the special algebraic structure resident in BCH and RS codes allows an algebraic solution for the error pattern directly from the computed syndrome.

The first formulation of an algebraic decoding procedure for binary BCH codes was given by Peterson [40] and extended to nonbinary codes by Gorenstein and Zierler [41]. Subsequent important contributions were made by Chien [42], Berlekamp [43], Forney [44], and Massey [45]. Recently, a transform-domain perspective has become popular, as described by Blahut [4, 46]. Our discussion cannot be comprehensive in treating a large body of literature, but will focus on a single transform-oriented approach that highlights the digital signal-processing aspects of the problem. This approach may not always provide the most computationally efficient procedure, but it is relatively simple to grasp.

The procedure presented in this section is generally referred to as ***errors-only decoding***, to distinguish the algorithm from generalizations that process errors and erasures simultaneously. The latter is discussed in the following section and will be seen to be a modification of the errors-only procedure. The algorithm presented here is a ***bounded distance*** (incomplete) decoder, as are virtually all algebraic decoding procedures. This means that decoding will correctly occur if, and only if, the number of errors present is less than or equal to $\lfloor(\delta - 1)/2\rfloor$, where δ is the design distance of the code. When the number of errors exceeds this value, the decoder may either be unable to decode or may decode incorrectly.

We begin by assuming that the error pattern $\mathbf{e}$ has at most $t = \lfloor(\delta - 1)/2\rfloor$ errors. Let the error positions be designated $\mathbf{m} = (m_1, m_2, \ldots, m_t)$ and the corresponding error values, or error types, be represented by $\mathbf{e} = (e_{m_1}, e_{m_2}, \ldots, e_{m_t})$. We then have that $m_i \in \{0, 1, \ldots, n-1\}$ and $e_{m_i} \in \mathrm{GF}(q)$. In these terms, we write the error polynomial as

$$e(D) = e_{m_1} D^{m_1} + e_{m_2} D^{m_2} + \cdots + e_{m_t} D^{m_t}. \tag{5.5.13}$$

(If fewer than t errors occur, we merely define some of the error values to be the zero symbol.)

We now define the ***syndrome sequence*** S_i, $i = 0, 1, \ldots, \delta - 2$, to be the value of the received polynomial evaluated at the $\delta - 1$ consecutive roots used to define the BCH (or RS) code; that is, we evaluate the received polynomial $r(D)$ at the field elements $\alpha^j, \alpha^{j+1}, \ldots, \alpha^{j+\delta-2}$:

$$\begin{aligned} S_i &= r(D)|_{D=\alpha^{j+i}} = e(D)|_{D=\alpha^{j+i}} \\ &= \sum_{p=1}^{t} e_{m_p} (\alpha^{j+i})^{m_p}, \qquad i = 0, 1, \ldots, \delta - 2. \end{aligned} \tag{5.5.14}$$

The second step follows because $r(D) = x(D) + e(D)$, and $x(D)$ evaluated at the prescribed roots yields zero, by definition.

An efficient way to compute the syndrome symbol S_i is by polynomial evaluation using what is known as Horner's rule, a nested multiply-and-add algorithm:

$$S_i = ((\ldots(r_{n-1}\alpha^{i+j} + r_{n-2})\alpha^{i+j} + r_{n-3})\alpha^{i+j} + \cdots + r_1)\alpha^{i+j} + r_0. \tag{5.5.15}$$

Thus, all syndrome digits could be computed in parallel using $\delta - 1$ recursive digital filters, as indicated in Figure 5.5.3.

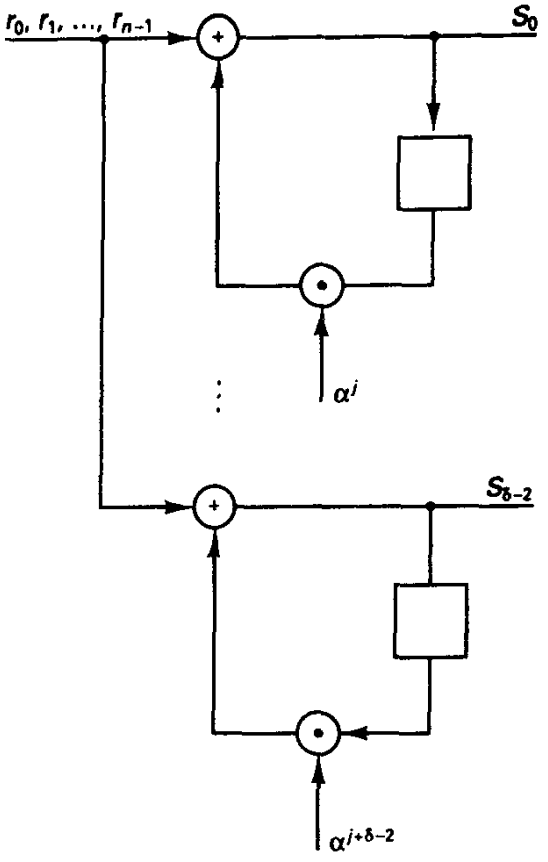

Figure 5.5.3 Recursive computation of syndrome digits for BCH/RS codes. All arithmetic over GF(q).

The nomenclature suggests that the syndrome symbols defined here, which are members of GF(q^m), are related to the syndrome polynomial $s(D)$ over GF(q) defined earlier. From the definition of $s(D)$, we have

$$r(D) = a(D)g(D) + s(D). \tag{5.5.16}$$

Then

$$S_i = r(D)|_{D=\alpha^{j+i}} = s(D)|_{D=\alpha^{j+i}}, \tag{5.5.17}$$

since α^{j+i} is a root of $g(D)$. In other words, the syndrome values defined by (5.5.17) are merely equivalent to evaluation of the syndrome polynomial, defined by $s(D) = r(D) \bmod g(D)$, at the prescribed code roots.

It is also important to notice that the syndrome sequence corresponds exactly to a portion of the DFT of the sequence $\mathbf{r}$; that is,

$$S_i = r(D)|_{D=\alpha^{j+i}} = R_{j+i}, \qquad i = 0, 1, \ldots, \delta - 2. \tag{5.5.18}$$

To see the implication of this, we consider performing the DFT of $\mathbf{r}$ to obtain, by linearity of the Fourier transform,

$$R_i = X_i + E_i, \qquad i = 0, 1, \ldots, n - 1, \tag{5.5.19}$$

where $\{X_i\}$ and $\{E_i\}$ are, respectively, the n-point transform values of the transmitted code vector and the error pattern, neither of which is known to the decoder. However,

we do know that in positions $i = j, j+1, \ldots, j+\delta-2$, $R_i \equiv E_i$ since the codeword transform values in these positions are zero (the alternative definition of the BCH/RS codes). In these positions, the DFT of $\mathbf{r}$ therefore reveals *exactly* the transform of $\mathbf{e}$. Furthermore, from (5.5.18) we have that these error transform entries are related to the syndromes by $S_i = E_{i+j}, i = 0, 1, \ldots, \delta-2$. If we could somehow extrapolate these transform coefficients to ascertain the entire transform $\mathbf{E}$, and then an inverse transform could recover $\mathbf{e}$. All that would remain is subtraction of the estimated $\mathbf{e}$ from $\mathbf{r}$ to perform error correction. Alternatively, we could first subtract in the transform domain to repair $\mathbf{R}$ and then inverse transform to obtain $\mathbf{r}$. Since, however, there are many such $\mathbf{e}$ sequences having the observed transform coefficients in $\delta-1$ consecutive positions, we again seek the minimum-weight error pattern whose transform satisfies the given syndrome (or DFT) constraints. The primary difficulty in BCH/RS decoding is making this step.

Before proceeding, we note that the syndrome equations formed by (5.5.14) are a set of $\delta-1$ *nonlinear* equations over GF(q) in the (up to) $2t$ unknowns $\{m_i\}$ and $\{e_{m_i}\}$. In the binary case, there are really only t unknowns, since the only value of an error is 1, and it may appear that the system of equations is inconsistent. However, it should be remembered that not all the $\delta-1$ parity check equations are independent in the binary case. In the more general case of RS decoding, we need all the syndrome equations to solve for the t error locations and the t error values.

The desired solution is a vector $(\mathbf{e}, \mathbf{m})$ for which $\mathbf{e}$ has minimum weight, that is, the smallest number of nonzero entries in $\mathbf{e}$. Any procedure for solving these nonlinear equations, including trial and error, is a decoding algorithm. Early researchers [40, 41] discovered a clever trick: rather than seek a direct solution of this system, we manipulate the problem to one of solving two linear systems of equations.

To do so, we define

$$L_p = \alpha^{m_p}, \qquad p = 1, 2, \ldots, t, \tag{5.5.20}$$

as the ***error locator*** for the pth error. Note that L_p is an element of GF(q^m), the extension field invoked to define the BCH or RS code, but the decoder does not know the error locations at the outset.[19] With this definition, the syndrome equations (5.5.14) may be written in the form

$$S_i = \sum_{p=1}^{t} e_{m_p} (L_p)^{j+i}, \qquad i = 0, 1, \ldots, \delta-2, \tag{5.5.21}$$

which we observe is a system of *linear* equations for the error values, once the error locators become known.

We next introduce the ***connection polynomial***,[20] so named for reasons soon to be clear:

$$B(D) = \prod_{p=1}^{t} (1 - L_p D) = 1 + B_1 D + B_2 D^2 + \cdots + B_t D^t. \tag{5.5.22}$$

---

[19] In the Reed–Solomon case with $m = 1$, the code symbol field and the locator field are the same.

[20] This polynomial is also frequently known as the ***error locator polynomial.***

Of course, the decoder does not know *a priori* the polynomial $B(D)$, but note that we have defined the zeros of the connection polynomial to be the reciprocals of the error locators, L_p; that is, $B(\alpha^{-m_p}) = 0$, $p = 1, 2, \ldots, t$. Even more importantly, the inverse DFT of the sequence **B** has special behavior. Suppose that we pad the **B** sequence defined in (5.5.22) with zeros. The inverse transform coefficients are $b_{m_p} = B(\alpha^{-m_p}) = 0$, $p = 1, 2, \ldots, t$, and we have thus defined a frequency-domain sequence, **B**, whose time-domain sequence, **b**, is identically *zero* in precisely those positions where the error sequence, **e**, is *nonzero*.

Because e_m is assumed to be zero in the remaining positions, we have the time-domain relation

$$z_m = b_m e_m = 0, \qquad m = 0, 1, \ldots, n-1. \tag{5.5.23}$$

Now recall from Section 5.1 that multiplication of two sequences corresponds to cyclic convolution of their respective DFT sequences, by the convolution theorem for the DFT. Thus, in the frequency domain we find that the transform sequence **E** obeys

$$\sum_{p=0}^{n-1} B_p E_{i-p} = 0, \qquad i = 0, 1, \ldots, n-1, \tag{5.5.24}$$

with $B_0 = 1$ and indexes interpreted modulo n. However, since we have defined the polynomial $B(D)$ to have degree at most t, (5.5.24) may be reduced to

$$\sum_{p=0}^{t} B_p E_{i-p} = 0, \qquad i = 0, 1, \ldots, n-1, \quad B_0 = 1. \tag{5.5.25a}$$

An alternative way of writing (5.5.25a) is in recursive form:

$$E_i = -\sum_{p=1}^{t} B_p E_{i-p}, \qquad i = 0, \ldots, n-1, \tag{5.5.25b}$$

since $B_0 = 1$.

This recursion establishes a constraint between the complete transform, **E**, of the error pattern and the coefficients of the connection polynomial, once it is known. As mentioned, a fragment of the error pattern transform is known exactly, equivalent to the syndrome sequence, but the remaining entries in the transform are unknowns. It should be clear that there exist different connection polynomials that satisfy (5.5.25), corresponding to different choices for the error locations.

The recursion in (5.5.25b) holds in particular for $i = j, j+1, \ldots, j+\delta-1$. If we recall that the syndromes are related to the error transform coefficients through $S_i = E_{i+j}$, then (5.5.25) could be recast as a linear system of t equations relating the $\delta - 1$ syndromes and the t unknown connection polynomial (or error locator polynomial) coefficients. The earliest formulation of a decoding procedure simply solved this linear system for the B_i coefficients, using standard methods of matrix algebra. It was shown that if t or fewer errors occurred the solution was unique. However, since the true number of errors is unknown, we need to begin by assuming t errors, test the determinant of the system matrix in (5.5.25), and, if nonzero, determine the B_i solutions. If the determinant is zero, one less error is assumed and the solution reattempted. Once the coefficients of the polynomial $B(D)$ are found, we can then determine its roots, the reciprocals of error locators, using what is known as Chien search [42]. Once the locators are known, they

may be back-substituted into (5.5.14), leaving a *linear* system of equations for the error values. (In the binary case, this last step is unnecessary.)

In summary, introduction of the error locators and the connection polynomial $B(D)$ has changed the solution of the nonlinear system (5.5.14) into two stages of solving linear systems. The primary difficulty with this general method is one of solving the linear systems of equations for codes with large minimum distance, since their solution requires computation proportional to the cube of t.

Another interpretation is due to Massey [45], who realized that (5.5.25) could be synthesized by the linear feedback shift register (LFSR) shown in Figure 5.5.4, where the B_i are the *connection tap weights* defining the feedback. Our task is still to find the minimum-degree connection polynomial that is consistent with the observed error pattern transform. In a general context, the problem is one of determining the minimal order and coefficients of an autoregressive filter capable of synthesizing a given finite-length sequence. We say that a filter can synthesize a finite-length sequence if its output produces the desired sequence when loaded with appropriate initial conditions.

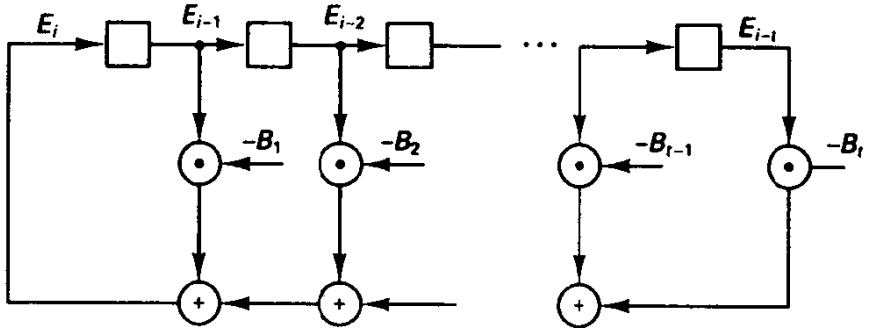

Figure 5.5.4 Feedback shift register recursively producing E_i sequence.

Massey's minimal-order linear feedback shift register synthesis technique is iterative. We begin with a zeroth-order system, $B(D) = 1$, and test whether it is capable of producing the sequence $E_i, i = j+1, \ldots, j+\delta-1$, when initialized with the first element, E_j, of the sequence. As soon as a discrepancy is uncovered, that is, the recursion (5.5.25b) fails to be satisfied, the coefficients and/or the order of the feedback shift register are corrected appropriately. Figure 5.5.5 provides a flow chart description of Massey's linear feedback shift register (LFSR) algorithm, which we shall not further justify. (A superb description is found in Blahut's text [4].)

The linkage of the LFSR procedure to decoding of BCH/RS codes is rigorously established by Massey [45]: provided t or fewer errors occur in a given codeword, the proper connection polynomial is obtained by applying the LFSR algorithm just presented to the syndrome sequence, $S_0, S_1, \ldots, S_{\delta-2}$, which again is just the portion of the transform vector known to exactly reflect the error pattern $E_j, E_{j+1}, \ldots, E_{j+\delta-1}$. The connection polynomial solution is unique if t or fewer errors occur. If too many errors are present, an incorrect connection polynomial may be determined, or it may be that a connection polynomial with degree t or less is not achievable, whence the error pattern is declared not correctable. Once the connection polynomial is determined, we merely need to find the reciprocals of the roots of the polynomial to establish the error

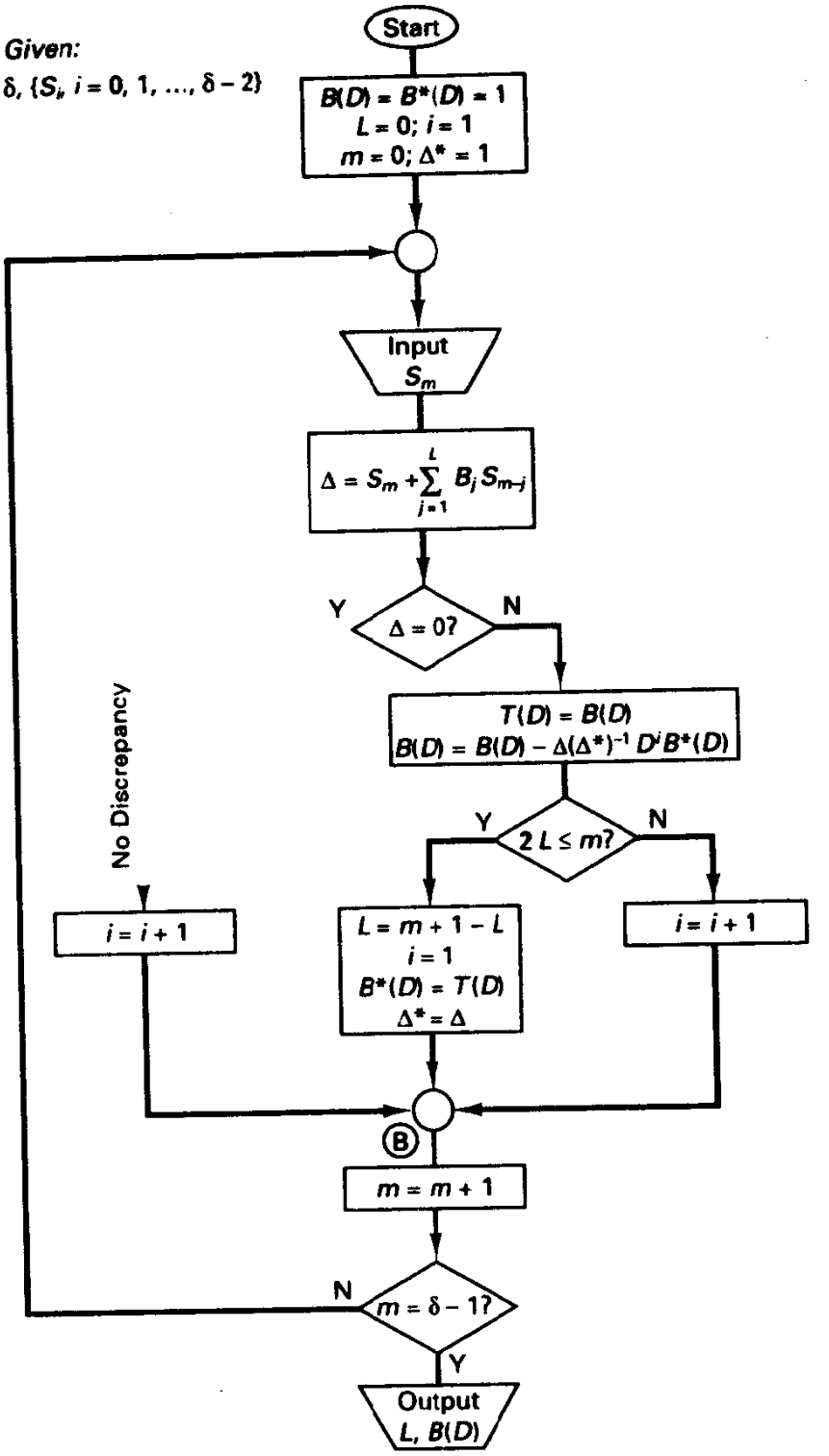

Figure 5.5.5 Algorithm for synthesizing LFSR from syndromes (after J. L. Massey, Jr.).

locators. Back substitution into (5.5.14) then leaves a linear system of equations relating the unknown error types. In any case the algorithm is a bounded-distance decoder, guaranteeing correction of up to t errors, but no more.

There is a slightly different procedure that relies even more heavily on transform methodology and that avoids solution for polynomial roots and the subsequent linear system solution. The received sequence $\mathbf{r}$ is first transformed to produce $\mathbf{R}$, whose positions $j, j+1, \ldots, j+\delta-2$ are known to exactly constitute the transform of the error pattern in these positions. [These are also the syndrome values defined in (5.5.14).] These transform coefficients are subjected to the LFSR synthesis procedure. Once the LFSR structure has been obtained, it is simple to recursively generate the remaining E_i terms, using the circuit of Figure 5.5.4 and periodic extension of the DFT coefficients. Then we compute an inverse DFT to produce $\hat{\mathbf{e}}$, our estimate of the error vector, which is finally subtracted from $\mathbf{r}$ to produce the estimated code vector. We claim this will always be correctly done if the actual number of errors is t or fewer, since the proper connection polynomial is found and the error locators thus properly determined [45].

Although we have described the procedure along the lines of Massey's shift-register synthesis formulation, a largely similar idea was earlier proposed by Berlekamp [43] for iteratively finding the error locator polynomial (called connection polynomial here), and the basic procedure is known commonly as the Berlekamp–Massey algorithm. When combined with the transform-based decoding perspective, we obtain the complete decoding flow chart indicated in Figure 5.5.6. The steps are summarized next.

TRANSFORM-BASED BCH/RS DECODER

1. Compute the $\delta - 1$ syndromes, either using Horner's rule or by extracting the appropriate fragment of the DFT of $\mathbf{r}$.
2. Use the syndromes to determine the LFSR solution of order t or less, if any such solution exists.
3. Extend the transform fragment recursively to produce the complete error transform, $\hat{\mathbf{E}}$.
4. Perform the inverse transform on $\hat{\mathbf{E}}$ to determine $\hat{\mathbf{e}}$, and subtract this solution from $\mathbf{r}$.
5. Finally, check if the result is a valid codeword by recomputing syndromes.

There are several checkpoints along the way. First, it is possible that the final value of L in the LFSR algorithm exceeds the degree of the corresponding connection polynomial $B(D)$, or that this length exceeds t, in which case we declare that the codeword has uncorrectable errors (this is a decoding failure). Also, upon inverse transformation to obtain the error pattern estimate, we may find that the number of errors (the Hamming weight of the inverse transform) does not equal L, the apparent number of errors found by the filter synthesis procedure. If so, we declare a decoding failure. Another check is whether all components of $\hat{\mathbf{e}}$ are in the field of the code symbols, GF(q). Recall from our discussion of finite field transforms that such inverse transforms may not be in the ground field unless the transform domain symbols have special relationships. (For RS codes, this last test is unnecessary, since

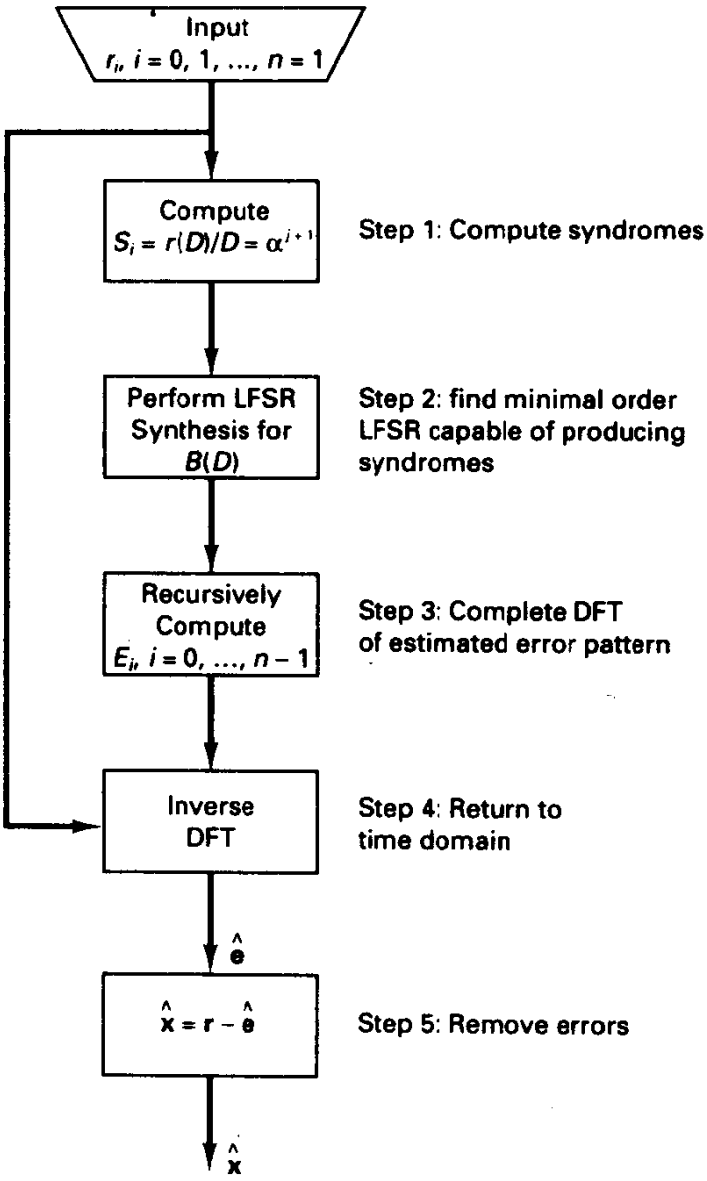

Figure 5.5.6 Flow chart for BCH/RS decoding, "errors only."

the two fields involved are identical.) Finally, it may be that the $\mathbf{r} - \hat{\mathbf{e}}$ solution does not constitute a valid codeword (the algorithm may "decode" to an invalid codeword when the number of errors exceeds t). This can be determined by a final syndrome check.

We shall now study the application of the algorithm to two cases. Our first example is a (7, 3) RS code over GF(8). This code has $d_{\min} = 5$ and is thus capable of correcting up to two errors in a code block. Normal applications involve longer codes than this, but the example code provides a simple, yet nontrivial, vehicle for studying the decoding algorithm.

Example 5.25 Decoding of a (7, 3) Reed–Solomon Code

Recall that all elements, other than 0 or 1, in GF(8) are primitive. Suppose in the enumeration of GF(8) provided in Figure 5.1.1 that we define the element 2 as α so that $\alpha^3 = 3$, for example. Now define $\alpha, \alpha^2, \ldots, \alpha^4$ as the four consecutive roots of the RS code; that is, we take $j = 1$. Suppose that the all-zeros sequence in the code is selected for transmission. (This entails no loss of generality and eases the computations.) Let symbol errors occur in positions 0 and 2, with error types $\alpha^0 = 1$ and α^2, respectively. Thus, the received polynomial is

$$r(D) = x(D) + e(D) = 1 + \alpha^2 D^2. \tag{5.5.26}$$

[The decoder will not have our inside knowledge that $r(D)$ is in fact the error polynomial.] With $j = 1$, the four syndromes that are to be computed are given by $S_i = r(\alpha^{i+1}), i = 0, 1, 2, 3$, and are determined to be

$$\begin{aligned} S_0 &= r(\alpha^1) = \alpha + \alpha^2\alpha^2 = \alpha^5, \\ S_1 &= r(\alpha^2) = \alpha^2, \\ S_2 &= r(\alpha^3) = \alpha^3, \\ S_3 &= r(\alpha^4) = \alpha^1. \end{aligned} \tag{5.5.27}$$

We reemphasize that this sequence reveals exactly the partial Fourier transform of the error sequence $E_1, \ldots, E_4$.

Given our prior knowledge, it might be instructive to write down the error locators and the corresponding connection polynomial before proceeding with decoding. The two error locators are

$$L_1 = \alpha^0 = 1, \qquad L_1 = \alpha^2, \tag{5.5.28}$$

and substitution into (5.5.20) gives the connection polynomial as

$$B(D) = (1 - D)(1 - \alpha^2 D) = 1 - \alpha^6 D + \alpha^2 D^2. \tag{5.5.29}$$

(It is perhaps worth reiterating that, because our field has characteristic 2, addition and subtraction are equivalent, and signs are unimportant). We could also observe prior to decoding here that the syndrome sequence (5.5.27) obeys the recursion (5.5.25b). For example, $S_2 = -B_1 S_1 - B_2 S_0 = \alpha^6\alpha^2 + \alpha^2\alpha^5 = \alpha^3$, consistent with (5.5.27).

Figure 5.5.7 lists the complete calculations of the decoding process, including specifics of the LFSR synthesis. We have listed in Figure 5.5.7 the LFSR variables at the point in the algorithm labeled point B near the bottom of the flow chart (Figure 5.5.5). The sequence of operations basically amounts to beginning with the zero-order recursion, $B(D) = 1$, finding a discrepancy, modifying the connection polynomial to an appropriate first-order recursion, trying this, and ultimately settling on a second-order connection polynomial.

Using the obtained connection polynomial, the remainder of the transform values can be extrapolated using (5.5.25b), and we find that

$$\mathbf{E} = (\alpha^6, \alpha^5, \alpha^2, \alpha^3, \alpha^1, \alpha^4, 0). \tag{5.5.30a}$$

Upon performing the inverse transform, we indeed discover that

$$\mathbf{e} = (1, 0, \alpha^2, 0, 0, 0, 0), \tag{5.5.30b}$$

which when subtracted from $\mathbf{r}$ repairs the errors. All the necessary consistency checks are passed.

Step 1: $S_i = r(D)|_{D=\alpha^{j+i}} \Rightarrow S_0 = \alpha^5,\ S_1 = \alpha^2,\ S_2 = \alpha^3,\ = \alpha$

Step 2: LFSR Synthesis

	m	s_m	Δ	$T(D)$	$B(D)$	L	$B^*(D)$	Δ^*	i
(initial)	0	–	–	–	1	0	1	1	1
	0	α^5	α^5	1	$1-\alpha^5 D$	1	1	α^5	1
	1	α^2	α^5	$1-\alpha^5 D$	$1-\alpha^4 D$	1	1	α^5	2
	2	α^3	α^4	$1-\alpha^4 D$	$1-\alpha^4 D-\alpha^6 D^2$	2	$1-\alpha^4 D$	α^4	1
	3	α	1	$1-\alpha^4 D-\alpha^6 D^2$	$1-\alpha^6 D-\alpha^2 D^2$	2	$1-\alpha^4 D$	α^4	2
	4	(stop)							

Conclude 2 errors present and $B(D) = 1 - \alpha^6 D - \alpha^2 D^2$

Step 3: Recursive Extensions for Complete E_i Sequence

$\hat{E}_i = \alpha^6 E_{i-1} + \alpha^2 E_{i-2}$, initialized with $E_1 = S_0$, $E_2 = S_1$

$\hat{\mathbf{E}} = (\alpha^6, \alpha^5, \alpha^2, \alpha^3, \alpha, \alpha^4, 0)$

Step 4: $\hat{\mathbf{e}} = \text{IDFT}(\hat{\mathbf{E}})$

$= (1, 0, \alpha^2, 0, 0, 0, 0)$

Check: Number errors $\leq L$ (✓)
Errors in field of code symbols (✓)

Step 5: $\hat{\mathbf{x}} = \mathbf{r} - \hat{\mathbf{e}} = (0, 0, 0, 0, 0, 0, 0)$

Check: Syndrome of $\hat{\mathbf{x}} = \mathbf{0}$ (✓)

Figure 5.5.7 Decoding calculations for (7, 3) RS code, Example 5.25.

Example 5.26 Decoding of (15, 5) Binary BCH Code

Let's consider the binary $t = 3$ error-correcting code described earlier in Example 5.20. We adopt $\alpha^1, \ldots, \alpha^6$ as the roots of $g(D)$, so we have a narrow-sense code ($j = 1$). Assume that three errors occur in positions 2, 7, and 9. Thus, assuming the all-zeros vector is selected for transmission, we have

$$r(D) = D^2 + D^7 + D^9. \tag{5.5.31}$$

Notice that the code is over the binary field here, but syndrome values, error locators, and transform values will lie in GF(16).

Computation of the $\delta - 1 = 6$ syndromes (or six elements of $\mathbf{R}$) yields

$$\mathbf{S} = (\alpha^8, \alpha^1, \alpha^{12}, \alpha^2, 0, \alpha^9). \tag{5.5.32}$$

Performing the iterative LFSR procedure gives a connection polynomial

$$B(D) = 1 - \alpha^8 D - \alpha^5 D^2 - \alpha^3 D^3. \tag{5.5.33}$$

Recursive extrapolation for the remaining nine values of $\mathbf{E}$ produces

$$\mathbf{E} = (1, \alpha^8, \alpha^1, \alpha^{12}, \alpha^2, 0, \alpha^9, \alpha^1, \alpha^4, \alpha^6, 0, \alpha^8, \alpha^3, \alpha^4, \alpha^2). \tag{5.5.34}$$

Upon inverse transformation, we will indeed find that the *binary* error vector

$$\mathbf{e} = (0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0) \tag{5.5.35}$$

is produced. Try the same exercise with one additional error in the message and notice whether the decoder completes its decoding or, if failing, where in the algorithm.

As earlier noted, the method presented here is a ***bounded distance*** (incomplete) decoder, capable of correct decoding if and only if the received vector is within t units of the transmitted codeword, with t determined by δ, the design distance. For BCH codes the true minimum distance may exceed the design distance, and thus a truly intelligent decoder could occasionally decode "beyond the BCH bound." Such procedures are rather special case and not of general interest. This is not an issue for RS codes since the true minimum distance equals the design distance.

When the bounded-distance decoder does not correctly decode, it frequently simply fails to decode, rather than produce an incorrect decoding. To see why, consider the space of n-tuples over GF(q). Incorrect decoding will occur only when $\mathbf{r} = \mathbf{x} + \mathbf{e}$ falls inside a radius-t ball about an incorrect codeword, and decoding failure (detected error) occurs when $\mathbf{r}$ lies in interstitial space between decoding balls. A simple volume calculation reveals that the fraction of n-space consumed by radius-t spheres is rather small for many BCH/RS codes. This is especially true for nonbinary codes. (See Exercise 5.5.2.) Thus, we expect that, with high probability, decoding failures are much more likely than incorrect decodings. We shall say more about such performance calculations in Section 5.10.

It is simple to modify the preceding decoding algorithm to perform bounded-distance decoding of patterns with $t_1 < t$ errors, providing less error correction in exchange for lower probability of undetected error. This corresponds to the mode 3 decoding strategy posed earlier in Section 5.2. In the LFSR algorithm, we simply announce decoding failure if the apparent number of errors, L, exceeds the *correction radius* t_1.

We complete this discussion with a brief analysis of computational complexity for decoding. Using Horner's rule, calculation of the syndrome values (step 1 in Figure 5.5.6) requires $n(\delta - 1)$ multiplications and additions, ignoring the skipping of possible zeros in $\mathbf{r}$, so the complexity is about $2n\delta$ operations[21] in GF(q^m). The LFSR algorithm requires on the order of $4L(\delta - 1)$ operations, but since we can exit the procedure if $L \geq \delta/2$, the LFSR complexity is upper-bounded by $2\delta^2$ operations. Once the feedback structure is identified, the remaining transform symbols can be computed with $2L(n-\delta-1) < \delta(n-1)$ operations. Step 4, the inverse transformation, requires $2n^2$ additions/multiplications for brute-force DFT evaluation. However, assuming that n is highly composite, for example, $n = 63 = 3 \cdot 3 \cdot 7$, then fast algorithms may reduce the transform complexity significantly, as with DFTs over the real or complex numbers.[22] Correction of the output vector requires $k < n$ operations in GF(q). In summary, if we hold code rate R fixed and set $\delta \approx n(1-R)$ as with RS codes, then the computations grow roughly as the square of block length for fixed-rate codes. Normalized per message symbol, the effort grows roughly linearly in

[21] An operation here is an addition or multiplication.

[22] Zero padding can be used if necessary to obtain a desired DFT block length.

block length. In particular, complexity is much less than exponential in block length. Actual computation needs to be evaluated in the context of specific codes, however, and there are undoubtedly clever approaches to speeding the decoding process.

5.5.3 Errors-and-Erasures Decoding

The previous discussion pertains to what is known as *errors-only* decoding, meaning that the received symbol alphabet is the same as the transmitted alphabet, GF(q), and symbols in a received codeword are simply either correct or in error. An important variation in decoding is possible when the demodulator can provide additional information in the form of erasures when the reliability of a certain symbol is poor. In a simple binary situation, the code symbols could be modulated with an antipodal signal set, but the demodulator could provide a three-level quantization of the received random variable, with the middle zone of the quantizer (values near zero) representing the low-confidence erasure zone. In other cases, the presence of strong interference during a given symbol may be detectable, causing the demodulator to pass along erasure information to the decoder. We have earlier seen that the number of fillable erasures is twice the number of correctable errors, so accommodating erasures seems very worthwhile. Of course, we must be prudent in setting erasure thresholds so that erasing of correctly decided symbols is not too frequent.

In the *binary* case, errors-and-erasures decoding is relatively simple. We suppose that there are s erasures and r errors, such that $2r + s \leq \delta - 1$. We first attempt decoding with all erasure positions set to 0. (If the code is a binary BCH code, we can use the preceding decoding algorithm, but this errors-and-erasures procedure is more general, not even restricted to cyclic codes, and only requires a decoder capable of correctly processing r errors.) If decoding is successful, that is, the decoder locates an error pattern with r or fewer errors, we save the error pattern and note its Hamming weight. Next, we place 1's in all erasure locations and decode again; if successful, we note the weight of the error pattern. Among the two possible candidate error patterns, we decide in favor of that having smallest Hamming weight. We can demonstrate that this two-pass procedure guarantees correct decoding provided r and s meet the condition stated; this is left as an exercise. Of course, it is still possible for decoding to be in error or, more often, to simply fail when $2r + s \geq \delta$.

Errors-and-erasures decoding of *nonbinary* BCH and RS codes is possible using a modification of the algorithm just presented, due to Forney [44]. Suppose that there are r errors in positions $m_1, m_2, \ldots, m_r$ whose error types are e_{m_p}, $p = 1, \ldots, r$, respectively. Let there be s erasures declared in positions l_p, $p = 1, 2, \ldots, s$. As before, we define the error locators as $L_p = \alpha^{m_p}$, $p = 1, 2, \ldots, r$, and also in a similar vein define the ***erasure locators*** as $Z_p = \alpha^{l_p}$, $p = 1, 2, \ldots, s$. In the latter case of erasures, all the locator numbers are known, since erasure positions are specified by the demodulator. Finally, we let the correct values of the codeword in these erased locations be denoted by x_{l_p}, $p = 1, \ldots, s$.

We begin decoding by first stuffing 0 symbols into the erased positions, in effect introducing errors equal to $-x_{l_p}$ in code positions l_p. The decoder then proceeds to compute the syndromes S_i, $i = 0, 1, \ldots, \delta - 2$, exactly as before. Again, these may be

interpreted as a $\delta - 1$ consecutive entries, R_{j+i}, $i = 0, 1, \ldots, \delta - 2$, in the DFT of the received sequence, as modified by insertion of 0's.

The values of the syndromes are expressible as

$$S_i = \sum_{p=1}^{r} e_{m_p} L_p^{i+j} + \sum_{p=1}^{s} -x_{l_p} Z_p^{i+j}, \qquad i = 0, 1, \ldots, \delta - 2, \tag{5.5.36a}$$

where again j is the first power of α in the root set used to define the code. Notice that each syndrome symbol contains a term due to outright errors in the transmission, as in the previous section, plus a contribution due to the induced errors. Equivalently, the syndromes defined are related to the DFT of the *error-and-erasure* pattern by

$$S_i = E_{i+j}, \qquad i = 0, 1, \ldots, \delta - 2, \tag{5.5.36b}$$

where $\mathbf{E} = (E_0, E_1, \ldots, E_{n-1})$ is the DFT of the sequence of outright errors and induced errors due to zero stuffing.

Next, we define the ***erasure-locator polynomial*** as

$$\Lambda(D) = \prod_{p=1}^{s} (D - Z_p) = \lambda_0 D^s + \lambda_1 D^{s-1} + \cdots + \lambda_s, \qquad \lambda_0 = 1. \tag{5.5.37}$$

In contrast with the error locator polynomial, this polynomial is completely specified at the outset by the erasure positions. Notice also that $\Lambda(D)$ is defined to have the error locators $\{Z_p\}$, rather than their reciprocals, as its roots.

Forney [44] proposed a modification of the original syndromes according to the linear transformation

$$T_i = \sum_{p=0}^{s} \lambda_p S_{i+s-p}, \qquad 0 \le i \le \delta - s - 2. \tag{5.5.38}$$

This expresses the input/output relation for an $(s+1)$-tap linear transversal filter excited by the syndrome sequence S_i. The resulting $\delta - 1 - s$ ***modified syndromes*** have erasure and error information incorporated in a manner that will allow the previous strategy for errors-only decoding to determine the r error locations. To see how, we write the modified syndromes using (5.5.36a) and (5.5.38) to obtain

$$\begin{aligned} T_i &= \sum_{m=0}^{s} \lambda_m \left[\sum_{p=1}^{r} e_{m_p} L_p^{i+s-m+j} + \sum_{p=1}^{s} -x_{l_p} Z_p^{i+s-m+j} \right] \\ &= \sum_{p=1}^{r} e_{m_p} L_p^{i+j} \sum_{m=0}^{s} \lambda_m L_p^{s-m} + \sum_{p=1}^{r} -x_{l_p} Z_p^{i+j} \sum_{m=0}^{s} \lambda_m Z_p^{s-m}. \end{aligned} \tag{5.5.39}$$

The final summation is zero since it can be recognized as the erasure locator polynomial $\Lambda(D)$ evaluated at one of its roots; thus, the second product of sums in (5.5.39) vanishes. Also, we can note that the second sum in the remaining term is the erasure locator polynomial evaluated at one of the *error* locators, L_p, and write this sum as $\Lambda(L_p)$.

Next, we introduce the quantity $W_p = e_{m_p} L_p \Lambda(L_p)$, $p = 1, 2, \ldots, r$, and we are then able to write (5.5.39) as

$$T_i = \sum_{m=1}^{r} W_m L_m^{i+j}, \qquad 0 \le i \le \delta - s - 2. \tag{5.5.40}$$

The important observation now is that (5.5.40) relates the modified syndromes T_i to the error locators in identical form to that found in (5.5.21) for errors-only processing. Specifically, it can be shown that the T_i obey the recursion

$$T_i = -\sum_{p=1}^{r} B_p T_{i-p}, \qquad i = r, r+1, \ldots, \tag{5.5.41}$$

where the B_p's are coefficients of the connection polynomial introduced in the previous section. The task is to find the minimal-order recursion consistent with the calculated modified syndromes. The Berlekamp–Massey algorithm is a convenient procedure for doing this task.

This LFSR device can be used to extend the T_i sequence to cover $0 \le i \le n-1$. This sequence is related to the transform of the errors-and-erasures pattern through (5.5.38) and (5.5.36b), and we must invert (5.5.38) to recover this sequence. Specifically, if we write out the system (5.5.38) beginning with the equation for $T_{\delta-s-2}$, we find it involves only a single unknown, $S_{\delta-1}$. Once this is found, we move to the equation for $T_{\delta-s-1}$, solving for S_δ, and so on until we have extended the initial syndrome sequence to an n-tuple. Recalling that the syndromes sequence is shifted relative to the error transform by j positions, as in (5.5.36b), we equivalently have recovered the transform of the errors-and-erasures pattern. This recovery of the errors-and-erasures transform from the extended modified syndrome sequence is effectively the operation of an autoregressive digital filter with feedback coefficients given by the error-locator polynomial. This follows from the fact that (5.5.38) is a digital transversal filter relation with $\{S_i\}$ as the input and $\{T_i\}$ as the output, with the filter coefficients given by λ_m. To solve this inverse problem, we inject **T** into a *feedback* filter having feedback taps specified by λ_m, as shown in Figure 5.5.8, and the desired **S** sequence is produced. This is related to the transform of the error-and-erasure pattern by

$$S_i = E_{i+j}, \qquad i = 0, 1, \ldots, n-1, \tag{5.5.42}$$

and, as before, we merely need to perform the inverse transform to determine $\hat{e}$, the

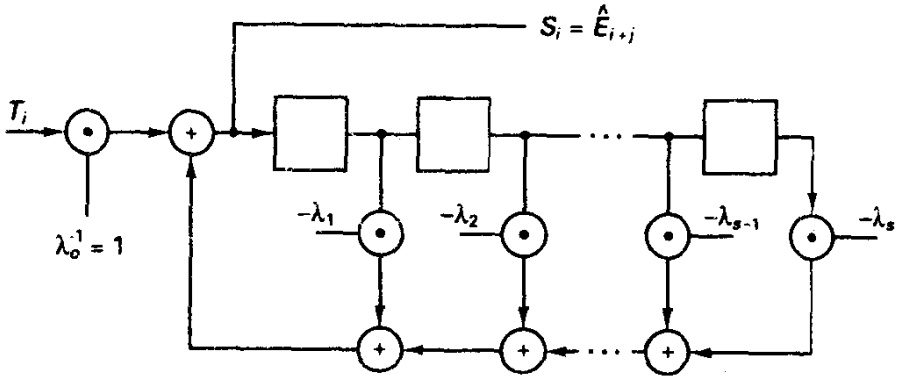

Figure 5.5.8 Feedback circuit for recovery of $S_i = E_{i+j}$ from T_i in errors-and-erasures decoding.

combined estimate of the original errors e_i and the discrepancies x_i to add to the inserted 0's.

Figure 5.5.9 illustrates the sequence of operations to be performed in this formulation of the errors-and-erasures algorithm. Correct decoding ensues whenever $2r + s \leq \delta - 1$. Again, there are multiple points where decoding failure can be announced.

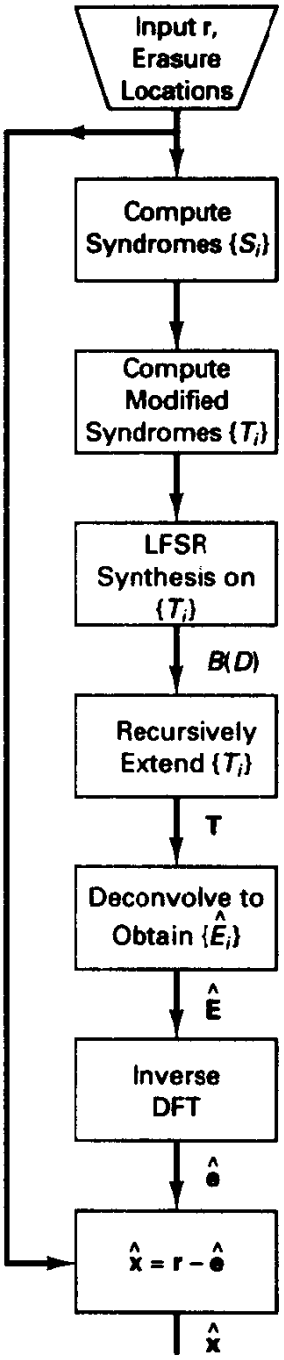

Figure 5.5.9 Flow chart for BCH/RS errors-and-erasures decoding.

Example 5.27 Errors-and-Erasures Decoding of (7, 3) RS Code

We pursue the decoding of the (7, 3) RS code discussed in Example 5.25, illustrating the erasure processing power of such codes. Specifically, let the received vector possess erasures in positions 0 and 1, along with a single error of type α^2 in position 2. This corresponds to a correctable error-and-erasure pattern. After placing 0's into the erased positions, the received polynomial is $r(D) = \alpha^2 D^2$, and the four syndromes are

$$\begin{aligned} S_0 &= r(\alpha^1) = \alpha^4, \\ S_1 &= r(\alpha^2) = \alpha^6, \\ S_2 &= r(\alpha^3) = \alpha^1, \\ S_3 &= r(\alpha^4) = \alpha^3. \end{aligned} \tag{5.5.43}$$

From the given erasure locations and appeal to GF(8) tables, the erasure locator polynomial is

$$\Lambda(D) = (D - \alpha^0)(D - \alpha^1) = \alpha^1 + \alpha^3 D + D^2. \tag{5.5.44}$$

This allows computation of the $\delta - s - 1 = 2$ modified syndromes, which follows from performing GF(8) arithmetic according to Figure 5.1.1:

$$\begin{aligned} T_0 &= \lambda_0 S_2 + \lambda_1 S_1 + \lambda_2 S_0 = 1, \\ T_1 &= \lambda_0 S_3 + \lambda_1 S_2 + \lambda_2 S_1 = \alpha^2. \end{aligned} \tag{5.5.45}$$

From these modified syndromes, it is found that the connection polynomial is a first-order polynomial

$$B(D) = 1 + \alpha^2 D. \tag{5.5.46}$$

Use of the recursion $T_j = -B_1 T_{j-1}$ for $j = 2, 3, \ldots, 6$ yields

$$T_2 = \alpha^4, \quad T_3 = \alpha^6, \quad T_4 = \alpha^1, \quad T_5 = \alpha^3, \quad T_6 = \alpha^5. \tag{5.5.47}$$

We now back-solve for S_4 using (5.5.38), which in this case becomes

$$T_2 = \lambda_0 S_4 + \lambda_1 S_3 + \lambda_2 S_2, \quad \text{or} \quad S_4 = (T_2 - \lambda_1 S_3 - \lambda_2 S_1)\lambda_0^{-1}. \tag{5.5.48}$$

Thus, $S_4 = \alpha^5$. Proceeding in similar manner to solve for the entire syndrome sequence, we obtain

$$\mathbf{S} = (\alpha^4, \alpha^6, \alpha^1, \alpha^3, \alpha^5, \alpha^0, \alpha^2). \tag{5.5.49}$$

We can then say that the estimate of the errors-and-erasures transform is a one-place rotation of this sequence (since $j = 1$), yielding $\hat{\mathbf{E}} = (\alpha^2, \alpha^4, \alpha^6, \alpha^1, \alpha^3, \alpha^5, \alpha^0)$, which inverse transforms to the time-domain sequence $\hat{\mathbf{e}} = (0, 0, \alpha^2, 0, 0, 0, 0)$, as desired.

If we had stuffed any other value into the erasure locations, the recovered time-domain estimate of the error/erasure pattern would have been adjusted accordingly.

5.5.4 ML and Near-ML Decoding

In many digital transmission applications, the demodulator is able to provide to the decoder more than its best estimate of a given symbol. We have argued in Chapter 4 *in favor of providing full likelihood information*, or perhaps finely quantized versions of symbol likelihoods to the decoder. Errors-and-erasures decoding represents a simple

step in this direction; the demodulator either makes a minimum-error probability decision on each symbol or produces an erasure if confidence in any decision is poor. Of course, if the threshold is set too high, a large fraction of correct decisions is erased and performance becomes poor. Conversely, setting a low threshold for being confident about a decision allows a higher probability of errors entering the decoder, again lowering decoder performance. On benign channels such as the AWGN channel, it has been found that errors-and-erasures decoding buys little gain (< 1 dB) in performance, even with optimized erasure declaration. On the other hand, fading channels and jamming environments benefit greatly from erasure processing, provided side information on fading amplitude or instantaneous noise level is available to assist in erasure declaration.

In this section, we return to the general ML decoding problem, posed initially in Section 5.2. On a memoryless channel, the codeword likelihood, or metric, is the sum of log likelihoods for each symbol in the codeword. The obvious difficulty with computing the ML decision is the exponential complexity; there are q^k codeword metrics to evaluate.

We might suspect there are ways to avoid this brute-force search policy. One category of decoding procedures makes use of ***trellis*** formulations of the code, utilizing the finite-state machine representation of a cyclic encoder as in Figure 5.4.4. Trellis search for the ML codeword can generally be performed with far less computation than with brute-force exhaustive search, although trellis decoding may still be prohibitively time consuming (the trellis has a maximum number of states given by $\max[q^k, q^{n-k}]$). We will defer this idea, however, until Chapter 6, where trellis codes are presented. The decoding of block codes using this viewpoint was initiated by Wolf [47] and lately studied by Forney [48].

Another category of more heuristic algorithms, due to Chase [49], is approximately ML. Bounded Hamming distance decoding will locate the one and only one codeword within t units of the received hard-decision vector, if one exists. Even if we successfully decode, the resulting codeword estimate may not be best under the ML metric for the channel. We expect such candidates will lie close in Hamming distance to the received vector, but perhaps not within t units.

Here is the general idea introduced by Chase. Let's assume an (n, k) cyclic code over GF(q) with minimum distance $d_{\min}$, and let $\mathbf{r}$ be the demodulator's hard-decision (symbol-by-symbol) estimate of the transmitted sequence. We employ a bounded-distance algebraic decoder to process $\mathbf{r}$, arriving perhaps at a codeword estimate $\hat{\mathbf{x}}$. (It may be that the decoder cannot decode if the Hamming distance between $\mathbf{r}$ and any codeword exceeds t.) The decoded codeword implies an apparent error pattern $\hat{\mathbf{e}}$, through

$$\hat{\mathbf{e}} = \mathbf{r} - \hat{\mathbf{x}}. \tag{5.5.50}$$

This error pattern is the best error sequence estimate under the criterion of minimum Hamming distance; however, it is not guaranteed to be the most likely error pattern when scored according to the ML metric for the problem. For example, if the binary code symbols are transmitted with antipodal signaling by a white Gaussian noise channel, the proper decision is in favor of the codeword that is closest in Euclidean distance. Nonetheless, we expect that the hard-decision sequence $\mathbf{r}$ is not a poor estimate and that small perturbations of this sequence, when decoded, may produce other candidate

codeword choices, among which the ML choice lies. In essence, the procedure explores the vicinity of space near $\mathbf{r}$, trying to locate codewords by the algebraic decoder, and compare the candidates thus produced according to the ML metric. An efficient procedure then is one that minimizes the number of perturbations and repeated decodings, while still locating the ML codeword with high probability.

With reference to Figure 5.5.10, let $\mathbf{z}$ be a ***test pattern***, an n-tuple over GF(q), and let the perturbed received sequence be

$$\tilde{\mathbf{r}} = \mathbf{r} - \mathbf{z}_j, \qquad j = 1, 2, \ldots, J. \tag{5.5.51}$$

We attempt decoding with this new input vector, and note that the new decision $\tilde{\mathbf{x}}_j$ has attached to it a (possibly new)[23] error pattern $\tilde{\mathbf{e}}_j = \tilde{\mathbf{r}} - \tilde{\mathbf{x}}_j$. This apparent error pattern is clearly related to the previous by

$$\tilde{\mathbf{e}}_j = \hat{\mathbf{x}} + \tilde{\mathbf{x}}_j + \hat{\mathbf{e}}. \tag{5.5.52}$$

Note that if both decodings produce the same decision the two implied error patterns are equivalent. If we repeat this process with different perturbations, we expect to produce multiple candidate codewords. These should ultimately be judged using the appropriate metric for the problem, for example, Euclidean distance. Chase refers to this as selecting in favor of the minimum analog weight sequence.

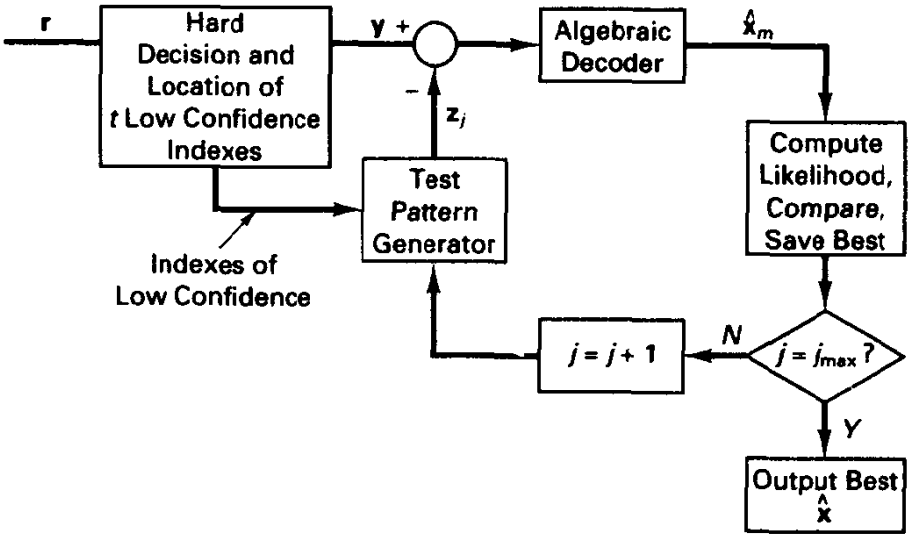

Figure 5.5.10 Chase's iterative decoder.

Chase argued that the test vectors $\mathbf{z}$ should not need to have Hamming weight larger than $\lfloor (d_{\min} - 1)/2 \rfloor$, for this will allow location and likelihood scoring of code vectors within $2t$ Hamming units of $\mathbf{r}$. In fact, the set of test patterns can be much smaller without significant sacrifice in performance, at least on the AWGN channel, and Chase's algorithm 2 represents an illustration of the method.

[23] It may not differ from the initial decoder choice.

CHASE'S ALGORITHM 2 (GENERALIZED TO q-ARY CASE)

1. Locate the $\lfloor d_{\min}/2 \rfloor$ positions in the codeword having the smallest likelihood or reliability.
2. Let T denote the set of vectors over GF(q) with at most $\lfloor d_{\min}/2 \rfloor$ nonzero symbols in the low-reliability positions. (This includes the all-zeros vector.)
3. Perform algebraic decoding for all vectors $\tilde{\mathbf{r}} = \mathbf{r} - \tilde{\mathbf{z}}_j$, $\tilde{\mathbf{z}}_j \in T$.
4. For each successful decoding, compute the codeword likelihood, or metric, and decide in favor of the best.

The number of test patterns is ostensibly $q^{\lfloor d_{\min}/2 \rfloor}$, although some preprocessing can eliminate test vectors that would produce the same error pattern upon bounded-distance decoding. Thus, such algorithms remain only of interest for small alphabet sizes and, more importantly, small minimum distance. Chase formulated the algorithm for the binary case; it seems logical that rather than trying all q choices in the low-reliability positions, most of the q choices could be deemed poor at the outset, thereby reducing complexity back toward $2^{\lfloor d_{\min}/2 \rfloor}$.

Still another related notion is that of ***generalized minimum distance decoding (GMD)*** introduced by Forney [50] to attempt to approach the performance of maximum likelihood decoding. The demodulator is assumed to provide reliability measures attached to each code symbol (hard) decision, reflecting the confidence attached to a particular decision. The n decisions are rank ordered, and decoding first proceeds with zero erasures, that is, errors-only decoding. This may or may not produce a decoding success, and, if it does, the codeword produced may be incorrect. Next we erase *two* least confident symbols and perform errors-and-erasures decoding. Again, this may or may not succeed and may produce another choice for the proper codeword. We proceed until $\delta - 1$ least-reliable symbols have been erased, which for RS codes ensures a successful decoding. There are thus $\lfloor (\delta - 1)/2 \rfloor$ decoding passes. Among the possibly several candidate code vectors produced, final arbitration is based on the overall likelihood metric.

5.6 MODIFYING BLOCK CODES

We have by now encountered a variety of block codes that, as defined, have restricted values of n, k, and q. For example, the primitive binary BCH codes have block lengths $n = 2^m - 1$ and special values for the message length k. Reed–Solomon codes have block lengths related to the alphabet size q. Frequently, these natural code parameters are not convenient for implementation, or operational requirements may dictate certain other parameters, such as codeword lengths, n, being a power of 2. Therefore, we are faced with modifying a base code, and we consider several means of doing so in this section. These constructions, while not necessarily producing optimal codes for a desired (n, k) pair, are generally quite good and preserve the ease of implementation of the parent code. We shall discuss code modification in the context of the binary cyclic (7, 4) Hamming code and derivatives, as shown in Figure 5.6.1, but the same ideas pertain to general q-ary codes as well.

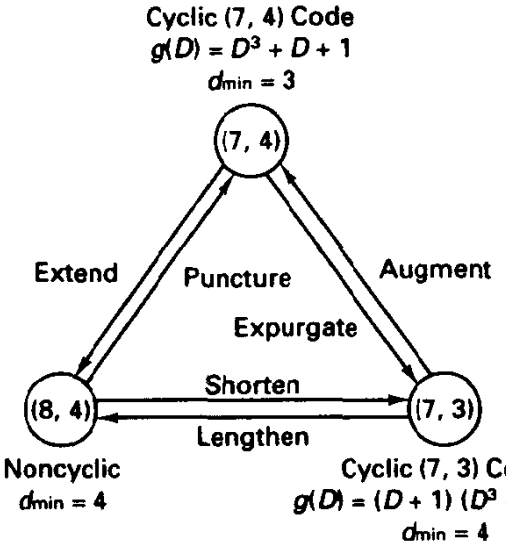

Figure 5.6.1 Basic code modifications surrounding (7, 4) cyclic Hamming code.

5.6.1 Extending and Puncturing

Both of these operations on a base code retain the same code size, but modify the block length. ***Extending*** is a process of adding more code symbols, presumably to increase the minimum distance, while ***puncturing*** is a process of deleting code symbols. As an object of study, let's again consider the (7, 4) cyclic Hamming code, generated by $g(D) = D^3+D^1+1$. Recall that this code is a perfect code, with $d_{min} = 3$. We may singly extend this code to an (8, 4) code by appending an extra bit that makes the overall codeword parity even. In Figure 5.6.2 we show the systematic generator matrix for this code, as well as the parity check matrix, derived from the generator matrix for the (7, 4) code and the additional requirement that x_7 must equal the sum of x_0 through x_6. In adding this extra bit, we increase the minimum distance to 4, since all former odd-weight codewords have been increased in weight by one unit. More generally, this single extension

$$\mathbf{G}_{Hamming} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$\mathbf{G}_{ext\ Hamming} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Appended Bit x_7 Satisfies $x_7 = \sum_{i=0}^{6} x_i = u_0 + u_2 + u_3$

Figure 5.6.2 Generator matrices for standard and singly extended Hamming (7, 4) code.

increases $d_{\min}$ for any code that has odd $d_{\min}$. A second example would be the extension of the Golay (23, 12) perfect code to a (24, 12) code with $d_{\min} = 8$. Extended cyclic codes are no longer cyclic, but encoders and decoders for singly extended codes are simple modifications of those for the base code.

The singly extended Hamming codes are sometimes referred to as ***SEC–DED***, an acronym for single error correcting, double error detecting. To see why these codes are capable of such, consider the parity check matrix for the (8, 4) code. Its columns are all of odd weight; consequently, all single-error patterns produce odd-weight syndromes, while any two-error pattern gives a nonzero, even-weight syndrome. In fact, *all* even-weight error patterns produce even-weight syndromes and are therefore detectable. In geometric terms, the minimum distance between codewords is 4, so radius-1 spheres about each codeword are disjoint, and any two-error pattern lies outside a radius-1 sphere and is thus detectable. The extended Hamming codes are all quasi-perfect, since all possible received n-tuples are within $t + 1$ Hamming units of a codeword, where $t = 1$ here. In the case of the (8, 4) code, the code is equivalent to a biorthogonal 16-ary signal set discussed in Chapter 3 (see Exercise 5.6.1) and to a first-order Reed–Muller code.

Puncturing is the opposite process of extending; code symbols are merely deleted. Puncturing symbols eventually reduces $d_{\min}$, but the choice of punctured coordinate affects on the decrease in distance as block length is shortened. Because relatively little control is maintained on distance, puncturing is rather uncommon in block coding practice. It is, however, important in trellis coding.

5.6.2 Expurgation and Augmentation

This pair of operations either increases or decreases the code size, while maintaining the original codeword length. ***Expurgation*** of a code over GF(q) eliminates a fraction $(q - 1)/q$ of the codewords in a parent code for every expurgation step, reducing the dimension of the linear space. Thus, from an (n, k) code we arrive at an $(n, k - 1)$ code. In our working example, the (7, 4) code becomes a (7, 3) code by eliminating any row of the **G** matrix. It happens that this corresponds to purging the Hamming code of its eight weight 3 codewords. This construction works in all cases with odd $d_{\min}$ codes. For example, the (23, 12) Golay code could be expurgated to achieve a (23, 11) code with minimum distance 8.

If the parent code is cyclic and generated by $g(D)$, multiplying $g(D)$ by any other factor of $D^n - 1$ produces an expurgated cyclic code. Since $D - 1$ is always a factor of $D^n - 1$, we can always expurgate a code whose $g(D)$ does not already incorporate $D - 1$ as a factor and retain a cyclic code. Thus, in the example, the generator for the (7, 3) code is $(D + 1)(D^3 + D + 1)$.[24] It is also interesting to observe that the (7, 3) code is the dual of the (7, 4) code, and that the (7, 3) codewords, when antipodally modulated, form an 8-ary simplex in seven-dimensional space. (Simplex signal sets were discussed earlier in Section 3.4).

The reverse of expurgation is ***augmentation***, where in essence we glue codes together to form codes of larger size having the same block length. Thus, two (7, 3)

[24] Recall that binary polynomials $D + 1$ and $D - 1$ are equivalent.

codes, one differing from the other by the addition of the all-1's vector to each codeword in the first code, form the (7, 4) code. The proof of the Gilbert lower bound on $d_{\min}$ indicates how, in principle, augmentation can be applied to low-rate codes to obtain codes meeting the Gilbert bound; however, the resultant codes, while linear, do not necessarily carry the structure needed to implement "good" long block codes.

5.6.3 Lengthening and Shortening

In ***lengthening*** a code, we increase both the size and block length. For example, the (7, 3) code lengthens to (8, 4), both codes having $d_{\min} = 4$. Note that in doing so the code rate increases, and in this case at least, $d_{\min}$ remains constant. After several steps of lengthening, we must expect the minimum distance to decrease. For example, there is no linear (9, 5) code with minimum distance of 4.

As earlier noted, Wolf [34] has shown that up to two information symbols may be added to a Reed–Solomon code over GF(q) with block length $n = q - 1$ without decrease in $d_{\min}$. Thus, a (31, 27) code over GF(32) with $\delta = 5$ could be lengthened to (32, 28), while still retaining double-error-correcting ability. The latter code parameters might be more convenient for implementation, since the code rate is $\frac{7}{8}$, and codewords are exactly four bytes long. To accomplish such lengthening, we need to append additional columns to the parity check matrix $\mathbf{H}$, without changing the property that any combination of $\delta - 1$ columns is linearly independent. Such a pair of columns is $(1, 0, 0, \ldots, 0)^T$ and $(0, 0, \ldots, 1)^T$.

Finally, and probably of most practical interest, is ***shortening*** an (n, k) code to $(n - j, k - j)$ by simply forcing j leading message positions to be zero and then deleting these message positions from a systematic-form codeword. The code amounts to a lower-order cross section (the coordinates remaining after omitting the leading 0 positions) of a subcode of the original code. Therefore, a shortened code has $d_{\min}$ at least as large as the parent code's $d_{\min}$, and with enough shortening, the distance will eventually increase. Shortening of systematic cyclic codes is particularly simple to accomplish; we utilize the parent encoder and decoder, but force the leading j information symbols to be zero and do not include these preordained zeros as part of the transmission process.

Example 5.28 Obtaining a $k = 32$ Binary SEC–DED Code

A single-error correcting, double-error detecting code for 32-bit computer memory error control is obtained by using the (63, 57) Hamming code, generated by the primitive polynomial $g(D) = D^6 + D + 1$ as a base code and then expurgating the odd-weight codewords to form a (63, 56) code with $d_{\min} = 4$. The resulting generator polynomial is $g(D) = (D + 1)(D^6 + D + 1)$. Next, we shorten this code to (39, 32) by forcing 24 of the 56 information bits to zero. Feedback shift register implementation of encoder and decoder is possible, but with high-speed application in mind, direct matrix encoding and syndrome forming is used, followed by logical determination of the error location or the detection of a double error. Similar schemes have been routinely employed in most modern mainframe computers for memory error protection and are supported by LSI integrated circuit encoder/decoders, which may be cascaded to achieve desirable word sizes (16, 32, 64, and so on). One interesting aspect of error control

in memory systems is that many memory systems are implemented using 4 Meg × 1 memory chips (or whatever size is in vogue), paralleling as many as needed to achieve the required word size. The SEC-DED scheme described can provide continuous operation of the memory despite the complete failure of one chip or its removal. In effect, the memory organization has provided a natural interleaving so that errors common to a chip effect at most one bit of a codeword. Shortened RS codes have also been suggested for memory error control when memory is organized in b-bit wide "nibbles" [31].

Example 5.29 INTELSAT's[25] Modified BCH Code for TDMA Transmission

INTELSAT, in a high-speed TDMA satellite network, uses a (128, 112) binary code derived from a length-127 BCH code. The ratio 112/128 is precisely 7/8, making clock generation easier, and both n and k are multiples of 8, the length of a byte. The double-error-correcting BCH code of length 127 has $d_{\min} = 5$ with 113 information symbols, as discussed in Section 5.4. By expurgating the odd-weight codewords, we achieve $d_{min} = 6$, and by extending this code, we arrive at the (128, 112) design. Actually, this last step is of marginal use, for the minimum distance remains at 6. Regarding decoding, the syndrome former is a 16-bit feedback shift register. Given the memory technology available today, table lookup of the error location in a 64K read-only memory is an attractive alternative to the algebraic decoding approach. In this way, some triple-error patterns are correctable, if desired. Alternatively, the decoder could be operated in mode 3, correcting up to two errors and detecting triple-error patterns.

Example 5.30 Modifying the (23, 12) Golay Code

The Golay (23, 12) cyclic code can be modified in all the ways depicted in Figure 5.6.1. The code can be extended to (24, 12), making the rate exactly 1/2 and increasing the distance to 8. Or we may expurgate the odd-weight codewords by multiplying the former generator polynomial by $D + 1$, obtaining a (23, 11) cyclic code with $d_{\min} = 8$. The weight spectra of these three codes are listed in Figure 5.6.3.

w	A_w
0	1
7	253
8	506
11	1288
12	1288
15	506
16	253
23	1

(23, 12) Code

w	A_w
0	1
8	759
12	2576
16	759
24	1

(24, 12) Code

w	A_w
0	1
8	506
12	1288
16	253

(23, 11) Code

Figure 5.6.3 Weight spectra for (23, 12) Golay code, extended code, and expurgated code.

[25]INTELSAT is a consortium managing international commercial satellite traffic.

5.7 ERROR DETECTION WITH CYCLIC CODES

In many data transmission applications we are interested in the ***reliable detection*** of transmission errors, and when such errors are detected, message retransmission is requested. Such schemes are used in writing and reading of floppy disks, in checking the validity of commands sent to spacecraft (where invalid messages could be disastrous), and in checking the validity of packets in packetized data communication networks. In the packet network application, the term ***ARQ*** is often applied to denote the automatic request of retransmission upon detection of message errors. Often, this error detection takes place after good error-correction coding has taken place, to detect the presence of residual error. In that case, the error-detection scheme is concatenated with another (usually more powerful) coding technique.

The popularity of error detection stems from its simple implementation and high reliability for surprisingly little message overhead, or redundancy. As we have seen, the encoder and syndrome generator are simple shift registers with feedback, and integrated circuits are now available for processing of certain international standard codes. The same circuit can be easily configured to perform either encoding or decoding functions. The codes for such applications are usually referred to as ***cyclic redundancy check*** (CRC) codes in the literature.

In error-detection applications, we are interested only in the probability that an error pattern goes undetected at the decoder, denoted P_{UE}. As earlier noted, this is exactly the probability that the error pattern corresponds to a nonzero code vector. Thus, knowledge of the complete weight spectrum of a code provides, in principle, the tools for analyzing P_{UE}.

For general binary (n, k) codes, Korzhik [51] has shown by ensemble-averaging arguments that binary codes exist whose probability of undetected error on a memoryless binary symmetric channel is bounded by

$$P_{\text{UE}}(\epsilon) \leq 2^{-(n-k)}[1 - (1 - \epsilon)^n]. \tag{5.7.1}$$

where ϵ is the channel error probability. This implies that for *all* ϵ, or no matter how poor the channel,

$$P_{\text{UE}} \leq 2^{-(n-k)}. \tag{5.7.2}$$

For q-ary codes on uniform channels, the corresponding result is $P_{\text{UE}} \leq q^{-(n-k)}$.

This result suggests that even under very poor channel conditions, where error correction might be prohibitively difficult, error detection can be quite reliable, being exponentially dependent on the number of parity symbols appended to the message. On typical channels for which the error probability is small, we expect the P_{UE} performance to be much better than the $q^{-(n-k)}$ bound, governed by $d_{\min}$ and the code's weight spectrum.

This existence proof does not describe the codes, nor does it even hold that easily instrumented cyclic codes behave in this manner. It is rather widely assumed, incorrectly, that (5.7.2) holds for any specific (n, k) code, for any ϵ. It has been shown [52] that Hamming codes, extended Hamming codes, and double-error-correcting BCH codes satisfy (5.7.2), called the $2^{-(n-k)}$ bound, and other good codes seem to reflect this closely, although as we shall see the bound does not strictly hold in all cases of interest, especially under substantial shortening of cyclic codes.

We now analyze the performance of a specific (not necessarily cyclic) code. A q-ary memoryless symmetric channel model with error probability P_s is assumed. If A_w denotes the number of codewords of weight w, then

$$P_{\mathrm{UE}} = \sum_{w=d_{\min}}^{n} A_w \left(\frac{P_s}{q-1}\right)^w (1-P_s)^{(n-w)}. \tag{5.7.3}$$

This is just the probability that a q-ary error pattern is produced by the channel that takes an input codeword to another valid codeword. For small error probability, the probability of undetected error is approximately

$$P_{\mathrm{UE}} \approx A_{d_{\min}} \left(\frac{P_s}{q-1}\right)^{d_{\min}}. \tag{5.7.4}$$

Typically, error detection schemes have low redundancy; that is, k is nearly as large as n. Consequently, it is often convenient to find the weight spectrum of the smaller dual code and the MacWilliams identity to determine the required weight spectrum of the object code. If B_w denotes the number of weight w words in the dual code, then (5.2.21) provides the weight spectrum of the desired code, from which (5.7.3) gives P_{UE}. In the case of binary codes, this reduces to

$$P_{\mathrm{UE}}(\epsilon) = 2^{-(n-k)} \sum_{w=0}^{n} B_w (1-2\epsilon)^w - (1-\epsilon)^n. \tag{5.7.5}$$

For *cyclic codes*, other important error-detection claims can be made. Recall that codewords may be represented (in nonsystematic form) as

$$x(D) = u(D)g(D). \tag{5.7.6}$$

Since $g(D)$ has degree $n-k$, any nonzero code polynomial will have exponents that span at least $n-k+1$ positions. Consequently, no nonzero codeword can have all its nonzero symbols confined to $n-k$ or fewer positions, including end-around counting of positions. Since undetectable error patterns are the same set as the nonzero codewords, we conclude that all error bursts confined to $n-k$ contiguous positions, including end-around bursts, are detectable. Furthermore, among error patterns of length $n-k+1$ bits, the fraction of undetectable events is known to be $q^{-(n-k-1)}/(q-1)$, and for still longer error events, the undetectable fraction is $q^{-(n-k)}$ [1, 3], independent of channel quality. These results also attest to the importance of the number of parity symbols, independent of the message length.

Example 5.31 CCITT Code Error Detection Performance

An international standard adopted by CCITT for binary communication in several protocols uses the 16-bit CRC parity word formed by the generator polynomial

$$\begin{aligned} g(D) &= (D+1)(D^{15}+D^{14}+D^{13}+D^{12}+D^4+D^3+D^2+D^1+1) \\ &= D^{16}+D^{12}+D^5+1. \end{aligned} \tag{5.7.7}$$

where the second polynomial in the first expression is primitive of degree 15 and thus, by itself, generates a (32767, 32752) Hamming code, with $d_{\min}=3$. The adopted $g(D)$ thus would produce an expurgated (32767, 32751) cyclic code with $d_{\min}=4$.

The transmitted code polynomial is

$$x(D) = D^{n-k}u(D) + [D^{n-k}u(D)] \bmod g(D), \tag{5.7.8}$$

so the code is in systematic form with message bits in the leading positions, as described in Section 5.4. The encoder can be implemented with a 16-bit shift register with feedback, and the syndrome former is essentially identical. This and other CRC codes have been implemented in integrated circuit form by several manufacturers.

If we were content with messages of length $k = 32,751$, we could readily evaluate the P_{UE} for this code since it is an expurgated Hamming code, and the weight spectrum of the parent code is given in (5.2.33). Substitution into (5.7.3) for various ϵ gives P_{UE} shown in Figure 5.7.1. Note that for small ϵ error-detection performance on the BSC behaves as $A_4\epsilon^{-4}$, since $d_{min} = 4$ for this code, and quadruple-error patterns are required to deceive the decoder. Note also for high ϵ that P_{UE} still remains below 2^{-16}, the 2^{n-k} bound. For this code we may also say that all error bursts confined to 16 bits are detected, and all but a fraction 2^{-15} of error patterns having length 17 bits are detected, as well as all but a fraction 2^{-16} of longer error patterns. For a random-error channel, these are correctly subsumed in the calculation of (5.7.3). However, if the channel exhibits a bursty tendency, the detection performance remains very robust, and performance is actually better than (5.7.3) for equivalent average channel error probability.

In practice, we usually deal with message lengths other than the natural length of the code, typically much shorter packets of perhaps 1000 bits. A common procedure is to

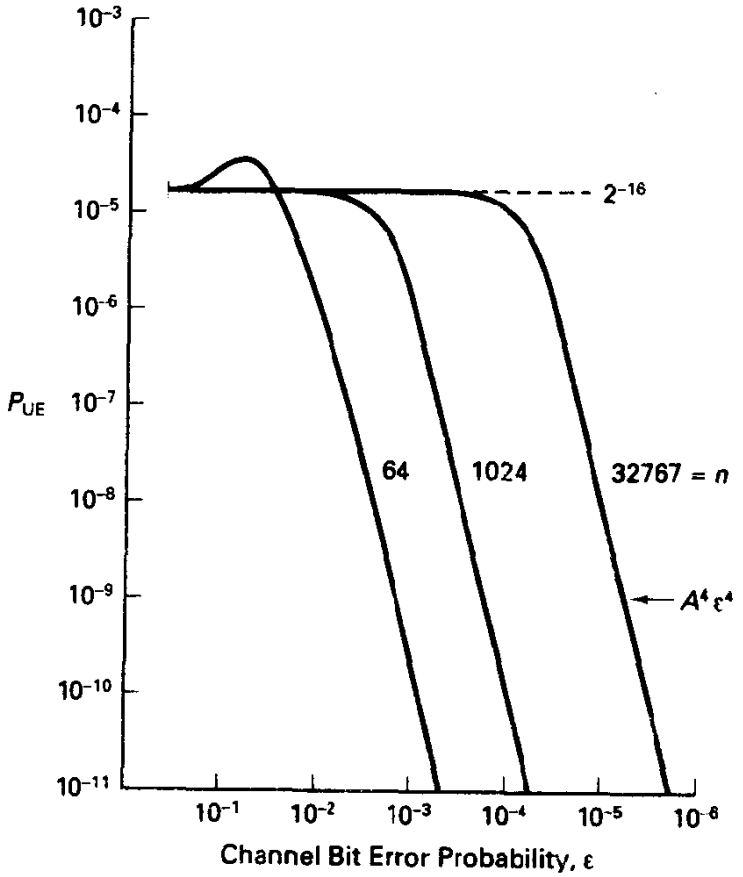

Figure 5.7.1 Probability of undetected error on BSC, block length varying, $g(D) = D^{12} + D^{11} + D^5 + 1$.

shorten the code to $(n-j, k-j)$ by using the original encoder and syndrome former, except forcing the leading j bits to be zero and omitting their transmission. These leading zeros have no effect on the parity vector, nor will they alter the syndrome. Also, as we saw in the previous section, $d_{\min}$ is at least as large for the shortened code as for the parent code, since the shortened code is the truncated subcode formed by codewords having zeros in the leading j positions. Even though the shortened code is not in general cyclic, we can still claim detection of all bursts confined to $n-k$ bits. On the other hand, it is somewhat surprising that, upon shortening, the $2^{-(n-k)}$ bound may be slightly exceeded for large ϵ on a BSC. For example, when the preceding CCITT code is shortened to have $n = 1024$ (roughly a 1000-bit message), calculations performed in [53] give the results shown in Figure 5.7.1. Note that for small ϵ the slope remains the same as for the parent code, and in this region P_{UE} is better than for the parent code, as we would expect. The latter is attributable to the smaller numbers appearing in the weight spectrum. For shortening to $n = 1024$, it appears that the 2^{-16} bound holds, but further shortening to $n = 64$, say, will show this bound is exceeded slightly for a small range of ϵ. The reader is referred to references [52–54] for further analysis of this effect, where it is also shown that for very noisy channels some generator polynomials are better than others when shortened versions of the original code are used.

Discussion of a 32-bit CRC code employed in the Ethernet protocol is provided in Exercise 5.7.2. The ATM cell header code is treated in Exercise 5.7.3.

5.8 LAYERED CODES: PRODUCT CODES AND CONCATENATED CODES

In this section, we present techniques for combining two or more block codes to produce a more powerful error control technique. In essence, both methods seek the power of long block codes, without the complexity of the associated decoding, by employing shorter component codes.

5.8.1 Product Codes

We will describe product codes in a two-dimensional context, although the extension to higher levels of coding should become obvious. Given a $k = k_1 k_2$ symbol message for transmission, we imagine storing the sequence in an array with k_2 rows and k_1 columns. Suppose that we encode each row using an (n_1, k_1) block code, employing any one of a number of techniques already presented. Let the minimum Hamming distance of the row code be d_1. In the process, we have filled an array of k_2 rows and n_1 columns. To each of these columns, we next apply an (n_2, k_2) block code (over the same field), having $d_{\min} = d_2$. This populates a two-dimensional array of size $n_1 n_2$, and we have produced a two-dimensional code with parity constraints on rows and columns. (The order in which the encodings are actually performed is unimportant, and it is possible that the same code is used in both dimensions.)

The entire array can be viewed as having a message section, a section of row parity symbols, or row checks, a section of column checks, and a set of checks on checks, as

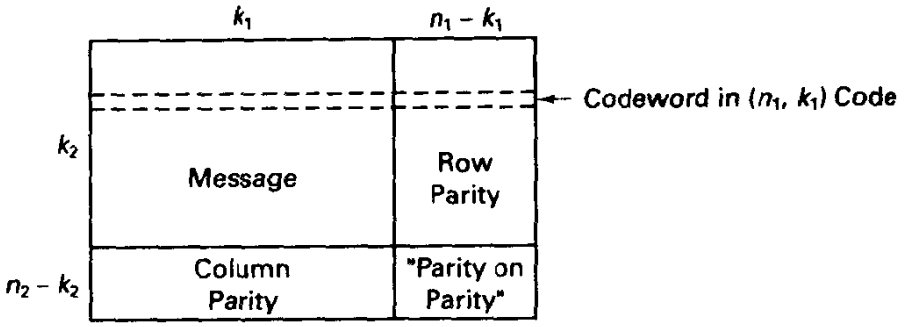

Figure 5.8.1 Product code layout.

illustrated in Figure 5.8.1. It is clear that we could view the entire code as an (n_1n_2, k_1k_2) block code, whose aggregate rate is

$$R = \frac{k_1k_2}{n_1n_2} = R_1R_2. \tag{5.8.1}$$

However (and this is the important property), encoding and decoding retain the algorithms and complexity of the component codes.

Specifically, decoding could take place in the obvious manner. First, each row is decoded, with the corresponding information positions repaired as indicated. Of course, incorrect row decoding is possible, in which case residual errors are left. Next, columns are decoded separately. This two-step decoder has decoding effort, or operation count, given approximately by the sum of the individual decoder complexities.

The minimum distance of the code can be shown (Exercise 5.8.1) to be the product of the minimum distances of the row and column codes; that is,

$$d_{\min} = d_1d_2. \tag{5.8.2}$$

This would, as for standard codes, imply that the guaranteed error-correction capability of a maximum likelihood decoder is

$$t = \left\lfloor \frac{d_1d_2 - 1}{2} \right\rfloor. \tag{5.8.3}$$

Unfortunately, this two-step decoding procedure, row decoding followed by column decoding, is in general incapable of correcting all error patterns with t or fewer errors in the array. Nonetheless, this idea is rather simple, and we will see that many error patterns with more than t errors are correctable with this method.

We can easily determine the smallest number of errors that prohibits correct decoding in row/column decoding. Let t_1 and t_2 be the guaranteed error-correcting capability of the row and column codes, respectively. For it to be *possible* for an array to fail, we must have a certain number of row failures. Row failures may happen when $t_1 + 1$ errors occur in any row. For the column decodings to fail, it must be true that $t_2 + 1$ row failures occurred and, even then, these errors must be placed rather specially. Thus, we claim that any error pattern with $(t_1 + 1)(t_2 + 1) - 1$ errors or less is correctable. Of course, this is smaller than t in (5.8.3) (roughly half as large for large distance codes), but we should also observe that most error patterns with more than the guaranteed limit are in fact correctable.

Example 5.32 Product Code Involving Binary Hamming and BCH Codes

Suppose that the row code is a (15, 11) binary code with $d_1 = 3$, and the column code is a (63, 51) binary BCH code with $d_2 = 5$. The overall rate is roughly $R = 0.6$, and the block length is $n = n_1 n_2 = 945$. The minimum distance of the product code is then 15, but a row/column decoder can only guarantee correction of $2 \cdot 3 - 1 = 5$ errors in the array, rather than the 7 ensured by minimum-distance considerations. In Figure 5.8.2 we illustrate a 6-error pattern that is uncorrectable in the row/column decoding order (notice the special requirements on error placement) and an 11-error pattern that is correctable. Product codes are apparently powerful because of the high likelihood of correcting beyond the guaranteed limit with simple decoders.

As a design comparison, we might consider other nonproduct codes with approximately the same block length and rate. A BCH code with $n = 1023$ and $k = 618$, hence rate $R = 0.6$, can always correct 44 errors, a far better guarantee than for the product code. Decoding effort for this code is certainly much larger than that for either component code given, although total decoding effort must reflect the need for multiple decodings in rows and columns in the product code situation.

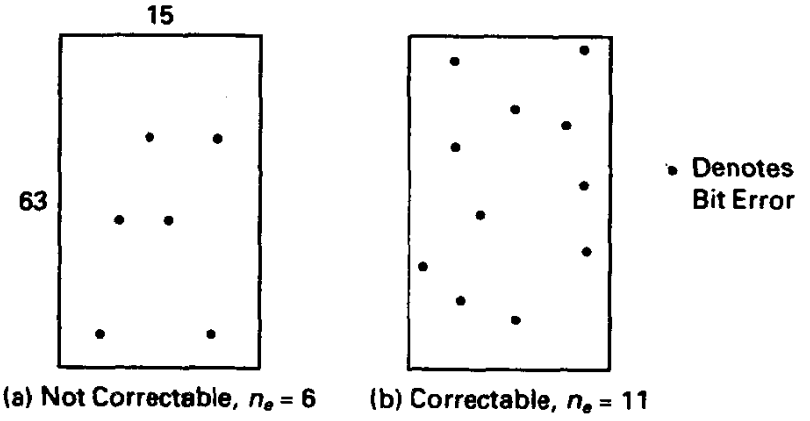

Figure 5.8.2 Uncorrectable and correctable error patterns for product code of Example 5.32.

A particularly simple application of the product code concept is that involving single symbol parity encoding in each dimension, giving each code a minimum distance of 2, and the complete code a minimum distance of 4. Neither row nor column code is capable by itself of correcting any errors, but their combination admits a very simple scheme for correcting one error or detecting three errors. Specifically, any single error is located at the intersection of the row and column where parity checks fail. At the same time, any double-error pattern is detectable, so the system is SEC–DED. In this case, we can attain the error-correcting capability suggested by (5.8.3) with a simple decoder.

Blahut [4] describes an intricate procedure whereby the first decoder declares erasure of the entire word when decoding fails and passes suspected error counts to the second decoder otherwise. By judiciously processing this side information, the second decoder is able to ensure correction of all error patterns with up to t errors, where t is given by (5.8.3). One simpler alternative, which under certain approximations achieves the same result, is to perform correction of up to $r_1 < t_1$ errors on rows, declaring row erasures when row decoding fails. (Usually, when the error limit r_1 is exceeded, the

decoding is not to an incorrect codeword, but simply a failure, so we can assume that residual errors apart from erasures are negligible.) The column decoder can accommodate up to $d_2 - 1$ erasures in a column, assuming that no residual errors are left after row decoding. Thus, we can guarantee repair of error patterns with up to $(d_2)(r_1 + 1) - 1$ errors. It should be clear that reversal of the order of decoding would guarantee (approximately) $(d_1)(r_2 + 1) - 1$ error correction, and it is wise to choose the best order.

Example 5.32 *continued*

Since the Hamming code is a perfect code with $t_1 = 1$, and thus row decoding failure normally would not occur, we could choose $r_1 = 0$; that is, the row code will be used only for error detection. At least three errors in a row are required to cause incorrect decoding, but single or double errors will cause the row to be erased. Column decoding with erasure filling will guarantee correction of $d_2(r_1 + 1) - 1 = 4$ errors. (Five errors in any one column will cause product code failure here.)

Another approach is to perform column decoding first, adopting $r_2 = 1$ as the error-correcting radius, instead of the guaranteed limit of two-error correction. Accordingly, four errors in a column are required to cause incorrect decoding, and we may assume that whenever two or more errors in a column occur then a column erasure is declared. The row decoder is capable of handling $d_1 - 1$ erasures. Consequently, the guaranteed correction capability with this option is $d_1(r_2 + 1) - 1 = 5$. In this case, the result is no stronger than the simple row/column decoding discussed previously; however, for more powerful component codes, this method begins to excel and approach the possibilities given by (5.8.3).

Product codes have natural burst-error handling capability as well. To see how, suppose we form the complete array as in Figure 5.8.1 and transmit *by columns*. Decoding proceeds first on rows after the full array is received and then by columns, as described previously. Notice that any single error burst confined to $n_2 t_1$ transmission slots will not place more than t_1 errors in any row of the received array and is thus correctable. Thus, $n_2 t_1$ is the guaranteed burst-correcting capability. Obviously, interchange of orders could achieve $n_1 t_2$, and if burst correction is of principal importance, the better choice should be adopted. On this topic, the reader is directed to [55] for a related discussion on burst-error correction with "array" codes.

As a prelude to the next topic, we show in Figure 5.8.3 a schematic illustration of product encoding, which depicts the layering of one code on top of another.

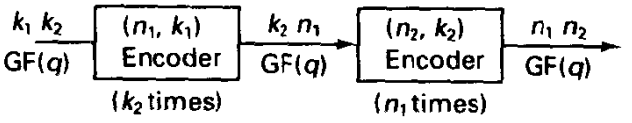

Figure 5.8.3 Layering of row/column codes.

5.8.2 Concatenated Codes

A closely allied concept is that of code ***concatenation***, introduced by Forney [38]. Again, we shall limit our discussion to one level of concatenation. Figure 5.8.4 illustrates a general concatenation approach, and remarkable similarity with Figure 5.8.3 is seen. The primary difference in concatenation is that the two codes normally are defined over different field sizes, for reasons that will become clear shortly.

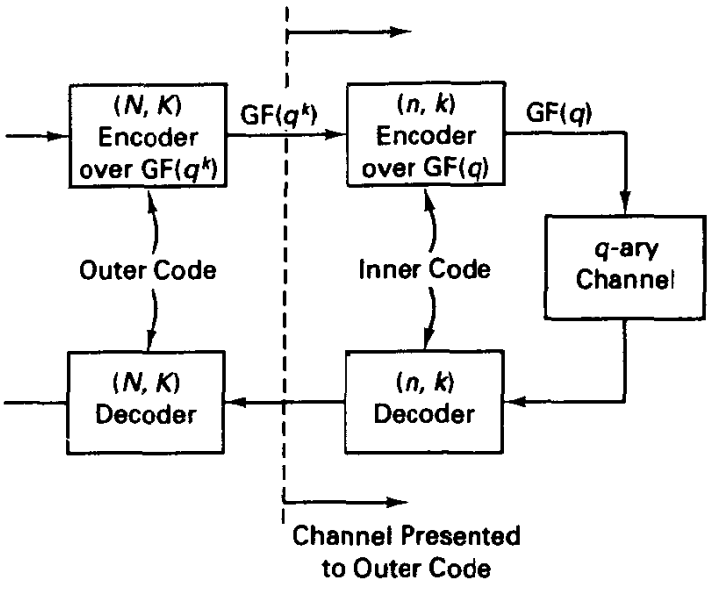

Figure 5.8.4 Diagram of concatenated coding/decoding.

We usually speak of an ***outer code*** and an ***inner code***. The outer code is taken to be an (N, K) code over GF(q^k), and each output symbol in the outer codeword is represented as a k-tuple over GF(q). These k symbols are then further encoded by an (n, k) code over GF(q) whose symbols are compatible with the modulator and channel. Typically, the inner code is binary ($q = 2$), and the outer code is a Reed–Solomon code over GF(2^k). It is helpful to visualize the inner encoder/channel/decoder sequence as providing a superchannel on the field of size q^k for use by the outer coding system. Also, the similarity with product codes is made stronger if we treat the outer code as a (single) column code, with the row code playing the role of the inner code.

Example 5.33 (16, 12) R–S Code Concatenated with (8, 4) Binary Code

Suppose we adopt as an outer code the lengthened (16, 12) Reed–Solomon code over GF(16), whose minimum distance is $n - k + 1 = 5$. Symbols in the RS codeword are represented as binary 4-tuples, and the inner code can be selected as an extended Hamming (8, 4) binary code. These binary code symbols are then transmitted using a binary modulation technique. Recall that the inner code minimum distance is 4. The overall encoding rate is $R = R_{\text{outer}} R_{\text{inner}} = \left(\frac{3}{4}\right)\left(\frac{1}{2}\right) = 0.375$ bit/channel use.

Decoding of concatenated codes proceeds inside-out; that is, each inner codeword is separately decoded, and each k-tuple estimate is then regarded as a symbol in the field of the outer code. It is worth noting that when the inner decoder errs it matters not at all whether one of the k symbols or all of them are incorrect, for the outer code is symbol oriented. In fact, the superchannel may be regarded as a bursty channel for which Reed–Solomon and other codes over bigger fields are well suited.

Performance analysis of a given scheme involves iterating previous calculations for block codes. Let P_{in} denote the probability of an inner code decoding error, which will depend on the code employed, the channel, demodulator quantization, and SNR.

The probability of outer code decoding failure is (for a bounded-distance decoder)

$$(1 - P_{\text{CD}})_{\text{out}} = \sum_{i=t_{\text{out}}+1}^{N} C_i^N P_{\text{in}}^i (1 - P_{\text{in}})^{N-i}. \tag{5.8.4}$$

Just as for product codes, concatenation schemes can be viewed as a recipe for building long codes for the desired channel alphabet, while preserving the complexity of the composite decoders. Error probability curves can be extremely steep, and concatenation is generally regarded as an efficient way to achieve extremely small error probabilities without resort to error detection and retransmission protocols.

As a matter of theoretical interest, concatenation was used by Justesen [56] to produce the first constructive procedure for producing codes whose normalized minimum distance remains bounded away from zero as block length increases indefinitely. The normalized distance in this construction is still, however, below the Varshamov–Gilbert asymptotic bound. The construction involves a special allocation of redundancy, or code rate, in a multilevel concatenation scheme. Even in simple examples, like the one just given, it is clear that the overall performance is sensitive to the allocation of rate between inner and outer codes.

Typical practice has been that the inner code is designed to be a powerful, lower-rate code, while the outer code is a high-rate code, normally a Reed–Solomon code. It is important that the inner decoder produce the very best superchannel possible in the sense of channel capacity or R_0; when feasible, this implies maximum likelihood decoding of the inner code. In the previous example, if the (8, 4) code is used on an AWGN channel, correlation decoding should be employed if analog demodulator outputs are available.

5.9 INTERLEAVING FOR CHANNELS WITH MEMORY

In many practical situations, the mapping from encoder sequences to demodulator output sequences is not a memoryless relation; that is, if we let $\tilde{\mathbf{r}} = (\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{n-1})$ denote the collection of received observations,

$$P(\tilde{\mathbf{r}}|\mathbf{x}) \neq \prod_{j=0}^{n-1} P(\mathbf{r}_j|x_j). \tag{5.9.1}$$

We say such channels exhibit memory, in that the action of the channel is not a sequence of independent trials. Physical mechanisms for this memory, or dependence between channel actions from symbol to symbol, are quite varied and may be due to amplitude fading on a radio link, sporadic burst noise due to inadvertent interference or hostile jamming, magnetic or optical disc recordings with surface defects, and intersymbol interference effects due to channel time dispersion. References [57–59] provide good surveys of channels with memory and applications of coding to them.

The codes we have been studying are essentially designed for memoryless channels. When used on channels with memory, these codes tend to be overwhelmed by the rare periods of poor channel conditions and do not perform as we might predict based on "average" conditions and memoryless channel analysis. A hypothetical illus-

tration is the following: we employ the (23, 12) binary Golay code on a binary channel whose error mechanism is admittedly atypical. A code block is either error free or there are exactly four bit errors present. The probability that a block is corrupted is 10^{-3}, so the average or marginal error probability of this channel is $\left(\frac{4}{23}\right)(0.001)$. Because the Golay code is perfect, a complete decoder will always decode incorrectly on corrupted blocks and, in fact, will induce additional errors in the codeword. Error-free blocks are processed correctly. Nonetheless, the postdecoding error probability is larger than it would have been without coding, rendering this coding technique a poor choice.

A more typical illustration of memory effects is provided by the Gilbert [60] model for a binary channel. The channel is assumed to have two underlying states, good = G and bad = B. In the good state, the channel crossover probability is, say, $\epsilon_G = 10^{-5}$, while in the bad state, the channel parameter is $\epsilon_B = 10^{-1}$. Furthermore, there is a stochastic model for transitioning between states at every channel time, as shown in Figure 5.9.1. This represents a two-state ergodic Markov chain whose steady-state probabilities can be obtained as in Chapter 2. It is easy to show that the steady-state probability of being in the good state is $\frac{5}{6}$ for this example, and furthermore the long-term probability of error is $1.67 \cdot 10^{-2}$. Also, the expected time of persistence in state B is $1/0.05 = 20$ bits. If a code is used on this channel with a block length of, say, $n = 31$ bits, the performance will be much different than predicted by a memoryless error channel analysis with $\epsilon = 0.0167$. To see why, we show in Figure 5.9.2 a cumulative distribution function for the number of errors per block for a memoryless channel with $\epsilon = 0.0167$ and for the actual channel, in qualitative terms. For the memoryless channel the expected number of errors is about 0.5, and a double-error-correcting code would have rather good performance. On the actual channel, however, there is significantly higher probability of two or more errors, due to the persistence in the bad state once it arrives. We may again find the decoded error rate is actually poorer than with no coding! Obviously, we could design a code with better error-correcting power, but it is clear that the design philosophy is not well matched to the problem at hand.

Another relevant example is the Rayleigh fading channel, as perhaps experienced in digital cellular radio communication. If the correlation time of the fading process is longer than the block length of the code, we will encounter the same difficulties as before, even in the best situation where the demodulator presents likelihood information to the decoder and perhaps channel state information as well.

It is clear that a more intelligent code design could do better if the design anticipates the clustering tendency of errors. Such designs have been widely studied under the name ***burst-correcting codes*** [61], but have seen relatively little practical application, with the possible exception of magnetic disk and tape units [62]. This is because of several factors. First, the real channel error model is seldom known with sufficient accuracy to have

$P(\text{error}|G) = 10^{-5}$, $P(\text{error}|B) = 0.1$
$P(G) = 5/6$, $P(B) = 1/6$, $P(\text{error}) = 0.0167$

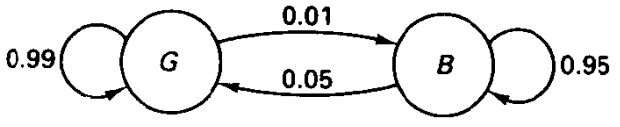

Figure 5.9.1 Gilbert two-state model for binary channel with memory.

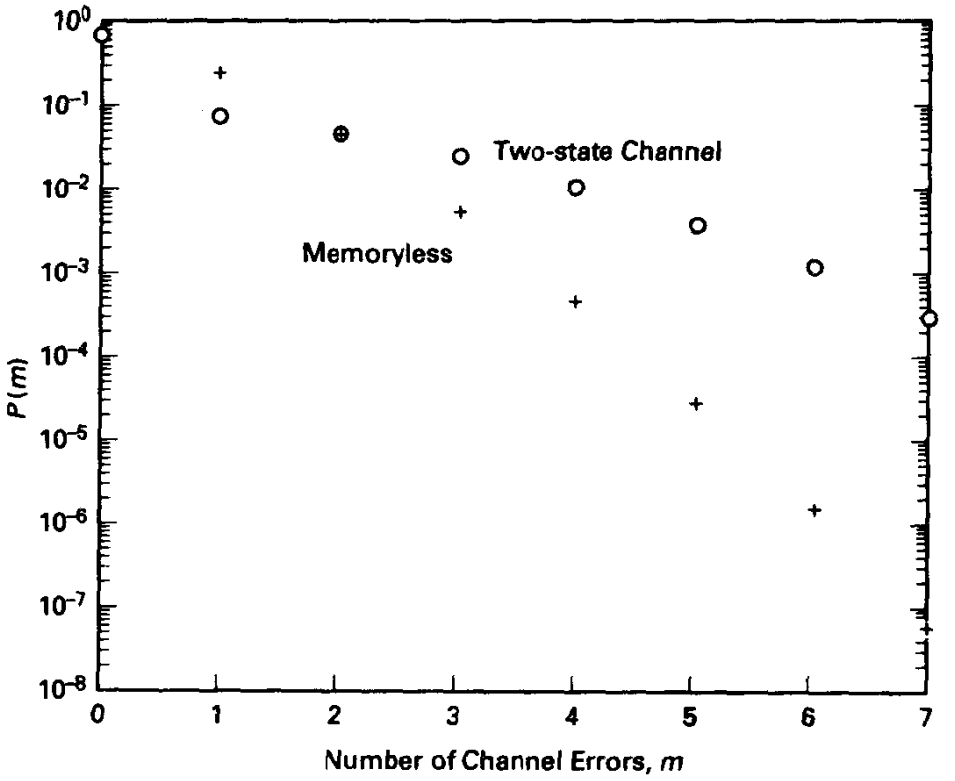

Figure 5.9.2 Block error histograms for memoryless and Gilbert channel model, $n = 31$.

confidence in any special-case design. Second, the burst-correcting approaches do not readily provide for incorporation of soft decisions or channel state information, if available.

A quite different, rather intuitive, approach to coding on these channels is that of ***interleaving***[26] of code symbols [63]. Interleaving is nothing more than a regular permutation of the order of the encoder output sequence prior to channel transmission. ***Deinterleaving*** of the stream produced by the demodulator then unscrambles the order, restoring the original time ordering of code symbols. The intent is to make channel actions on symbols in one decoder block appear to be independent, with the same first-order probability model. In more everyday terms, the interleaving technique attempts to uniformly mix the good and bad intervals, so for the Gilbert model just posed the channel would appear to the decoder as a memoryless channel with $\epsilon = 0.0167$.

Before providing the details, we should note that such scrambling, or destruction of statistical dependencies, is antithetical to the information-theoretic view of the coding problem. Indeed, such scrambling will in general reduce the *apparent* channel capacity relative to that of the actual channel. Wolfowitz [64] confirms our intuition that, for two channels with the same marginal error probability, the one with memory will have larger capacity. (See Exercise 5.9.1 associated with the Gilbert model.) However, exploitation of the channel memory in an intelligent way requires possibly long block length codes

[26] Interleaving is sometimes referred to as interlacing.

with complicated decoders employing accurate channel state information. (An exception is the design of burst-correcting codes for the dense burst channel mentioned previously [61].) Most importantly, interleaving represents a universal technique applicable to various codes and channel behaviors and is simple to instrument.

5.9.1 Block Interleaving

Suppose, that we encode as usual with an (n, k) block encoder, and we write successive codewords $\mathbf{x}^i$ into a data array that is n symbols wide, as depicted in Figure 5.9.3. We proceed to write D such codewords in vertical sequence, filling a rectangular array of nD symbols. The sequence for transmission, however, is taken "by column"; that is, the entire first column is transmitted, followed by the second, and so on, until the array is exhausted. These transmissions will be acted on by some channel with memory. The demodulator outputs, whether a vector of real numbers or a hard-decision symbol from the code alphabet, will be written into another array by columns, in transmitted order, until the array is full. Conceivably, the demodulator supplies channel state information as well that could be employed in decoding, and, if so, this is written in similar order. Decoding then commences in row-by-row fashion.

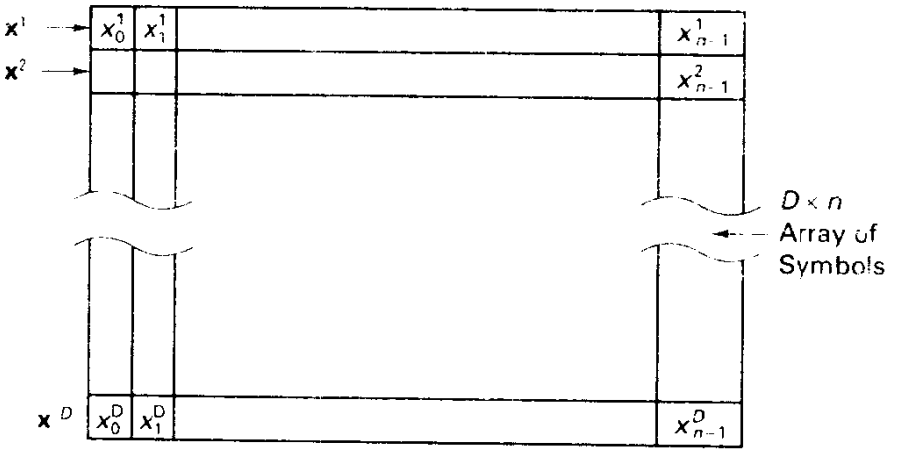

Figure 5.9.3 Arrangement of symbols in block interleaver. Codewords $\mathbf{x}_i$ are loaded by rows and then transmitted by columns.

It is clear upon deinterleaving at the receiver that the proper channel outputs corresponding to the same codeword have been grouped together in proper order. However, the channel time indexes affecting a given decoding cycle are separated by D units (see Figure 5.9.4). For example, the channel time indexes attached to the symbols of the

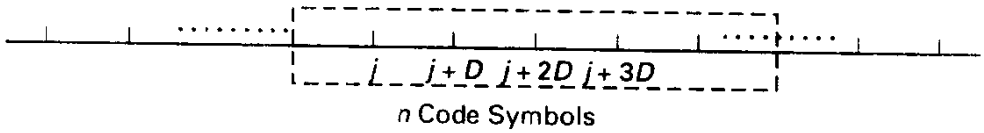

Figure 5.9.4 Channel time indexes as seen at output of deinterleaver.

first codeword, after deinterleaving, are $0, D, 2D, \ldots$. The parameter D is known as the *interleaving depth*. The design goal should now be apparent: if D is sufficiently large relative to the time scale over which the channel evolves through its various modes, then the channel has been converted into an apparently memoryless channel with the same first-order behavior. A code designed for a memoryless channel has much better prospects for success.

If we employ algebraic decoding, the burst-correcting capability for block interleaving is easy to specify. Suppose that the basic block code has error-correcting capability t. Then, with the aid of Figure 5.9.3, it is readily seen that a single error pattern confined to tD contiguous symbols will place no more than t symbol errors in any single row and will thereby be corrected by the decoder operating on each row sequentially. Alternatively, up to t shorter bursts of length D or less can be accommodated, dependent on the alignment of these bursts.

Example 5.34 Block Interleaving for a Tape Recording Application

A video cassette recorder is to be used to archive digital data, recording at a rate of several megabits per second, and storing perhaps 5 gigabytes of data on one video cassette. By measurements, it is determined that the long-term average error probability is less than 10^{-5}, but that errors are found in isolated clusters of 256 bits or less in length (due to media defects, tape-to-head flutter, and timing jitter). A BCH (31, 21) double-error-correcting binary code is to be used for error correction, and we add a block interleaver of width 31 bits and depth 256 words to enhance burst correction. Over a span of 31(256) bits, the decoder is capable of correcting any two error bursts confined to 256 bits appearing anywhere in the array, as shown in Figure 5.9.5. Alternatively, a single 512-bit burst of errors is correctable.

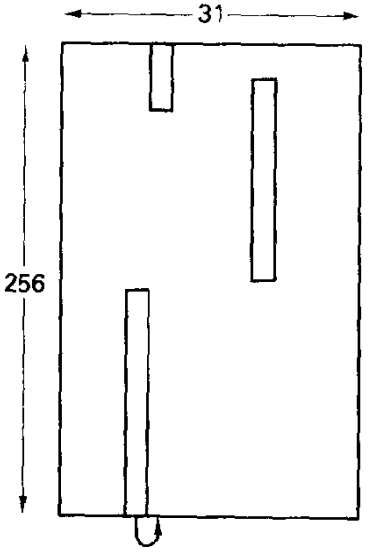

Figure 5.9.5 Deinterleaver array for Example 5.33 showing two correctable error bursts.

The two negative aspects attached to block interleaving are the required storage at the encoder and decoder (nB symbols each) and the end-to-end delay. We may not start transmission until the array fills, and likewise for decoding, so the waiting time, relative to time attached to the generation of the first information symbol, is $2nB$ symbols. (Of course, other burst-error-correcting approaches are required to have a suitably long delay

as well.) In some applications, the end-to-end delay constraint places real limitations on the effectiveness of interleaving. For example, in two-way speech communication, one-way delay of more than 100 ms makes verbal communication awkward. In this case, if the channel fading process has a correlation time on the order of 100 ms, as it may be in a vehicular communication system with slow-moving vehicles, the interleaving capability is very limited.

Decoder synchronization is more difficult when interleaving is employed. Not only are codeword boundaries necessary (as they are in any case at the decoder), but we must know where the first codeword in an array sequence appears or, equivalently, establish interleaver frame synchronization. Sequential testing of hypothesized partitions of the input stream can be used to achieve proper codeword synchronization.

5.9.2 Convolutional Interleaving[27]

A more efficient permutation scheme is performed by the system shown in Figure 5.9.6. The n positions of codewords are delayed by progressively larger amounts at the encoder output prior to transmission. The delay schedule is reversed at the demodulator output, however, bringing the symbols into proper time alignment for decoding. Essentially, we are operating with a triangular memory version of the block interleaver. It may be seen that symbols within an n-symbol codeword, examined at the decoder input, are influenced by channel actions spaced $D - 1$ units of channel time apart. Said another way, we have diffused a given codeword over a time interval of nD time units. With this arrangement, however, the total memory is only nD symbols, and the end-to-end delay is nD channel time units. Furthermore, synchronization of the deinterleaver is somewhat simpler, as there is only an n-fold ambiguity in placing demodulator outputs.

A related technique, called interlacing, is shown in Figure 5.9.7. We imagine D copies of encoders and decoders that are simply time-multiplexed on the channel. Successive inputs to any one decoder are separated by B channel time units, and the end-to-end delay is again nD symbols. It is possible to avoid D copies of hardware (or software) encoders and decoders to implement the scheme. Indeed, one of each suffices if a sufficient amount of temporary storage is available. For cyclic codes, the implementation is especially easy; we merely replace each delay cell in the encoder and decoder by D units of delay (Figure 5.9.7b).

Example 5.35 Interleaving for NASA/ESA Deep-space Coding

The CCSDS (Consultative Committee on Space Data Systems) coding standard for deep-space communication employs some of the most powerful techniques in coding theory to operate with smallest possible power and antenna requirements. Specifically, an outer (255, 223) RS code over GF(256) is symbol-interleaved to depth 8 or 16. This could be either a block or convolutional interleaver since delay issues are not paramount, nor is memory a real problem, but a frame-oriented block interleaver was selected. The 8-bit symbols appearing at the interleaver output are convolutionally encoded with a binary code, and binary PSK modulation of the transmitter carrier is utilized. The decoder uses the Viterbi's algorithm combined with soft-decision decoding (see Chapter 6) to provide a powerful inner

[27] Not to be confused with convolutional codes, to be studied in Chapter 6; convolutional interleavers may be employed either with block codes or convolutional codes.

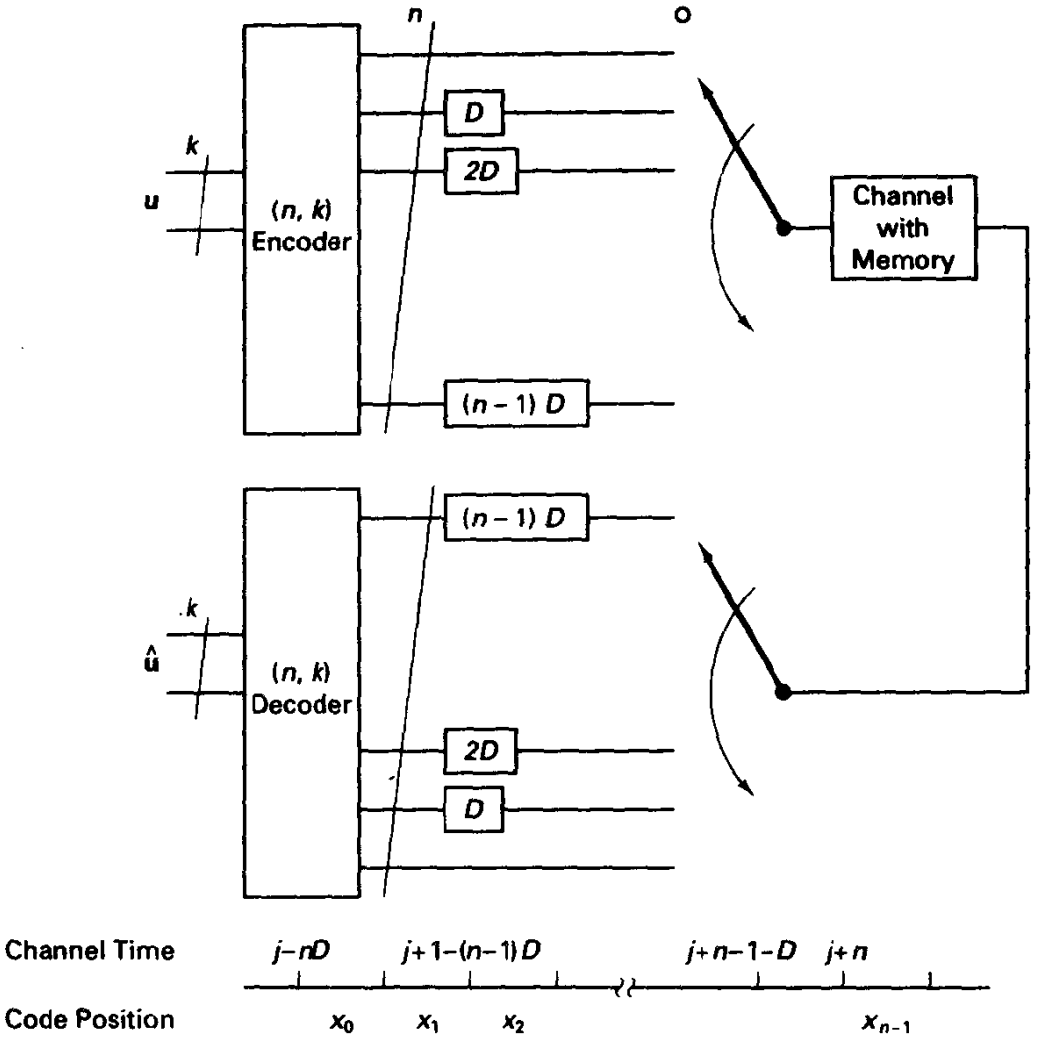

Figure 5.9.6 Convolutional interleaver. Channel time indexes attached to consecutive code symbols are $D + 1$ time units apart. Total delay $= nD$; total memory $= nD$.

channel. Figure 5.9.8 illustrates the block diagram. Typically, the inner channel decoded bit error probability is 10^{-3} for this inner channel when $E_b/N_0 = 3.5$ dB. When the inner decoder does make errors, these are typically confined to perhaps 8-bit spans; however, synchronization jitter and other effects could cause the inner channel error patterns to span several RS code symbols. For this reason, the code is interleaved as described. The outer decoder can correct up to 16 symbol errors in a codeword, which is many inner channel decoding error events. The concatenation and interleaving allow the system to operate acceptably with E_b/N_0 in the range of 2.5 dB. Most of the occasions where a frame is not decoded correctly are simply decoding failures.

Still another interleaving architecture is known as ***helical interleaving*** [65]. The primary advantage is that reliability information (erasures) can be obtained from previous channel symbols for decoding current passages.

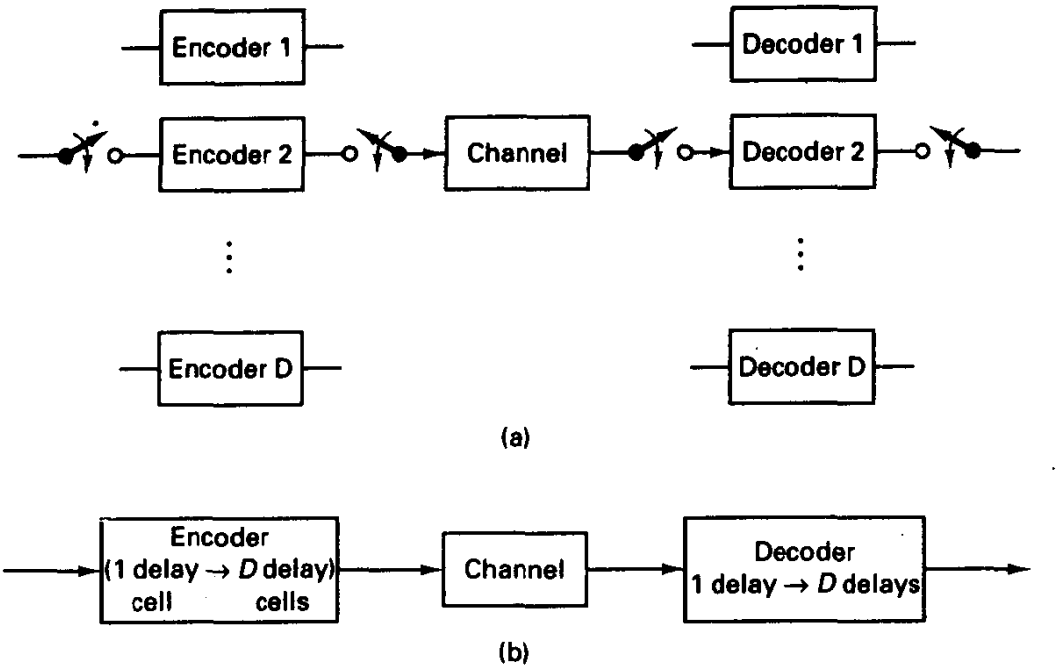

Figure 5.9.7 Interlaced encoder/decoder method of interleaving.

Interleaving is traditionally employed around hard-decision channels, and the philosophy is literally one of burst-error correction. We may just as well apply the method to soft-decision demodulator outputs and, if useful, could carry along channel state information with the demodulator outputs. The only cost is one of additional deinterleaver memory.

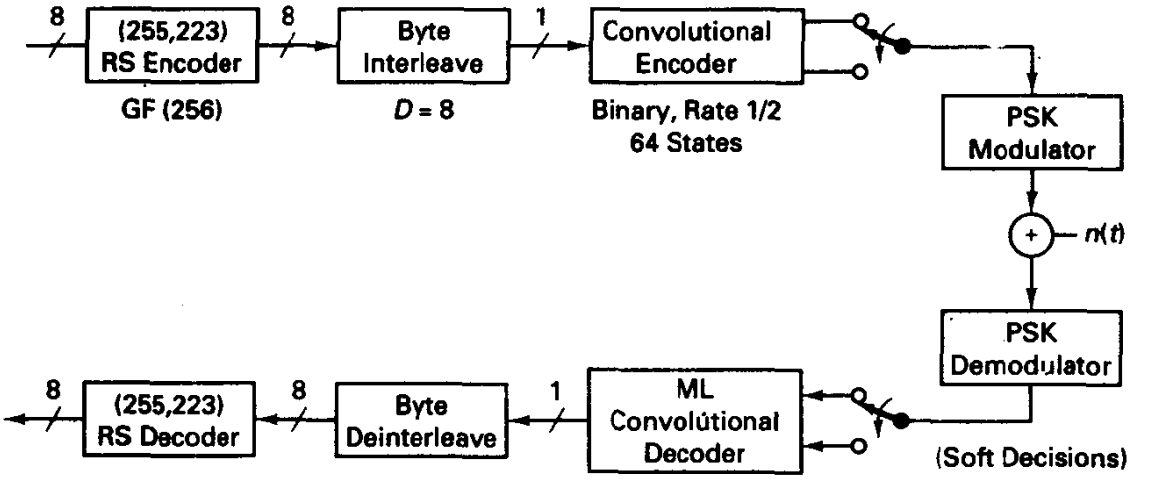

Figure 5.9.8 *Concatenated RS/convolutional coding scheme in NASA/ESA CCSDS standard.*