GScom

Reed-Solomon Codec

System Design

GScom

Reed-Solomon Codec

Table of Contents

1. Introduction

There are two types of errors in communication environment: random errors and burst errors. The Viterbi, Turbo, or LDPC code is good at taking care of the random errors and Reed-Solomon (R-S) code is good at taking care of the burst errors.

The Reed-Solomon code falls in the category of the forward error correction code and it is optimized for the burst error correction rather than random error correction. The Reed-Solomon code provides a compromise between efficiency and complexity, so it can be easily implemented using hardware or FPGA.

GScom has developed a Reed-Solomon codec which provides a real time processing speed by exploiting the system parameter, efficiency and complexity of hardware implementation.

2. Reed-Solomon System Description

Many digital signaling applications in wireless communication systems use a forward error correction (FEC) coding technique where redundant information is added to the signal to allow the receiver to detect and correct the errors that may have occurred in the transmission through a noisy channel. The Reed-Solomon (R-S) code is a FEC code and has been proven to be a good compromise between efficiency and complexity.

R-S codes are non-binary cyclic codes with symbols made up of m-bit sequences, where m is any positive integer having a value greater than 2. This makes the code particularly good at dealing with burst errors. A R-S code is a block code, i.e. the message to be transmitted is divided up into separate blocks of data. Each block of data has parity protection information added to form a self-contained code word.

An R-S code is a systematic code, i.e. the encoding process does not alter the message symbols and the protection symbols are added as a separate part of the block. An R-S code is a linear and cyclic code. Cyclic codes are widely used in data communication systems because their structure allows the encoder and decoder circuits to be relatively simple.

Choosing different parameters for a code provides a different levels of protection and affects the complexity of the implementation. An R-S code can be describe as a (n,k) code, where n is the block length in symbols and k is the number of parity information symbols in the message (or block).

In general,

$$n \leq 2^m - 1,$$

where m is the number of bits in symbol. There are n-k parity symbols and t symbol errors which can be corrected in a block, where

t = (n-k)/2.

2.1 Reed-Solomon Encoder

The values of the message and parity symbols of an R-S coder are the elements of a Galois field. Thus for a code based on m-bits symbols, the Galois field has 2^m elements. GScom uses (255, k, t) R-S code so that the k=255-2t symbols of the input packet will be extended with 2t parity symbols to produce a code block length of 255 symbols. For this code, the Galois filed has 256 (m=8) elements and GScom used the following field generator polynomial to construct the 256 elements of GF(256);

$\mathrm{p(x)} = x^8 + x^4 + x^3 + x^2 + 1$ ---------- (1)

2.1.1 The Code Generator Polynomial

A (n, k) R-S code is constructed by forming the code generator polynomial g(x), consisting of n-k=2t factors, the roots of which are consecutive elements of the Galois field. Choosing the consecutive elements ensures that the distance properties of the code are maximized.

The code generator polynomial of a t-error-correcting R-S code of length 2^m-1 takes the following form;

$g(X) = (X+\alpha)\ (X+\alpha^2)\ \ldots.\ (X+\alpha^{2t})$ ---------- (2)

where α is a primitive element of GF(2^m).

Note that the primitive element α is the root of the code generating polynomial.

2.1.2 Reed-Solomon Encoding

The k information symbols that form the message to be encoded as one block is represented by a polynomial M(x) of order k-1, so that;

$\mathrm{M(x)} = M_{k-1}x^{k-1} + \cdots + M_1x + M_0$, where each of the coefficients M_{k-1} …..M_1, M_0 is an m-bit message symbols and is an element of GF(2^m).

To encode the message, the message polynomial is multiplied by x^{n-k} and the result is divided by the generator polynomial, g(x). Division by g(x) produces a quotient q(x) and a remainder r(x) which is of a degree up to n-k-1. Thus:

$$\frac{M(x)\,x^{n-k}}{g(x)} = q(x) + \frac{r(x)}{g(x)} \quad \text{---------- (3)}$$

Having produced the r(x) by division, the transmitted code word (R-S code word), T(x), can then be formed by combining M(x) and r(x) as following:

$\mathrm{T(x)} = M(x)x^{n-k} + r(x) = M_{k-1}x^{n-1} + \cdots + M_0x^{n-k} + r_{n-k-1}x^{n-k-1} + \cdots + r_0$ -------- (4)

which shows that the R-S code word is produced in the required systematic form.

Adding the remainder, r(x), ensures that the encoded message polynomial will always be divisible by the generator polynomial without a remainder.

2.1.2.1 Implementation of Reed-Solomon Encoder

R-S encoding is obtained by the division of the message polynomial M(x) by generating polynomial g(x) to produce the parity symbols r(x), and then by combining the message and the remainder as seen in the equations (3) and (4). The division of the M(x) by the g(x) can be accomplished by a polynomial division process in the R-S field.

GScom uses a hardware encoder to perform the polynomial long division process. The hardware encoder operates on pipelined data; the division calculation is made using the message symbols one at a time as they are presented. The pipelined division calculation is performed using the conventional linear feedback shift register (LFSR) encoder which enables the encoding to perform real-time processing.

2.2 Reed-Solomon Decoder

Reed-Solomon (R-S) codes are block, systematic, and cyclic code with a finite field of size of (n, k). R-S decoder recovers the transmitted signal, T(x), which was encoded by the R-S encoder from a received signal through a communication channel. The channel may introduce errors, E(x), into the transmitted signal in a communication environment. Thus the received signal, R(x), is given by;

R(x) = T(x) + E(x),

where T(x) is given in equation (4) and

$$\mathrm{E(x)} = E_{n,-1}x^{n-1} + \cdots + E_1 x + E_0 \quad \text{---------- (5)}$$

and each of the coefficients $E_{n-1}, \ldots, E_0$ is an m-bit error symbol represented by an element of the GF(2^m) with the positions of the errors in the code word determined by the degree of x for that term. If more than t=(n-k)/2 of the E values are non-zero, then the capacity of the code is exceeded and the errors are not correctable.

The R-S decoder corrects the error signal if the number of error symbol is less than or equal to t=(n-k)/2 using the following four major steps: 1) calculating the syndromes; 2) finding the error location polynomial; 3) determination of the error location number; 4) calculation of the error values.

2.2.1 Syndrome Calculation

Section 2.1.2 showed that the encoded message polynomial or transmitted code word, T(x), is divisible with the generator polynomial without remainder and that this property extends to the individual factors of the generator polynomial.

The first step of decoding a code is to calculate the syndrome from the received signal R(x). The syndrome can be obtained by dividing the received signal polynomial, R(x), by each of the factors $(\mathrm{x} + \alpha^i)$ of the generator polynomial, g(x), in equation (2). This produces a quotient and a remainder as follows;

$$\frac{\mathrm{R(x)}}{(x+\alpha^i)} = Q_i(x) + \frac{S_i}{(x+\alpha^i)} \qquad for \quad 0 \le i \le 2t-1 \qquad \text{---------- (6)}$$

The remainders, S_i, resulting from these division are known as the syndromes.

The syndromes S_i can be represented as follows by rearranging the equation (6);

$S_i = Q_i \times (x + \alpha^i) + R(x)$ ---------- (7)

The syndrome, S_i, is reduced to the following equation (8) when x = α^i in equation (7);

$S_i = R(\alpha^i) = R_{n-1}(\alpha^i)^{n-1} + R_{n-2}(\alpha^i)^{n-2} + \cdots + R_1\alpha^i + R_0$, for i = 1, 2, …, 2t ---------- (8)

where the coefficients $R_{n-1} \ldots . R_0$ are the symbols of the received code word. The physical meaning of the above equation (8) is that each of the syndrome values can also be obtained by substituting x=α^i in the received signal polynomial, as an alternative to the division of the R(x) by $(x - \alpha^i)$ to form the remainder.

Note also that $R(\alpha^i) = T(\alpha^i) + E(\alpha^i)$ and $T(\alpha^i) = 0$ because $x + \alpha^i$ is a factor of g(x), which is a factor of T(x). As a result

$S_i = R(\alpha^i) = E(\alpha^i)$, for i = 1, 2, …, 2t ---------- (9)

This means that the syndrome values are only dependent on the error pattern and are not affected by data values. In addition, all the syndrome values are zeros when no errors have occurred.

2.2.2 Error Location Polynomial

Assume that E(x) is an error pattern of ν, where $\nu \leq t$, say;

$E(x) = Y_1 x^{e_1} + Y_2 x^{e_2} + \cdots + Y_\nu x^{e_\nu}$ ---------- (10)

where $e_1, \ldots, e_\nu$ are the locations of the errors in the code word as the corresponding powers of x, while $Y_1, \ldots, Y_\nu$ represent the error values at those locations. Substituting the equation (10) into the equation (9) produces;

$S_i = E(\alpha^i) = Y_1\alpha^{ie_1} + Y_2\alpha^{ie_2} + \cdots + Y_\nu\alpha^{ie_\nu} = Y_1x_1^i + Y_2x_2^i + \cdots + Y_\nu x_\nu^i$, for i=1,..,2t -------- (11)

Where $x_1 = \alpha^{e_1}, \ldots x_\nu = \alpha^{e_\nu}$ are known as error locators. Note that the syndromes are written as $S_1, \ldots, S_{2t}$ to correspond with the roots of $\alpha^1, \ldots . \alpha^{2t}$ and the powers of x are dependent on the chosen roots in the generator polynomial of the equation (2).

GScom uses the error location polynomial to represent the number of errors and their locations in the received signal (or polynomial). The error location polynomial is constructed to have ν factors of the form (1+x_jx) and therefore has the inverses $x_1^{-1}, \ldots \ldots, x_\nu^{-1}$ of the ν error locators as its roots;

$\Lambda(x) = (1 + x_1x)(1 + x_2x) \ldots (1 + x_\nu x) = 1 + \Lambda_1 x + \cdots + \Lambda_{\nu-1}x^{\nu-1} + \Lambda_\nu x^\nu$ ---------- (12)

For each error, there is a corresponding root x_j^{-1} that makes $\Lambda(x)$ equal to zero

$$1 + \Lambda_1 x_j^{-1} + \cdots + \Lambda_{\nu-1}x_j^{-\nu+1} + \Lambda_\nu x_j^{-\nu} = 0$$

Multiplying through by $Y_j x_j^{i+\nu}$;

$$Y_j x_j^{i+v} + \Lambda_1 Y_j x_j^{i+v-1} + \cdots + \Lambda_{v-1} Y_j x_j^{i+1} + \Lambda_v Y_j x_j^{i} = 0, \quad for\ i = 1, 2, \ldots, 2t$$

GScom obtains similar equations for all error and by taking the summation of all the terms;

$$\sum_{j=1}^{v} Y_j x_j^{i+v} + \Lambda_1 \sum_{j=1}^{v} Y_j x_j^{i+v-1} + \cdots + \Lambda_v \sum_{j=1}^{v} Y_j x_j^{i} = 0$$

or

$$S_{i+v} + \Lambda_1 S_{i+v-1} + \cdots + \Lambda_v S_i = 0, \quad for\ i = 1, 2, \ldots, 2t \quad \text{---------- (13)}$$

The 2t sets of simultaneous linear equation have ν unknown coefficients of the error location polynomial. The first ν simultaneous equations can be used to find the coefficients of $\Lambda_{1,\ldots,}\Lambda_v$ of the error location polynomial (12). There are several methods to find the coefficients of the error location polynomial. The Berlekamp's algorithm and the Euclidean algorithm are the most efficient algorithms to solve the coefficients of the error location polynomial. The Berlekamp's algorithm is a concise and efficient algorithm to find the coefficients using micro-processor. GScom believes that the Euclidean algorithm is a more efficient algorithm to find the coefficients by implementing with hardware for high speed data and for low-power application.

GScom describes the Berlekamp's algorithm very briefly and the Euclidean algorithm in more detail. GScom will use the Euclidean algorithm to implement the error location polynomial and its related process to implement R-S decoder in a hardware based system for high speed and low power purposes.

2.2.2.1 The Coefficient of the Error Location Polynomial

2.2.2.1.1 Berlekamp's Algorithm

Berlekamp's algorithm is an efficient iterative technique for solving equation (13). This is done by forming an approximation to the error location polynomial, starting with $\Lambda(\mathrm{x}) = 1$. Then at each stage, an error value is formed by substituting the approximate coefficients into the equations corresponding to that value of ν. The error is then used to refine a correction polynomial, which is then added to improve the approximated $\Lambda(\mathrm{x})$. The process ends when the approximate error location polynomial checks consistently with the remaining equations.

2.2.2.1.2 Euclidean Algorithm

The Euclidean algorithm is another efficient algorithm for obtaining the coefficient of the error location polynomial. This algorithm uses the relationship between the errors and the syndromes expressed in the form of an equation based on polynomials. The Euclidean algorithm is also referred to as the key equation and requires two new polynomials, the syndrome and error magnitude polynomials, which will be introduced in the following sub-sections.

Syndrome Polynomial

The syndrome polynomial is defined as

$$S(x) = S_{2t}x^{2t-1} + \cdots + S_2x + S_1$$

where the coefficients are the 2t syndrome values obtained from the received code word using equation (9).

Error Magnitude Polynomial

The error magnitude polynomial is defined as

$$\Omega(x) = \Omega_v x^{v-1} + \cdots + \Omega_2 x + \Omega_1$$

Key Equation

The key equation can be written as;

$$\Omega(x) = [\, S(x)\, \Lambda(x)\,]\ mod\ x^{2t}$$

Where S(x) is the syndrome polynomial and Λ(x) is the error location polynomial. So that

$\Omega_1 = S_1$

$\Omega_2 = S_2 + S_1\Lambda_1$

.

$\Omega_v = S_v + S_{v-1}\Lambda_1 + \cdots + S_1\Lambda_v$

Applying the Euclid's Method to the Key Equation

Euclid's method can find the highest common factor d of two elements a and b, such that:

ua + vb = d ---------- (14)

where u and v are coefficients produced by the algorithm.

The product of S(x) and Λ(x) have degree 2t+v-1. So the product can be expressed as:

S(x) X Λ(x) = $F(x)\ X\ x^{2t} + \Omega(x)$

in which the terms of x^{2t} and above are represented by the F(x) term and the remaining part is represented by $\Omega(x)$. This can be rearranged as:

$S(x)\ X\ \Lambda(x) + F(x)\ X\ x^{2t} = \Omega(x)$. ---------- (15)

so that the known terms S(x) and x^{2t} correspond to the a and b terms in equation (14). The algorithm consists of dividing the x^{2t} by S(x) to produce a remainder. S(x) then becomes the dividend and the remainder becomes the divider to provide a new remainder. This process is

continued until the degree of the remainder becomes less than t. At this point, both the remainder $\Omega(x)$ and the multiplying factor Λ(x) are available as terms in the calculation.

Note that the equation (15) has the same form as of the equation (3).

Solving the Error Location Polynomial – The Chien Search

After obtaining the coefficients values, $\Lambda_1, \ldots, \Lambda_v$, of the error location polynomial, it is possible to find its roots. If the polynomial is written in the form;

$$\Lambda(\mathrm{x}) = x_1(x + x_1^{-1})x_2(x + x_2^{-1}) \ldots$$

then the function value will be zero if x = $x_1^{-1}, x_2^{-2}, \ldots$, that is $\mathrm{x} = \alpha^{-e_1}, \alpha^{-e_2}, \ldots.$.

The roots, and hence the values of $x_1, \ldots.$, x_v, are found by trial and error, known as the Chien search, in which all the possible values of the roots (the field values α^i, $0 \leq \mathrm{i} \leq \mathrm{n} - 1$) are substituted into equation (12) and the results are evaluated. If the expression reduces to zero, then that values of x is a root and identifies the error position.

The Forney's Algorithm for Calculating the Error Values

This is a means of calculating the error value Y_j having established the error location polynomial $\Lambda(\mathrm{x})$ and the error magnitude polynomial, $\Omega(\mathrm{x})$. The algorithm makes use of the derivative of the error location polynomial.

For a polynomial f(x) given by:

$\mathrm{f}(\mathrm{x}) = 1 + f_1 x + f_2 x^2 + \cdots + f_v x^v$

the derivative is given by:

$f'(\mathrm{x}) = f_1 + 2f_2 x + \cdots + v f_v x^{v-1}$

However, for the error location polynomial, $\Lambda(\mathrm{x})$, for $\mathrm{x} = x_j^{-1}$, the derivative reduces to:

$$\Lambda'(x_j^{-1}) = \Lambda_1 + \Lambda_3 x_j^{-2} + \Lambda_5 x_j^{-4} + \cdots$$

which amounts to setting even-powered terms of the error location polynomial to zero and dividing through by $\mathrm{x} = x_j^{-1}$.

Forney's algorithm is very efficient for the calculation of error values $Y_1 \ldots . Y_v$ and is very easily implemented into hardware effectively. According to the Forney's algorithm, the error value is given by:

$$Y_j = x_j^1 \frac{\Omega(x_j^{-1})}{\Lambda'(x_j^{-1})}$$

where $\Lambda'(x_j^{-1})$ is the derivative of $\Lambda(\mathrm{x})$ for $\mathrm{x} = x_j^{-1}$.

Having located the symbols containing errors, identified by x_j through the usage of the Chien Search, and calculated the values of Y_j of those errors, the errors can be corrected by adding the error

polynomial E(x) to the received signal polynomial R(x). It should be noted that conventionally the highest term of the received polynomial corresponds to the first symbol of the received code word.

2.3 Implementation of a Reed-Solomon Decoder

There are several approaches to implement the Reed-Solomon(R-S) decoder. For the high speed backhaul communication applications, GScom decided to implement the hardware based R-S decoder for the purpose of high speed and low power.

The following figure is the arrangement of main units of the R-S decoder.

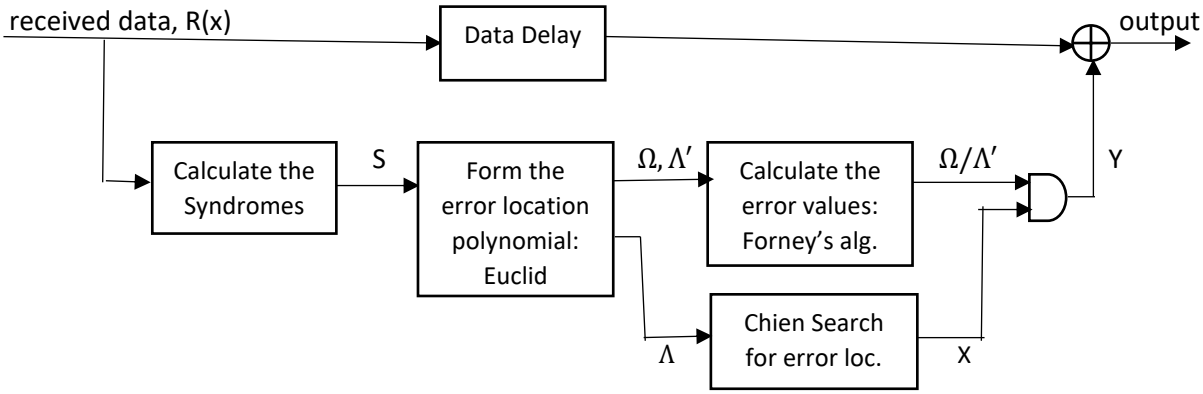

Figure 1. Main Processes of a Reed-Solomon Decoder

The first process in the R-S decoder is to calculate the syndrome values from the incoming code word (received data). These are then used to find the coefficients of the error location polynomial $\Lambda_1, \ldots, \Lambda_v$ and the error value (or magnitude) polynomial $\Omega_1, \ldots, \Omega_v$ using the Euclidean algorithm. The error locations (i.e. positions) are identified by the Chien search and the error values are calculated using Forney's algorithm. As these calculations involve all the symbols of the received code word, it is necessary to store the message (or the received data) until the results of the calculation are available. Then, to correct the errors, each error values is added (modulo 2) to the symbol at the appropriate location in the received data (or code word).

2.3.1 Implementation of Syndrome Calculator

Section 2.2.1 shows that the syndrome corresponding to each root of the α^i of the generator polynomial could be calculated either by dividing the received polynomial R(x) by $x + \alpha^i$, or by evaluating $R(\alpha^i)$.

For the direct division process for the syndrome calculation, GScom noted that the syndrome calculation process of equation (6) has a very similar form as that of the equation (3) which was used for R-S encoding. Following the same reasoning for the R-S encoder implementation, GScom uses a

hardware syndrome calculator to perform the polynomial long division process. The hardware syndrome calculator operates on pipelined data, therefore the division calculation is made using the received symbols one at a time as they are presented. The pipelined division calculation is performed using the conventional linear feedback shift register (LFSR) which enables the syndrome calculator to perform real-time processing.

In the case of polynomial division, there is only one feedback term with one multiplier, α^i, and only one register; GScom uses a single feedback shift register(SFSR) to calculate the syndrome S_i. There is a need to calculate 2t sets of syndromes, $S_i \ for\ i = 1, \dots\ , 2t$. GScom builds a parallel pipelined syndrome circuits for fast speed and low power consumption.

2.3.2 Implementation of Error Location Polynomial using Euclidean Algorithm

As discussed in Section 2.2.2.1.2, the first step to find the error location polynomial is to find the syndrome polynomial. After obtaining the syndrome values S_i, $\mathrm{i} = 1, 2, \dots, 2\mathrm{t}$, the syndrome polynomial can be given as:

$$\mathrm{S(x)} = S_v x^{v-1} + S_{v-1} x^{v-2} + \cdots + S_2 x + S_1$$

As discussed with the equation (15) and the following paragraph in Section 2.2.2.1.2, the Euclidean algorithm consists of dividing the x^{2t} by S(x) to produce a remainder. S(x) then becomes the dividend and the remainder becomes the divider to provide a new remainder. This process is continued until the degree of the remainder becomes less than t. At this point, both the remainder $\Omega(x)$ and the multiplying factor Λ(x) are available as terms in the calculation. Note that the $\Omega(x)$ is the error value (or magnitude) polynomial and Λ(x) is the error location polynomial. Note also that both of $\Omega(x)$ and Λ(x) the output of Euclid algorithm in Figure 1.

Note that equation (15) used in the Euclid algorithm has a similar form to that of the equation (3), which was used in the encoding process. However, there is a difference between the contents of those two equations. Equation (3) has two known information and one unknown which is the remainder of $\mathrm{r(x)}$ and equation (15) has two known information and two unknown information which are the error location polynomial Λ(x) and the error value (or magnitude) polynomial $\Omega(x)$.

For direct division process for the calculation of the error value polynomial, the process of equation (15) has a similar form to that of the equation (3) which was used for R-S encoding. However, Euclid algorithm requires a multiplication process on top of the division process to calculate the error location polynomial. GScom uses a hardware based error value polynomial calculator and hardware based error location polynomial to perform the polynomial long division process and the polynomial multiplication process, respectively. The hardware based error value polynomial calculator operates on pipelined data, so the division calculation is made using the received symbols one at a time as they are presented. The hardware based error location polynomial calculator, also, operates on pipelined data, so the multiplication calculation is made using the received symbols one at a time as they are presented. The pipelined division and multiplication are performed using four conventional single feedback shift registers (SFSR) which enables the Euclidean algorithm to perform a real time processing.

2.3.3 Implementation of Error Location Searcher --- Chien Searcher

After obtaining the coefficients values, $\Lambda_1, \ldots, \Lambda_v$, of the error location polynomial, it is possible to find the roots of the error location polynomial using a Chien searcher. If the polynomial is written in the form:

$$\Lambda(\mathrm{x}) = (1 + x_1 x)(1 + x_2 x) \ldots (1 + x_v x) = 1 + \Lambda_1 x + \cdots + \Lambda_{v-1} x^{v-1} + \Lambda_v x^v \quad \text{---------- (12)}$$

then the function value will be zero if x = $x_1^{-1}, x_2^{-2}, \ldots\ x_v^{-1}$, that is $\mathrm{x} = \alpha^{-e_1}, \alpha^{-e_2}, \ldots . \alpha^{-e_v}$.

The roots, and hence the values of $x_1, \ldots .\ , x_v$, are found by trial and error, known as the Chien search, in which all the possible values of the roots (the field values α^i, $0 \leq \mathrm{i} \leq \mathrm{n} - 1$) are substituted into equation (12) and the results are evaluated. If the expression reduces to zero, then that values of x is a root and identifies the error position.

The Chien searcher consists of the ν-sets of accumulative multipliers and adder. The accumulative multiplier consists of a multiplier and a register to store each term in the error location polynomial. The value of each term in the polynomial is calculated by loading the coefficient value Λ_i and multiplying it by the appropriate power of α. Then, at each successive clock period, the next value of the term is produced by multiplying the previous result by the power of α. Adding the values of the individual terms together produces the value of the polynomial for each symbol position. The detection of zero values of the sum identifies symbol positions where the error occurs.

2.3.4 Implementation of Error Value Calculator

Hardware calculation of the two polynomials, $\Omega(\mathrm{x})$ and $\Lambda'(x)$, can be performed in a similar manner to that of the Chien search, in particular, the function value is calculated for each symbol position in the code word in successive clock periods.

2.3.5 Error Correction

Errors are corrected by adding the error values Y to the received error symbols R at the positions located by the x values.