialization bit sequence 1011 followed by E_1 = 10101011_0000, the same CRC 1001 is obtained as shown in Figure 12.

Cycle	C3-0	IN
0	0000	1011_10101011_0000
1	0001	011_10101011_0000
2	0010	11_10101011_0000
3	0101	1_10101011_0000
4	1011	10101011_0000
5	1110	0101011_0000
6	0101	101011_0000
7	1011	01011_0000
8	1111	1011_0000
9	0110	011_0000
10	1100	11_0000
11	0000	1_0000
12	0001	0000
13	0010	000
14	0100	00
15	1000	0
16	1001	

Figure 12: PST-LFSR1 Operations with Initial Value 0000.

Note that the CRC value in the PST-LFSR1 is 1011 after 4 cycles of 1011.

Run the PRE-LFSR1 created for G_1 = 11001 for M_1 = 10101011 after feeding the initialization sequence 1011. The result is the same CRC 1001 as shown in Figure 13.

Cycle	C3–0	IN
0	0000	1011_10101011
1	1001	011_10101011
2	1011	11_10101011
3	0110	1_10101011
4	0101	10101011
5	0011	0101011
6	0110	101011
7	0101	01011
8	1010	1011
9	0100	011
10	1000	11
11	0000	1
12	1001	

Figure 13: PRE-LFSR1 Operations with Initialization Sequence 1011.

Non-Zero Initial Values in Pre-Multiply LFSR

It is not obvious to find the initialization sequence for a particular initial value for the pre-multiply LFSR because of its structure. The easiest way is to create a table by running the pre-multiply LFSR for all the possible initialization sequences. There are 24 initialization sequences for a 4-bit pre-multiply LFSR, as shown in Table 9.

Table 9: Initialization Sequences of Pre-Multiply LFSR for G_1 = 11001.

Init Seq	CRC	Init Seq	CRC
0000	0000	1000	0111
0001	1001	1001	1110
0010	1011	1010	1100
0011	0010	1011	0101
0100	1111	1100	1000
0101	0110	1101	0001
0110	0100	1110	0011
0111	1101	1111	1010

If the PRE-LFSR1 is run for G_1 = 11001 with the initial value 1111, the CRC of 0001 is obtained, as shown in Figure 14.

Cycle	C3–0	IN
0	1111	10101011
1	1110	0101011
2	0101	101011
3	0011	01011
4	0110	1011
5	0101	011
6	1010	11
7	0100	1
8	0001	

Figure 14: PRE-LFSR1 with Initial Value 1100.

FromTable 9, the initialization sequence for the initial CRC value of 1111 is 0100. If the PST-LFSR1 created for G1 = 11001 is run for the input string 0100_10101011_0000 with the initial value 0000, the same CRC value of 0001 is obtained as shown in Figure 15.

Cycle	C3-0	IN
0	0000	0100_10101011_0000
1	0000	100_10101011_0000
2	0001	00_10101011_0000
3	0010	0_10101011_0000
4	0100	10101011_0000
5	1001	0101011_0000
6	1011	101011_0000
7	1110	01011_0000
8	0101	1011_0000
9	1011	011_0000
10	1111	11_0000
11	0110	1_0000
12	1101	0000
13	0011	000
13	0110	00
14	1100	0
15	0001	

Figure 15: PST-LFSR1 Operations with Initialization Sequence 0100.

TABLE LOOKUP ALGORITHMS

If a message is composed of multiple words of the CRC length, a table lookup method can be used. There are two table methods: large and small. The tables are generated by running the PST-LFSR created for a given generator string.

Use the example set of polynomials:

$$G_1 = 11001\,,$$
$$M_1 = 10101011\,,$$
$$E_1 = 10101011\_0000\,.$$

The CRC length of the generator polynomial G_1 = 11001 is 4. Assume that a message is composed of multiple words of 4 bits.

Large Table Method

The LRG-TABLE is created by running a PST-LFSR for 16 initial CRC values and 16 message input values. The LRG-TABLE1 created for G_1 = 11001 is shown in Table 10.

Using this LRG-TABLE1, the new 4-bit CRC (NewCrc) can be found from the old 4-bit CRC (OldCrc) and the 4-bit message input (Input) as follows:

$$\mathrm{Index} = (\mathrm{OldCrc} << 4) \mid \mathrm{Input}\,;$$
$$\mathrm{NewCrc} = \mathrm{LRG-TABLE}[\mathrm{Index}]\,;$$

The 4-bit OldCrc and 4-bit Input make an 8-bit index to the LRG-TABLE1. The entry at the index provides a new 4-bit CRC value. Note that the OldCrc value determines the row position and the input determines the column position in the LRG-TABLE1.

Table 10: LRG-TABLE1 created for G1 = 11001.

Old Crc	Input															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
2	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
3	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
5	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
6	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
7	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
9	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
10	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
11	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
12	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
13	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
14	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
15	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5

For the input message E_1 = 1010_1011_0000, the CRC C1 = 1101 (13) is obtained as follows:

Table 11: Example LRG-TABLE1 operations.

Step	OldCrc	Input	NewCrc
1	0	10	10
2	10	11	7
3	7	0	13

Small Table Method

The large table method requires a table with 2^{2g} entries where g is the degree of the generator polynomial, which is equal to the CRC bit length.

The small table method (SML-TABLE) requires a small table with only 2^g entries. The SML-TABLE is created with the 2^g entries in the LRG-TABLE with Input = 0.

Table 12 shows the SML-TABLE1 entries created from the LRG-TABLE1 entries.

Using this SML-TABLE1, the new 4-bit CRC (NewCrc) can be obtained from the old 4-bit CRC (OldCrc) and the 4-bit message input (Input) as follows:

$$\mathrm{Crc} = \mathrm{SML{-}TABLE}\ [\ \mathrm{OldCrc}\ ]\ ;$$

$$\mathrm{NewCrc} = \mathrm{Crc} + \mathrm{Input}\ ;$$

where + is the bitwise exclusive OR operator.

Table 13 shows the SML-TABLE1 operations for E_1 = 1010_1011_0000, which produces the same CRC value of 1101 (13).

Table 12: SML-TABLE for SENT4 CRC.

OldCrc	Crc
0	0
1	9
2	11
3	2
4	15
5	6
6	4
7	13

OldCrc	Crc
8	7
9	14
10	12
11	5
12	8
13	1
14	3
15	10

Table 13: SML-TABLE1 operations.

Step	OldCrc	Crc	Input	NewCrc
1	0	0	10	10
2	10	12	11	7
3	7	13	0	13

ALLEGRO SENSOR CRCS FOR MANCHESTER AND SPI

Allegro sensors use CRC in Manchester and SPI communications.

Manchester CRC

The generator polynomial and initial value of the Manchester CRC are defined for PRE-LFSR as follows:

Generator Polynomial: $GMAN(x) = x^3 + x + 1 = 1011$,
Initial CRC Value: 111.

The PRE-LFSR for this generator polynomial is shown in Figure 16.

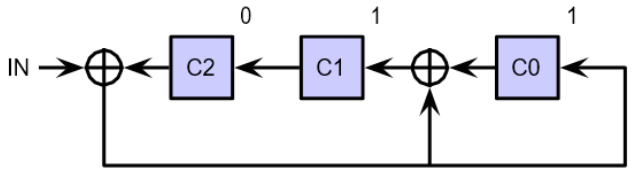

Figure 16: PRE-LFSR for GMAN = 1011.

The code for this PRE-LFSR is shown below:

```
manchester_crc(msg, len)
{
   crc = crc_pre_lfsr (011, 3 , 111, msg, len);
   return crc;
}
```

Figure 17: An Example PRE-LFSR Function Code for Manchester CRC.

Table 14 shows the Manchester CRCs for different messages.

Table 14: Manchester CRCs for Different Messages.

Message	CRC
101	010
101_111	100
101_111_110	110
101_111_110_011	100

SPI CRC

The generator polynomial and initial value of the SPI CRC are defined for the PRE-LFSR as follows:

Generator Polynomial: $GSPI(x) = x^4 + x + 1 = 10011$,
Initial Value: 1111 .

The PRE-LFSR for this generator polynomial is shown in Figure 18.

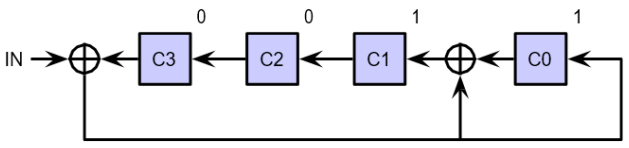

Figure 18: PRE-LFSR for GSPI = 10011.

The code for this PRE-LFSR is shown below:

```
spi_crc(msg, len)
{
   crc = crc_pre_lfsr (0011, 4, 1111, msg, len);
   return crc;
}
```

Figure 19: An Example PRE-LFSR Function Code for SPI CRC.

Table 15 shows the SPI CRCs for different messages.

Table 15: SPI CRCs for Different Messages.

Message	CRC
1001	1110
1001_1101	0101
1001_1101_1000	0100
1001_1101_1000_1101	1000

ALLEGRO SENSOR CRCS FOR SENT OUTPUTS

SENT 4-Bit CRC

The generator polynomial and initial value of the 4-bit SENT CRC are defined for the PST-LFSR as follows:

Generator Polynomial:
$G_{SENT4}(x) = x^4 + x^3 + x^2 + 1 = 11101$,
Initial Value: 0101 .

The PST-LFSR for G_{SENT4} is shown in Figure 20.

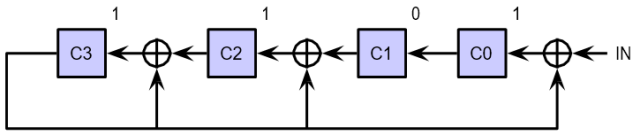

Figure 20: PST-LFSR for GSENT4 = 11101.

The code for this PST-LFSR is shown below.

```
sent4_crc(msg, len)
{
   crc = crc_pst_lfsr(1101, 4 , 0101, msg, len);
   return crc;
}
```

Figure 21: An Example PRE-LFSR Function Code for SPI CRC.

In the SENT communication, the message is composed of multiple 4-bit nibbles. Therefore the large and small table methods can be used.

The LRG-TABLE for this SENT4 CRC is shown in Table 16.

The SML-TABLE for this SENT4 CRC is shown in Table 17.

Table 16: LRG-TABLE for SENT4 CRC.

Old Crc	Input															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
2	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
3	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
4	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
5	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
6	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
7	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
8	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
9	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
10	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
11	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
13	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
14	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
15	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10

Table 17: SML-TABLE for SENT4 CRC.

OldCrc	Crc
0	0
1	13
2	7
3	10
4	14
5	3
6	9
7	4

OldCrc	Crc
8	1
9	12
10	6
11	11
12	15
13	2
14	8
15	5

Table 18 shows the SENT4 CRCs for different messages.

Table 18: SENT4 CRCs for Different Messages.

Message	CRC
0101_0011_1110_0101_0011_1110	1111
0111_0100_1000_0111_0100_1000	0011
0100_1010_1100_0100_1010_1100	1010
0111_1000_1111_0111_1000_1111	0101
1001_0001_1101_1001_0001_1101	0110
0000_0000_0000_0000_0000_0000	0101

SENT 6-Bit CRC

The generator polynomial and initial value of the 6-bit SENT CRC are defined for the PST-LFSR as follows:

Generator Polynomial:

$$G_{SENT6}(x) = x^6 + x^4 + x^3 + 1 = 1011001\ ,$$

Initial Value: 010101 .

The PST-LFSR for G_{SENT6} is shown in Figure 22.

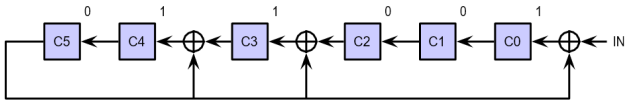

Figure 22: PST-LFSR for G_{SENT6} = 1011001.

The code for this PST-LFSR is shown below.

```
sent6_crc(msg, len)
{
   crc = crc_pst_lfsr(011001, 6, 010101, msg, len);
   return crc;
}
```

Figure 23: An Example PRE-LFSR Function Code for SENT6 CRC.

The LRG-TABLE for G_{SENT6} is too large to print. The SML-TABLE for G_{SENT6} is shown in Table 19.

Table 19: SML-TABLE for SENT6 CRC.

OldCrc	Crc	OldCrc	Crc	OldCrc	Crc	OldCrc	Crc
0	0	16	31	32	62	48	33
1	25	17	6	33	39	49	56
2	50	18	45	34	12	50	19
3	43	19	52	35	21	51	10
4	61	20	34	36	3	52	28
5	36	21	59	37	26	53	5
6	15	22	16	38	49	54	46
7	22	23	9	39	40	55	55
8	35	24	60	40	29	56	2
9	58	25	37	41	4	57	27
10	17	26	14	42	47	58	48
11	8	27	23	43	54	59	41
12	30	28	1	44	32	60	63
13	7	29	24	45	57	61	38
14	44	30	51	46	18	62	13
15	53	31	42	47	11	63	20

Table 20 shows the SENT6 CRCs for different messages.

Table 20: SENT6 CRCs for Different Messages.

Message	CRC
000000_000000_000000_000000	100110
111111_111111_111111_111111	000010
010101_010101_010101_010101	001101
101010_101010_101010_101010	101001
010100_111110_010100_111110	000010
011101_001000_011101_001000	010110
010010_101100_010010_101100	100101
011110_001111_011110_001111	111011

CRC ERROR COVERAGE

In the digital data communication with CRC, the sender sends the transmission data D with the message M and its CRC C to the receiver, and the receiver checks the received transmission data D′ with the message M' and CRC C' to see if C' is valid for M'.

Consider a generator polynomial of degree g, which generates a g-bit CRC for a message. The LFSR created for this generator polynomial generates 2^g unique g-bit CRCs for 2^g different g-bit messages. If this LFSR runs for all the possible (n+g)-bit messages, it will generate the same g-bit CRC value for 2^n different (n+g)-bit messages.

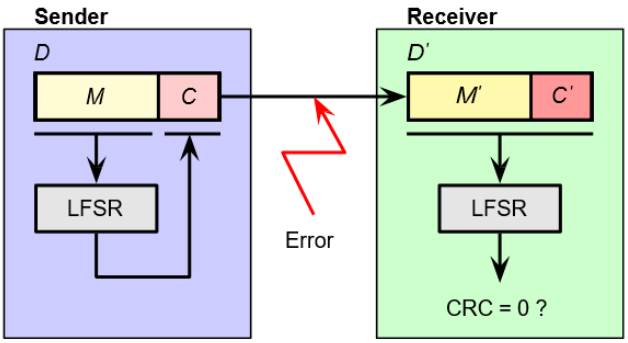

Figure 24: Communication Error Detection with CRC.

CRC error coverage R is defined by:

$$R = \frac{D}{T} = 1 - \frac{U}{T}$$

where

$T = \text{Total \# Error Messages}$,
$U = \text{\# Undetectable Error Messages}$,
$D = T - U = \text{\# Detectable Error Messages}$.

Note that the undetectable error messages are the messages that are not correct but accepted as good.

For the g-bit CRC and (n+g)-bit messages ($n \geq 0$):

$$\begin{aligned} T &= 2^{n+g} - 1, \\ U &= 2^n - 1, \\ D &= T - U = 2^{n+g} - 2^n, \\ R &= \frac{D}{T} = \frac{2^{n+g} - 2^n}{2^{n+g} - 1} = 1 - \frac{2^n - 1}{2^{n+g} - 1}. \end{aligned}$$

Table 21: CRC Error Coverage R (%) for $3 \leq g \leq 12$.

	n + g									
g	3	4	5	6	7	8	9	10	11	12
3	88.89	94.49	97.25	98.63	99.32	99.66	99.83	99.91	99.96	99.98
4	–	94.12	97.06	98.53	99.27	99.63	99.82	99.91	99.95	99.98
5	–	–	96.97	98.49	99.24	99.62	99.81	99.91	99.95	99.98
6	–	–	–	98.46	99.23	99.62	99.81	99.90	99.95	99.98
7	–	–	–	–	99.22	99.61	99.81	99.90	99.95	99.98
8	–	–	–	–	–	99.61	99.81	99.90	99.95	99.98
9	–	–	–	–	–	–	99.81	99.90	99.95	99.98
10	–	–	–	–	–	–	–	99.90	99.95	99.98
11	–	–	–	–	–	–	–	–	99.95	99.98
12	–	–	–	–	–	–	–	–	–	99.98

Assuming $2^{n+g} \gg 1$, the approximate CRC coverage can be obtained by:

$$R = 1 - \frac{2^{-g} - 2^{-(n+g)}}{1 - 2^{-(n+g)}} \approx 1 - 2^{-g}.$$

From this approximation, a better CRC error coverage can be obtained by using a large g value (a large generator polynomial) and/or the long message.

The above CRC error coverage calculation assumes that the CRC value is not changed. But a communication error may change the CRC value, too.

If a communication error changes the CRC value, then the following CRC error coverage R' is obtained:

$$\begin{aligned} T &= 2^{n+g} - 1, \\ U' &= 2^{n}, \\ D' &= T - U' = 2^{n+g} - 2^{n} - 1, \\ R' &= \frac{D'}{T} = \frac{2^{n+g} - 2^{n} - 1}{2^{n+g} - 1} = 1 - \frac{2^{n}}{2^{n+g} - 1}. \end{aligned}$$

Assuming $2^{n+g} \gg 1$, then:

$$R' \approx 1 - 2^{-g},$$

which is the same approximation of R.

Table 22: Approximate Error Coverages.

Generator Polynomial Degree (g)	Approximate Error Coverage (%)
3	87.50
4	93.75
5	96.88
6	98.44
7	99.22
8	99.61
9	99.80
10	99.90
11	99.95
12	99.98

CONCLUSION

In this note, the following topics have been covered:

- The CRC algorithm is based on the polynomial division over GF(2), which uses bitwise logical exclusive OR for addition/subtraction and bitwise logical AND for multiplication;
- The CRC algorithm, characterized by the generator polynomial of g degree, divides the message polynomial by the generator polynomial and finds its remainder as the g-bit CRC value;
- The hand calculation algorithm is based on the straightforward paper-and-pencil method to calculate the CRC value by the polynomial division;
- The CRC value can be calculated using the post-multiply or pre-multiply linear feedback shift register algorithm (PST-LFSR or PRE-LFSR);
- The PST-LFSR algorithm, starting with the initial CRC value, runs for each of the message bits followed by g zero bits;
- The PRE-LFSR algorithm, starting with the initial CRC value, runs for each of the message bits;
- If the message consists of groups of g bits, the CRC value can be calculated using the large or small lookup table algorithm (LRG-TABLE or SML-TABLE);
- The LRG-TABLE algorithm finds the new CRC value from the old CRC value and the input bits for each of the message bit groups followed one group of g zero bits;
- The SML-TABLE algorithm calculates the new CRC value by taking the bitwise logical exclusive OR on the table entry indexed by the old CRC value and the input bits for each of the message bit groups followed one group of g zero bits;
- The calculations of the Manchester and SPI CRC used in the Allegro sensor ICs can be done by any of the CRC algorithms; and
- The calculations of the 4- and 6-bit SENT CRC used in the Allegro sensor ICs can also be done by any of the CRC algorithms.

REFERENCES

[1] Allegro MicroSystems. A1333 Precision, High Speed, Hall-Effect Angle Sensor IC. Datasheet, September 25, 2017.

[2] Allegro MicroSystems. A1335 Precision Hall-Effect Angle Sensor IC. Datasheet, Revision 3, July 5, 2016.

[3] Allegro MicroSystems. A1337 Precision, Hall-Effect Angle Sensor IC. Datasheet, Revision 3, January 24, 2018.

[4] Allegro MicroSystems. A1339 Precision, High Speed, Hall-Effect Angle Sensor IC. Datasheet, September 25, 2017.

[5] Allegro MicroSystems. A1346 Dual-Die Programmable Linear Hall IC. Datasheet, Revision 2, September 20, 2017.

[6] Allegro MicroSystems. A1363 Low Noise, High Precision, Programmable Linear Hall Effect Sensor IC. Datasheet, Revision 6, December 16, 2015.

[7] Allegro MicroSystems. A1367 Low-Noise, High-Precision, Programmable Linear Hall-Effect Sensor IC, Datasheet, Revision 4, November 13, 2017.

[8] SAE Intentional. SENT – Single Edge Nibble Transmission for Automotive Applications. Surface Vehicle Information Report, J2716-APR2016, April 2016.

[9] W. W. Peterson and D. T. Brown. “Cyclic Codes for Error Detection,” Proceedings of the IRE, January 1961, pp.228–235.

[10] T. V. Ramabadran and S. S. Gaitonde. “A Tutorial on CRC Computations,” IEEE Micro, August 1988, pp.62–75.

[11] R. N. Williams. “A Painless Guide to CRC Error Detection Algorithms,” Internet Document, August 19, 1993.

[12] A. S. Tanenbaum and D. J. Wetherall. “Error Detecting Codes.” In Computer Networks. 5th Ed. Prentice-Hall, 2011, pp.209–215.

[13] H. S. Warren, Jr. “Cyclic Redundancy Check.” In Hacker’s Delight, 2nd Ed. Addison-Wesley, 2013, pp.319–330.

[14] M. Paulitsch, J. Morris, B. Hall, K. Driscoll, E. Latronico, and P. Koopman. “Coverage and the Use of Cyclic Redundancy Codes in Ultra-Dependable Systems”. In Proceedings of the 2005 International Conference on Dependable Systems and Networks, June/July 2005, pp.346–355.

APPENDIX: MORE EXAMPLE CRCS

Example 1 CRC

Table 23: CRCs for G1 = 11001 with Initial CRC = 1101.

Message	CRC
1011	1100
1011_0101	1110
1011_0101_1110	0000
1011_0101_1110_0010	1011
1011_0101_1110_0010_0110	0001
1011_0101_1110_0010_0110_1101	1000

Example 2 CRC

Table 24: CRCs for G2 = 11011 with Initial CRC = 1011.

Message	CRC
1101	1001
1101_0110	0101
1101_0110_1101	0010
1101_0110_1101_0101	0111
1101_0110_1101_0101_0111	0000
1101_0110_1101_0101_0111_1000	0010

Example 3 CRC

Table 25: CRCs for G3 = 1100101 with Initial CRC = 101001.

Message	CRC
101010	010001
101010_010101	111011
101010_010101_110011	010011
101010_010101_110011_000111	011101
101010_010101_110011_000111_011001	111011
101010_010101_110011_000111_011001_100110	101011

Manchester CRC

Table 26: CRCs for GMAN = 1011 with Initial CRC = 111.

Message	CRC
100	001
100_101	111
100_101_010	100
100_101_010_110	110
100_101_010_110_011	100
100_101_010_110_011_111	101

SPI CRC

Table 27: CRCs for GSPI = 10011 with Initial CRC = 1111.

Message	CRC
1010	1011
1010_1011	0000
1010_1011_0011	0101
1010_1011_0011_1101	1011
1010_1011_0011_1101_0101	0001
1010_1011_0011_1101_0101_0100	1111

SENT4 CRC

Table 28: CRCs for GSENT4 = 11101 with Initial CRC = 0101.

Message	CRC
0101	1001
0101_1101	1110
0101_1101_0101	1011
0101_1101_0101_1001	0111
0101_1101_0101_1001_0000	0100
0101_1101_0101_1001_0000_1010	1000

SENT6 CRC

Table 29: CRCs for GSENT4 = 1011001 with Initial CRC = 010101.

Message	CRC
110100	110101
110100_010110	010101
110100_010110_011010	110101
110100_010110_011010_000101	100001
110100_010110_011010_000101_100111	001111
110100_010110_011010_000101_100111_111001	101110

Revision History

Number	Date	Description	Responsibility
–	July 9, 2020	Initial release	TakaHide Ohkami

955 PERIMETER ROAD • MANCHESTER, NH 03103 • USA
+1-603-626-2300 • FAX: +1-603-641-5336 • ALLEGROMICRO.COM