JRM Consultants, Inc.

ECC White Paper

Jon Miller

December 2009

Reed-Solomon Error Correction Coding

J. Galstad

www.jrmconsultants.com

Reed-Solomon Error Correction Coding

Julia Galstad, JRM Consultants, Inc.

This document summarizes the well-known algorithms used for error correction coding with Reed-Solomon codes as described in the references.

Contents

1 Galois Fields

The arithmetic used for error correction coding with Reed-Solomon occurs in a Galois field, GF(2^m), a field of order 2^m. The field GF(2^m) is constructed with a primitive polynomial of order m.

1.1 Constructing GF(2^3)

We'll use the primitive polynomial $x^3 + x + 1$ to generate GF(2^3). Start with α. The elements of GF(2^3) in index notation are

$$\{0, 1 = \alpha^0, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}.$$

The polynomial tells us that α satisfies the relationship

$$\alpha^3 + \alpha + 1 = 0.$$

Adding is the same as subtracting in any GF(2^m), so use

$$\alpha^3 = \alpha + 1$$

to change from index notation to polynomial representation of the field elements. The coefficients of the polynomial give the binary representation. See Table 1.

index	polynomial	binary	decimal
0	0	0	0
α^0	1	1	1
α^1	α	10	2
α^2	α^2	100	4
α^3	$\alpha + 1$	11	3
α^4	$\alpha^2 + \alpha$	110	6
α^5	$\alpha^2 + \alpha + 1$	111	7
α^6	$\alpha^2 + 1$	101	5

Table 1: Generating Table for GF(8)

1.2 Adding, Multiplying and Inverting in GF(2^3)

Using the binary representation of field elements, addition is the exclusive or binary operation, "xor." Note that subtraction is the same as addition. See Table 2.

Multiplication is best calculated in index form. Zero times any number is zero. Otherwise, compute

$$\alpha^i \times \alpha^j = \alpha^{(i+j) \pmod{2^3-1}}.$$

For example,

$$\alpha^3 \times \alpha^6 = \alpha^2,$$

since $3 + 6 = 9$, and $9 = 2 \pmod 7$. See Table 3.

The multiplicative inverse of α^i is α^j, where $j \equiv -i \bmod 7$. If i is given in reduced form, then $j = 7 - i$. See Table 4

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

Table 2: Addition Table for GF(8) in Decimal Form

·	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

Table 3: Multiplication Table for GF(8) in Decimal Form

inverse(index)	index	polynomial	bin	dec	inverse(dec)
na	0	0	0	0	na
α^0	α^0	1	1	1	1
α^6	α^1	α	10	2	5
α^5	α^2	α^2	100	4	7
α^4	α^3	$\alpha+1$	11	3	6
α^3	α^4	$\alpha^2+\alpha$	110	6	3
α^2	α^5	$\alpha^2+\alpha+1$	111	7	4
α^1	α^6	α^2+1	101	5	2

Table 4: Table for GF(8) with inverses in left and right columns

2 Encoding

The basic Reed-Solomon (n,k)-code will correct at least t errors, where $n=2^m-1$ and $n-k=2t$. Our example will be a (7,3)-code over GF(2^3), which will correct up to 2 errors.

2.1 The Generator Polynomial

The generator polynomial is

$$g(x)=(x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{2t}).$$

In the (7,3)-code,

$$\begin{aligned} g(x) &= (x+\alpha)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4) \\ &= x^4+3x^3+x^2+2x+3. \end{aligned}$$

2.2 Encoding using Long Division

Given the message polynomial, $M(x)$, calculate the remainder polynomial using the division algorithm. The remainder polynomial, $R_{g(x)}[M(x)x^{n-k}]$ is the remainder when $M(x)x^{n-k}$ is divided by $g(x)$. Then the code word is

$$c(x)=M(x)x^{n-k}+R_{g(x)}(M(x)x^{n-k}).$$

For example, the binary message 011100101 becomes $M(x)=3x^2+4x+5$ in the $(7,3)$-code. Check that:

$$R_{g(x)}[M(x)x^4]=3x^3+2x^2+2x+4$$

and

$$c(x)=3x^6+4x^5+5x^4+3x^3+2x^2+2x+4.$$

3 Decoding

3.1 Error Values and Error Locators

The code word polynomial $c(x)$ was transmitted. The received polynomial, $r(x)$, may be different from $c(x)$, so call their difference the error polynomial, $e(x)$. Then

$$c(x) = r(x) + e(x).$$

Since we are only interested in the nonzero coefficients of e(x), assume that there are ν errors and write

$$e(x) = \sum_{l=1}^{\nu} e_{i_l} x^{i_l}.$$

The values e_{i_l} are *error values*, with the index i_l giving the location of the error. The corresponding *error locator* is

$$X_l = \alpha^{i_l}.$$

3.2 Options for Decoding

3.2.1 Using the Euclidean Algorithm

1. Calculate the syndromes. Form $S(x)$, the syndrome polynomial. If all syndromes are zero, the remaining steps are unnecessary.

2. Run the (Extended) Euclidean Algorithm using the Key Equation to simultaneously calculate $\Lambda(x)$, the error-locator polynomial, and $\Omega(x)$, the error-evaluator polynomial.

3. Use the Chien Search to find the roots of $\Lambda(x)$, the error-locator polynomial. The roots are inverses of the error locators.

4. Use Forney's algorithm to find the error values. Forney's algorithm uses $\Omega(x)$, the formal derivative of $\Lambda(x)$, and the roots of $\Lambda(x)$ (the inverses of the error locators).

5. Use the error locations and error values to form $e(x)$. Then $c(x) = r(x) + e(x)$.

3.2.2 Using the Berlekamp-Massey Algorithm

1. Calculate the syndromes. Form $S(x)$, the syndrome polynomial. If all syndromes are zero, the remaining steps are unnecessary.

2. Run the Berlekamp-Massey Algorithm to find $\Lambda(x)$, the error locator polynomial.

3. Use the Key Equation (and polynomials $\Lambda(x)$ and $S(x)$) to find $\Omega(x)$, the error-evaluator polynomial.

4. Use the Chien Search to find the roots of $\Lambda(x)$, the error-locator polynomial. The roots are inverses of the error locators.

5. Use Forney's algorithm to find the error values. Forney's algorithm uses $\Omega(x)$, the formal derivative of $\Lambda(x)$, and the roots of $\Lambda(x)$ (the inverses of the error locators).

6. Use the error locations and error values to form $e(x)$. Then $c(x) = r(x) + e(x)$.

3.3 Calculating Syndromes

Calculate the syndromes

$$S_i = r(\alpha^i) \qquad \text{for } i = 1, 2, \ldots, 2t.$$

Then the syndrome polynomial is

$$S(x) = S_{2t}x^{2t-1} + \cdots + S_1.$$

[Note: Some would multiply this polynomial by x and declare the result to be their syndrome polynomial. This makes a difference in the algorithms you use.]

Suppose the codeword $c(x)$ of our example is received as

$$r(x) = 3x^6 + 4x^5 + 2x^4 + 3x^3 + 2x^2 + 6x + 4.$$

Then $S_1 = 7$, $S_2 = 3$, and $S_3 = S_4 = 4$, so

$$S(x) = 4x^3 + 4x^2 + 3x + 7.$$

The syndromes and error locators and error values satisfy the *power-sum formulas.*

$$S_j = \sum_{l=1}^{\nu} e_{i_l}(X_l)^j \qquad \text{for } j = 1, 2, \ldots, 2t.$$

3.4 Error Locator Polynomial

For ν errors, the error locator polynomial is

$$\Lambda(x) = \prod_{l=1}^{\nu} (1 - X_l x) .$$

By design, $\Lambda_o = 1$ and the zeros of $\Lambda(x)$ are the reciprocals of the error locators.

The syndromes (S_i) and the coefficients (Λ_i) of the error locator polynomial are related by Newton's identities:

$$\Lambda_\nu S_{j-\nu} + \Lambda_{\nu-1} S_{j-\nu+1} + \cdots + \Lambda_1 S_{j-1} + S_j = 0, \qquad \text{for } 1 \le j - \nu \le 2v.$$

3.5 The Key Equation

The Key Equation and Forney's Algorithm depend on the definition of the syndrome polynomial. Here, we use the definition that agrees with Moon. For a description of both versions, see Clarke.

The Key Equation is

$$\Omega(x) = S(x)\Lambda(x) \pmod{x^{2t}},$$

where $\Omega(x)$ is defined to be the error evaluator polynomial, and

$$S(x) = S_{2t}x^{2t-1} + S_{2t-1}x^{2t-2} + \cdots + S_2 x + S_1.$$

3.6 The Euclidean Algorithm

Use the Euclidean Algorithm to solve the Key Equation,

$$\Omega(x) = S(x)\Lambda(x) \pmod{x^{2t}},$$

for $\Omega(x)$ and $\Lambda(x)$. For the key equation to be satisfied, there must exist an $f(x)$ such that

$$x^{2t} f(x) + S(x)\Lambda(x) = \Omega(x). \tag{1}$$

If we apply the Euclidean Algorithm with to x^{2t} and $S(x)$, we'll find polynomials $h(x)$, $k(x)$, and $\gcd(x^{2t}, S(x))$ such that

$$x^{2t} h(x) + S(x)k(x) = \gcd(x^{2t}, S(x)). \tag{2}$$

Note that Equations 1 and 2 differ by a factor γ, which must be the inverse of the constant term of $y = k(x)$.

In the (7,3)-code example, the Euclidean Algorithm results in the following.

$$\begin{aligned} \Lambda(x) &= 7x^2 + 4x + 1 \\ \Omega(x) &= 2x + 7 \end{aligned}$$

3.6.1 The Euclidean Algorithm Outline

1. Initialize $r_{-1} = x^{2t}$, $r_0 = S(x)$, $y_{-1} = 0$, and $y_0 = 1$.
2. Iterate the following steps starting at $i = 1$, stopping when $\deg(r_i) < t$.

$$\begin{aligned} q_i &= \lfloor r_{i-2}/r_{i-1} \rfloor \\ r_i &= r_{i-2} + q_i r_{i-1} \\ y_i &= y_{i-2} + q_i y_{i-1} \end{aligned}$$

3. Calculate γ: Set γ equal to the inverse of the constant term of y_i.
4. Set

$$\Lambda(x) = \gamma y_i,$$
$$\Omega(x) = \gamma r_i.$$

3.7 The Berlekamp-Massey Algorithm

Newton's identity (Equation 3) describes the output of a linear feedback shift register (LFSR) with coefficients Λ_1, Λ_2,..., Λ_ν.

$$\Lambda_\nu S_{j-\nu} + \Lambda_{\nu-1} S_{j-\nu+1} + \cdots + \Lambda_1 S_{j-1} + S_j = 0 \tag{3}$$

The idea is to successively amend an LSFR to produce the entire sequence of syndromes. This algorithm finds the shortest such LSFR, whose connection polynomial will be the error locator polynomial. At each step k, the discrepancy between the k^{th} syndrome S_k and the k^{th} output of the current LSFR is calculated. Based on the discrepancy, the connection polynomial is updated and the process repeats until the connection polynomial produces the correct sequence of syndromes.

3.7.1 The Berlekamp-Massey Algorithm Outline

The variable k is the algorithm iteration counter, L is the counter for the current length of the LSFR, the polynomial $c(x)$ is the current connection polynomial, the polynomial $p(x)$ is the connection polynomial before the last length change, the variable l is the counter for the amount of length change of the LSFR, the variable d_k is the current computed discrepancy, and e is the previous nonzero discrepancy. The algorithm takes the syndromes S_1, S_2, ..., S_{2t}, as input.

1. Initialize $L = 0$, $c(x) = p(x) = l = e = 1$.

2. For $k = 1$ to $2t$, calculate

$$d_k = S_k + \sum_{i=1}^{L} c_i S_{k-i}.$$

- If $d_k = 0$, set $l = l + 1$.
- If $d_k \neq 0$ and if $2L \geq K$, then set

$$\begin{aligned} c(x) &= c(x) + d_k e^{-1} x^l p(x) \\ l &= l + 1 \end{aligned}$$

- If $d_k \neq 0$ and if $2L \not\geq K$, then set

$$\begin{aligned} p(x) &= c(x) \\ c(x) &= c(x) + d_k e^{-1} x^l p(x) \\ L &= k - L \\ e &= d_k \\ l &= 1 \end{aligned}$$

3. Set the error locator polynomial $\Lambda(x)$ equal to $c(x)$, the final connection polynomial.

3.8 Chien Search

The purpose of the Chien Search is to find the zeros of $\Lambda(x)$, the inverses of the error locators.

In the (7,3)-code example, the Chien Search finds $X_1^{-1} = \alpha^3$ and $X_2^{-1} = \alpha^6$. Taking inverses, we compute $X_1 = \alpha^4$ and $X_2 = \alpha^1$.

3.8.1 The Chien Search Outline

1. Set $\nu =$deg$(\Lambda(x))$.
2. Load ν registers with the coefficients $\Lambda_1, \ldots, \Lambda_\nu$.
3. Add the values of the registers. If the sum is 1, then α^0 is a zero of $\Lambda(x)$.
4. Multiply the first register by α^1, the second by α^2, ..., and the ν^{th} register by α^ν. (Usually, we have arranged for $\alpha = 2$.) Add the values of the registers. If the sum is 1, then α^1 is a zero of $\Lambda(x)$.

5. After checking if α^i is a zero, multiply the first register by α^1, the second by α^2, ..., and the ν^{th} register by α^ν. Add the values of the registers. If the sum is 1, then α^{i+1} is a zero of $\Lambda(x)$. Continue until α^{2^m-2} has been checked.

If α^{i_1} is the first zero found in the Chien Search, set $X_1^{-1} = \alpha^{i_1}$. Set X_2^{-1} to the second zero found, and so on. Invert these elements to find X_1, X_2, ..., X_ν, the error locators.

3.8.2 Detecting Decoding Failure

A decoding failure has occurred if either $\Lambda(x)$ has a double zero, or $\Lambda(x)$ has a zero that is in an extension field of GF(2^m). The Chien Search will detect a decoding failure if the number of distinct zeros found is less than ν, the degree of $\Lambda(x)$.

3.9 Forney's Algorithm

[Forney's Algorithm depends on the choice of definition of the syndrome polynomial and key equation.]

Forney's Algorithm solves for the error values using the equation

$$e_{i_k} = \frac{-\Omega(X_k^{-1})}{\Lambda'(X_k^{-1})},$$

where $\Lambda'(x)$ denotes the formal derivative of $\Lambda(x)$.

In our (7,3)-code example, we first compute $\Lambda'(x) = 4$. Then Forney's Algorithm results in $e_{i_1} = 7$ and $e_{i_2} = 4$.

4 References

Clarke, C.K.P. Reed-Solomon Error Correction: BBC R&D White Paper WHP031. BBC. Retrieved from http://www.bbc.co.uk/rd/pubs/whp/whp031.shtml January 2009.
Moon, T.K. Error Correction Coding. Wiley. Hoboken: 2005.