FROM: R. E. Blahut, Theory and Practice of Error Control Codes, Addison Wesley, 1983.

7.4 SYNTHESIS OF AUTOREGRESSIVE FILTERS

Most of the computations required to decode BCH codes using the algorithm of Section 7.2 centers on the solution of the matrix equation

$$\begin{bmatrix} S_1 & S_2 & S_3 & \cdots & S_v \\ S_2 & S_3 & S_4 & \cdots & S_{v+1} \\ S_3 & S_4 & S_5 & \cdots & S_{v+2} \\ \vdots & & & & \vdots \\ S_v & S_{v+1} & S_{v+2} & \cdots & S_{2v-1} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ \\ \vdots \\ -S_{2v} \end{bmatrix}.$$

For moderate v, the obvious method of solution using matrix inversion is not unreasonable. The number of computations necessary to invert a v by v matrix is proportional to v^3. In many applications, however, one wants to use codes that correct large numbers of errors. In such cases, one wishes for a more efficient method of solution. Berlekamp found such a method. This method relies on the fact that the above matrix equation is not arbitrary in its form; rather, the matrix is highly structured. This structure is used to advantage to obtain the vector Λ by a method that is conceptually more intricate than direct matrix inversion, but computationally much simpler.

We will describe a variation of the algorithm by Massey, who recognized that the best way to derive the algorithm is as a problem in the design of linear-feedback shift registers. Suppose the vector Λ is known. Then the first row of the above matrix equation defines S_{v+1} in terms of $S_1, \ldots, S_v$. The second row defines S_{v+2} in terms of $S_2, \ldots, S_{v+1}$, and so forth. This sequential process is summarized by the equation

$$S_j = -\sum_{i=1}^{v} \Lambda_i S_{j-i} \qquad j = v+1, \ldots, 2v.$$

For fixed Λ, this is the equation of an autoregressive filter. It may be implemented as a linear-feedback shift register with taps given by Λ.

Seen in this way, the problem becomes one of designing the linear-feedback shift register shown in Fig. 7.2 that will generate the known sequence of syndromes. Many such shift registers exist, but we wish to find the smallest linear-feedback shift register with this property. This gives the least-weight error pattern that will explain the received data, that is, $\Lambda(x)$ of smallest degree. The polynomial of smallest degree will have degree v, and there is

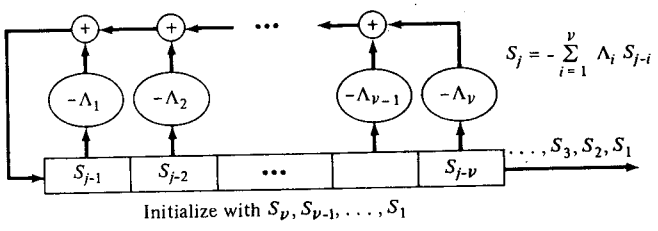

Figure 7.2 Error locator polynomial as a shift-register circuit.

only one of degree v because the v by v matrix of the original problem is invertible.

Any procedure for designing the autoregressive filter is also a method for solving the matrix equation for the Λ vector. We shall develop such a shift-register design procedure. The procedure applies in any field and does not assume any special properties for the sequence $S_1, S_2, \ldots, S_{2t}$. They need not be syndromes, but if they are the syndromes of a correctable error pattern, the procedure always designs a shift register with the rightmost tap nonzero. For an arbitrary linear-feedback shift register with feedback polynomial $\Lambda(x)$, the length of the shift register might be larger than the degree of $\Lambda(x)$, because some rightmost stages might not be tapped.

To design the required shift register, we must determine two quantities: the shift register length L, and the feedback-connection polynomial $\Lambda(x)$:

$$\Lambda(x) = \Lambda_v x^v + \Lambda_{v-1} x^{v-1} + \cdots + \Lambda_1 x + 1,$$

where $\deg \Lambda(x) \leq L$. We denote this pair by $(L, \Lambda(x))$. We must find a feedback shift register of shortest length that will produce the sequence $S_1, \ldots, S_{2t}$ when properly initialized.

The design procedure is inductive. For each r, starting with $r = 1$, we will design a shift register for generating the first r syndromes. Shift register $(L_r, \Lambda^{(r)}(x))$ is a minimum-length shift register for producing $S_1, \ldots, S_r$. This shift register need not be unique. Several choices may exist, but all will have the same length. At the start of iteration r, we will have constructed a list of shift registers

$$(L_1, \Lambda^{(1)}(x)),$$
$$(L_2, \Lambda^{(2)}(x)),$$
$$\vdots$$
$$(L_{r-1}, \Lambda^{(r-1)}(x)).$$

The main trick of the Berlekamp-Massey algorithm is to find a way to compute a new shortest-length shift-register design $(L_r, \Lambda^{(r)}(x))$ that will generate the sequence $S_1, \ldots, S_{r-1}, S_r$. This will be done by using the most recent shift register, and if necessary, modifying the length and the tap weights.

At iteration r, compute the next output of the $(r-1)$th shift register:

$$\hat{S}_r = -\sum_{j=1}^{n-1} \Lambda_j^{(r-1)} S_{r-j}.$$

Many terms in the sum are equal to zero because $n-1$ is larger than the degree of $\Lambda^{(r-1)}$, and thus the sum could be written as a sum from 1 to deg $\Lambda^{(r-1)}(x)$. The notation is less cumbersome, however, if we write the sum extending to $n-1$.

Subtract $\hat{S}_r$ from the desired output S_r to get a quantity Δ_r, known as the *rth discrepancy*:

$$\Delta_r = S_r - \hat{S}_r = S_r + \sum_{j=1}^{n-1} \Lambda_j^{(r-1)} S_{r-j}.$$

Equivalently,

$$\Delta_r = \sum_{j=0}^{n-1} \Lambda_j^{(r-1)} S_{r-j}.$$

If Δ_r is zero, then set $(L_r, \Lambda^{(r)}(x)) = (L_{r-1}, \Lambda^{(r-1)}(x))$, and the rth iteration is complete. Otherwise, the shift-register taps are modified as follows:

$$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) + Ax^l \Lambda^{(m-1)}(x),$$

where A is a field element, l is an integer, and $\Lambda^{(m-1)}(x)$ is one of the shift-register polynomials appearing earlier on the list. Now recompute the discrepancy (call it Δ_r') with this new polynomial:

$$\begin{aligned}\Delta_r' &= \sum_{j=0}^{n-1} \Lambda_j^{(r)} S_{r-j} \\ &= \sum_{j=0}^{n-1} \Lambda_j^{(r-1)} S_{r-j} + A \sum_{j=0}^{n-1} \Lambda_j^{(m-1)} S_{r-j-l}.\end{aligned}$$

We are ready to specify m, l, and A. Choose an m smaller than r for which $\Delta_m \neq 0$, choose $l = r - m$, and choose $A = -\Delta_m^{-1}\Delta_r$. Then

$$\Delta_r' = \Delta_r - \frac{\Delta_r}{\Delta_m}\Delta_m = 0,$$

and thus the new shift register will generate the sequence $S_1, \ldots, S_{r-1}, S_r$. We do not want just any such shift register, however, we want one of smallest length. We still have not specified which m for which $\Delta_m \neq 0$ should be chosen. If we choose m as the most recent iteration at which $L_m > L_{m-1}$, we will get a shortest-length shift register at every iteration, but this last refinement will take some time to develop.

A physical interpretation of the development up to this point is shown in Fig. 7.3. The two shift registers at iteration m and at iteration r are shown in Fig. 7.3(a). Iteration m is chosen such that at iteration m, the shift register $(L_{m-1}, \Lambda^{(m-1)}(x))$ failed to produce syndrome S_m, and the shortest shift register that produced the required syndrome at iteration m was longer than L_{m-1}. We can also suppose that shift register $(L_{r-1}, \Lambda^{(r-1)}(x))$ failed to produce S_r because otherwise, we need not do anything to it.

In Fig. 7.3(b), we show the shift register $(L_{m-1}, \Lambda^{(m-1)}(x))$ made into an auxillary shift register by lengthening it, positioning it, and doctoring its output so that it can compensate for the failure of $(L_{r-1}, \Lambda^{(r-1)}(x))$ to produce S_r. Notice that the auxillary shift register has an extra tap with weight 1 coming from the term $\Lambda_0^{(m-1)}$. During the first $r-1$ iterations, the remaining taps are producing the negative of the term coming from the tap with weight 1, and thus a string of zeros comes from the auxillary register and does not affect the generated syndromes. At the rth iteration, the two terms do not cancel, and a nonzero comes from the auxillary shift register. The coefficient A is selected to adjust this nonzero value so that it can be added to the rth

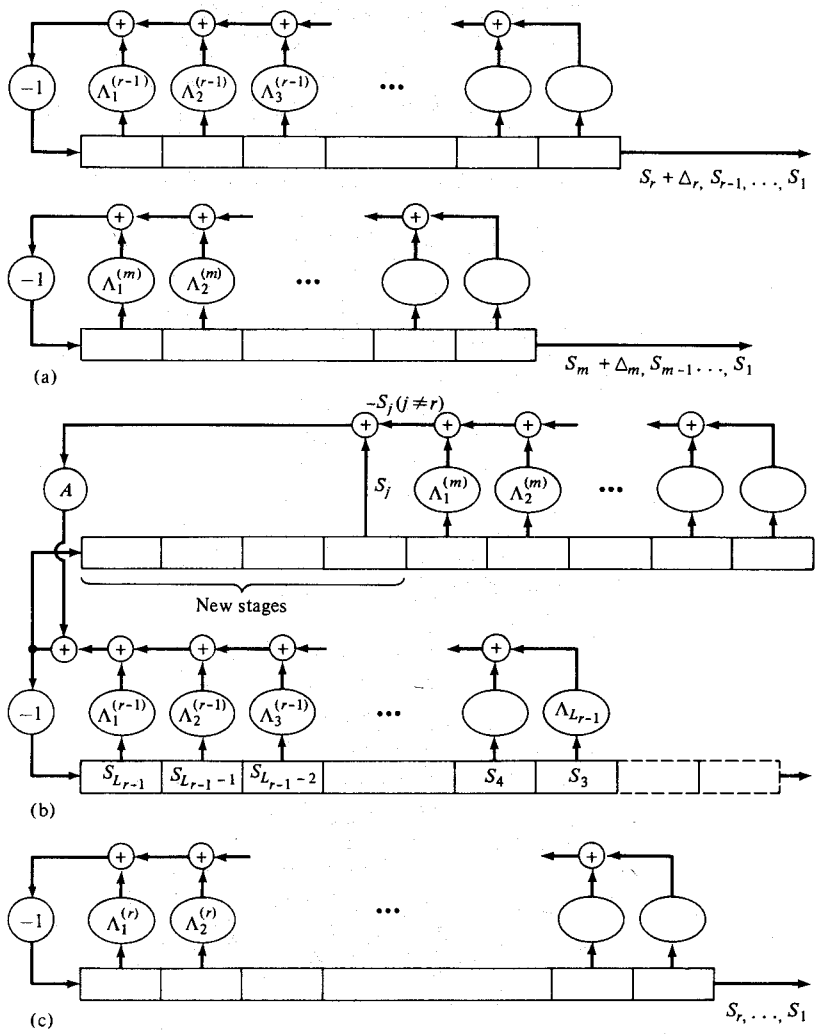

Figure 7.3 Berlekamp-Massey construction.

feedback of the main shift register, thereby producing the required syndrome S_r.

In Fig. 7.3(c), we show the two shift registers of part (b) physically merged into one shift register, which, because of superposition, does not change the behavior. This gives $(L_r, \Lambda^{(r)}(x))$. Sometimes $L_r = L_{r-1}$, sometimes $L_r > L_{r-1}$. When the latter happens, we replace m with r for use in future iterations.

A precise procedure for doing this is given by the following theorem, which asserts that it does produce a shortest shift register with the desired property. The proof is lengthy and occupies the remainder of the section.

□ **Theorem 7.4.1 (The Berlekamp-Massey Algorithm)** In any field, let $S_1, \ldots, S_{2t}$ be given. Under the initial conditions $\Lambda^{(0)}(x) = 1$, $B^{(0)}(x) = 1$, and $L_0 = 0$, let the following set of recursive equations be used to compute $\Lambda^{(2t)}(x)$:

$$\Delta_r = \sum_{j=0}^{n-1} \Lambda_j^{(r-1)} S_{r-j}$$

$$L_r = \delta_r(r - L_{r-1}) + (1 - \delta_r)L_{r-1}$$

$$\begin{bmatrix} \Lambda^{(r)}(x) \\ B^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix} \begin{bmatrix} \Lambda^{(r-1)}(x) \\ B^{(r-1)}(x) \end{bmatrix}$$

$r = 1, \ldots, 2t$ where $\delta_r = 1$ if both $\Delta_r \neq 0$ and $2L_{r-1} \leqslant r - 1$, and otherwise $\delta_r = 0$. Then $\Lambda^{(2t)}(x)$ is the smallest-degree polynomial with the properties that $\Lambda_0^{(2t)} = 1$, and

$$S_r + \sum_{j=1}^{n-1} \Lambda_j^{(2t)} S_{r-j} = 0 \qquad r = L_{2t} + 1, \ldots, 2t. \qquad \square$$

In the theorem, Δ_r may be zero, but only when δ_r is zero. The term $\Delta_r^{-1}\delta_r$ is then understood to be zero.

Notice that the matrix update requires at most $2t$ multiplications per iteration, and that the calculation of Δ_r requires no more than t multiplications per iteration. There are $2t$ iterations, and thus at most $6t^2$ multiplications. Thus using the algorithm will usually be much better than using a matrix inversion, which requires on the order of t^3 operations.

The proof of Theorem 7.4.1 is broken into two lemmas. First, in Lemma 7.4.2, we will find an inequality relationship between L_r and L_{r-1}. Then in Lemma 7.4.3, we will use the algorithm of Theorem 7.4.1 for the construction of a shift register that generates $S_1, \ldots, S_r$ from the shortest-length shift register that generates $S_1, \ldots, S_{r-1}$. We will conclude in Lemma 7.4.3 that the construction of Theorem 7.4.1 is the shortest such shift register because it satisfies the bound of Lemma 7.4.2.

□ **Lemma 7.4.2** Suppose that $(L_{r-1}, \Lambda^{(r-1)}(x))$ is a linear-feedback shift register of shortest length that generates $S_1, \ldots, S_{r-1}$; $(L_r, \Lambda^{(r)}(x))$ is a linear-feedback shift register of shortest length that generates $S_1, \ldots, S_{r-1}, S_r$; and $\Lambda^{(r)}(x) \neq \Lambda^{(r-1)}(x)$. Then

$$L_r \geqslant \max[L_{r-1}, r - L_{r-1}].$$

Proof The inequality to be proved is a combination of two inequalities:

$$L_r \geqslant L_{r-1} \qquad \text{and} \qquad L_r \geqslant r - L_{r-1}.$$

The first inequality is obvious, because if a linear-feedback shift register generates a sequence, it must also generate any beginning portion of the

sequence. The second inequality is obvious if $L_{r-1} \geq r$. Hence, assume $L_{r-1} < r$. Suppose the second inequality is not satisfied, and look for a contradiction. Then $L_r \leq r - 1 - L_{r-1}$. As a temporary shorthand, let $c(x) = \Lambda^{(r-1)}(x)$, let $b(x) = \Lambda^{(r)}(x)$, let $L = L_{r-1}$, and let $L' = L_r$. By assumption, we have: $r \geq L + L' + 1$, and $L < r$. Next, by the assumptions of the lemma,

$$S_r \neq -\sum_{i=1}^{L} c_i S_{r-i},$$

$$S_j = -\sum_{i=1}^{L} c_i S_{j-i} \qquad j = L+1, \ldots, r-1,$$

and

$$S_j = -\sum_{k=1}^{L'} b_k S_{j-k} \qquad j = L'+1, \ldots, r.$$

Now establish the contradiction. First,

$$S_r = -\sum_{k=1}^{L'} b_k S_{r-k} = \sum_{k=1}^{L'} b_k \sum_{i=1}^{L} c_i S_{r-k-i},$$

where the expansion of S_{r-k} is valid because $r-k$ runs from $r-1$ down to $r-L'$, which is in the range $L+1, \ldots, r-1$ because of the assumption $r \geq L + L' + 1$. Second,

$$S_r \neq -\sum_{i=1}^{L} c_i S_{r-i} = \sum_{i=1}^{L} c_i \sum_{k=1}^{L'} b_k S_{r-i-k},$$

where the expansion of S_{r-i} is valid because $r-i$ runs from $r-1$ down to $r-L$, which is in the range $L'+1, \ldots, r-1$ again because of the assumption $r \geq L + L' + 1$. The summations on the right side can be interchanged to agree with the right side of the previous equation. Hence we get a contradiction: $S_r \neq S_r$, and the contradiction proves the lemma. □

If we can find a shift-register design that satisfies the inequality of Lemma 7.4.2 with equality, then it must be of shortest length. The following lemma shows that Theorem 7.4.1 does this.

□ **Lemma 7.4.3** Suppose that $(L_i, \Lambda^{(i)}(x))$ with $i = 1, \ldots, r$ is a sequence of minimum-length linear-feedback shift registers such that $\Lambda^{(i)}(x)$ generates $S_1, \ldots, S_i$. If $\Lambda^{(r)}(x) \neq \Lambda^{(r-1)}(x)$, then

$$L_r = \max [L_{r-1}, r - L_{r-1}],$$

and any shift register that generates $S_1, \ldots, S_r$ and has length equal to the right-hand side is a minimum-length shift register. Theorem 7.4.1 gives such a shift register.

Proof By Lemma 7.4.2, L_r cannot be smaller than the right-hand side. If we can construct any shift register that generates the required sequence and whose length equals the right-hand side, then this must be a minimum-length shift register. The proof is by induction. We give a construction for a shift register satisfying the theorem, assuming that we have iteratively constructed such shift registers for all $k \leqslant r-1$. For each k, $k=1,\ldots,r-1$, let $(L_k, \Lambda^{(k)}(x))$ be the minimum-length shift register that generates $S_1,\ldots,S_k$. Assume for the induction argument that

$$L_k = \max[L_{k-1}, k-L_{k-1}] \qquad k=1,\ldots,n-1$$

whenever $\Lambda^{(k)}(x) \neq \Lambda^{(k-1)}(x)$. This is clearly true for $k=0$ because $L_0=0$ and $L_1=1$. Let m denote the value k had at the most recent iteration step that required a length change. That is, at the end of iteration $r-1$, m is that integer such that

$$L_{r-1} = L_m > L_{m-1}.$$

We now have

$$S_j + \sum_{i=1}^{L_{r-1}} \Lambda_i^{(r-1)} S_{j-i} = \sum_{i=0}^{L_{r-1}} \Lambda_i^{(r-1)} S_{j-i} = \begin{cases} 0 & j=L_{r-1},\ldots,r-1 \\ \Delta_r & j=r. \end{cases}$$

If $\Delta_r = 0$, then the shift register $(L_{r-1}, \Lambda^{(r-1)}(x))$ also generates the first r digits, and thus

$$L_r = L_{r-1} \qquad \text{and} \qquad \Lambda^{(r)}(x) = \Lambda^{(r-1)}(x).$$

If $\Delta_r \neq 0$, then a new shift register must be designed. Recall that a change in shift-register length occurred at $k=m$. Hence

$$S_j + \sum_{i=1}^{L_{m-1}} \Lambda_i^{(m-1)} S_{j-i} = \begin{cases} 0 & j=L_{m-1},\ldots,m-1 \\ \Delta_m \neq 0 & j=m, \end{cases}$$

and by the induction hypothesis,

$$\begin{aligned} L_{r-1} = L_m &= \max\,[L_{m-1}, m-L_{m-1}] \\ &= m - L_{m-1} \end{aligned}$$

because $L_m > L_{m-1}$. Now choose the new polynomial

$$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) - \Delta_r \Delta_m^{-1} x^{r-m} \Lambda^{(m-1)}(x),$$

and let $L_r = \deg\, \Lambda^{(r)}(x)$. Then, because $\deg\, \Lambda^{(r-1)}(x) \leqslant L_{r-1}$ and $\deg[x^{r-m}\Lambda^{(m-1)}(x)] \leqslant r-m+L_{m-1}$,

$$L_r \leqslant \max[L_{r-1}, r-m+L_{m-1}] \leqslant \max[L_{r-1}, r-L_{r-1}].$$

Hence recalling Lemma 7.4.2, if $\Lambda^{(r)}(x)$ generates $S_1,\ldots,S_r$, then $L_r = \max[L_{r-1}, r-L_{r-1}]$. It only remains to prove that the shift register $(L_r, \Lambda^{(r)}(x))$ generates the required sequence. This is done by

direct computation of the difference between S_j and the shift-register feedback:

$$S_j - \left(- \sum_{i=1}^{L_r} \Lambda_i^{(r)} S_{j-i} \right) = S_j + \sum_{i=1}^{L_{r-1}} \Lambda_i^{(r-1)} S_{j-i} - \Delta_r \Delta_m^{-1} \left[S_{j-r+m} + \sum_{i=1}^{L_{m-1}} \Lambda_i^{(m-1)} S_{j-r+m-i} \right]$$

$$= \begin{cases} 0 & j = L_r, L_r + 1, \ldots, r-1 \\ \Delta_r - \Delta_r \Delta_m^{-1} \Delta_m = 0 & j = r \end{cases}$$

Hence the shift register $(L_r, \Lambda^{(r)}(x)$ generates $S_1, \ldots, S_r$. In particular, $(L_{2t}, \Lambda^{(2t)}(x))$ generates $S_1, \ldots, S_{2t}$, and the lemma is proved. □

7.5 FAST DECODING OF BCH CODES

Understanding the Peterson-Gorenstein-Zierler decoder is the best way to understand the decoding of BCH codes. But when building a decoder, one gives up the conceptually clear in favor of the computationally efficient. The Peterson-Gorenstein-Zierler decoder, as described in Section 7.2, requires that two t by t matrices be inverted. Although matrix inverses in a finite field do not suffer from problems of round-off error, the computational work may still be excessive, especially for large t. Both of these matrix inversions can be circumvented. The first matrix inversion is in the computation of the error-locator polynomial; it can be circumvented by using the Berlekamp-Massey algorithm. The second matrix inversion is in the computation of the error magnitudes; it can be circumvented by a procedure known as the Forney algorithm. We begin this section with a derivation of the Forney algorithm; then we return to the Berlekamp-Massey algorithm.

The Forney algorithm is derived starting with the error-locator polynomial

$$\Lambda(x) = \Lambda_v x^v + \Lambda_{v-1} x^{v-1} + \cdots + \Lambda_1 x + 1,$$

which was defined to have zeros at X_l^{-1} for $l = 1, \ldots, v$:

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

Define the syndrome polynomial

$$S(x) = \sum_{j=1}^{2t} S_j x^j = \sum_{j=1}^{2t} \sum_{i=1}^{v} Y_i X_i^j x^j,$$

and define the error-evaluator polynomial $\Omega(x)$ in terms of these known polynomials:

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

The error-evaluator polynomial will play a minor role from time to time. It can be related to the error-locations and error magnitudes as follows.

□ **Theorem 7.5.1** The error-evaluator polynomial can be written

$$\Omega(x) = x \sum_{i=1}^{v} Y_i X_i \prod_{l \neq i} (1 - X_l x).$$

Proof By the definition of the terms in $\Omega(x)$,

$$\Omega(x) = \left[\sum_{j=1}^{2t} \sum_{i=1}^{v} Y_i X_i^j x^j \right] \left[\prod_{l=1}^{v} (1 - X_l x) \right] \pmod{x^{2t}}$$

$$= \sum_{i=1}^{v} Y_i X_i x \left[(1 - X_i x) \sum_{j=1}^{2t} (X_i x)^{j-1} \right] \prod_{l \neq i} (1 - X_l x) \pmod{x^{2t}}.$$

The bracketted term is a factorization of $(1 - X_i^{2t} x^{2t})$. Therefore

$$\Omega(x) = \sum_{i=1}^{v} Y_i X_i x (1 - X_i^{2t} x^{2t}) \prod_{l \neq i} (1 - X_l x) \pmod{x^{2t}}.$$

Because this is modulo x^{2t}, it is the same as the expression to be proved. □

Now we are ready to give an expression for the error magnitudes that is much simpler than matrix inversion.

□ **Theorem 7.5.2 (The Forney Algorithm)** The error magnitudes are given by

$$Y_l = \frac{\Omega(X_l^{-1})}{\prod_{j \neq l} (1 - X_j X_l^{-1})} = -\frac{\Omega(X_l^{-1})}{X_l^{-1} \Lambda'(X_l^{-1})}.$$

Proof Evaluate Theorem 7.5.1 at X_l^{-1} to get

$$\Omega(X_l^{-1}) = Y_l \prod_{j \neq l} (1 - X_j X_l^{-1})$$

which yields the first half of the theorem.

On the other hand, the derivative of $\Lambda(x)$ is

$$\Lambda'(x) = -\sum_{i=1}^{v} X_i \prod_{j \neq i} (1 - x X_j).$$

Hence

$$\Lambda'(X_l^{-1}) = -X_l \prod_{j \neq l} (1 - X_j X_l^{-1}),$$

from which the theorem follows. □

The Forney algorithm is a considerable improvement over matrix inversion but does require division. In Chapter 9, we will give alternative solutions that do not involve division.

Now we turn to the calculation of the error-locator polynomial using the Berlekamp-Massey algorithm of Theorem 7.4.1. Massey's view of the problem

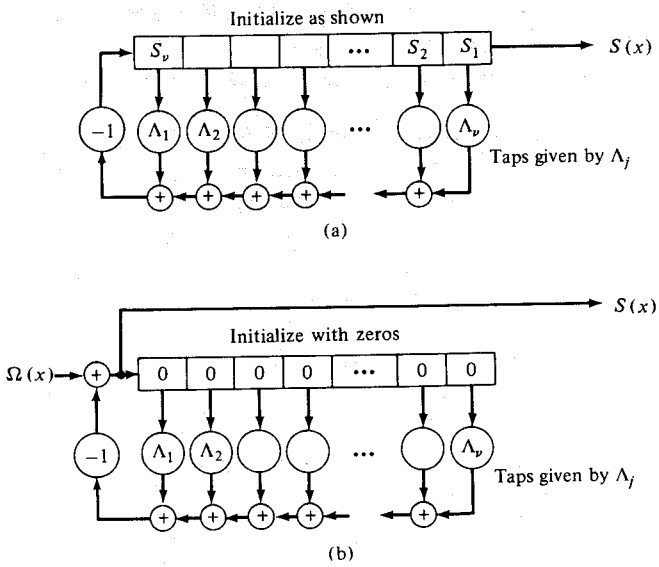

Figure 7.4 Generation of error spectrum. (a) Massey viewpoint. (b) Berlekamp viewpoint.

is shown in Fig. 7.4(a). Given S_j for $j=1,\ldots,2t$, find the smallest length vector Λ that satisfies the t equations

$$S_j+\sum_{k=1}^{t} S_{j-k}\Lambda_k=0 \qquad j=t+1,\ldots,2t.$$

That is, one is asked to solve a convolution given $2t$ components of $\mathbf{S}$, and the a priori knowledge that $\Lambda_j=0$ for j greater than t. The vector Λ that satisfies this equation gives the coefficients of the error-locator polynomial,

$$\Lambda(x)=\prod_{l=1}^{v}(1-xX_l),$$

and X_l for $l=1,\ldots,v$ are the error locations. The error-evaluator polynomial is not computed by the algorithm but can be computed later from $\Lambda(x)$ and $S(x)$ from the definition $\Omega(x)=S(x)\Lambda(x) \pmod{x^{2t}}$.

Berlekamp's formulation of the problem, shown in Fig. 7.4(b), portrays the error-evaluator polynomial in a more central role. This formulation asks for the vectors Λ and $\boldsymbol{\Omega}$, both zero for $t<k\leqslant n$, that satisfy the $2t$ equations

$$S_j+\sum_{k=1}^{t} S_{j-k}\Lambda_k=\Omega_j \qquad j=1,\ldots,2t,$$

where $S_j=0$ for $j\leqslant 0$. Notice that j now runs over $2t$ values. The solution is described by two polynomials: the error-locator polynomial and the error-evaluator polynomial.

The two forms of the problem are equivalent. We have treated Massey's form of the problem. If desired, the Berlekamp-Massey algorithm can be modified to solve the Berlekamp form of the problem directly by introducing both $\Lambda^{(r)}(x)$ and $\Omega^{(r)}(x)$ as iterates.

Figure 7.5 gives the Berlekamp-Massey algorithm graphically in the form of a flowchart. As shown, the algorithm will compute the error-locator polynomial from the $2t$ syndromes $S_1, \ldots, S_{2t}$. If the code has some j_0 other than 1, simply define $S_j = V_{j+j_0-1}$ for $j = 1, \ldots, 2t$. These syndromes are

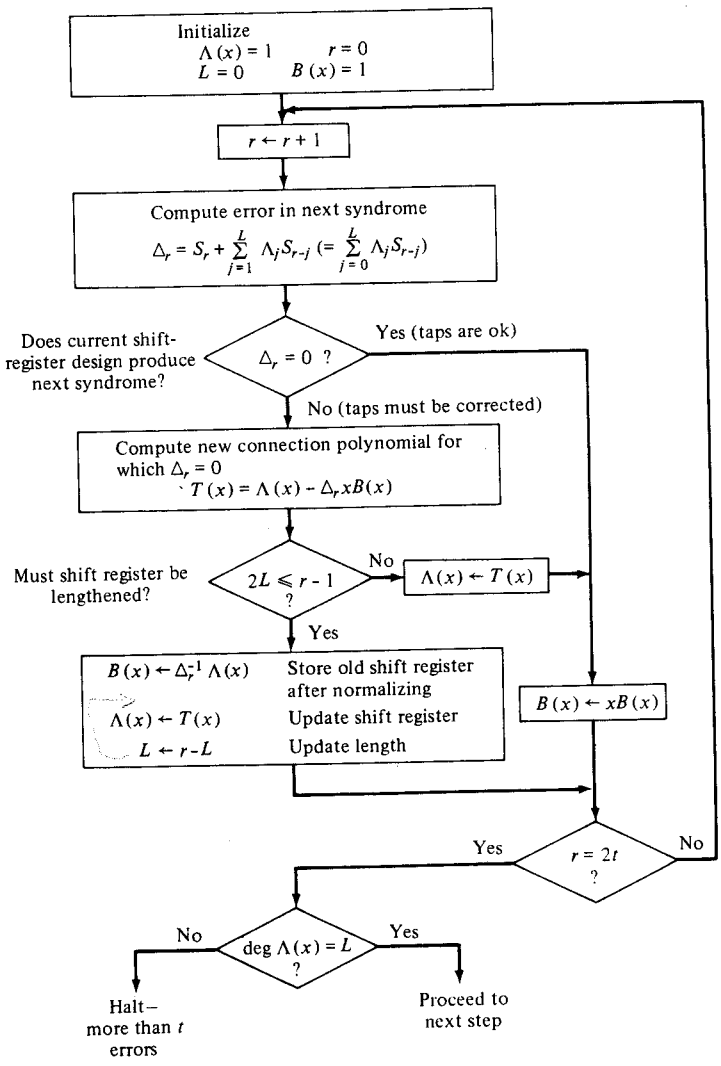

Figure 7.5 Berlekamp-Massey algorithm.

used just as before. There is no need to rederive the algorithm; this is merely a matter of properly labeling the internal variables of the algorithm.

The algorithm and its proof might be better understood by working through the details of Fig. 7.5 for specific examples. Table 7.3 gives the calculation for a (15, 9) Reed-Solomon triple-error-correcting code. These calculations should be checked by tracing through the six iterations in Fig. 7.5. Table 7.4 gives similar calculations for a (15, 5) BCH triple-error-correcting code.

The inner workings of the Berlekamp-Massey algorithm can appear somewhat mysterious. This difficulty might be softened by a few comments. At some iterations, say the rth, there may be more than one linear-feedback shift register of the minimum length that generate the required symbols.

Table 7.3 Sample Berlekamp-Massey computation

Reed-Solomon (15, 9) $t = 3$ code:

$$g(x) = (x+\alpha)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4)(x+\alpha^5)(x+\alpha^6)$$
$$= x^6 + \alpha^{10}x^5 + \alpha^{14}x^4 + \alpha^4 x^3 + \alpha^6 x^2 + \alpha^9 x + \alpha^6$$

$$i(x) = 0$$

$$v(x) = \alpha x^7 + \alpha^5 x^5 + \alpha^{11} x^2 = e(x)$$

$$S_1 = \alpha\alpha^7 + \alpha^5\alpha^5 + \alpha^{11}\alpha^2 = \alpha^{12}$$
$$S_2 = \alpha\alpha^{14} + \alpha^5\alpha^{10} + \alpha^{11}\alpha^4 = 1$$
$$S_3 = \alpha\alpha^{21} + \alpha^5\alpha^{15} + \alpha^{11}\alpha^6 = \alpha^{14}$$
$$S_4 = \alpha\alpha^{28} + \alpha^5\alpha^{20} + \alpha^{11}\alpha^8 = \alpha^{13}$$
$$S_5 = \alpha\alpha^{35} + \alpha^5\alpha^{25} + \alpha^{11}\alpha^{10} = 1$$
$$S_6 = \alpha\alpha^{42} + \alpha^5\alpha^{30} + \alpha^{11}\alpha^{12} = \alpha^{11}$$

r	Δ_r	$T(x)$	$B(x)$	$\Lambda(x)$	L
0			1	1	0
1	α^{12}	$1+\alpha^{12}x$	α^3	$1+\alpha^{12}x$	1
2	α^7	$1+\alpha^3 x$	$\alpha^3 x$	$1+\alpha^3 x$	1
3	1	$1+\alpha^3 x+\alpha^3 x^2$	$1+\alpha^3 x$	$1+\alpha^3 x+\alpha^3 x^2$	2
4	1	$1+\alpha^{14}x$	$x+\alpha^3 x^2$	$1+\alpha^{14}x$	2
5	α^{11}	$1+\alpha^{14}x+\alpha^{11}x^2+\alpha^{14}x^3$	$\alpha^4+\alpha^3 x$	$1+\alpha^{14}x+\alpha^{11}x^2+\alpha^{14}x^3$	3
6	0	$1+\alpha^{14}x+\alpha^{11}x^2+\alpha^{14}x^3$	$\alpha^4 x+\alpha^3 x^2$	$1+\alpha^{14}x+\alpha^{11}x^2+\alpha^{14}x^3$	3

$$\Lambda(x) = 1+\alpha^{14}x+\alpha^{11}x^2+\alpha^{14}x^3 = (1+\alpha^7 x)(1+\alpha^5 x)(1+\alpha^2 x)$$

Table 7.4 Sample Berlekamp-Massey computation

BCH (15, 5) $t = 3$ code:

$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$$

$$i(x) = 0$$

$$v(x) = x^7 + x^5 + x^2 = e(x)$$

$$S_1 = \alpha^7 + \alpha^5 + \alpha^2 = \alpha^{14}$$

$$S_2 = \alpha^{14} + \alpha^{10} + \alpha^4 = \alpha^{13}$$

$$S_3 = \alpha^{21} + \alpha^{15} + \alpha^6 = 1$$

$$S_4 = \alpha^{28} + \alpha^{20} + \alpha^8 = \alpha^{11}$$

$$S_5 = \alpha^{35} + \alpha^{25} + \alpha^{10} = \alpha^5$$

$$S_6 = \alpha^{42} + \alpha^{30} + \alpha^{12} = 1$$

r	Δ_r	$T(x)$	$B(x)$	$\Lambda(x)$	L
0			1	1	0
1	α^{14}	$1+\alpha^{14}x$	α	$1+\alpha^{14}x$	1
2	0	$1+\alpha^{14}x$	αx	$1+\alpha^{14}x$	1
3	α^{11}	$1+\alpha^{14}x+\alpha^{12}x^2$	$\alpha^4+\alpha^3x$	$1+\alpha^{14}x+\alpha^{12}x^2$	2
4	0	$1+\alpha^{14}x+\alpha^{12}x^2$	$\alpha^4x+\alpha^3x^2$	$1+\alpha^{14}x+\alpha^{12}x^2$	2
5	α^{11}	$1+\alpha^{14}x+\alpha^{11}x^2+\alpha^{14}x^3$	$\alpha^4+\alpha^3x+\alpha x^2$	$1+\alpha^{14}x+\alpha^{11}x^2+\alpha^{14}x^3$	3
6	0	$1+\alpha^{14}x+\alpha^{11}x^2+\alpha^{14}x^3$	$\alpha^4x+\alpha^3x^2+\alpha x^3$	$1+\alpha^{14}x+\alpha^{11}x^2+\alpha^{14}x^3$	3

$$\Lambda(x) = 1 + \alpha^{14}x + \alpha^{11}x^2 + \alpha^{14}x^3 = (1+\alpha^7x)(1+\alpha^5x)(1+\alpha^2x)$$

These all produce the same required sequence of r syndromes but differ in the future syndromes, which are not required of the rth iteration. During later iterations, whenever possible, another of the linear-feedback shift registers of this length is selected to produce the next required syndrome. If none such exists, then the shift register is lengthened. The test to see if lengthening is required, however, does not involve the next syndrome other than to determine that $\Delta_{r+1} \neq 0$. Hence, whenever one alternative shift register is available, then there are at least as many alternative shift registers as one less than the number of values that the $(r+1)$th syndrome can assume.

The algorithm can be programmed on a general-purpose computer or on a special-purpose computer designed for Galois field computations. If very high decoding speeds are required, one can build special hardware circuits, possibly using shift registers, to implement the algorithm directly.

A shift-register implementation is outlined in Fig. 7.6. Registers are provided for the three polynomials $S(x)$, $\Lambda(x)$, and $B(x)$, and the length of each register is large enough to hold the largest degree of its polynomial, and possibly a little larger. Shorter polynomials are stored simply by filling out the register with zeros. The $S(x)$ and $B(x)$ registers are each one stage longer than needed to store the largest polynomial; notice the extra symbol in the syndrome register. This, and the number of clocks applied to each shift register during an iteration, are set up so that the polynomials will precess one position during each iteration. This supplies the multiplications of $B(x)$ by x and also offsets the index of S_j by r to provide S_{j-r}, which appears in the expression for Δ_r. To see this, refer to Fig. 7.6; the syndrome register is shown as it is initialized. During each iteration, it will be shifted to the right by one symbol so that it will be timed properly for multiplication with Λ. During one iteration, the Λ register is cycled twice, first to compute Δ, then to be updated.

In Fig. 7.7, we show all of the computations involved in a decoder that uses the Berlekamp-Massey algorithm and the Forney algorithm. This decoder will produce the correct error-locator polynomial as long as at most t errors occur. If more than t errors occur, the algorithm will fail, either by

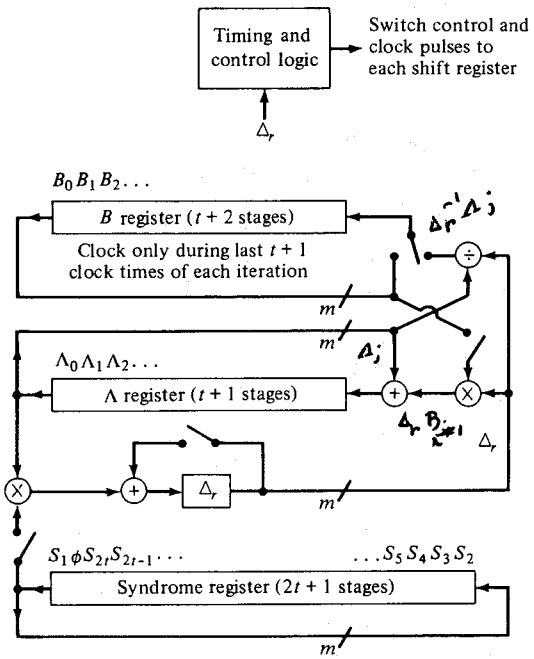

Figure 7.6 Outline of circuit for Berlekamp-Massey algorithm.
Notes: $2t$ iterations required. $2(t + 1)$ clocks per iterations. All data paths are m bits wide.

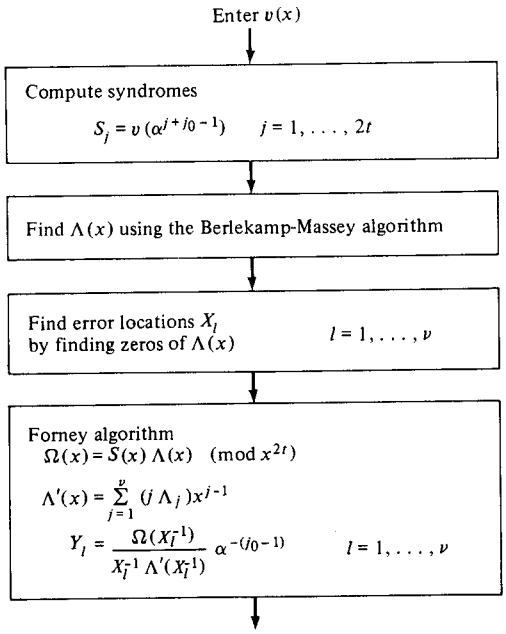

Figure 7.7 Fast decoding of BCH codes.

producing a polynomial that does not fit the requirements of an error-locator polynomial or by producing a legitimate, but incorrect, error-locator polynomial. We can recognize when the first case occurs, and thus the decoder can flag the message as uncorrectable. The tests that can be used to flag an uncorrectable error pattern are the following:

Number of distinct zeros of $\Lambda(x)$ in $GF(q^m)$ different from L.
Error symbols not in the symbol field.

The error-locator polynomial is tested first. If the error-locator polynomial is legitimate, the decoder proceeds to compute the error polynomial. If the symbol field is smaller than the error-locator field, it is possible that the decoder will find some error symbols outside of the symbol field. Such a symbol could not come through the channel, and hence this indicates a pattern of more than t errors has occurred.

In Section 9.6 we will discuss ways in which to decode BCH codes for some of the patterns of more than t errors. To end this section we will discuss the opposite situation—decoders that intentionally stop short of the designed distance. In practice, such decoders are used to greatly reduce the probability of decoding error, but the penalty is a greater probability of decoding failure.

To reduce the chance of a false decoding, a decoder may be designed to correct only up to t errors where $2t + 1$ is strictly smaller than d^* (this must

always happen if d^* is even). The decoder will then be sure to detect an uncorrectable error pattern if v, the number of errors that actually occur, satisfies

$$t + v < d^*,$$

because at least $d^* - t$ errors are needed to move any codeword into an incorrect decoding sphere.

The Berlekamp-Massey algorithm nicely fits into such a decoder. The procedure is to make $2t$ passes through the algorithm to generate $\Lambda(x)$. An additional $\tau - t$ iterations are then made to check if the discrepancy Δ_r is zero for each of these additional iterations where τ equals $d^* - 1 - t$. If Δ_n is nonzero for any of these iterations, the received word is flagged as having more than t errors.

The following theorem provides the justification for this procedure.

□ **Theorem 7.5.3** Given S_j for $j = 1, \ldots, t + \tau$, where $\tau > t$, and that at most τ errors occur, let $(L_{2t}, \Lambda^{(2t)}(x))$ describe a linear-feedback shift register that will produce S_j for $j = 1, \ldots, 2t$, and let

$$\Delta_r = \sum_{j=0}^{n-1} \Lambda_j^{(2t)} S_{r-j}.$$

Suppose that Δ_r equals zero for $r = 2t + 1, \ldots, \tau$ and $L_{2t} \leq t$. Then at most t errors have occurred, and $\Lambda^{(2t)}(x)$ is the correct error-locator polynomial.

Proof We are given syndromes S_j for $j = 1, \ldots, t + \tau$. If we had $\tau - t$ more syndromes, S_j for $j = t + \tau + 1, \ldots, 2\tau$, then we could correct τ errors; that is, we could correct every error pattern assumed to occur. Suppose we are given these extra syndromes by a genie or an imaginary side channel. Then by the Berlekamp-Massey algorithm, we could compute an error-locator polynomial whose degree equals the number of errors. But at iteration $2t$, $L_{2t} \leq t$, and by assumption Δ_r equals zero for $r = 2t + 1, \ldots, t + \tau$, and thus L is not updated before iteration $t + \tau + 1$. Hence, by the rule for updating L,

$$\begin{aligned} L_{2\tau} &\geq (t + \tau + 1) - L_{2t} \\ &\geq (t + \tau + 1) - t = \tau + 1, \end{aligned}$$

contrary to the assumption that there are at most τ errors. □

7.6 DECODING OF BINARY BCH CODES

The theory developed in this chapter holds for codes over any finite field. When the field is $GF(2)$, however, one can make some additional simplifications. An obvious simplification is that it only is necessary to find the error

location; the error magnitude is always one (but the decoder might compute the error magnitude as a check).

Further simplification is less obvious, but a clue suggesting a possibility occurs in the example of Table 7.4. Notice that Δ_r is always zero on even-numbered iterations. If this is always the case for binary codes, then even-numbered iterations can be skipped.

We shall prove in this section that this is so. The proof is based on the fact that over $GF(2)$ the even-numbered syndromes are easily determined from the odd-numbered syndromes by the formula

$$S_{2j} = \sum_{j=1}^{n} v(\alpha^{2j}) = \left[\sum_{j=1}^{n} v(\alpha^{j})\right]^2 = S_j^2,$$

as follows from Theorem 5.3.3. Let us calculate algebraic expressions for the coefficients of $\Lambda^{(r)}(x)$ for the first several values of r. Tracing through the algorithm of Fig. 7.5 and using the fact that $S_4 = S_2^2 = S_1^4$ for all binary codes gives

$$\Delta_1 = S_1 \qquad \Lambda^{(1)}(x) = S_1 x + 1$$

$$\Delta_2 = S_2 + S_1^2 = 0 \qquad \Lambda^{(2)}(x) = S_1 x + 1$$

$$\Delta_3 = S_3 + S_1 S_2 \qquad \Lambda^{(3)}(x) = (S_1^{-1} S_3 + S_2)x^2 + S_1 x + 1.$$

$$\Delta_4 = S_4 + S_1 S_3 + S_1^{-1} S_2 S_3 + S_2^2 = 0$$

For any binary BCH code, this shows that Δ_2 and Δ_4 are always zero, but it is impractical to continue indefinitely in this explicit way to find other even-numbered syndromes. We will formulate instead a general argument to show that $\Delta_r = 0$ for all even r.

□ **Theorem 7.6.1** In $GF(2)$, for any linear-feedback shift register $\Lambda(x)$ and any sequence $S_1, S_2, \ldots, S_{2v-1}$ satisfying $S_{2j} = S_j^2$ for $2j \leqslant 2v - 1$, suppose that

$$S_j = -\sum_{i=1}^{n-1} \Lambda_i S_{j-i} \qquad j = v, \ldots, 2v-1.$$

If the next member of the sequence is given by

$$S_{2v} = -\sum_{i=1}^{n-1} \Lambda_i S_{2v-i},$$

then

$$S_{2v} = S_v^2.$$

Proof The proof consists of giving identical expressions for S_v^2 and S_{2v}. First, we have

$$S_v^2 = \left(\sum_{i=1}^{n-1} \Lambda_i S_{v-i}\right)^2 = \sum_{i=1}^{n-1} \Lambda_i^2 S_{v-1}^2 = \sum_{i=1}^{n-1} \Lambda_i^2 S_{2v-2i},$$

and also

$$S_{2v} = \sum_{k=1}^{n-1} \Lambda_k S_{2v-k} = \sum_{k=1}^{n-1} \sum_{i=1}^{n-1} \Lambda_k \Lambda_i S_{2v-k-i}.$$

By symmetry, every term in the double sum with $i \neq k$ appears twice, and in $GF(2)$ the two terms add to zero. Hence, only the diagonal terms with $i = k$ contribute:

$$S_{2v} = \sum_{i=1}^{n-1} \Lambda_i^2 S_{2v-2i},$$

which agrees with the expression for S_v^2 and thus proves the theorem. □

Thus by induction, Δ_r is zero for even r, and we can analytically combine two iterations to give for odd r:

$$\Lambda^{(r)}(x) = \Lambda^{(r-2)}(x) - \Delta_r x^2 B^{(r-2)}(x),$$

$$B^{(r)}(x) = \delta_r \Delta_r^{-1} \Lambda^{(r-2)}(x) + (1 - \delta_r) x^2 B^{(r-2)}(x).$$

Using these formulas, iterations with r even can be skipped. This gives a faster decoder for binary codes. Notice that this improvement can be used even though the error pattern might contain more than t errors, because only the conjugacy relations of the syndromes were used in the proof of the theorem; nothing was assumed about the binary error pattern. Therefore subsequent tests for more than t errors are still valid.

7.7 DECODING WITH THE EUCLIDEAN ALGORITHM

The Euclidean algorithm can be used to develop alternative decoders to those we have discussed already. These decoders are a little easier to understand but have the reputation of being somewhat less efficient in practice. This latter observation, however, probably depends strongly on the application.

In Chapter 4 we gave the Euclidean algorithm as a recursive procedure for calculating the greatest common divisor of two polynomials. In a slightly expanded version the algorithm will also produce the polynomials $a(x)$ and $b(x)$ satisfying

$$\mathrm{GCD}[s(x), t(x)] = a(x)s(x) + b(x)t(x).$$

For any polynomials $s(x)$ and $t(x)$, we repeat the algorithm in a convenient matrix form using the notation $s(x) = \left\lfloor \frac{s(x)}{t(x)} \right\rfloor t(x) + r(x)$ to represent the division algorithm.

□ **Theorem 7.7.1. (Euclidean Algorithm for Polynomials)** Given two polynomials $s(x)$ and $t(x)$ with $\deg s(x) \geq \deg t(x)$, let $s^{(0)}(x) = s(x)$ and $t^{(0)}(x) = t(x)$, and let $\mathbf{A}^{(0)}(x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. The following recursive