

From the definition of S_i , this becomes

$$\begin{aligned}\sum_{i=0}^{\hat{e}} \hat{\sigma}_i S_{i-l} &= \sum_{i=0}^{\hat{e}} \hat{\sigma}_i \sum_{j=1}^e V_j U_j^{r+i-l} \\ &= \sum_{j=0}^{\hat{e}} V_j U_j^{r+i} \sum_{l=1}^e \hat{\sigma}_l U_j^{-l} \\ &= \sum_{j=1}^e V_j \hat{\sigma}(U_j^{-1}) U_j^{r+i} = 0; \quad \hat{e} \leq i \leq d-2\end{aligned}\quad (6.7.29)$$

Equation 6.7.29 can be regarded as a set of $d-1-\hat{e}$ linear equations in the e unknowns $V_j \hat{\sigma}(U_j^{-1})$ for $1 \leq j \leq e$. Since \hat{e} is the smallest integer for which (6.7.28) can be satisfied and since $[\sigma(D)S(D)]_e^{d-2} = 0$, we must have $\hat{e} \leq e$. Also, since $e \leq [(d-1)/2]$, it follows that $e \leq d-1-\hat{e}$. Now consider only the first e of the equations in (6.7.29), for $\hat{e} \leq i \leq \hat{e}+e-1$. As e equations in the e unknowns $V_j \hat{\sigma}(U_j^{-1})$, these equations are linearly independent by the same argument used to establish the linear independence of (6.7.10) in Theorem 6.7.1. Thus the only solution to these equations is given by

$$V_j \hat{\sigma}(U_j^{-1}) = 0; \quad 1 \leq j \leq e \quad (6.7.30)$$

Since $V_j \neq 0$, $1 \leq j \leq e$, it follows that U_j^{-1} is a root of $\hat{\sigma}(D)$ for $1 \leq j \leq e$. Since the degree of $\hat{\sigma}(D)$ is at most e , since $\hat{\sigma}(D)$ has the same e roots as $\sigma(D)$, and since $\hat{\sigma}_0 = \sigma_0$, it follows that $\hat{\sigma}(D) = \sigma(D)$. Since $\hat{\sigma}(D)$ has degree e , we also have $\hat{e} = e$, completing the proof. |

We next describe an iterative algorithm for finding $\sigma(D)$ in a particularly simple way.

Iterative Algorithm* for Finding $\sigma(D)$

We saw in the previous theorem that if at most $[(d-1)/2]$ errors occur, then $\sigma(D)$ is given in terms of the syndrome polynomial $S(D)$ by that solution to $[\sigma(D)S(D)]_e^{d-2} = 0$ for which e is smallest and $\sigma(D)$ has degree at most e with $\sigma_0 = 1$. This problem is most easily visualized in terms of the linear-feedback shift register (LFSR) shown in Figure 6.7.1.

The register is initially loaded with a sequence of elements S_0, S_1, \dots, S_{l-1} from a given field. The register then computes a new element S_l given in terms of the feedback connections $-C_1, \dots, -C_l$, by

$$S_l = -S_{l-1}C_1 - S_{l-2}C_2 - \dots - S_0C_l \quad (6.7.31)$$

The elements C_1, \dots, C_l are elements of the same field as the S_i . The register is then shifted to the right one position, S_l entering the register from

* This algorithm is due to Berlekamp (1967). We present here a modification of Berlekamp's algorithm due to Massey (1968) and follow Massey's approach closely.

ding

in (6.7.22) that $A(D)$ has $A(D)$ in (6.7.25) are un-

(6.7.26)

the product $\sigma(D)S(D)$ is (6.7.26) is equivalent to the

= 0

= 0

(6.7.27)

$2-e = 0$

set of $d-1-e$ linear [from (6.7.21), $\sigma_0 = 1$]. If U_1, \dots, U_e can be U_j^{-1} are the roots of $\sigma(D)$. procedure and, for future

ved sequence y.

locations of the errors.

V_e . Notice that, for binary e , by definition, they are

to discuss the implementa- 7.26) for $\sigma(D)$, the decoder wing theorem asserts that eforehand.

ors have occurred and that ger for which a polynomial isfying $[\hat{\sigma}(D)S(D)]_e^{d-2} = 0$.

e rewritten as

- 2 (6.7.28)

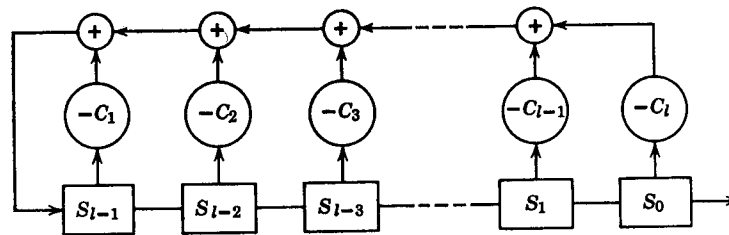


Figure 6.7.1. Linear feedback shift register (LFSR).

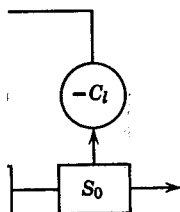
the left. On each successive shift to the right, a new element is computed, given by

$$S_i = -S_{i-1}C_1 - S_{i-2}C_2 - \cdots - S_{i-l}C_l; \quad i \geq l \quad (6.7.32)$$

We define the *register length* of an LFSR as the number of stages in the shift register (l in Figure 6.7.1). We also define the *connection polynomial* $C(D)$ of an LFSR as $C(D) = 1 + C_1D + C_2D^2 + \cdots + C_lD^l$ where C_1, C_2, \dots, C_l are the negatives of the feedback connections as shown in Figure 6.7.1. Since an LFSR is completely described (apart from the initial loading) by its register length and connection polynomial, we shall use the notation *register* $[C(D), l]$ to denote an LFSR with register length l and connection polynomial $C(D)$. Any or all of the feedback connections $-C_1, \dots, -C_l$ may be zero so that $C(D)$ may be an arbitrary polynomial with $C_0 = 1$ and degree at most l . Finally, if $S(D)$ is a polynomial (either finite or infinite), we shall say that the register $[C(D), l]$ *generates* $[S(D)]_0^L$ iff, when the register is initially loaded with S_0, \dots, S_{l-1} , the remaining elements (if any) S_l, \dots, S_L are those generated by the register, as given by (6.7.32). Observing that (6.7.32) is equivalent to the statement that the coefficient of D^i in $C(D)S(D)$ is zero, we see that register $[C(D), l]$ generates $[S(D)]_0^L$ iff

$$[C(D)S(D)]_l^L = 0 \quad (6.7.33)$$

The relationship between linear-feedback shift registers and finding $\sigma(D)$ should now be clear. We want to find the register $[\sigma(D), e]$ of smallest register length e that generates $[S(D)]_0^{q-2}$. Notice that the restrictions that $\sigma_0 = 1$ and that the degree of $\sigma(D)$ is at most e are built into the definition of register $[\sigma(D), e]$. The algorithm to be given below finds this shortest register. We shall show that the algorithm works for an arbitrary polynomial $S(D)$ over an arbitrary field. The algorithm operates by finding a sequence of registers, first finding the shortest register that generates S_0 , then the shortest register that generates $S_0 + S_1D$, and so forth. The register produced by the algorithm to generate $[S(D)]_0^{n-1}$ is denoted $[C_n(D), l_n]$.



LFSR).

element is computed,

$$i \geq l \quad (6.7.32)$$

or of stages in the shift register polynomial $C(D)$ where C_1, C_2, \dots, C_l in Figure 6.7.1. Since initial loading) by its the notation register and connection polynomial with $C_0 = 1$ and other finite or infinite), $S(D)]_0^L$ iff, when the missing elements (if any) by (6.7.32). Observing the coefficient of D^i in $S(D)]_0^L$ iff

$$(6.7.33)$$

sters and finding $\sigma(D)$ or $[\sigma(D), e]$ of smallest at the restrictions that built into the definition how finds this shortest for an arbitrary polynomial operates by finding a that generates S_0 , then so forth. The register is denoted $[C_n(D), l_n]$.

Roughly, the algorithm works as follows: given a register $[C_n(D), l_n]$ that generates $[S(D)]_0^{n-1}$, the algorithm tests to see whether $[C_n(D), l_n]$ also generates $[S(D)]_0^n$, that is, whether $[C_n(D)S(D)]_{l_n}^n = 0$. Since by assumption $[C_n(D)S(D)]_{l_n}^{n-1} = 0$, the question is whether the coefficient of D^n in $C_n(D)S(D)$ is equal to zero. This sum is called the *next discrepancy*, d_n , for the algorithm and expressing $C_n(D)$ by $1 + C_{n,1}D + C_{n,2}D^2 + \dots$, we have

$$d_n = S_n + \sum_{i=1}^{l_n} C_{n,i} S_{n-i} \quad (6.7.34)$$

Letting S_n' be the coefficient of D^n generated by the register, as given by (6.7.32), we have $d_n = S_n - S_n'$ so that d_n is the difference between the desired next output S_n and the actual register output S_n' . If $d_n = 0$, the algorithm increases n by one, but keeps the same register. If $d_n \neq 0$, a correction term is added to the connection polynomial to make it generate S_n correctly.

The detailed rules for the algorithm are as follows: for each n , the register $[C_{n+1}(D), l_{n+1}]$ is defined in terms of the register $[C_n(D), l_n]$ and a prior register in the sequence $[C_{k_n}(D), l_{k_n}]$ where $k_n < n$. The integer k_n is recursively defined for each $n > 0$ by

$$k_n = \begin{cases} k_{n-1} & \text{if } l_n = l_{n-1} \\ n-1 & \text{if } l_n > l_{n-1} \end{cases} \quad (6.7.35)$$

$C_{n+1}(D)$ and l_{n+1} are given by

$$C_{n+1}(D) = C_n(D) - \frac{d_n}{d_{k_n}} D^{n-k_n} C_{k_n}(D) \quad (6.7.36)$$

$$l_{n+1} = \begin{cases} l_n & ; \quad d_n = 0 \\ \max[l_n, n - (k_n - l_{k_n})] & ; \quad d_n \neq 0 \end{cases} \quad (6.7.37)$$

where d_n and d_{k_n} are given by (6.7.34), or more explicitly,

$$d_{k_n} = S_{k_n} + \sum_{i=1}^{l_{k_n}} C_{k_n,i} S_{k_n-i}$$

The algorithm starts at $n = 0$ with the initial conditions $C_0(D) = C_{-1}(D) = 1$, $l_0 = l_{-1} = 0$, $k_0 = -1$, $d_{-1} = 1$.

The following theorem asserts that $C_n(D)$ and l_n specify a register $[C_n(D), l_n]$ and that that register generates $[S(D)]_0^{n-1}$. We show, in a later theorem, that $[C_n(D), l_n]$ is the shortest register that generates $[S(D)]_0^{n-1}$.

Theorem 6.7.3. For each $n \geq 0$,

$$(a) \ k_n < n \quad (6.7.38)$$

$$(b) \ C_{n,0} = 1 \text{ [where } C_n(D) = C_{n,0} + C_{n,1}D + \cdots \text{]} \quad (6.7.39)$$

$$(c) \ \deg [C_n(D)] \leq l_n \quad (6.7.40)$$

$$(d) \ [C_n(D)S(D)]_{l_n}^{n-1} = 0 \quad (6.7.41)$$

Proof.

Part a. For $n = 0$, $k_0 < 0$ from the initial conditions. For $n > 0$, the proof is immediate from (6.7.35) using induction on n .

Part b. From the initial conditions $C_{0,0} = 1$. Now assume that $C_{n,0} = 1$ for any given n . Since $n - k_n > 0$, it follows from (6.7.36) that $C_{n+1,0} = 1$. Thus, by induction, $C_{n,0} = 1$ for all $n \geq 0$.

Parts c and d. We again use induction on n . From the initial conditions, (6.7.40) and (6.7.41) are satisfied for $n = -1, 0$. For any given n , assume that for $-1 \leq i \leq n$,

$$\deg [C_i(D)] \leq l_i \quad (6.7.42)$$

$$[C_i(D)S(D)]_{l_i}^{i-1} = 0 \quad (6.7.43)$$

The proof will be complete if we show that this implies that (6.7.42) and (6.7.43) are also satisfied for $i = n + 1$. We consider separately the case in which $d_n = 0$ and that in which $d_n \neq 0$. For $d_n = 0$, $C_{n+1}(D) = C_n(D)$ and $l_{n+1} = l_n$. Thus (6.7.42) for $i = n$ implies (6.7.42) for $i = n + 1$. Also (6.7.43) for $i = n$ implies that

$$[C_{n+1}(D)S(D)]_{l_{n+1}}^{n-1} = 0$$

From (6.7.34), we have $[C_{n+1}(D)S(D)]_n^n = d_n = 0$. Thus $[C_{n+1}(D)S(D)]_{l_{n+1}}^n = 0$, establishing (6.7.43) for $i = n + 1$. Now, assume that $d_n \neq 0$. From (6.7.36), we have

$$\begin{aligned} \deg [C_{n+1}(D)] &\leq \max \{ \deg [C_n(D)], n - k_n + \deg [C_{k_n}(D)] \} \\ &\leq \max \{ l_n, n - k_n + l_{k_n} \} = l_{n+1} \end{aligned}$$

where we have used (6.7.42) for $i = n$ and $i = k_n$ and then used (6.7.37). Finally, from (6.7.36)

$$[C_{n+1}(D)S(D)]_{l_{n+1}}^n = [C_n(D)S(D)]_{l_{n+1}}^n - \left[\frac{d_n}{d_{k_n}} D^{n-k_n} C_{k_n}(D)S(D) \right]_{l_{n+1}}^n \quad (6.7.44)$$

Since $l_{n+1} \geq l_n$, we have

$$[C_n(D)S(D)]_{l_{n+1}}^n = d_n D^n \quad (6.7.45)$$

$$(6.7.38)$$

$$(6.7.39)$$

$$(6.7.40)$$

$$(6.7.41)$$

ns. For $n > 0$, the proof

assume that $C_{n,0} = 1$
(6.7.36) that $C_{n+1,0} = 1$.

n the initial conditions,
or any given n , assume

$$(6.7.42)$$

$$(6.7.43)$$

mples that (6.7.42) and
er separately the case in
, $C_{n+1}(D) = C_n(D)$ and
2) for $i = n + 1$. Also

thus $[C_{n+1}(D)S(D)]_{l_{n+1}}^n =$
ame that $d_n \neq 0$. From

$$+ \deg [C_{k_n}(D)]\}$$

$$l_{n+1}$$

, and then used (6.7.37).

$$\left[\frac{d_n}{d_{k_n}} D^{n-k_n} C_{k_n}(D) S(D) \right]_{l_{n+1}}^n \quad (6.7.44)$$

$$(6.7.45)$$


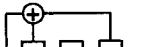
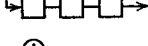
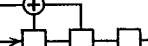
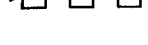
n	S_n	l_n	$C_n(D)$	LFSR	d_n	k_n	l_{k_n}	$C_{k_n}(D)$	d_{k_n}
0	1	0	1		1	-1	0	1	1
1	1	1	$1+D$		0	0	0	1	1
2	1	1	$1+D$		0	0	0	1	1
3	0	1	$1+D$		1	0	0	1	1
4	1	3	$1+D+D^3$		0	3	1	$1+D$	1
5	1	3	$1+D+D^3$		1	3	1	$1+D$	1
6	0	3	$1+D+D^2$		0	3	1	$1+D$	1
7	1	3	$1+D+D^2$		0	3	1	$1+D$	1
8		3	$1+D+D^2$						

Figure 6.7.2. Operation of algorithm in $GF(2)$ for $S(D) = 1 + D + D^2 + D^4 + D^5 + D^7$ up to $n = 8$.

For the final term in (6.7.44), we observe that D^{n-k_n} can be moved outside the brackets if the limits are simultaneously reduced by $n - k_n$. Thus

$$\begin{aligned} \left[\frac{d_n}{d_{k_n}} D^{n-k_n} C_{k_n}(D) S(D) \right]_{l_{n+1}}^n &= \frac{d_n}{d_{k_n}} D^{n-k_n} [C_{k_n}(D) S(D)]_{(l_{n+1}-n+k_n)}^{k_n} \\ &= \frac{d_n}{d_{k_n}} D^{n-k_n} d_{k_n} D^{k_n} \end{aligned} \quad (6.7.46)$$

where we have used (6.7.37) to see that $l_{n+1} - n + k_n \geq l_{k_n}$. Substituting (6.7.45) and (6.7.46) into (6.7.44), we have $[C_{n+1}(D)S(D)]_{l_{n+1}}^n = 0$, completing the proof. |

In Figure 6.7.2 we give an example of how the algorithm works, using polynomials in $GF(2)$ for simplicity. In Figure 6.7.3, l_n and $n - l_n$ are

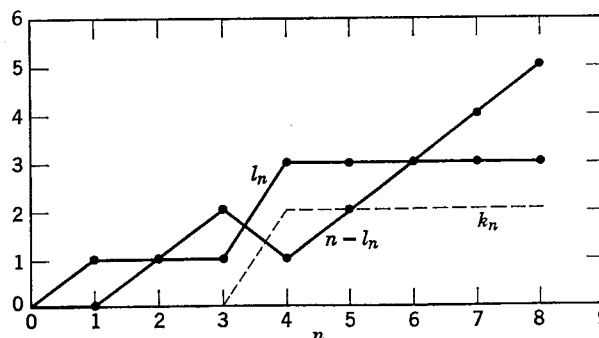


Figure 6.7.3. Sketch of l_n and $n - l_n$ as functions of n .

sketched as functions of n for this example. There are some important aspects of this relationship between l_n and $n - l_n$ that are valid in general. First, we observe that l_n is nondecreasing with n and it is obvious from (6.7.37) that this is valid in general. The next three results are more subtle.

LEMMA. For each $n \geq 0$,

$$k_n - l_{k_n} > i - l_i; \quad -1 \leq i < k_n \quad (6.7.47)$$

$$l_n = k_n - l_{k_n} + 1 \quad (6.7.48)$$

$$l_{n+1} > l_n \text{ iff } l_n < \frac{n}{2} \text{ and } d_n \neq 0 \quad (6.7.49)$$

Proof. We first show that (6.7.48) implies (6.7.49). From (6.7.37), $l_{n+1} > l_n$ iff both $d_n \neq 0$ and

$$n - (k_n - l_{k_n}) > l_n \quad (6.7.50)$$

From (6.7.48), (6.7.50) is equivalent to $n > 2l_n - 1$ or $l_n \leq n/2$, establishing (6.7.49). Equations 6.7.47 and 6.7.48 are clearly valid for $n = 0$ and we assume them to be valid for any given n . Using induction, the proof will be complete if we can show that the equations are valid for $n + 1$. If $l_{n+1} = l_n$, then $k_{n+1} = k_n$, and (6.7.47) and (6.7.48) are valid for $n + 1$. If $l_{n+1} > l_n$, then from (6.7.35), $k_{n+1} = n$ and

$$k_{n+1} - l_{k_{n+1}} = n - l_n \quad (6.7.51)$$

Since k_n is where the most recent register length change occurred prior to n , $l_i = l_n$ for $k_n < i < n$, and thus

$$k_{n+1} - l_{k_{n+1}} > i - l_i; \quad k_n < i < n = k_{n+1} \quad (6.7.52)$$

Also, from (6.7.37), $n - l_n > k_n - l_{k_n}$, so that combining (6.7.51) with (6.7.47),

$$k_{n+1} - l_{k_{n+1}} > i - l_i; \quad -1 \leq i \leq k_n \quad (6.7.53)$$

establishing (6.7.47) for $n + 1$. Still assuming $l_{n+1} > l_n$, we can combine (6.7.50), which is valid for n , and (6.7.51) to obtain

$$l_{n+1} = n - k_n + l_{k_n} = n - l_n + 1 = k_{n+1} - l_{k_{n+1}} + 1 \quad (6.7.54)$$

This establishes (6.7.48) for $n + 1$, completing the proof. |

From this lemma, we can see that $n - l_n$ as a function of n will have the appearance of an ascending sequence of peaks and for each n , k_n gives the location of the peak prior to n , which is higher than any of the preceding peaks (we consider a peak to occur at n if $n - l_n \geq (n + 1) - l_{n+1}$).

Before proving that the algorithm produces the shortest possible register for each n , we need two lemmas.

LEMMA 2. Suppose that $[A(D), l]$ and $[B(D), l]$ are two registers satisfying

$$[A(D)S(D)]_l^n = aD^n; \quad a \neq 0 \quad (6.7.55)$$

$$[B(D)S(D)]_l^n = 0 \quad (6.7.56)$$

then for some j , $0 \leq j \leq l$, there is a register $[F(D), l-j]$ satisfying

$$[F(D)S(D)]_{l-j}^{n-j} = fD^{n-j}; \quad f \neq 0 \quad (6.7.57)$$

Proof.

$$\{[A(D) - B(D)]S(D)\}_l^n = aD^n \quad (6.7.58)$$

Let j be the smallest integer for which $A_j \neq B_j$ and let $\gamma = A_j - B_j$. Let $F(D)$ be defined by

$$A(D) - B(D) = \gamma D^j F(D) \quad (6.7.59)$$

Now $F_0 = 1$ and

$$\deg F(D) < \min_{a, \gamma} [\deg A(D), \deg B(D)] - j \leq l - j$$

Thus $[F(D), l-j]$ is a register. Substituting (6.7.59) into (6.7.58) and observing that we can remove D^j from the brackets if we reduce the limits by j , we have

$$\gamma[F(D)S(D)]_{l-j}^{n-j} = aD^{n-j}$$

Since $a/\gamma \neq 0$, this completes the proof. |

LEMMA 3. Assume that for a given $S(D)$ and a given n , the register $[C_i(D), l_i]$ is the shortest register that generates $[S(D)]_0^{i-1}$ for each $i \leq n$. Then there exists no register $[A(D), l_A]$ such that, for some $n_A < n$, both

$$n_A - l_A > k_n - l_{k_n} \quad (6.7.60)$$

and

$$[A(D)S(D)]_{l_A}^{n_A} = aD^{n_A}; \quad a \neq 0 \quad (6.7.61)$$

Proof. We shall assume that the lemma is false and exhibit a contradiction. Let $[A(D), l_A]$ be the shortest register for which (6.7.60) and (6.7.61) are satisfied with $n_A < n$.

Case a. Assume that $n_A > k_{k_n}$. We have seen that $l_i = l_n$ for $k_n < i < n$ and thus $[C_n(D), l_n]$ is the shortest register that generates $[S(D)]_0^{i-1}$ for $k_n < i < n$. Taking $i = n_A$, this shows that $l_n \leq l_A$. Thus, since $n_A < n$, the register $[C_n(D), l_A]$ satisfies

$$[C_n(D)S(D)]_{l_A}^{n_A} = 0 \quad (6.7.62)$$

From the previous lemma, (6.7.61) and (6.7.62) assert the existence of a register $[F(D), l_A - j]$ for some $j > 0$ satisfying

$$[F(D)S(D)]_{l_A-j}^{n_A-j} = fD^{n_A-j}; \quad f \neq 0$$

This register is shorter than $[A(D), l_A]$ and satisfies (6.7.60) and (6.7.61), establishing a contradiction.

Case b. Assume $n_A \leq k_n$. The register $[C_{n_A}(D), l_{n_A}]$ is by hypothesis the shortest register generating $[S(D)]_0^{n_A-1}$, and thus $l_{n_A} \leq l_A$. Thus, using (6.7.47),

$$k_n - l_{k_n} \geq n_A - l_{n_A} \geq n_A - l_A$$

contradicting (6.7.60). |

Theorem 6.7.4. For any $S(D)$ and each $n \geq 0$, no register that generates $[S(D)]_0^{n-1}$ has a smaller register length than the register $[C_n(D), l_n]$ produced by the algorithm.

Proof. We use induction on n . The theorem is clearly valid for $n = 0$. Assume that, for any given $S(D)$, it is valid for a given n . If $l_{n+1} = l_n$, then clearly $[C_{n+1}(D), l_{n+1}]$ is the shortest register generating $[S(D)]_0^n$ since it is the shortest register generating $[S(D)]_0^{n-1}$ and any register generating $[S(D)]_0^n$ also generates $[S(D)]_0^{n-1}$. Now assume $l_{n+1} > l_n$, so that

$$[C_n(D)S(D)]_{l_n}^n = d_n D^n; \quad d_n \neq 0$$

Let $[B(D), l_B]$ be any register generating $[S(D)]_0^n$. We must have $l_B \geq l_n$, and, from Lemma 2, the registers $[C_n(D), l_n]$ and $[B(D), l_B]$ imply the existence of a register $[F(D), l_B - j]$ for some $j > 0$ so that

$$[F(D)S(D)]_{l_B-j}^{n-j} = f D^{n-j}; \quad f \neq 0$$

From Lemma 3, $(n-j) - (l_B - j) \leq k_n - l_{k_n}$. Thus $l_B \geq n - (k_n - l_{k_n}) = l_{n+1}$. Thus register $[B(D), l_B]$ is no shorter than $[C_{n+1}(D), l_{n+1}]$, completing the proof. |

The block diagram in Figure 6.7.4, from Massey (1968), suggests a way of implementing the algorithm. Notice that it uses (6.7.49) as a test for when l_n and k_n change. The length of the registers, j , in Figure 6.7.4, must be long enough to store the connection polynomial of the longest register expected. For decoding BCH codes, we choose $L = d - 2$ and $\sigma(D)$, except for the $\sigma_0 = 1$ term, is left in $R1$. We can choose $j = \lfloor (d-1)/2 \rfloor$ and be guaranteed of correcting all combinations of at most $\lfloor (d-1)/2 \rfloor$ errors. For binary BCH codes, the elements $\{S_i\}$ and $\{C_i\}$ are elements of $GF(2^m)$, and each of the registers in Figure 6.7.4 can be implemented by m binary registers. The $GF(2^m)$ multipliers can be implemented as in Figure 6.6.5. It can be seen that the equipment required for the registers and multipliers is proportional to $m d$. It can also be seen that the time required to find $\sigma(D)$ is proportional to $m d$ [or slightly more, depending on how $(d^*)^{-1}$ is calculated]. There are, of course, a number of detailed design questions to be answered in building

.60) and (6.7.61),

by hypothesis the l_A . Thus, using

ter that generates $n(D), l_n]$ produced

valid for $n = 0$.

If $l_{n+1} = l_n$, then $S(D)]_0^{k_n-1}$ since it is gister generating so that

ust have $l_B \geq l_n$, $D), l_B]$ imply the t

$n - (k_n - l_n) = l_{n+1}]$, completing

), suggests a way as a test for when .7.4, must be long register expected.)), except for the und be guaranteed . For binary BCH , and each of the try registers. The t can be seen that s proportional to)) is proportional lated]. There are, wered in building

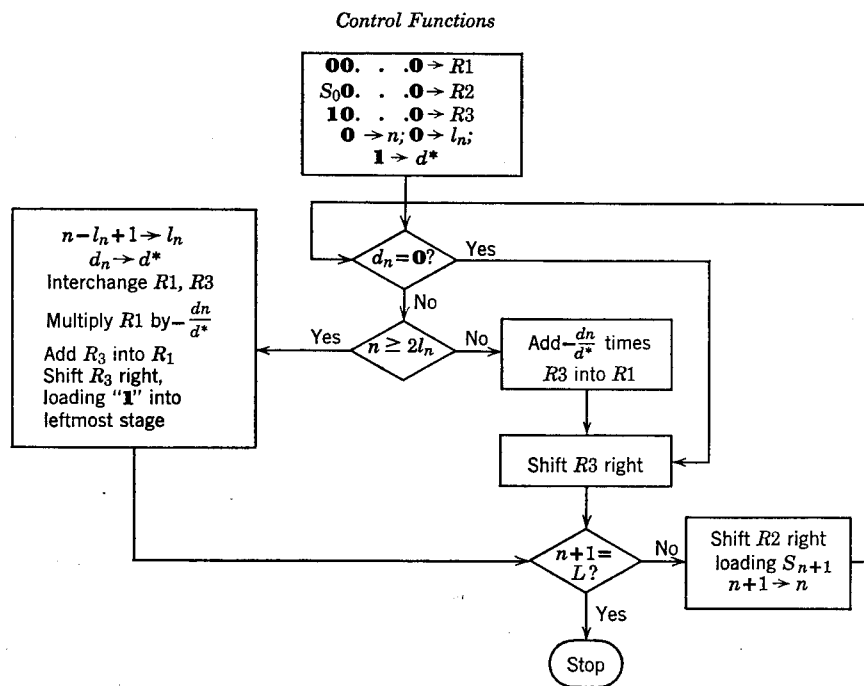
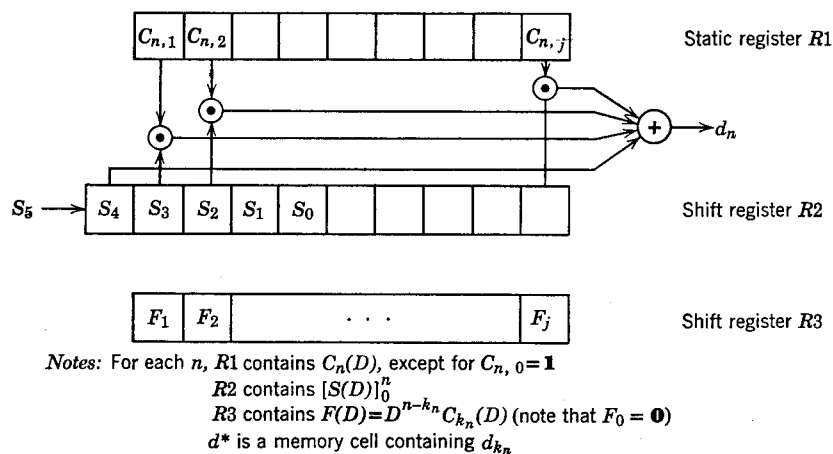


Figure 6.7.4. Implementation of LFSR algorithm to generate $[S(D)]_0^L$.

such a device and the only point to be made here is that such a device requires surprisingly little hardware and surprisingly little computing time. Berlekamp (1967) has also shown that, for binary codes with $r = 1$, d_n is always zero for n odd. Making use of this fact essentially cuts the computation time to find $\sigma(D)$ in half. This completes our discussion of step 2 in the BCH decoding procedure.

We now briefly discuss the implementation of steps 1, 3, and 4 in a BCH decoder. For step 1, the elements of the syndrome can be calculated by

$$S_i = \sum_{n=0}^{N-1} y_n \alpha^{(r+i)n} = (\cdots (y_{N-1} \alpha^{r+i} + y_{N-2}) \alpha^{r+i} + y_{N-3}) \alpha^{r+i} \cdots + y_0$$

Thus S_i can be calculated by adding each successive received digit into an initially empty register, the sum to be multiplied by α^{r+i} and returned to the register awaiting the next received digit.

Step 3 is most easily implemented by a procedure due to Chien (1964). If at most $\lfloor (d-1)/2 \rfloor$ errors have occurred, then $\sigma(D)$, as calculated in step 2, will be given by (6.7.21) and an error will have occurred in position n (that is, $z_n \neq 0$) iff $\sigma(\alpha^{-n}) = 0$, or equivalently iff

$$\sum_{i=0}^e \sigma_i \alpha^{-ni} = 0 \quad (6.7.63)$$

If we define

$$\sigma_{i,n} = \sigma_i \alpha^{-ni} \quad (6.7.64)$$

then

$$\sigma_{i,N-1} = \sigma_i \alpha^{-(N-1)i} = \sigma_i \alpha^i \quad (6.7.65)$$

and for each n

$$\sigma_{i,n-1} = \sigma_{i,n} \alpha^i \quad (6.7.66)$$

This suggests using the implementation in Figure 6.7.5 to perform the test in (6.7.63), first for $n = N-1$, then for $n = N-2$, and so forth.

We have already pointed out that, for a binary BCH code, step 4 in the decoding procedure is unnecessary. Thus the received digits can be fed out of the decoder synchronously with the operation of the circuit in Figure 6.7.5, simply adding y_n to z_n , modulo 2, as n goes from $N-1$ to 0, yielding the transmitted code word if at most $\lfloor (d-1)/2 \rfloor$ errors have occurred.

If more than $\lfloor (d-1)/2 \rfloor$ errors have occurred for a binary code, any one of the following three events might occur. First, the register $[\sigma(D), l]$ generated in step 2 might have $l > \lfloor (d-1)/2 \rfloor$. If $j = \lfloor (d-1)/2 \rfloor$ in the block diagram of Figure 6.7.4, $\sigma(D)$ will not be found in this case, but it is trivial to detect the event. It is also possible for l , as found in step 2, to not exceed $\lfloor (d-1)/2 \rfloor$, but for $\sigma(D)$ not to have l roots in $GF(2^m)$. In this case, fewer than l corrections will be made in step 3, but the decoded sequence will not be a code word. This

at such a device requires
putting time. Berlekamp
= 1, d_n is always zero
ne computation time to
2 in the BCH decoding

is 1, 3, and 4 in a BCH
in be calculated by

$$y_{N-3}\alpha^{r+i} \dots + y_0$$

the received digit into an
 x^{r+i} and returned to the

due to Chien (1964). If
as calculated in step 2,
ed in position n (that is,

$$(6.7.63)$$

$$(6.7.64)$$

$$(6.7.65)$$

$$(6.7.66)$$

7.5 to perform the test
, and so forth.

CH code, step 4 in the
d digits can be fed out
f the circuit in Figure
m $N - 1$ to 0, yielding
rs have occurred.

a binary code, any one
ister $[\sigma(D), l]$ generated
2] in the block diagram
ut it is trivial to detect
not exceed $\lfloor (d-1)/2 \rfloor$,
fewer than l corrections
ot be a code word. This

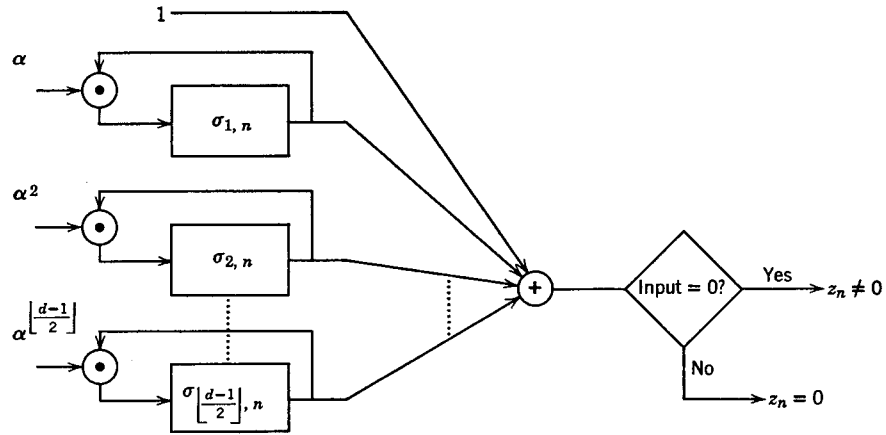


Figure 6.7.5. Step 3 of BCH decoding: finding error locations. Register initially loaded with $\sigma_1, \dots, \sigma_{\lfloor (d-1)/2 \rfloor}$. Then multiplication takes place, and then test for $z_{N-1} = 0$; then multiplication and test for $z_{N-2} = 0$; and so on to $z_0 = 0$.

again can be easily detected, either by counting the number of corrections or by checking whether the decoded sequence is a code word. Finally, it is possible that the register $[\sigma(D), l]$ has $l \leq \lfloor (d-1)/2 \rfloor$ and that l errors are found in decoding. In this case, the decoded sequence will be a code word and differ from the received sequence in at most $\lfloor (d-1)/2 \rfloor$ positions. The decoding error cannot be detected in this case, but at least we know that the decoder has made the maximum-likelihood decision for a binary symmetric channel.

We next turn to finding the error values (step 4) in the decoding process for nonbinary BCH codes. We defined the polynomial $A(D)$ in (6.7.22) as

$$A(D) = \sum_{j=1}^e V_j U_j^r \prod_{i \neq j} (1 - U_i D) \quad (6.7.67)$$

Also $A(D)$ is determined in terms of $\sigma(D)$ by (6.7.25),

$$A(D) = [\sigma(D)S(D)]_0^{d-2} \quad (6.7.68)$$

$A(D)$ can be calculated directly from (6.7.68) or, more elegantly, it can be incorporated as part of the iterative algorithm for finding $\sigma(D)$. In particular, we use the initial conditions $A_{-1}(D) = -D^{-1}$ and $A_0(D) = 0$ and for each $n \geq 0$, calculate $A_{n+1}(D)$ from

$$A_{n+1}(D) = A_n(D) - \frac{d_n}{d_{k_n}} D^{n-k_n} A_{k_n}(D) \quad (6.7.69)$$

where d_n and k_n are given by (6.7.34) and (6.7.35). This requires two extra registers in the block diagram of Figure 6.7.4 and virtually no extra control logic since the operations on the registers for $A_n(D)$ and $D^{n-k_n}A_{k_n}(D)$ are the same as those on the registers for $C_n(D)$ and $D^{n-k_n}C_{k_n}(D)$. The proof that $[C_n(D)S(D)]_0^{n-1} = A_n(D)$ for each $n \geq 0$ is almost the same as the proof of Theorem 6.7.3 and is treated in Problem 6.35.

After $A(D)$ has been found, we see from (6.7.67) that

$$A(U_j^{-1}) = V_j U_j^r \prod_{l \neq j} (1 - U_l U_j^{-1}) \quad (6.7.70)$$

The term on the right can be simplified if we define the derivative of $\sigma(D) = \sigma_0 + \sigma_1 D + \cdots + \sigma_e D^e$ as

$$\sigma'(D) = \sigma_1 + 2\sigma_2 D + \cdots + e\sigma_e D^{e-1} \quad (6.7.71)$$

This calculation of $\sigma'(D)$ can be easily instrumented from $\sigma(D)$, and if q is a power of 2, it is simply the odd power terms of $\sigma(D)$ divided by D . Since

$$\sigma(D) = \prod_j (1 - U_j D)$$

we also have (see Problem 6.36):

$$\begin{aligned} \sigma'(D) &= - \sum_{j=1}^e U_j \prod_{l \neq j} [1 - U_l D] \\ \sigma'(U_j^{-1}) &= - U_j \prod_{l \neq j} (1 - U_l U_j^{-1}) \end{aligned} \quad (6.7.72)$$

Substituting (6.7.72) into (6.7.70),

$$V_j = - U_j^{1-r} \frac{A(U_j^{-1})}{\sigma'(U_j^{-1})} \quad (6.7.73)$$

Recalling the definitions of U_j and V_j in (6.7.16), each nonzero noise digit z_n is given by

$$z_n = - \frac{\alpha^{n(1-r)} A(\alpha^{-n})}{\sigma'(\alpha^{-n})} \quad (6.7.74)$$

Each of the three terms on the right-hand side of (6.7.74) can be calculated successively for n going from $N-1$ to 0 by the same type of circuit as in Figure 6.7.5.

This concludes our discussion of decoding for BCH codes. The major point to be remembered is that, although the decoding is conceptually complicated, it is very simple in terms of decoding time and required circuitry. Apart from storage of the received word and a circuit to take the inverse of elements in $GF(q^m)$, the amount of hardware required is proportional to

$m d$. The
4 is pro
Let u
the limi
estimate

where
fixed R
number
drops l
has ca
binary
does d
occurs
import
The
BCH
field in
In this

Thus
to see
of cho
tion c
 d nor
Sin
of q
sizes.
input
used
synt
code
arbit
decr
prop
also
each

$m d$. The decoding time in step 2 is proportional to $m d$ and that in steps 3 and 4 is proportional to $m N$.

Let us see what can be said about the behavior of binary BCH codes in the limit as $N \rightarrow \infty$. Assume for the moment that $m(d-1)/2$ is a good estimate of the number of check digits in the code so that

$$\frac{m(d-1)}{2N} \approx 1 - R$$

where R is the rate in binary digits. Since $m \geq \log_2(N+1)$, we see that for fixed R , $(d-1)/2N$ must approach 0 as N approaches infinity. Thus the number of errors that can be corrected by the decoding algorithm eventually drops below the expected number of errors on the channel. Peterson (1961) has calculated the exact number of check digits required in a variety of binary BCH codes, and his results indicate that, for fixed R , $(d-1)/2N$ does decrease toward zero with increasing N . On the other hand, this decrease occurs at such large values of N that this limitation is of little practical importance.

The Reed Solomon (1960) codes are a particularly interesting class of BCH codes in which the parameter m is 1; that is, in which the extension field in which α is defined is the same as the symbol field for the code letters. In this case, the minimal polynomial of α^i is simply $D - \alpha^i$, so we have

$$g(D) = \prod_{i=r}^{r+d-2} (D - \alpha^i)$$

Thus this code has $d-1$ check digits and a minimum distance of d . It is easy to see that no group code with this alphabet size, block length and number of check digits can have a larger minimum distance, since if all the information digits but 1 are chosen to be zero, the resulting code word has at most d nonzero digits.

Since the block length N of a Reed-Solomon code is $q-1$ or a divisor of $q-1$, it can be seen that these codes are useful only for larger alphabet sizes. They can be used effectively on continuous time channels where the input alphabet is chosen as a large set of waveforms. They have also been used effectively by Forney (1965) in a concatenation scheme where the symbols in the Reed-Solomon code are code words in a smaller, imbedded code. Forney has shown that such codes can be used at transmission rates arbitrarily close to capacity. The error probability is an exponentially decreasing function of the block length, and the decoding complexity is proportional to a small power of the block length. Reed-Solomon codes can also be used directly on a channel with a small input alphabet by representing each letter in a code word by a sequence of channel letters. Such a technique