Reed-Solomon Codes

by

Bernard Sklar

Introduction

In 1960, Irving Reed and Gus Solomon published a paper in the *Journal of the Society for Industrial and Applied Mathematics* [1]. This paper described a new class of error-correcting codes that are now called *Reed-Solomon* (R-S) *codes*. These codes have great power and utility, and are today found in many applications from compact disc players to deep-space applications. This article is an attempt to describe the paramount features of R-S codes and the fundamentals of how they work.

Reed-Solomon codes are *nonbinary cyclic* codes with symbols made up of m-bit sequences, where m is any positive integer having a value greater than 2. R-S (n, k) codes on m-bit symbols exist for all n and k for which

$$0 < k < n < 2^m + 2 \tag{1}$$

where k is the number of data symbols being encoded, and n is the total number of code symbols in the encoded block. For the most conventional R-S (n, k) code,

$$(n, k) = (2^m - 1, 2^m - 1 - 2t) \tag{2}$$

where t is the symbol-error correcting capability of the code, and $n - k = 2t$ is the number of parity symbols. An extended R-S code can be made up with $n = 2^m$ or $n = 2^m + 1$, but not any further.

Reed-Solomon codes achieve the *largest possible* code minimum distance for any linear code with the same encoder input and output block lengths. For nonbinary codes, the distance between two codewords is defined (analogous to Hamming distance) as the number of symbols in which the sequences differ. For Reed-Solomon codes, the code minimum distance is given by [2]

$$d_{\min} = n - k + 1 \tag{3}$$

The code is capable of correcting any combination of t or fewer errors, where t can be expressed as [3]

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor = \left\lfloor \frac{n - k}{2} \right\rfloor \tag{4}$$

where $\lfloor x \rfloor$ means the largest integer not to exceed x. Equation (4) illustrates that for the case of R-S codes, correcting t symbol errors requires no more than $2t$ parity symbols. Equation (4) lends itself to the following intuitive reasoning. One can say that the decoder has $n - k$ redundant symbols to "spend," which is twice the amount of correctable errors. For each error, one redundant symbol is used to locate the error, and another redundant symbol is used to find its correct value.

The erasure-correcting capability, ρ, of the code is

$$\rho = d_{\min} - 1 = n - k \tag{5}$$

Simultaneous error-correction and erasure-correction capability can be expressed as follows:

$$2\alpha + \gamma < d_{\min} < n - k \tag{6}$$

where α is the number of symbol-error patterns that can be corrected and γ is the number of symbol erasure patterns that can be corrected. An advantage of nonbinary codes such as a Reed-Solomon code can be seen by the following comparison. Consider a binary $(n, k) = (7, 3)$ code. The entire n-tuple space contains $2^n = 2^7 = 128$ n-tuples, of which $2^k = 2^3 = 8$ (or 1/16 of the n-tuples) are codewords. Next, consider a *nonbinary* $(n, k) = (7, 3)$ code where each symbol is composed of $m = 3$ bits. The n-tuple space amounts to $2^{nm} = 2^{21} = 2{,}097{,}152$ n-tuples, of which $2^{km} = 2^9 = 512$ (or 1/4096 of the n-tuples) are codewords. When dealing with nonbinary symbols, each made up of m bits, only a small fraction (i.e., 2^{km} of the large number 2^{nm}) of possible n-tuples are codewords. This fraction decreases with increasing values of m. The important point here is that when a small fraction of the n-tuple space is used for codewords, a large $d_{\min}$ can be created.

Any linear code is capable of correcting $n - k$ symbol erasure patterns if the $n - k$ erased symbols all happen to lie on the parity symbols. However, R-S codes have the remarkable property that they are able to correct *any* set of $n - k$ symbol erasures within the block. R-S codes can be designed to have any redundancy. However, the complexity of a high-speed implementation increases with redundancy. Thus, the most attractive R-S codes have high code rates (low redundancy).

Reed-Solomon Error Probability

The Reed-Solomon (R-S) codes are particularly useful for *burst-error correction*; that is, they are effective for channels that have memory. Also, they can be used efficiently on channels where the set of input symbols is large. An interesting feature of the R-S code is that as many as two information symbols can be added to an R-S code of length n without reducing its minimum distance. This extended R-S code has length $n + 2$ and the same number of parity check symbols as the original code. The R-S decoded symbol-error probability, P_E, in terms of the channel symbol-error probability, p, can be written as follows [4]:

$$P_E \approx \frac{1}{2^m - 1} \sum_{j=t+1}^{2^m-1} j \binom{2^m - 1}{j} p^j \left(1 - p\right)^{2^m - 1 - j} \tag{7}$$

where t is the symbol-error correcting capability of the code, and the symbols are made up of m bits each.

The bit-error probability can be upper bounded by the symbol-error probability for specific modulation types. For MFSK modulation with $M = 2^m$, the relationship between P_B and P_E is as follows [3]:

$$\frac{P_B}{P_E} = \frac{2^{m-1}}{2^m - 1} \tag{8}$$

Figure 1 shows P_B versus the channel symbol-error probability p, plotted from Equations (7) and (8) for various (t-error-correcting 32-ary orthogonal Reed-Solomon codes with $n = 31$ (thirty-one 5-bit symbols per code block).

Figure 2 shows P_B versus E_b/N_0 for such a coded system using 32-ary MFSK modulation and noncoherent demodulation over an AWGN channel [4]. For R-S codes, error probability is an exponentially decreasing function of block length, n, and decoding complexity is proportional to a small power of the block length [2]. The R-S codes are sometimes used in a concatenated arrangement. In such a system, an inner convolutional decoder first provides some error control by operating on soft-decision demodulator outputs; the convolutional decoder then presents hard-decision data to the outer Reed-Solomon decoder, which further reduces the probability of error.

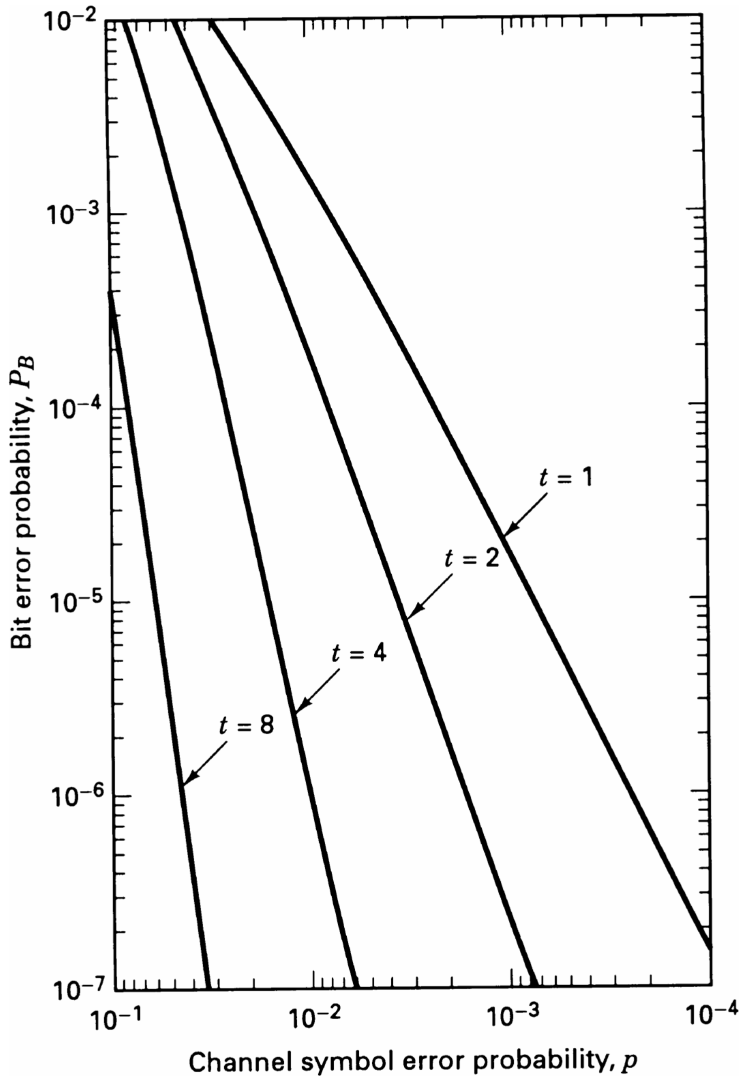

Figure 1

P_B versus p for 32-ary orthogonal signaling and $n = 31$, t-error correcting Reed-Solomon coding [4].

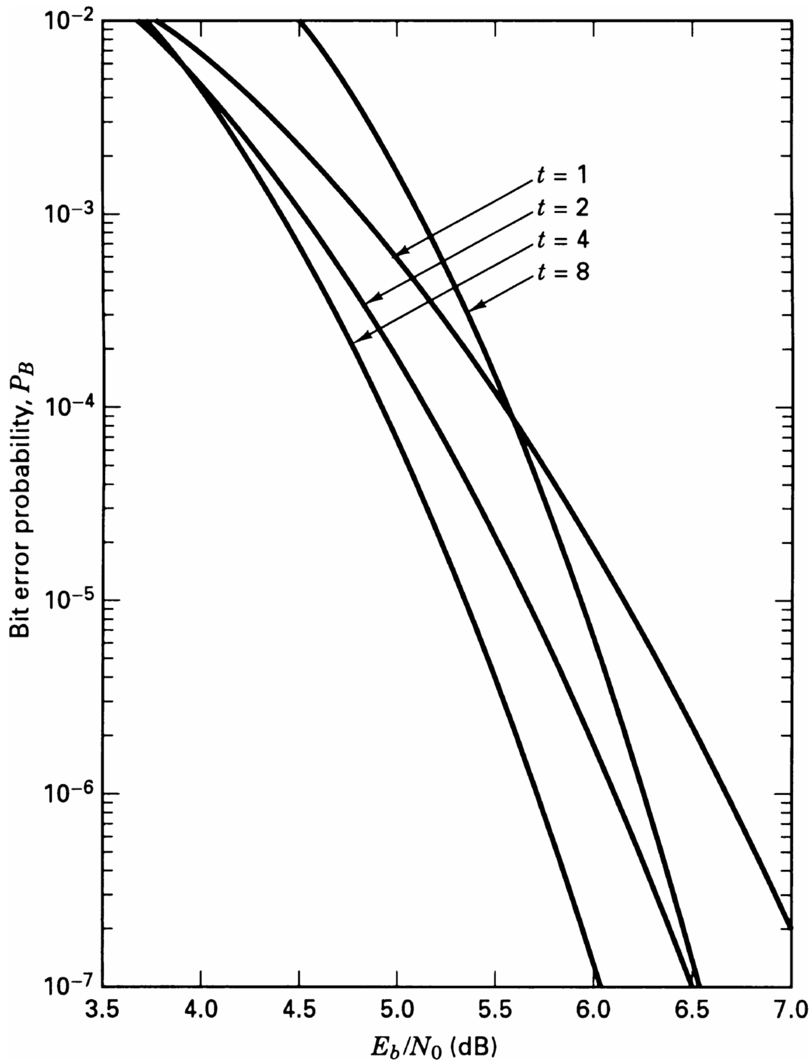

Figure 2

Bit-error probability versus E_b/N_0 performance of several $n = 31$, t-error correcting Reed-Solomon coding systems with 32-ary MPSK modulation over an AWGN channel [4].

Why R-S Codes Perform Well Against Burst Noise

Consider an $(n, k) = (255, 247)$ R-S code, where each symbol is made up of $m = 8$ bits (such symbols are typically referred to as *bytes*). Since $n - k = 8$, Equation (4) indicates that this code can correct any four symbol errors in a block of 255. Imagine the presence of a noise burst, lasting for 25-bit durations and disturbing one block of data during transmission, as illustrated in Figure 3.

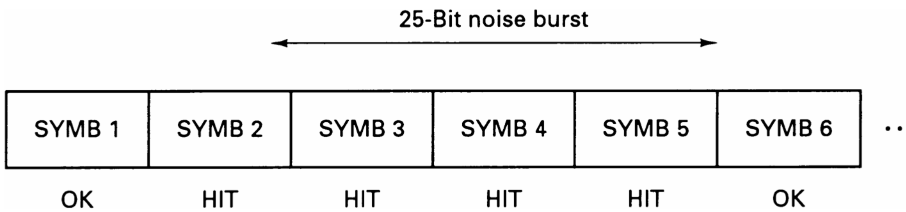

Figure 3

Data block disturbed by 25-bit noise burst.

In this example, notice that a burst of noise that lasts for a duration of 25 contiguous bits must disturb exactly four symbols. The R-S decoder for the (255, 247) code will correct *any* four-symbol errors without regard to the type of damage suffered by the symbol. In other words, when a decoder corrects a byte, it replaces the incorrect byte with the correct one, whether the error was caused by one bit being corrupted or all eight bits being corrupted. Thus if a symbol is wrong, it might as well be wrong in all of its bit positions. This gives an R-S code a tremendous burst-noise advantage over binary codes, even allowing for the interleaving of binary codes. In this example, if the 25-bit noise disturbance had occurred in a random fashion rather than as a contiguous burst, it should be clear that many more than four symbols would be affected (as many as 25 symbols might be disturbed). Of course, that would be beyond the capability of the (255, 247) code.

R-S Performance as a Function of Size, Redundancy, and Code Rate

For a code to successfully combat the effects of noise, the noise duration has to represent a relatively small percentage of the codeword. To ensure that this happens most of the time, the received noise should be averaged over a long period of time, reducing the effect of a freak streak of bad luck. Hence, error-correcting codes become more efficient (error performance improves) as the code block size increases, making R-S codes an attractive choice whenever long block lengths are desired [5]. This is seen by the family of curves in Figure 4, where the rate of the code is held at a constant 7/8, while its block size increases from n = 32 symbols (with m = 5 bits per symbol) to n = 256 symbols (with m = 8 bits per symbol). Thus, the block size increases from 160 bits to 2048 bits.

As the redundancy of an R-S code increases (lower code rate), its implementation grows in complexity (especially for high-speed devices). Also, the bandwidth expansion must grow for any real-time communications application. However, the benefit of increased redundancy, just like the benefit of increased symbol size, is the improvement in bit-error performance, as can be seen in Figure 5, where the code length n is held at a constant 64, while the number of data symbols decreases from k = 60 to k = 4 (redundancy increases from 4 symbols to 60 symbols).

Figure 5 represents transfer functions (output bit-error probability versus input channel symbol-error probability) of hypothetical decoders. Because there is no system or channel in mind (only an output-versus-input of a decoder), you might get the idea that the improved error performance versus increased redundancy is a monotonic function that will continually provide system improvement even as the code rate approaches zero. However, this is not the case for codes operating in a real-time communication system. As the rate of a code varies from minimum to maximum (0 to 1), it is interesting to observe the effects shown in Figure 6. Here, the performance curves are plotted for BPSK modulation and an R-S (31, k) code for various channel types. Figure 6 reflects a real-time communication system, where the price paid for error-correction coding is bandwidth expansion by a factor equal to the inverse of the code rate. The curves plotted show clear optimum code rates that minimize the required E_b/N_0 [6]. The optimum code rate is about 0.6 to 0.7 for a Gaussian channel, 0.5 for a Rician-fading channel (with the ratio of direct to reflected received signal power, K = 7 dB), and 0.3 for a Rayleigh-fading channel. Why is there an E_b/N_0 degradation for very large rates (small redundancy) and very low rates (large redundancy)? It is easy to explain the degradation at high rates compared to the optimum rate. Any code generally provides a coding-gain benefit; thus, as the code rate approaches unity (no coding), the system will suffer worse error performance. The degradation at low code rates is more subtle because in a real-time communication system using both modulation and coding, there are two mechanisms at work. One mechanism works to improve error performance, and the other works to degrade it. The improving mechanism is the coding; the greater the redundancy, the greater will be the error-correcting capability of the code. The degrading mechanism is the energy reduction per channel symbol (compared to the data symbol) that stems from the increased redundancy (and faster signaling in a real-time communication system). The reduced symbol energy causes the demodulator to make more errors. Eventually, the second mechanism wins out, and thus at very low code rates the system experiences error-performance degradation.

Let's see if we can corroborate the error performance versus code rate in Figure 6 with the curves in Figure 2. The figures are really not directly comparable because the modulation is BPSK in Figure 6 and 32-ary MFSK in Figure 2. However, perhaps we can verify that R-S error performance versus code rate exhibits the same general curvature with MFSK modulation as it does with BPSK. In Figure 2, the error performance over an AWGN channel improves as the symbol error-correcting capability, t, increases from $t = 1$ to $t = 4$; the $t = 1$ and $t = 4$ cases correspond to R-S (31, 29) and R-S (31, 23) with code rates of 0.94 and 0.74 respectively. However, at $t = 8$, which corresponds to R-S (31, 15) with code rate = 0.48, the error performance at $P_B = 10^{-5}$ degrades by about 0.5 dB of E_b/N_0 compared to the $t = 4$ case. From Figure 2, we can conclude that if we were to plot error performance versus code rate, the curve would have the same general "shape" as it does in Figure 6. Note that this manifestation cannot be gleaned from Figure 1, since that figure represents a decoder transfer function, which provides no information about the channel and the demodulation. Therefore, of the two mechanisms at work in the channel, the Figure 1 transfer function only presents the output-versus-input benefits of the decoder, and displays nothing about the loss of energy as a function of lower code rate.

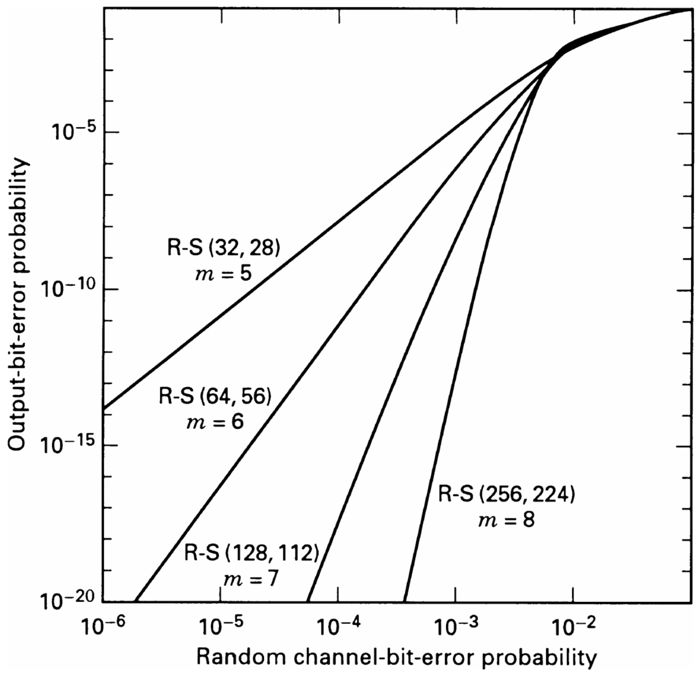

Figure 4

Reed-Solomon rate 7/8 decoder performance as a function of symbol size.

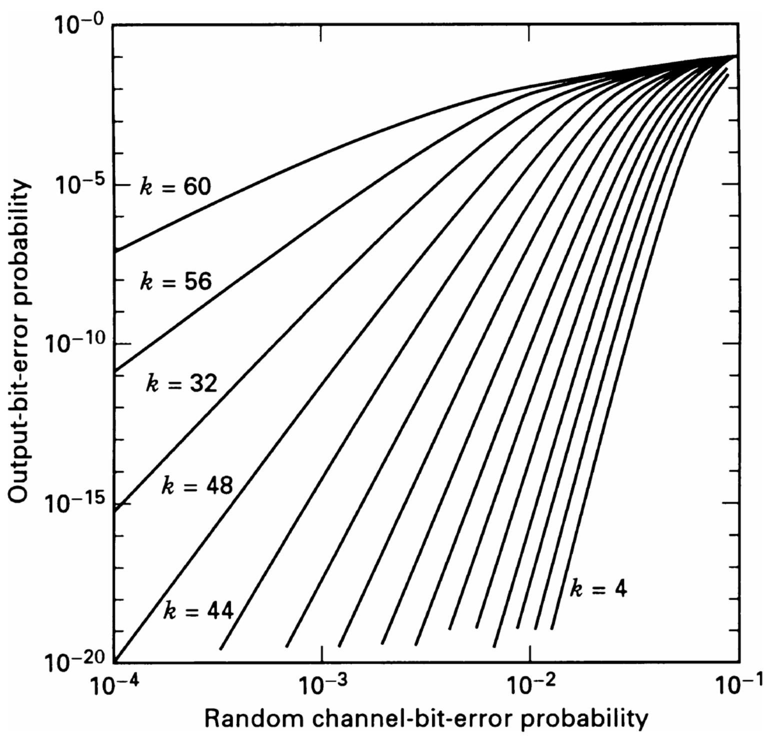

Figure 5

Reed-Solomon (64, *k*) decoder performance as a function of redundancy.

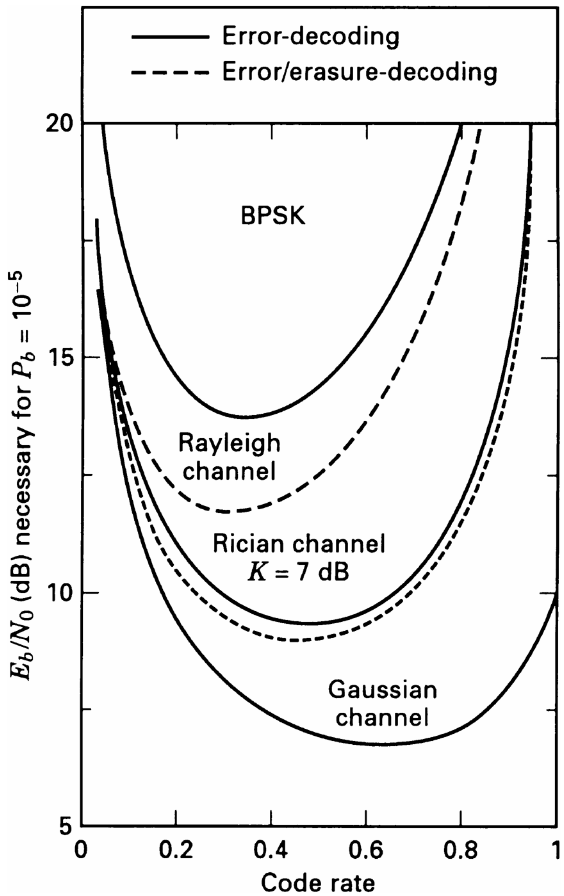

Figure 6

BPSK plus Reed-Solomon (31, *k*) decoder performance as a function of code rate.

Finite Fields

In order to understand the encoding and decoding principles of nonbinary codes, such as Reed-Solomon (R-S) codes, it is necessary to venture into the area of finite fields known as *Galois Fields* (GF). For any prime number, p, there exists a finite field denoted GF(p) that contains p elements. It is possible to extend GF(p) to a field of p^m elements, called an *extension field* of GF(p), and denoted by GF(p^m), where m is a nonzero positive integer. Note that GF(p^m) contains as a subset the elements of GF(p). Symbols from the extension field GF(2^m) are used in the construction of Reed-Solomon (R-S) codes.

The binary field GF(2) is a subfield of the extension field GF(2^m), in much the same way as the real number field is a subfield of the complex number field.

Besides the numbers 0 and 1, there are additional unique elements in the extension field that will be represented with a new symbol α. Each nonzero element in GF(2^m) can be represented by a power of α. An *infinite* set of elements, F, is formed by starting with the elements {0, 1, α}, and generating additional elements by progressively multiplying the last entry by α, which yields the following:

$$F = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^j, \ldots\} = \{0, \alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^j, \ldots\} \tag{9}$$

To obtain the *finite* set of elements of GF(2^m) from F, a condition must be imposed on F so that it may contain only 2^m elements and is closed under multiplication. The condition that closes the set of field elements under multiplication is characterized by the irreducible polynomial shown below:

$$\alpha^{(2^m-1)} + 1 = 0$$

or equivalently

$$\alpha^{(2^m-1)} = 1 = \alpha^0 \tag{10}$$

Using this polynomial constraint, any field element that has a power equal to or greater than 2^m - 1 can be reduced to an element with a power less than 2^m - 1, as follows:

$$\alpha^{(2^m+n)} = \alpha^{(2^m-1)}\ \alpha^{n+1} = \alpha^{n+1} \tag{11}$$

Thus, Equation (10) can be used to form the finite sequence F^* from the infinite sequence F as follows:

$$\begin{aligned} F^* &= \left\{\ 0,\ 1,\ \alpha,\ \alpha^2, \ldots,\ \alpha^{2^m-2},\ \alpha^{2^m-1},\ \alpha^{2^m}, \ldots \right\} \\ &= \left\{\ 0,\ \alpha^0, \alpha^1,\ \alpha^2, \ldots,\ \alpha^{2^m-2},\ \alpha^0,\ \alpha^1,\ \alpha^2, \ldots \right\} \end{aligned} \tag{12}$$

Therefore, it can be seen from Equation (12) that the elements of the finite field, GF(2^m), are as follows:

$$\mathrm{GF}(2^m) = \left\{\ 0,\ \alpha^0,\ \alpha^1,\ \alpha^2,\ .\ .\ .\ ,\ \alpha^{2^m-2}\ \right\} \tag{13}$$

Addition in the Extension Field GF(2^m)

Each of the 2^m elements of the finite field, GF(2^m), can be represented as a distinct polynomial of *degree* m - 1 or less. The degree of a polynomial is the value of its highest-order exponent. We denote each of the nonzero elements of GF(2^m) as a polynomial, $a_i(X)$, where at least one of the m coefficients of $a_i(X)$ is nonzero. For $i = 0,1,2,\ldots,2^m - 2$,

$$\alpha^i = a_i(X) = a_{i,0} + a_{i,1}X + a_{i,2}X^2 + \ldots + a_{i,m-1}X^{m-1} \tag{14}$$

Consider the case of $m = 3$, where the finite field is denoted GF(2^3). Figure 7 shows the mapping (developed later) of the seven elements $\{\alpha^i\}$ and the zero element, in terms of the basis elements $\{X^0, X^1, X^2\}$ described by Equation (14). Since Equation (10) indicates that $\alpha^0 = \alpha^7$, there are seven nonzero elements or a total of eight elements in this field. Each row in the Figure 7 mapping comprises a sequence of binary values representing the coefficients $a_{i,0}$, $a_{i,1}$, and $a_{i,2}$ in Equation (14). One of the benefits of using extension field elements $\{\alpha^i\}$ in place of binary elements is the compact notation that facilitates the mathematical representation of nonbinary encoding and decoding processes. Addition of two elements of the finite field is then defined as the modulo-2 sum of each of the polynomial coefficients of like powers,

$$\alpha^i + \alpha^j = (a_{i,0} + a_{j,0}) + (a_{i,1} + a_{j,1})X + \ldots + (a_{i,m-1} + a_{j,m-1})X^{m-1} \tag{15}$$

Basis elements

Field elements	X^0	X^1	X^2
0	0	0	0
α^0	1	0	0
α^1	0	1	0
α^2	0	0	1
α^3	1	1	0
α^4	0	1	1
α^5	1	1	1
α^6	1	0	1
α^7	1	0	0

Figure 7

Mapping field elements in terms of basis elements for GF(8) with $f(x) = 1 + x + x^3$.

A Primitive Polynomial Is Used to Define the Finite Field

A class of polynomials called *primitive polynomials* is of interest because such functions define the finite fields $GF(2^m)$ that in turn are needed to define R-S codes. The following condition is necessary and sufficient to guarantee that a polynomial is primitive. An irreducible polynomial $f(X)$ of degree m is said to be primitive if the smallest positive integer n for which $f(X)$ divides $X^n + 1$ is $n = 2^m - 1$. Note that the statement A divides B means that A divided into B yields a nonzero quotient and a zero remainder. Polynomials will usually be shown low order to high order. Sometimes, it is convenient to follow the reverse format (for example, when performing polynomial division).

Example 1: Recognizing a Primitive Polynomial

Based on the definition of a primitive polynomial given above, determine whether the following irreducible polynomials are primitive.

a. $1 + X + X^4$

b. $1 + X + X^2 + X^3 + X^4$

Solution

a. We can verify whether this degree $m = 4$ polynomial is primitive by determining whether it divides $X^n + 1 = X^{(2^m - 1)} + 1 = X^{15} + 1$, but does not divide $X^n + 1$, for values of n in the range of $1 \leq n < 15$. It is easy to verify that $1 + X + X^4$ divides $X^{15} + 1$ [3], and after repeated computations it can be verified that $1 + X + X^4$ will not divide $X^n + 1$ for any n in the range of $1 \leq n < 15$. Therefore, $1 + X + X^4$ is a primitive polynomial.

b. It is simple to verify that the polynomial $1 + X + X^2 + X^3 + X^4$ divides $X^{15} + 1$. Testing to see whether it will divide $X^n + 1$ for some n that is less than 15 yields the fact that it also divides $X^5 + 1$. Thus, although $1 + X + X^2 + X^3 + X^4$ is irreducible, it is not primitive.

The Extension Field $GF(2^3)$

Consider an example involving a primitive polynomial and the finite field that it defines. Table 1 contains a listing of some primitive polynomials. We choose the first one shown, $f(X) = 1 + X + X^3$, which defines a finite field $GF(2^m)$, where the degree of the polynomial is $m = 3$. Thus, there are $2^m = 2^3 = 8$ elements in the field defined by $f(X)$. Solving for the roots of $f(X)$ means that the values of X that correspond to $f(X) = 0$ must be found. The familiar binary elements, 1 and 0, do not satisfy (are not roots of) the polynomial $f(X) = 1 + X + X^3$, since $f(1) = 1$ and $f(0) = 1$ (using modulo-2 arithmetic). Yet, a fundamental theorem of algebra states that a polynomial of degree m must have precisely m roots. Therefore for this example, $f(X) = 0$ must yield three roots. Clearly a dilemma arises, since the three roots do not lie in the same finite field as the coefficients of $f(X)$. Therefore, they must lie somewhere else; the roots lie in the extension field, GF(2^3). Let α, an element of the extension field, be defined as a root of the polynomial $f(X)$. Therefore, it is possible to write the following:

$$\begin{aligned} f(\alpha) &= 0 \\ 1 + \alpha + \alpha^3 &= 0 \\ \alpha^3 &= -1 - \alpha \end{aligned} \tag{16}$$

Since in the binary field $+1 = -1$, α^3 can be represented as follows:

$$\alpha^3 = 1 + \alpha \tag{17}$$

Thus, α^3 is expressed as a weighted sum of α-terms having lower orders. In fact all powers of α can be so expressed. For example, consider α^4, where we obtain

$$\alpha^4 = \alpha \bullet \alpha^3 = \alpha \bullet (1 + \alpha) = \alpha + \alpha^2 \tag{18a}$$

Now, consider α^5, where

$$\alpha^5 = \alpha \bullet \alpha^4 = \alpha \bullet (\alpha + \alpha^2) = \alpha^2 + \alpha^3 \tag{18b}$$

From Equation (17), we obtain

$$\alpha^5 = 1 + \alpha + \alpha^2 \tag{18c}$$

Now, for α^6, using Equation (18c), we obtain

$$\alpha^6 = \alpha \bullet \alpha^5 = \alpha \bullet (1 + \alpha + \alpha^2) = \alpha + \alpha^2 + \alpha^3 = 1 + \alpha^2 \tag{18d}$$

And for α^7, using Equation (18d), we obtain

$$\alpha^7 = \alpha \bullet \alpha^6 = \alpha \bullet (1 + \alpha^2) = \alpha + \alpha^3 = 1 = \alpha^0 \tag{18e}$$

Note that $\alpha^7 = \alpha^0$, and therefore the eight finite field elements of GF(2^3) are

$$\{0, \alpha^0, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\} \tag{19}$$

Table 1

Some Primitive Polynomials

m		m	
3	$1+X+X^3$	14	$1+X+X^6+X^{10}+X^{14}$
4	$1+X+X^4$	15	$1+X+X^{15}$
5	$1+X^2+X^5$	16	$1+X+X^3+X^{12}+X^{16}$
6	$1+X+X^6$	17	$1+X^3+X^{17}$
7	$1+X^3+X^7$	18	$1+X^7+X^{18}$
8	$1+X^2+X^3+X^4+X^8$	19	$1+X+X^2+X^5+X^{19}$
9	$1+X^4+X^9$	20	$1+X^3+X^{20}$
10	$1+X^3+X^{10}$	21	$1+X^2+X^{21}$
11	$1+X^2+X^{11}$	22	$1+X+X^{22}$
12	$1+X+X^4+X^6+X^{12}$	23	$1+X^5+X^{23}$
13	$1+X+X^3+X^4+X^{13}$	24	$1+X+X^2+X^7+X^{24}$

The mapping of field elements in terms of basis elements, described by Equation (14), can be demonstrated with the linear feedback shift register (LFSR) circuit shown in Figure 8. The circuit generates (with $m = 3$) the $2^m - 1$ nonzero elements of the field, and thus summarizes the findings of Figure 7 and Equations (17) through (19). Note that in Figure 8 the circuit feedback connections correspond to the coefficients of the polynomial $f(X) = 1 + X + X^3$, just like for binary cyclic codes [3]. By starting the circuit in any nonzero state, say 1 0 0, and performing a right-shift at each clock time, it is possible to verify that each of the field elements shown in Figure 7 (except the all-zeros element) will cyclically appear in the stages of the shift register. Two arithmetic operations, addition and multiplication, can be defined for this GF(2^3) finite field. Addition is shown in Table 2, and multiplication is shown in Table 3 for the nonzero elements only. The rules of addition follow from Equations (17) through (18e), and can be verified by noticing in Figure 7 that the sum of any field elements can be obtained by adding (modulo-2) the respective coefficients of their basis elements. The multiplication rules in Table 3 follow the usual procedure, in which the product of the field elements is obtained by adding their exponents modulo-($2^m - 1$), or for this case, modulo-7.

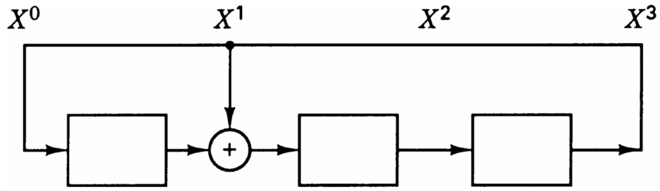

Figure 8

Extension field elements can be represented by the contents of a binary linear feedback shift register (LFSR) formed from a primitive polynomial.

Table 2

Addition Table

	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	0	α^3	α^6	α^1	α^5	α^4	α^2
α^1	α^3	0	α^4	α^0	α^2	α^6	α^5
α^2	α^6	α^4	0	α^5	α^1	α^3	α^0
α^3	α^1	α^0	α^5	0	α^6	α^2	α^4
α^4	α^5	α^2	α^1	α^6	0	α^0	α^3
α^5	α^4	α^6	α^3	α^2	α^0	0	α^1
α^6	α^2	α^5	α^0	α^4	α^3	α^1	0

Table 3

Multiplication Table

	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^1	α^1	α^2	α^3	α^4	α^5	α^6	α^0
α^2	α^2	α^3	α^4	α^5	α^6	α^0	α^1
α^3	α^3	α^4	α^5	α^6	α^0	α^1	α^2
α^4	α^4	α^5	α^6	α^0	α^1	α^2	α^3
α^5	α^5	α^6	α^0	α^1	α^2	α^3	α^4
α^6	α^6	α^0	α^1	α^2	α^3	α^4	α^5

A Simple Test to Determine Whether a Polynomial Is Primitive

There is another way of defining a primitive polynomial that makes its verification relatively easy. For an irreducible polynomial to be a primitive polynomial, at least one of its roots must be a primitive element. A *primitive element* is one that when raised to higher-order exponents will yield all the nonzero elements in the field. Since the field is a finite field, the number of such elements is finite.

Example 2: A Primitive Polynomial Must Have at Least One Primitive Element

Find the $m = 3$ roots of $f(X) = 1 + X + X^3$, and verify that the polynomial is primitive by checking that at least one of the roots is a primitive element. What are the roots? Which ones are primitive?

Solution

The roots will be found by enumeration. Clearly, $\alpha^0 = 1$ is not a root because $f(\alpha^0) = 1$. Now, use Table 2 to check whether α^1 is a root. Since

$$f(\alpha) = 1 + \alpha + \alpha^3 = 1 + \alpha^0 = 0$$

α is therefore a root.

Now check whether α^2 is a root:

$$f(\alpha^2) = 1 + \alpha^2 + \alpha^6 = 1 + \alpha^0 = 0$$

Hence, α^2 is a root.

Now check whether α^3 is a root.

$$f(\alpha^3) = 1 + \alpha^3 + \alpha^9 = 1 + \alpha^3 + \alpha^2 = 1 + \alpha^5 = \alpha^4 \neq 0$$

Hence, α^3 is *not* a root. Is α^4 a root?

$$f(\alpha^4) = \alpha^{12} + \alpha^4 + 1 = \alpha^5 + \alpha^4 + 1 = 1 + \alpha^0 = 0$$

Yes, it is a root. Hence, the roots of $f(X) = 1 + X + X^3$ are α, α^2, and α^4. It is not difficult to verify that starting with any of these roots and generating higher-order exponents yields all of the seven nonzero elements in the field. Hence, each of the roots is a primitive element. Since our verification requires that at least one root be a primitive element, the polynomial is primitive.

A relatively simple method to verify whether a polynomial is primitive can be described in a manner that is related to this example. For any given polynomial under test, draw the LFSR, with the feedback connections corresponding to the polynomial coefficients as shown by the example of Figure 8. Load into the circuit-registers any nonzero setting, and perform a right-shift with each clock pulse. If the circuit generates each of the nonzero field elements within one period, the polynomial that defines this GF(2^m) field is a primitive polynomial.

Reed-Solomon Encoding

Equation (2), repeated below as Equation (20), expresses the most conventional form of Reed-Solomon (R-S) codes in terms of the parameters n, k, t, and any positive integer $m > 2$.

$$(n, k) = (2^m - 1, 2^m - 1 - 2t) \tag{20}$$

where $n - k = 2t$ is the number of parity symbols, and t is the symbol-error correcting capability of the code. The generating polynomial for an R-S code takes the following form:

$$\mathbf{g}(X) = g_0 + g_1 X + g_2 X^2 + \ldots + g_{2t-1} X^{2t-1} + X^{2t} \tag{21}$$

The degree of the generator polynomial is equal to the number of parity symbols. R-S codes are a subset of the Bose, Chaudhuri, and Hocquenghem (BCH) codes; hence, it should be no surprise that this relationship between the degree of the generator polynomial and the number of parity symbols holds, just as for BCH codes. Since the generator polynomial is of degree $2t$, there must be precisely $2t$ successive powers of α that are roots of the polynomial. We designate the roots of $\mathbf{g}(X)$ as $\alpha, \alpha^2, \ldots, \alpha^{2t}$. It is not necessary to start with the root α; starting with any power of α is possible. Consider as an example the (7, 3) double-symbol-error correcting R-S code. We describe the generator polynomial in terms of its $2t = n - k = 4$ roots, as follows:

$$\begin{aligned}
\mathbf{g}(X) &= \left(X - \alpha\right)\left(X - \alpha^2\right)\left(X - \alpha^3\right)\left(X - \alpha^4\right) \\
&= \left(X^2 - \left(\alpha + \alpha^2\right)X + \alpha^3\right)\left(X^2 - \left(\alpha^3 + \alpha^4\right)X + \alpha^7\right) \\
&= \left(X^2 - \alpha^4 X + \alpha^3\right)\left(X^2 - \alpha^6 X + \alpha^0\right) \\
&= X^4 - \left(\alpha^4 + \alpha^6\right)X^3 + \left(\alpha^3 + \alpha^{10} + \alpha^0\right)X^2 - \left(\alpha^4 + \alpha^9\right)X + \alpha^3 \\
&= X^4 - \alpha^3 X^3 + \alpha^0 X^2 - \alpha^1 X + \alpha^3
\end{aligned}$$

Following the low order to high order format, and changing negative signs to positive, since in the binary field $+1 = -1$, $\mathbf{g}(X)$ can be expressed as follows:

$$\mathbf{g}(X) = \alpha^3 + \alpha^1 X + \alpha^0 X^2 + \alpha^3 X^3 + X^4 \tag{22}$$

Encoding in Systematic Form

Since R-S codes are cyclic codes, encoding in systematic form is analogous to the binary encoding procedure [3]. We can think of shifting a message polynomial, $\mathbf{m}(X)$, into the rightmost k stages of a codeword register and then appending a parity polynomial, $\mathbf{p}(X)$, by placing it in the leftmost $n - k$ stages. Therefore we multiply $\mathbf{m}(X)$ by X^{n-k}, thereby manipulating the message polynomial algebraically so that it is right-shifted $n - k$ positions. Next, we divide $X^{n-k}\,\mathbf{m}(X)$ by the generator polynomial $\mathbf{g}(X)$, which is written in the following form:

$$X^{n-k}\,\mathbf{m}(X) = \mathbf{q}(X)\,\mathbf{g}(X) + \mathbf{p}(X) \tag{23}$$

where $\mathbf{q}(X)$ and $\mathbf{p}(X)$ are quotient and remainder polynomials, respectively. As in the binary case, the remainder is the parity. Equation (23) can also be expressed as follows:

$$\mathbf{p}(X) = X^{n-k}\,\mathbf{m}(X) \text{ modulo } \mathbf{g}(X) \tag{24}$$

The resulting codeword polynomial, $\mathbf{U}(X)$ can be written as

$$\mathbf{U}(X) = \mathbf{p}(X) + X^{n-k}\,\mathbf{m}(X) \tag{25}$$

We demonstrate the steps implied by Equations (24) and (25) by encoding the following three-symbol message:

$$\underbrace{010}_{\alpha^1}\;\underbrace{110}_{\alpha^3}\;\underbrace{111}_{\alpha^5}$$

with the (7, 3) R-S code whose generator polynomial is given in Equation (22). We first multiply (upshift) the message polynomial $\alpha^1 + \alpha^3 X + \alpha^5 X^2$ by $X^{n-k} = X^4$, yielding $\alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6$. We next divide this upshifted message polynomial by the generator polynomial in Equation (22), $\alpha^3 + \alpha^1 X + \alpha^0 X^2 + \alpha^3 X^3 + X^4$. Polynomial division with nonbinary coefficients is more tedious than its binary counterpart, because the required operations of addition (subtraction) and multiplication (division) must follow the rules in Tables 2 and 3, respectively. It is left as an exercise for the reader to verify that this polynomial division results in the following remainder (parity) polynomial.

$$\mathbf{p}(X) = \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3$$

Then, from Equation (25), the codeword polynomial can be written as follows:

$$\mathbf{U}(X) = \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 + \alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6$$

Systematic Encoding with an (*n* - *k*)–Stage Shift Register

Using circuitry to encode a three-symbol sequence in systematic form with the (7, 3) R-S code described by $\mathbf{g}(X)$ in Equation (22) requires the implementation of a linear feedback shift register (LFSR) circuit, as shown in Figure 9. It can easily be verified that the multiplier terms in Figure 9, taken from left to right, correspond to the coefficients of the polynomial in Equation (22) (low order to high order). This encoding process is the nonbinary equivalent of cyclic encoding [3]. Here, corresponding to Equation (20), the (7, 3) R-S nonzero codewords are made up of $2^m - 1 = 7$ symbols, and each symbol is made up of $m = 3$ bits.

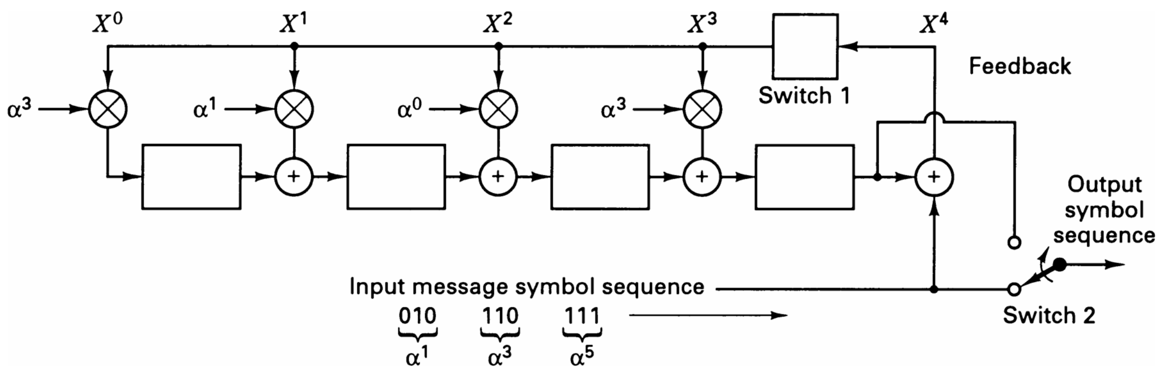

Figure 9

LFSR encoder for a (7, 3) R-S code.

Here the example is nonbinary, so that each stage in the shift register of Figure 9 holds a 3-bit symbol. In the case of binary codes, the coefficients labeled g_1, g_2, and so on are binary. Therefore, they take on values of 1 or 0, simply dictating the presence or absence of a connection in the LFSR. However in Figure 9, since each coefficient is specified by 3-bits, it can take on one of eight values.

The nonbinary operation implemented by the encoder of Figure 9, forming codewords in a systematic format, proceeds in the same way as the binary one. The steps can be described as follows:

1. Switch 1 is closed during the first k clock cycles to allow shifting the message symbols into the $(n - k)$–stage shift register.

2. Switch 2 is in the down position during the first k clock cycles in order to allow simultaneous transfer of the message symbols directly to an output register (not shown in Figure 9).

3. After transfer of the kth message symbol to the output register, switch 1 is opened and switch 2 is moved to the up position.

4. The remaining $(n - k)$ clock cycles clear the parity symbols contained in the shift register by moving them to the output register.

5. The total number of clock cycles is equal to n, and the contents of the output register is the codeword polynomial $\mathbf{p}(X) + X^{n-k}\,\mathbf{m}(X)$, where $\mathbf{p}(X)$ represents the parity symbols and $\mathbf{m}(X)$ the message symbols in polynomial form.

We use the same symbol sequence that was chosen as a test message earlier:

$$\underbrace{010}_{\alpha^1}\ \underbrace{110}_{\alpha^3}\ \underbrace{111}_{\alpha^5}$$

where the rightmost symbol is the earliest symbol, and the rightmost bit is the earliest bit. The operational steps during the first $k = 3$ shifts of the encoding circuit of Figure 9 are as follows:

INPUT QUEUE			CLOCK CYCLE	REGISTER CONTENTS				FEEDBACK
α^1	α^3	α^5	0	0	0	0	0	α^5
	α^1	α^3	1	α^1	α^6	α^5	α^1	α^0
		α^1	2	α^3	0	α^2	α^2	α^4
		-	3	α^0	α^2	α^4	α^6	-

After the third clock cycle, the register contents are the four parity symbols, α^0, α^2, α^4, and α^6, as shown. Then, switch 1 of the circuit is opened, switch 2 is toggled to the up position, and the parity symbols contained in the register are shifted to the output. Therefore the output codeword, $\mathbf{U}(X)$, written in polynomial form, can be expressed as follows:

$$\mathbf{U}(X) = \sum_{n=0}^{6} u_n X^n$$

$$\begin{aligned} \mathbf{U}(X) &= \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 + \alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6 \\ &= (100) + (001)\,X + (011)\,X^2 + (101)\,X^3 + (010)\,X^4 + (110)\,X^5 + (111)\,X^6 \end{aligned} \tag{26}$$

The process of verifying the contents of the register at various clock cycles is somewhat more tedious than in the binary case. Here, the field elements must be added and multiplied by using Table 2 and Table 3, respectively.

The roots of a generator polynomial, $\mathbf{g}(X)$, must also be the roots of the codeword generated by $\mathbf{g}(X)$, because a valid codeword is of the following form:

$$\mathbf{U}(X) = \mathbf{m}(X)\,\mathbf{g}(X) \tag{27}$$

Therefore, an arbitrary codeword, when evaluated at any root of $\mathbf{g}(X)$, must yield zero. It is of interest to verify that the codeword polynomial in Equation (26) does indeed yield zero when evaluated at the four roots of $\mathbf{g}(X)$. In other words, this means checking that

$$\mathbf{U}(\alpha) = \mathbf{U}(\alpha^2) = \mathbf{U}(\alpha^3) = \mathbf{U}(\alpha^4) = \mathbf{0}$$

Evaluating each term independently yields the following:

$$\begin{aligned}
\mathbf{U}(\alpha) &= \alpha^0 + \alpha^3 + \alpha^6 + \alpha^9 + \alpha^5 + \alpha^8 + \alpha^{11} \\
&= \alpha^0 + \alpha^3 + \alpha^6 + \alpha^2 + \alpha^5 + \alpha^1 + \alpha^4 \\
&= \alpha^1 + \alpha^0 + \alpha^6 + \alpha^4 \\
&= \alpha^3 + \alpha^3 = \mathbf{0}
\end{aligned}$$

$$\begin{aligned}
\mathbf{U}(\alpha^2) &= \alpha^0 + \alpha^4 + \alpha^8 + \alpha^{12} + \alpha^9 + \alpha^{13} + \alpha^{17} \\
&= \alpha^0 + \alpha^4 + \alpha^1 + \alpha^5 + \alpha^2 + \alpha^6 + \alpha^3 \\
&= \alpha^5 + \alpha^6 + \alpha^0 + \alpha^3 \\
&= \alpha^1 + \alpha^1 = \mathbf{0}
\end{aligned}$$

$$\begin{aligned}
\mathbf{U}(\alpha^3) &= \alpha^0 + \alpha^5 + \alpha^{10} + \alpha^{15} + \alpha^{13} + \alpha^{18} + \alpha^{23} \\
&= \alpha^0 + \alpha^5 + \alpha^3 + \alpha^1 + \alpha^6 + \alpha^4 + \alpha^2 \\
&= \alpha^4 + \alpha^0 + \alpha^3 + \alpha^2 \\
&= \alpha^5 + \alpha^5 = \mathbf{0}
\end{aligned}$$

$$\begin{aligned}
\mathbf{U}(\alpha^4) &= \alpha^0 + \alpha^6 + \alpha^{12} + \alpha^{18} + \alpha^{17} + \alpha^{23} + \alpha^{29} \\
&= \alpha^0 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 \\
&= \alpha^2 + \alpha^0 + \alpha^5 + \alpha^1 \\
&= \alpha^6 + \alpha^6 = \mathbf{0}
\end{aligned}$$

This demonstrates the expected results that a codeword evaluated at any root of $\mathbf{g}(X)$ must yield zero.

Reed-Solomon Decoding

Earlier, a test message encoded in systematic form using a (7, 3) R-S code resulted in a codeword polynomial described by Equation (26). Now, assume that during transmission this codeword becomes corrupted so that two symbols are received in error. (This number of errors corresponds to the maximum error-correcting capability of the code.) For this seven-symbol codeword example, the error pattern, $\mathbf{e}(X)$, can be described in polynomial form as follows:

$$\mathbf{e}(X) = \sum_{n=0}^{6} e_n X^n \tag{28}$$

For this example, let the double-symbol error be such that

$$\begin{aligned} \mathbf{e}(X) &= 0 + 0X + 0X^2 + \alpha^2 X^3 + \alpha^5 X^4 + 0X^5 + 0X^6 \\ &= (000) + (000)X + (000)X^2 + (001)X^3 + (111)X^4 + (000)X^5 + (000)X^6 \end{aligned} \tag{29}$$

In other words, one parity symbol has been corrupted with a 1-bit error (seen as α^2), and one data symbol has been corrupted with a 3-bit error (seen as α^5). The received corrupted-codeword polynomial, $\mathbf{r}(X)$, is then represented by the sum of the transmitted-codeword polynomial and the error-pattern polynomial as follows:

$$\mathbf{r}(X) = \mathbf{U}(X) + \mathbf{e}(X) \tag{30}$$

Following Equation (30), we add $\mathbf{U}(X)$ from Equation (26) to $\mathbf{e}(X)$ from Equation (29) to yield $\mathbf{r}(X)$, as follows:

$$\begin{aligned} \mathbf{r}(X) &= (100) + (001)X + (011)X^2 + (100)X^3 + (101)X^4 + (110)X^5 + (111)X^6 \\ &= \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^0 X^3 + \alpha^6 X^4 + \alpha^3 X^5 + \alpha^5 X^6 \end{aligned} \tag{31}$$

In this example, there are four unknowns—two error locations and two error values. Notice an important difference between the nonbinary decoding of $\mathbf{r}(X)$ that we are faced with in Equation (31) and binary decoding; in binary decoding, the decoder only needs to find the error locations [3]. Knowledge that there is an error at a particular location dictates that the bit must be "flipped" from 1 to 0 or vice versa. But here, the nonbinary symbols require that we not only learn the error locations, but also determine the correct symbol values at those locations. Since there are four unknowns in this example, four equations are required for their solution.

Syndrome Computation

The *syndrome* is the result of a parity check performed on **r** to determine whether **r** is a valid member of the codeword set [3]. If in fact **r** is a member, the syndrome **S** has value **0**. Any nonzero value of **S** indicates the presence of errors. Similar to the binary case, the syndrome **S** is made up of *n* - *k* symbols, $\{S_i\}$ (i = 1, … , *n* - *k*). Thus, for this (7, 3) R-S code, there are four symbols in every syndrome vector; their values can be computed from the received polynomial, $\mathbf{r}(X)$. Note how the computation is facilitated by the structure of the code, given by Equation (27) and rewritten below:

$$\mathbf{U}(X) = \mathbf{m}(X)\,\mathbf{g}(X)$$

From this structure it can be seen that every valid codeword polynomial $\mathbf{U}(X)$ is a multiple of the generator polynomial $\mathbf{g}(X)$. Therefore, the roots of $\mathbf{g}(X)$ must also be the roots of $\mathbf{U}(X)$. Since $\mathbf{r}(X) = \mathbf{U}(X) + \mathbf{e}(X)$, then $\mathbf{r}(X)$ evaluated at each of the roots of $\mathbf{g}(X)$ should yield zero only when it is a valid codeword. Any errors will result in one or more of the computations yielding a nonzero result. The computation of a syndrome symbol can be described as follows:

$$S_i = \mathbf{r}(X)\Big|_{X=\alpha^i} = \mathbf{r}(\alpha^i) \qquad i = 1, \cdots, n-k \tag{32}$$

where $\mathbf{r}(X)$ contains the postulated two-symbol errors as shown in Equation (29). If $\mathbf{r}(X)$ were a valid codeword, it would cause each syndrome symbol S_i to equal 0. For this example, the four syndrome symbols are found as follows:

$$\begin{aligned} S_1 = \mathbf{r}(\alpha) &= \alpha^0 + \alpha^3 + \alpha^6 + \alpha^3 + \alpha^{10} + \alpha^8 + \alpha^{11} \\ &= \alpha^0 + \alpha^3 + \alpha^6 + \alpha^3 + \alpha^2 + \alpha^1 + \alpha^4 \\ &= \alpha^3 \end{aligned} \tag{33}$$

$$\begin{aligned} S_2 = \mathbf{r}(\alpha^2) &= \alpha^0 + \alpha^4 + \alpha^8 + \alpha^6 + \alpha^{14} + \alpha^{13} + \alpha^{17} \\ &= \alpha^0 + \alpha^4 + \alpha^1 + \alpha^6 + \alpha^0 + \alpha^6 + \alpha^3 \\ &= \alpha^5 \end{aligned} \tag{34}$$

$$\begin{aligned} S_3 = \mathbf{r}(\alpha^3) &= \alpha^0 + \alpha^5 + \alpha^{10} + \alpha^9 + \alpha^{18} + \alpha^{18} + \alpha^{23} \\ &= \alpha^0 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha^4 + \alpha^4 + \alpha^2 \\ &= \alpha^6 \end{aligned} \tag{35}$$

$$\begin{aligned} S_4 = \mathbf{r}(\alpha^4) &= \alpha^0 + \alpha^6 + \alpha^{12} + \alpha^{12} + \alpha^{22} + \alpha^{23} + \alpha^{29} \\ &= \alpha^0 + \alpha^6 + \alpha^5 + \alpha^5 + \alpha^1 + \alpha^2 + \alpha^1 \\ &= 0 \end{aligned} \tag{36}$$

The results confirm that the received codeword contains an error (which we inserted), since **S**≠**0**.

Example 3: A Secondary Check on the Syndrome Values

For the (7, 3) R-S code example under consideration, the error pattern is known, since it was chosen earlier. An important property of codes when describing the standard array is that each element of a coset (row) in the standard array has the same syndrome [3]. Show that this property is also true for the R-S code by evaluating the error polynomial $\mathbf{e}(X)$ at the roots of $\mathbf{g}(X)$ to demonstrate that it must yield the same syndrome values as when $\mathbf{r}(X)$ is evaluated at the roots of $\mathbf{g}(X)$. In other words, it must yield the same values obtained in Equations (33) through (36).

Solution

$$S_i = \mathbf{r}(X)\Big|_{X=\alpha^i} = \mathbf{r}(\alpha^i) \qquad i = 1, 2, \cdots, n-k$$

$$S_i = \left[\, \mathbf{U}(X) + \mathbf{e}(X) \,\right]\Big|_{X=\alpha^i} = \mathbf{U}(\alpha^i) + \mathbf{e}(\alpha^i)$$

$$S_i = \mathbf{r}(\alpha^i) = \mathbf{U}(\alpha^i) + \mathbf{e}(\alpha^i) = 0 + \mathbf{e}(\alpha^i)$$

From Equation (29),

$$\mathbf{e}(X) = \alpha^2 X^3 + \alpha^5 X^4$$

Therefore,

$$\begin{aligned} S_1 = \mathbf{e}(\alpha^1) &= \alpha^5 + \alpha^9 \\ &= \alpha^5 + \alpha^2 \\ &= \alpha^3 \end{aligned}$$

$$\begin{aligned} S_2 = \mathbf{e}(\alpha^2) &= \alpha^8 + \alpha^{13} \\ &= \alpha^1 + \alpha^6 \\ &= \alpha^5 \end{aligned}$$

continues ➤

➤ *continued*

$$
\begin{aligned}
S_3 = \mathbf{e}(\alpha^3) &= \alpha^{11} + \alpha^{17} \\
&= \alpha^4 + \alpha^3 \\
&= \alpha^6
\end{aligned}
$$

$$
\begin{aligned}
S_4 = \mathbf{e}(\alpha^4) &= \alpha^{14} + \alpha^{21} \\
&= \alpha^0 + \alpha^0 \\
&= 0
\end{aligned}
$$

These results confirm that the syndrome values are the same, whether obtained by evaluating $\mathbf{e}(X)$ at the roots of $\mathbf{g}(X)$, or $\mathbf{r}(X)$ at the roots of $\mathbf{g}(X)$.

Error Location

Suppose there are ν errors in the codeword at location $X^{j_1}, X^{j_2}, \ldots, X^{j_\nu}$. Then, the error polynomial $\mathbf{e}(X)$ shown in Equations (28) and (29) can be written as follows:

$$\mathbf{e}(X) = e_{j_1} X^{j_1} + e_{j_2} X^{j_2} + \ldots + e_{j_\nu} X^{j_\nu} \tag{37}$$

The indices 1, 2, … ν refer to the first, second, …, ν^{th} errors, and the index j refers to the error location. To correct the corrupted codeword, each error value e_{j_l} and its location X^{j_l}, where $l = 1, 2, \ldots, \nu$, must be determined. We define an error locator number as $\beta_l = \alpha^{j_l}$. Next, we obtain the $n - k = 2t$ syndrome symbols by substituting α^i into the received polynomial for $i = 1, 2, \ldots 2t$:

$$
\begin{aligned}
S_1 = \mathbf{r}(\alpha) &= e_{j_1}\beta_1 + e_{j_2}\beta_2 + \ldots + e_{j_\nu}\beta_\nu \\
S_2 = \mathbf{r}(\alpha^2) &= e_{j_1}\beta_1^2 + e_{j_2}\beta_2^2 + \ldots + e_{j_\nu}\beta_\nu^2 \\
&\;\;\vdots \\
S_{2t} = \mathbf{r}(\alpha^{2t}) &= e_{j_1}\beta_1^{2t} + e_{j_2}\beta_2^{2t} + \ldots + e_{j_\nu}\beta_\nu^{2t}
\end{aligned}
\tag{38}
$$

There are $2t$ unknowns (t error values and t locations), and $2t$ simultaneous equations. However, these $2t$ simultaneous equations cannot be solved in the usual way because they are nonlinear (as some of the unknowns have exponents). Any technique that solves this system of equations is known as a *Reed-Solomon decoding algorithm*.

Once a nonzero syndrome vector (one or more of its symbols are nonzero) has been computed, that signifies that an error has been received. Next, it is necessary to learn the location of the error or errors. An error-locator polynomial, $\boldsymbol{\sigma}(X)$, can be defined as follows:

$$\begin{aligned}\sigma(X) &= (1 + \beta_1 X)(1 + \beta_2 X) \ldots (1 + \beta_\nu X) \\ &= 1 + \sigma_1 X + \sigma_2 X^2 + \ldots + \sigma_\nu X^\nu\end{aligned} \tag{39}$$

The roots of $\boldsymbol{\sigma}(X)$ are $1/\beta_1, 1/\beta_2, \ldots, 1/\beta_\nu$. The reciprocal of the roots of $\boldsymbol{\sigma}(X)$ are the error-location numbers of the error pattern $\mathbf{e}(X)$. Then, using autoregressive modeling techniques [7], we form a matrix from the syndromes, where the first t syndromes are used to predict the next syndrome. That is,

$$\begin{bmatrix} S_1 & S_2 & S_3 & \cdots & S_{t-1} & S_t \\ S_2 & S_3 & S_4 & \cdots & S_t & S_{t+1} \\ & & & \vdots & & \\ S_{t-1} & S_t & S_{t+1} & \cdots & S_{2t-3} & S_{2t-2} \\ S_t & S_{t+1} & S_{t+2} & \cdots & S_{2t-2} & S_{2t-1} \end{bmatrix} \begin{bmatrix} \sigma_t \\ \sigma_{t-1} \\ \vdots \\ \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} -S_{t+1} \\ -S_{t+2} \\ \vdots \\ -S_{2t-1} \\ -S_{2t} \end{bmatrix} \tag{40}$$

We apply the autoregressive model of Equation (40) by using the largest dimensioned matrix that has a nonzero determinant. For the (7, 3) double-symbol-error correcting R-S code, the matrix size is 2 × 2, and the model is written as follows:

$$\begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix} \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} S_3 \\ S_4 \end{bmatrix} \tag{41}$$

$$\begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} \alpha^6 \\ 0 \end{bmatrix} \tag{42}$$

To solve for the coefficients $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ and of the error-locator polynomial, $\boldsymbol{\sigma}(X)$, we first take the inverse of the matrix in Equation (42). The inverse of a matrix $[A]$ is found as follows:

$$\text{Inv}\left[A\right] = \frac{\text{cofactor}\left[A\right]}{\det\left[A\right]}$$

Therefore,

$$\det \begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} = \alpha^3\alpha^6 - \alpha^5\alpha^5 = \alpha^9 + \alpha^{10} \tag{43}$$

$$= \alpha^2 + \alpha^3 = \alpha^5$$

$$\text{cofactor} \begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} = \begin{bmatrix} \alpha^6 & \alpha^5 \\ \alpha^5 & \alpha^3 \end{bmatrix} \tag{44}$$

$$\text{Inv}\begin{bmatrix}\alpha^3 & \alpha^5\\ \alpha^5 & \alpha^6\end{bmatrix} = \frac{\begin{bmatrix}\alpha^6 & \alpha^5\\ \alpha^5 & \alpha^3\end{bmatrix}}{\alpha^5} = \alpha^{-5}\begin{bmatrix}\alpha^6 & \alpha^5\\ \alpha^5 & \alpha^3\end{bmatrix}$$

$$= \alpha^2\begin{bmatrix}\alpha^6 & \alpha^5\\ \alpha^5 & \alpha^3\end{bmatrix} = \begin{bmatrix}\alpha^8 & \alpha^7\\ \alpha^7 & \alpha^5\end{bmatrix} = \begin{bmatrix}\alpha^1 & \alpha^0\\ \alpha^0 & \alpha^5\end{bmatrix} \tag{45}$$

Safety Check

If the inversion was performed correctly, the multiplication of the original matrix by the inverted matrix should yield an identity matrix.

$$\begin{bmatrix}\alpha^3 & \alpha^5\\ \alpha^5 & \alpha^6\end{bmatrix}\begin{bmatrix}\alpha^1 & \alpha^0\\ \alpha^0 & \alpha^5\end{bmatrix} = \begin{bmatrix}\alpha^4 + \alpha^5 & \alpha^3 + \alpha^{10}\\ \alpha^6 + \alpha^6 & \alpha^5 + \alpha^{11}\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \tag{46}$$

Continuing from Equation (42), we begin our search for the error locations by solving for the coefficients of the error-locator polynomial, $\boldsymbol{\sigma}(X)$.

$$\begin{bmatrix}\sigma_2\\ \sigma_1\end{bmatrix} = \begin{bmatrix}\alpha^1 & \alpha^0\\ \alpha^0 & \alpha^5\end{bmatrix}\begin{bmatrix}\alpha^6\\ 0\end{bmatrix} = \begin{bmatrix}\alpha^7\\ \alpha^6\end{bmatrix} = \begin{bmatrix}\alpha^0\\ \alpha^6\end{bmatrix} \tag{47}$$

From Equations (39) and (47), we represent $\boldsymbol{\sigma}(X)$ as shown below.

$$\begin{aligned}\sigma(X) &= \alpha^0 + \sigma_1 X + \sigma_2 X^2\\ &= \alpha^0 + \alpha^6 X + \alpha^0 X^2\end{aligned} \tag{48}$$

The roots of $\boldsymbol{\sigma}(X)$ are the reciprocals of the error locations. Once these roots are located, the error locations will be known. In general, the roots of $\boldsymbol{\sigma}(X)$ may be one or more of the elements of the field. We determine these roots by exhaustive testing of the $\boldsymbol{\sigma}(X)$ polynomial with each of the field elements, as shown below. Any element X that yields $\boldsymbol{\sigma}(X) = \mathbf{0}$ is a root, and allows us to locate an error.

$$\boldsymbol{\sigma}(\alpha^0) = \alpha^0 + \alpha^6 + \alpha^0 = \alpha^6 \neq \mathbf{0}$$

$$\boldsymbol{\sigma}(\alpha^1) = \alpha^0 + \alpha^7 + \alpha^2 = \alpha^2 \neq \mathbf{0}$$

$$\boldsymbol{\sigma}(\alpha^2) = \alpha^0 + \alpha^8 + \alpha^4 = \alpha^6 \neq \mathbf{0}$$

$$\boldsymbol{\sigma}(\alpha^2) = \alpha^0 + \alpha^8 + \alpha^4 = \alpha^6 \neq \mathbf{0}$$

$$\boldsymbol{\sigma}(\alpha^3) = \alpha^0 + \alpha^9 + \alpha^6 = \mathbf{0} \Rightarrow \text{ERROR}$$

$$\boldsymbol{\sigma}(\alpha^4) = \alpha^0 + \alpha^{10} + \alpha^8 = \mathbf{0} \Rightarrow \text{ERROR}$$

$$\boldsymbol{\sigma}(\alpha^5) = \alpha^0 + \alpha^{11} + \alpha^{10} = \alpha^2 \neq \mathbf{0}$$

$$\boldsymbol{\sigma}(\alpha^6) = \alpha^0 + \alpha^{12} + \alpha^{12} = \alpha^0 \neq \mathbf{0}$$

As seen in Equation (39), the error locations are at the inverse of the roots of the polynomial. Therefore $\boldsymbol{\sigma}(\alpha^3) = \mathbf{0}$ indicates that one root exits at $1/\beta_l = \alpha^3$. Thus, $\beta_{l =} 1/\alpha^3 = \alpha^4$. Similarly, $\boldsymbol{\sigma}(\alpha^4) = \mathbf{0}$ indicates that another root exits at $1/\beta_{l'} = \alpha^4$. Thus, $\beta_{l'} = 1/\alpha^4 = \alpha^3$, where l and l' refer to the first, second, …, ν^{th} error. Therefore, in this example, there are two-symbol errors, so that the error polynomial is of the following form:

$$\mathbf{e}(X) = e_{j_1} X^{j_1} + e_{j_2} X^{j_2} \tag{49}$$

The two errors were found at locations α^3 and α^4. Note that the indexing of the error-location numbers is completely arbitrary. Thus, for this example, we can designate the $\beta_l = \alpha^{j_l}$ values as $\beta_1 = \alpha^{j_1} = \alpha^3$ and $\beta_2 = \alpha^{j_2} = \alpha^4$.

Error Values

An error had been denoted e_{j_1}, where the index j refers to the error location and the index l identifies the l^{th} error. Since each error value is coupled to a particular location, the notation can be simplified by denoting e_{j_1}, simply as e_1. Preparing to determine the error values e_1 and e_2 associated with locations $\beta_1 = \alpha^3$ and $\beta_2 = \alpha^4$, any of the four syndrome equations can be used. From Equation (38), let's use S_1 and S_2.

$$S_1 = \mathbf{r}(\alpha) = e_1\beta_1 + e_2\beta_2 \tag{50}$$

$$S_2 = \mathbf{r}(\alpha^2) = e_1\beta_1^2 + e_2\beta_2^2$$

We can write these equations in matrix form as follows:

$$\begin{bmatrix} \beta_1 & \beta_2 \\ \beta_1{}^2 & \beta_2{}^2 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} \tag{51}$$

$$\begin{bmatrix} \alpha^3 & \alpha^4 \\ \alpha^6 & \alpha^8 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} \alpha^3 \\ \alpha^5 \end{bmatrix} \tag{52}$$

To solve for the error values e_1 and e_2, the matrix in Equation (52) is inverted in the usual way, yielding

$$\text{Inv} \begin{bmatrix} \alpha^3 & \alpha^4 \\ \alpha^6 & \alpha^1 \end{bmatrix} = \frac{\begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix}}{\alpha^3\alpha^1 - \alpha^6\alpha^4}$$

$$= \frac{\begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix}}{\alpha^4 + \alpha^3} = \alpha^{-6} \begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix} = \alpha^1 \begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix} \tag{53}$$

$$= \begin{bmatrix} \alpha^2 & \alpha^5 \\ \alpha^7 & \alpha^4 \end{bmatrix} = \begin{bmatrix} \alpha^2 & \alpha^5 \\ \alpha^0 & \alpha^4 \end{bmatrix}$$

Now, we solve Equation (52) for the error values, as follows:

$$\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} \alpha^2 & \alpha^5 \\ \alpha^0 & \alpha^4 \end{bmatrix} \begin{bmatrix} \alpha^3 \\ \alpha^5 \end{bmatrix} = \begin{bmatrix} \alpha^5 + \alpha^{10} \\ \alpha^3 + \alpha^9 \end{bmatrix} = \begin{bmatrix} \alpha^5 + \alpha^3 \\ \alpha^3 + \alpha^2 \end{bmatrix} = \begin{bmatrix} \alpha^2 \\ \alpha^5 \end{bmatrix} \tag{54}$$

Correcting the Received Polynomial with Estimates of the Error Polynomial

From Equations (49) and (54), the estimated error polynomial is formed, to yield the following:

$$\begin{aligned}\hat{\mathbf{e}}(X) &= e_1 X^{j_1} + e_2 X^{j_2} \\ &= \alpha^2 X^3 + \alpha^5 X^4\end{aligned} \tag{55}$$

The demonstrated algorithm repairs the received polynomial, yielding an estimate of the transmitted codeword, and ultimately delivers a decoded message. That is,

$$\hat{\mathbf{U}}(X) = \mathbf{r}(X) + \hat{\mathbf{e}}(X) = \mathbf{U}(X) + \mathbf{e}(X) + \hat{\mathbf{e}}(X) \tag{56}$$

$$\mathbf{r}(X) = (100) + (001)X + (011)X^2 + (100)X^3 + (101)X^4 + (110)X^5 + (111)X^6$$

$$\hat{\mathbf{e}}(X) = (000) + (000)X + (000)X^2 + (001)X^3 + (111)X^4 + (000)X^5 + (000)X^6$$

$$\hat{\mathbf{U}}(X) = (100) + (001)X + (011)X^2 + (101)X^3 + (010)X^4 + (110)X^5 + (111)X^6$$

$$= \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 + \alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6 \tag{57}$$

Since the message symbols constitute the rightmost $k = 3$ symbols, the decoded message is

$$\underbrace{010}_{\alpha^1}\ \underbrace{110}_{\alpha^3}\ \underbrace{111}_{\alpha^5}$$

which is exactly the test message that was chosen earlier for this example. For further reading on R-S coding, see the collection of papers in reference [8].

Conclusion

In this article, we examined Reed-Solomon (R-S) codes, a powerful class of nonbinary block codes, particularly useful for correcting burst errors. Because coding efficiency increases with code length, R-S codes have a special attraction. They can be configured with long block lengths (in bits) with less decoding time than other codes of similar lengths. This is because the decoder logic works with symbol-based rather than bit-based arithmetic. Hence, for 8-bit symbols, the arithmetic operations would all be at the byte level. This increases the complexity of the logic, compared with binary codes of the same length, but it also increases the throughput.

References

[1] Reed, I. S. and Solomon, G., "Polynomial Codes Over Certain Finite Fields," *SIAM Journal of Applied Math.*, vol. 8, 1960, pp. 300-304.

[2] Gallager, R. G., *Information Theory and Reliable Communication* (New York: John Wiley and Sons, 1968).

[3] Sklar, B., *Digital Communications: Fundamentals and Applications, Second Edition* (Upper Saddle River, NJ: Prentice-Hall, 2001).

[4] Odenwalder, J. P., *Error Control Coding Handbook*, Linkabit Corporation, San Diego, CA, July 15, 1976.

[5] Berlekamp, E. R., Peile, R. E., and Pope, S. P., "The Application of Error Control to Communications," *IEEE Communications Magazine*, vol. 25, no. 4, April 1987, pp. 44-57.

[6] Hagenauer, J., and Lutz, E., "Forward Error Correction Coding for Fading Compensation in Mobile Satellite Channels," *IEEE JSAC*, vol. SAC-5, no. 2, February 1987, pp. 215-225.

[7] Blahut, R. E., *Theory and Practice of Error Control Codes* (Reading, MA: Addison-Wesley, 1983).

[8] Wicker, S. B. and Bhargava, V. K., ed., *Reed-Solomon Codes and Their Applications* (Piscataway, NJ: IEEE Press, 1983).

About the Author

Bernard Sklar is the author of *Digital Communications: Fundamentals and Applications, Second Edition* (Prentice-Hall, 2001, ISBN 0-13-084788-7).