(Draft of December 11, 2000)

Chapter 8: Cyclic Codes

8.1 Introduction.

At the beginning of Chapter 7, we said that by restricting our attention to linear codes (rather than arbitrary, unstructured codes), we could hope to find some good codes which are reasonably easy to implement. And it is true that (via syndrome decoding, for example) a "small" linear code, say with dimension or redundancy at most 20, can be implemented in hardware without much difficulty. However, in order to obtain the performance promised by Shannon's theorems, it is necessary to use larger codes, and in general, a large code, even if it is linear, will be difficult to implement. For this reason, almost all block codes used in practice are in fact *cyclic codes*; cyclic codes form a very small and highly structured subset of the set of linear codes. In this chapter, we will give a general introduction to cyclic codes, discussing both the underlying mathematical theory (Sec. 8.1) and the basic hardware circuits used to implement cyclic codes (Sec. 8.2). In Section 8.3 we will show that Hamming codes can be implemented as cyclic codes, and in Sections 8.4 and 8.5 we will see how cyclic codes are used to combat *burst errors*. Our story will be continued in Chapter 9, where we will study the most important family of cyclic codes yet discovered: the BCH/Reed-Solomon family.

We begin our studies with the innocuous-appearing definition of the class of cyclic codes.

`Definition.` An (n, k) linear code over a field F is said to be *a cyclic code* if, for every codeword $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1})$, the right cyclic shift of $\mathbf{C}$, viz. $\mathbf{C}^R = (C_{n-1}, C_0, \ldots, C_{n-2})$, is also a codeword.

As we shall see, there are many cyclic codes, but compared to linear codes, they are quite scarce. For example, there are 11,811 linear $(7, 3)$ codes over $GF(2)$, but only two are cyclic!

Example 8.1. If F is any field, and n is an integer ≥ 3, there are always at least four cyclic codes of length n over F, usually called the four *trivial* cyclic codes:

- An $(n, 0)$ code, consisting of just the all-zero codeword, called the *no information code*.
- An $(n, 1)$ code, consisting of all codewords of the form $(a, a, \ldots, a)$, for $a \in F$, called the *repetition code*.
- An $(n, n-1)$ code, consisting of all vectors $(C_0, C_1, \ldots, C_{n-1})$ such that $\sum_i C_i = 0$, called the *single parity-check code*.
- An (n, n) code, consisting of all vectors of length n, called the *no parity code*. ❒

For some values of n and F, the trivial cyclic codes described in Example 8.1 are the only cyclic codes of length n over F (e.g., $n = 19$ and $F = GF(2)$). However, there are often other, more interesting cyclic codes, as the next two examples illustrate.

Example 8.2. Consider the $(7, 3)$ linear code over $GF(2)$ defined by the generator matrix

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

This code has eight codewords. Denoting the rows of G by $\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3$, the nonzero codewords are:

$$\begin{aligned} \mathbf{C}_1 &= 1011100 \\ \mathbf{C}_2 &= 0101110 \\ \mathbf{C}_3 &= 0010111 \\ \mathbf{C}_1 + \mathbf{C}_2 &= 1110010 \\ \mathbf{C}_1 + \mathbf{C}_3 &= 1001011 \\ \mathbf{C}_2 + \mathbf{C}_3 &= 0111001 \\ \mathbf{C}_1 + \mathbf{C}_2 + \mathbf{C}_3 &= 1100101. \end{aligned}$$

This code is in fact a cyclic code. To verify this, we need to check that the right cyclic shift of each codeword is also a codeword. For example, the right cyclic shift of $\mathbf{C}_1$ is $\mathbf{C}_2$. The complete list of right cyclic shifts follows:

$$\begin{aligned}
\mathbf{C}_1 &\to \mathbf{C}_2 \\
\mathbf{C}_2 &\to \mathbf{C}_3 \\
\mathbf{C}_3 &\to \mathbf{C}_1 + \mathbf{C}_3 \\
\mathbf{C}_1 + \mathbf{C}_2 &\to \mathbf{C}_2 + \mathbf{C}_3 \\
\mathbf{C}_1 + \mathbf{C}_3 &\to \mathbf{C}_1 + \mathbf{C}_2 + \mathbf{C}_3 \\
\mathbf{C}_2 + \mathbf{C}_3 &\to \mathbf{C}_1 \\
\mathbf{C}_1 + \mathbf{C}_2 + \mathbf{C}_3 &\to \mathbf{C}_1 + \mathbf{C}_2.
\end{aligned}$$

❒

Example 8.3. Consider the $(4, 2)$ linear code over $GF(3)$ defined by the generator matrix

$$G = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 1 & 1 & 2 & 2 \end{bmatrix}.$$

This code has nine codewords. Denoting the rows of G by $\mathbf{C}_1$ and $\mathbf{C}_2$ the nonzero codewords are:

$$\begin{aligned}
\mathbf{C}_1 &= 1020 \\
2\mathbf{C}_1 &= 2010 \\
\mathbf{C}_2 &= 1122 \\
\mathbf{C}_1 + \mathbf{C}_2 &= 2112 \\
2\mathbf{C}_1 + \mathbf{C}_2 &= 0102 \\
2\mathbf{C}_2 &= 2211 \\
\mathbf{C}_1 + 2\mathbf{C}_2 &= 0201 \\
2\mathbf{C}_1 + 2\mathbf{C}_2 &= 1221.
\end{aligned}$$

This code is also a cyclic code. For example, the right cyclic shift of $\mathbf{C}_1$ is $2\mathbf{C}_1 + \mathbf{C}_2$. The complete list of right cyclic shifts follows:

$$\begin{aligned}
\mathbf{C}_1 &\to 2\mathbf{C}_1 + \mathbf{C}_2 \\
2\mathbf{C}_1 &\to \mathbf{C}_1 + 2\mathbf{C}_2 \\
\mathbf{C}_2 &\to \mathbf{C}_1 + \mathbf{C}_2 \\
\mathbf{C}_1 + \mathbf{C}_2 &\to 2\mathbf{C}_2 \\
2\mathbf{C}_1 + \mathbf{C}_2 &\to 2\mathbf{C}_1 \\
2\mathbf{C}_2 &\to 2\mathbf{C}_1 + 2\mathbf{C}_2 \\
\mathbf{C}_1 + 2\mathbf{C}_2 &\to \mathbf{C}_1 \\
2\mathbf{C}_1 + 2\mathbf{C}_2 &\to \mathbf{C}_2.
\end{aligned}$$

❒

The definition of a cyclic code may seem arbitrary, but in fact there are good reasons for it, which begin to appear if we introduce the notion of the generating function of a codeword. If $\mathbf{C} = (C_0, \cdots, C_{n-1})$ is a codeword, its *generating function* is defined to be the polynomial

$$C(x) = C_0 + C_1 x + \cdots + C_{n-1} x^{n-1},$$

where x is an indeterminate. The reason this is useful is that by using generating functions, we can give a simple algebraic characterization of the right cyclic shift of a codeword. In order to give this characterization, we need to define the important "mod" operator for integers and polynomials.*

`Definition.` If p and m are integers with $m > 0$, then "$p \bmod m$" denotes the remainder obtained when p is divided by m; thus $p \bmod m$ is the unique integer r such that $p - r$ is divisible by m, and $0 \le r \le m-1$. Similarly, if $P(x)$ and $Q(x)$ are polynomials, $P(x) \bmod M(x)$ denotes the remainder when $P(x)$ is divided by $M(x)$; thus $P(x) \bmod M(x)$ is the unique polynomial $R(x)$ such that $P(x) - R(x)$ is divisible by $M(x)$, and $\deg R(x) < \deg M(x)$. ❒

Example 8.4. Here are some examples.

$$\begin{aligned}
7 \bmod 5 &= 2 \\
-6 \bmod 4 &= 2 \\
4 \bmod 6 &= 4 \\
21 \bmod 7 &= 0 \\
x^3 \bmod x^2 &= 0 \\
x^2 \bmod x^3 &= x^2 \\
x^{1000} \bmod (x^2+x+1) &= x \\
(5x^2+1) \bmod (x^2+1) &= -4 \qquad \text{(over the reals)} \\
(x+1)^3 \bmod (x^2+1) &= 0 \qquad \text{(over } GF(2)\text{)} \\
x^i \bmod (x^n-1) &= x^{i \bmod n}
\end{aligned}$$

❒

The following Lemma lists the most important properties of the mod operator for polynomials.

`Lemma 1.`

(i) If $\deg P(x) < \deg M(x)$, then $P(x) \bmod M(x) = P(x)$.
(ii) If $M(x) \,|\, P(x)$, then $P(x) \bmod M(x) = 0$.
(iii) $(P(x) + Q(x)) \bmod M(x) = P(x) \bmod M(x) + Q(x) \bmod M(x)$.
(iv) $(P(x)Q(x)) \bmod M(x) = (P(x)(Q(x) \bmod M(x))) \bmod M(x)$.
(v) If $M(x) \,|\, N(x)$, then $(P(x) \bmod N(x)) \bmod M(x) = P(x) \bmod M(x)$.

`Proof:` Left as Problem 8.5.

❒

Now we can give the promised algebraic characterization of the right cyclic shift operation.

`Theorem 8.1.` *If $\mathbf{C} = (C_0, C_1, \cdots, C_{n-1})$ is a codeword with generating function $C(x) = C_0 + C_1 x + \cdots + C_{n-1}x^{n-1}$, then the generating function $C^R(x)$ for the right cyclic shift $\mathbf{C}^R$ is given by the formula*

$$C^R(x) = xC(x) \bmod (x^n - 1).$$

`Proof:` Since $C(x) = C_0 + C_1 x + \cdots + C_{n-1}x^{n-1}$, we have

$$\begin{aligned}
xC(x) &= C_0 x + \cdots + C_{n-2}x^{n-1} + C_{n-1}x^n \\
C^R(x) &= C_{n-1} + C_0 x + \cdots + C_{n-2}x^{n-1}.
\end{aligned}$$

* This use of "mod" as a *binary operator* is closely related to, but not identical with, the more familair use of "mod" as an *equivalence relation*. Thus $Q(x) \equiv P(x) \pmod{M(x)}$ (the equivalence relation use of "mod") means only that $Q(x) - P(x)$ is divisible by $M(x)$, whereas $Q(x) = P(x) \bmod M(x)$ (the binary operator) means that $Q(x) - P(x)$ is divisible by $M(x)$ *and* $\deg Q(x) < \deg M(x)$.

Hence $xC(x) - C^R(x) = C_{n-1}(x^n - 1)$. Since $\deg C^R(x) < \deg(x^n - 1)$, and $xC(x) - C^R(x)$ is a multiple of $x^n - 1$, the result follows from the definition of the mod operation. ❒

The generating function for a codeword is so useful that we will often not bother to distinguish between a codeword and its generating function. Thus for example we might view an (n, k) linear code as a set of polynomials of (formal) degree $n - 1$, such that any linear combination of polynomials in the code is again in the code. From this viewpoint, by Theorem 8.1, a cyclic code is a linear code such that if $C(x)$ is a codeword, then $xC(x) \bmod (x^n - 1)$ is also. By repeatedly applying the right cyclic shift generation, we find that $x^i C(x) \bmod (x^n - 1)$ is a codeword, for all $i \geq 0$ (see Problem 8.6). The following theorem is a generalization of this observation. For convenience, we introduce the notation

$$[P(x)]_n$$

as a shorthand for

$$P(x) \bmod (x^n - 1).$$

`Theorem 8.2.` *If $\mathcal{C}$ is an (n, k) cyclic code, and if $C(x)$ is a codeword in $\mathcal{C}$, then for any polynomial $P(x)$, $[P(x)C(x)]_n$ is also a codeword in $\mathcal{C}$*

`Proof:` Suppose $P(x) = \sum_{i=0}^{k} P_i x^i$. Then

$$\begin{aligned}[P(x)C(x)]_n &= \left[(\sum_{i=0}^{k} P_i x^i)C(x)\right]_n \\ &= \sum_{i=0}^{k} P_i \left[x^i C(x)\right]_n\end{aligned}$$

by Lemma 1 (iii). But by the remarks immediately preceeding the statement of this theorem, $\left[x^i C(x)\right]_n$ is a codeword for each i, and so, since the code is linear, the linear combination $\sum P_i \left[x^i C(x)\right]_n$ is also a codeword. ❒

Example 8.5. Consider the $(7, 3)$ cyclic code of Example 8.2. The codeword $\mathbf{C}_1 + \mathbf{C}_3$, viewed as a polynomial, is $1 + x^3 + x^5 + x^6$. According to Theorem 8.2, if we multiply this polynomial by any other polynomial, and reduce the result mod $(x^7 - 1)$, the resulting polynomial will also be in the code. For example:

$$\begin{aligned}\left[(1 + x)(1 + x^3 + x^5 + x^6)\right]_7 &= x + x^3 + x^4 + x^5 \\ &= \mathbf{C}_2 \\ \left[(1 + x^{53} + x^{100})(1 + x^3 + x^5 + x^6)\right]_7 &= 1 + x + x^4 + x^6 \\ &= \mathbf{C}_1 + \mathbf{C}_2 + \mathbf{C}_3 \\ \left[(1 + x^2 + x^3)(1 + x^3 + x^5 + x^6)\right]_7 &= \mathbf{0}.\end{aligned}$$

❒

The key to the design and analysis of cyclic codes is the *generator polynomial*.

`Definition.` If $\mathcal{C}$ is a cyclic code, a nonzero polynomial of lowest degree in $\mathcal{C}$ is called a *generator polynomial* for $\mathcal{C}$. The symbol $g(x)$ is usually reserved to denote a generator polynomial.

Example 8.6. In the code of Example 8.2, the codeword $\mathbf{C}_1$, viewed as a polynomial, has lowest degree among all nonzero codewords, and so $g(x) = 1 + x^2 + x^3 + x^4$ is the generator polynomial for this code. In the code of Example 8.3, there are two lowest-degree polynomials, viz. $\mathbf{C}_1 = 2x^2 + 1$ and $2\mathbf{C}_1 = x^2 + 2$. The first part of the following lemma shows that the generator polynomial for a cyclic code is always unique up to multiplication by scalars, and so we are justified in referring to *the* generator polynomial of a cyclic code,

and assuming it is monic. Thus for the code of Example 8.3, normally one would say that $g(x) = x^2 + 2$ is the generator polynomial. ❒

`Lemma 2.` *Suppose that $\mathcal{C}$ is a cyclic code with generator polynomial $g(x)$.*
(a) If $g'(x)$ is another generator polynomial, then $g'(x) = \lambda g(x)$, for some nonzero element $\lambda \in F$.
(b) If $P(x)$ is a polynomial such that $[P(x)]_n$ is a codeword, then $g(x)$ divides $P(x)$.

`Proof:` To prove (a), suppose that $g(x) = g_r x^r + \cdots + g_0$ and $g'(x) = g'_r x^r + \cdots + g'_0$, with $g_r \neq 0$ and $g'_r \neq 0$. Then if $\lambda = g'_r/g_r$ the polynomial $g''(x) = g'(x) - \lambda g(x)$ has degree less than r and is in $\mathcal{C}$. But r is the lowest possible degree among *nonzero* codewords in $\mathcal{C}$, so that $g''(x) = 0$, i.e., $g'(x) = \lambda g(x)$.

To prove (b), let $Q(x)$ and $R(x)$ be the quotient and remainder obtained when $P(x)$ is divided by $g(x)$, i.e.,

$$P(x) = Q(x)g(x) + R(x), \tag{8.1}$$

with $\deg R < \deg g$. Reducing each of these polynomials mod $x^n - 1$, and using the fact that $\deg R < \deg g \leq n - 1$ (the latter inequality since $g(x)$ is a codeword), we obtain

$$R(x) = [P(x)]_n - [Q(x)g(x)]_n.$$

But $[P(x)]_n$ is a codeword by assumption, and $[Q(x)g(x)]_n$ is a codeword by Theorem 8.2. Thus, since $\mathcal{C}$ is linear, $R(x)$ is a codeword also. But $\deg R < \deg g$, and $g(x)$ is a nonzero codeword of least degree, so $R(x) = 0$, and (8.1) becomes

$$P(x) = Q(x)g(x),$$

which shows that $g(x)$ divides $P(x)$. ❒

We can now state and prove the main theorem about cyclic codes. It establishes a 1–1 correspondence between cyclic codes of length n and monic divisors of $x^n - 1$.

`Theorem 8.3.`
(a) *If $\mathcal{C}$ is an (n, k) cyclic code over F, then its generator polynomial is a divisor of $x^n - 1$. Furthermore, the vector $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1})$ is in the code if and only if the corresponding generating function $C(x) = C_0 + C_1 x + \cdots + C_{n-1}x^{n-1}$ is divisible by $g(x)$. If k denotes the dimension of $\mathcal{C}$, then $k = n - \deg g(x)$.*
(b) *Conversely, if $g(x)$ is a divisor of $x^n - 1$, then there is an (n, k) cyclic code with $g(x)$ as its generator polynomial and $k = n - \deg g(x)$, namely, the set of all vectors $(C_0, C_1, \ldots, C_{n-1})$ whose generating functions are divisible by $g(x)$.*

`Proof of (a):` First, let $P(x) = x^n - 1$ in Lemma 2(b); $[P(x)]_n = 0$, which is of course a codeword, and so $g(x)$ divides $x^n - 1$. Next, by Theorem 8.2, any vector of length n whose generating function is a multiple of $g(x)$ is a codeword. Conversely, if $C(x) = C_0 + C_1 x + \cdots + C_n x^{n-1}$ is a codeword, then $[C(x)]_n = C(x)$ and so Lemma 2 implies that $g(x)$ divides $C(x)$. Finally, the assertion about the degree of $g(x)$ follows since $C(x) = C_0 + C_1 x + \cdots + C_{n-1}x^{n-1}$ is a multiple of $g(x)$ if and only if $C(x) = g(x)I(x)$, where $\deg I \leq n - 1 - \deg g$.
`Proof of (b):` Suppose that $g(x)$ is a divisor of $x^n - 1$. Then $C(x) = (C_0, C_1, \ldots, C_{n-1})$ is a multiple of $g(x)$ if and only if $C(x) = g(x)I(x)$, where $\deg g + \deg I \leq n - 1$. Thus the set of all such words is an (n, k) linear code, where $k = n - \deg g$. To show that this code is cyclic, we must show that the right cyclic shift of any codeword is also a codeword. Thus let $I(x)g(x)$ be any codeword; by Theorem 8.1, its right cyclic shift is $[xI(x)g(x)]_n$. But since $g(x)$ divides $x^n - 1$, we have

$$\begin{aligned} [xI(x)g(x)]_n \bmod g(x) &= [xI(x)g(x)] \bmod g(x) \qquad \text{(by Lemma 1 (v))} \\ &= 0 \qquad \text{(by Lemma 1(ii))}, \end{aligned}$$

which proves that $[xI(x)g(x)]_n$ is a multiple of $g(x)$, so the code is indeed a cyclic code. ❒

Theorem 8.3 shows the importance of the generator polynomial of a cyclic code. A closely-related, and equally important, polynomial is the *parity-check* polynomial for a cyclic code, which is denoted by $h(x)$, and defined by

$$h(x) = \frac{x^n - 1}{g(x)}.$$

The following corollaries to Theorem 8.3 give explicit descriptions of generator and parity-check matrices for a cyclic code, in terms of $g(x)$ and $h(x)$.

`Corollary 1.` If $\mathcal{C}$ is an (n, k) cyclic code with generator polynomial $g(x) = g_0 + g_1 x + \cdots + g_r x^r$ (with $r = n - k$), and parity-check polynomial $h(x) = h_0 + h_1 x + \cdots + h_k x^k$, then the following matrices are generator and parity-check matrices for $\mathcal{C}$:

$$G_1 = \begin{bmatrix} g_0 & g_1 & \cdots & \cdots & g_r & 0 & \cdots & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & \cdots & g_r & 0 & \cdots & 0 \\ \vdots & & & & & & & & \vdots \\ 0 & \cdots & \cdots & 0 & g_0 & g_1 & \cdots & \cdots & g_r \end{bmatrix} = \begin{bmatrix} g(x) \\ xg(x) \\ \vdots \\ x^{k-1}g(x) \end{bmatrix}$$

$$H_1 = \begin{bmatrix} h_k & h_{k-1} & \cdots & \cdots & h_0 & 0 & \cdots & \cdots & 0 \\ 0 & h_k & h_{k-1} & \cdots & \cdots & h_0 & 0 & \cdots & 0 \\ \vdots & & & & & & & & \vdots \\ 0 & \cdots & \cdots & 0 & h_k & h_{k-1} & \cdots & \cdots & h_0 \end{bmatrix} = \begin{bmatrix} \tilde{h}(x) \\ x\tilde{h}(x) \\ \vdots \\ x^{r-1}\tilde{h}(x) \end{bmatrix},$$

where $\tilde{h}(x) = h_k + h_{k-1}x + \cdots + h_0 x^k$ is $h(x)$'s "reciprocal" polynomial. Furthermore, if the vector $\mathbf{I} = (I_0, I_1, \ldots, I_{k-1})$ is encoded as $\mathbf{C} = \mathbf{I}G_1$ (cf. Eq. (7.1)), then the generating functions $I(x) = I_0 + I_1 x + \cdots + I_{k-1}x^{k-1}$ and $C(x) = C_0 + C_1 x + \cdots + C_{n-1}x^{n-1}$ are related by

$$C(x) = I(x)g(x).$$

`Proof:` The ith row of G_1, by definition, is $x^i g(x)$, for $i = 0, 1, \ldots, k-1$. Each of these k vectors is in $\mathcal{C}$ by Theorem 8.2, and they are linearly independent since each row has a different degree. Since $\mathcal{C}$ is k-dimensional, it follows that G_1 is a generator matrix for $\mathcal{C}$. Furthermore, the vector $\mathbf{I}G_1$, where $\mathbf{I} = (I_0, I_1, \ldots, I_{k-1})$, has generating function $I_0 g(x) + I_1 x g(x) + \cdots + I_{n-1}x^{n-1}g(x) = I(x)g(x)$.

To prove that H_1 is a parity-check matrix for $\mathcal{C}$, note that the inner product of the ith row of G_1 and the jth row of H_1 is the coefficient of x^{k-i+j} in the product $g(x)h(x)$. But $g(x)h(x) = x^n - 1$, and since the index $k - i + j$ ranges from 1 (when $i = k-1$ and $j = 0$) to $n-1$ (when $i = 0$ and $j = r-1$), each of these inner products is zero. Thus each row of H_1 is in the nullspace of $\mathcal{C}$; but the nullspace of $\mathcal{C}$ is r-dimensional, and H_1 has r linearly independent rows, so H_1 is in fact a parity-check matrix for $\mathcal{C}$. ❒

The matrices in Corollary 1 are sometimes useful, but more often, the "systematic" matrices in the following Corollary 2 are used.

`Corollary 2.` Let $\mathcal{C}$ be an (n, k) cyclic code with generator polynomial $g(x)$. For $i = 0, 1, \ldots, k-1$, let $G_{2,i}$ be the length n vector whose generating function is $G_{2,i}(x) = x^{r+i} - x^{r+i} \bmod g(x)$. Then the $k \times n$ matrix

$$G_2 = \begin{bmatrix} G_{2,0} \\ G_{2,1} \\ \vdots \\ G_{2,k-1} \end{bmatrix}$$

is a generator matrix for $\mathcal{C}$. Similarly, if $H_{2,j}$ is the length r vector whose generating function is $H_{2,j}(x) = x^j \bmod g(x)$, then the $r \times n$ matrix

$$H_2 = \left[H_{2,0}^T, H_{2,1}^T, \ldots, H_{2,n-1}^T\right]$$

is a parity-check matrix for $\mathcal{C}$. Furthermore, if the vector $\mathbf{I} = (I_0, I_1, \ldots, I_{k-1})$ is encoded as $\mathbf{C} = \mathbf{I}G_2$, then the generating functions $I(x)$ and $C(x)$ are related by

$$C(x) = x^r I(x) - [x^r I(x)] \bmod g(x).$$

Also, if the syndrome of the vector $\mathbf{R} = (R_0, R_1, \ldots, R_{n-1})$ is calculated as $\mathbf{S}^T = H_2\mathbf{R}^T$, then the generating functions $R(x)$ and $S(x)$ are related by

$$S(x) = R(x) \bmod g(x).$$

`Proof:` The ith row of G_2 is a multiple of $g(x)$ and so a codeword of $\mathcal{C}$, because

$$\begin{aligned}[x^{r+i} - x^{r+i} \bmod g(x)] \bmod g(x) &= x^{r+i} \bmod g(x) - x^{r+i} \bmod g(x) \\ &= 0\end{aligned}$$

by Lemma 1 parts (iii) and (i). Since G_2 has k linearly independent rows (the last k columns of G_2 form a $k \times k$ identity matrix), each of which is a codeword, G_2 is a generator matrix of $\mathcal{C}$. Furthermore, the vector $\mathbf{I}G_2$, where $\mathbf{I} = (I_0, I_1, \ldots, I_{k-1})$, has generating function $I_0(x^r - x^r \bmod g(x)) + I_1(x^{r+1} - x^{r+1} \bmod g(x)) + \cdots + I_{k-1}(x^{r+k-1} - x^{r+k-1} \bmod g(x)) = x^r I(x) - [x^r I(x)] \bmod g(x)$.

To prove the assertion about H_2, note first that H_2 has rank r (its first r columns form an $r \times r$ identity matrix). Furthermore, if $\mathbf{R} = (R_0, R_1, \ldots, R_{n-1})$ is a received word, then $H_2\mathbf{R}^T$ is an $r \times 1$ column vector with generating function $\sum_{j=0}^{n-1} R_j(x^j \bmod g(x)) = (\sum_{j=0}^{n-1} R_j x^j) \bmod g(x) = R(x) \bmod g(x)$. If $R(x)$ is a codeword, then $R(x) \bmod g(x) = 0$ by Theorem 8.3(a), and so $H_2\mathbf{R}^T = \mathbf{0}$ for any codeword. Thus shows that H_2 is a parity-check matrix for $\mathcal{C}$. On the other hand, if $\mathbf{R}$ is an arbitrary received vector, its syndrome with respect to H_2 is $\mathbf{S} = H_2\mathbf{R}^T$, which, as we have just seen, has generating function $S(x) = R(x) \bmod g(x)$, as asserted. ❒

`Note`: The form of the syndrome relative to the parity-check matrix H_2 as described in Corollary 2 is so natural that it is common to refer to it simply as the *remainder syndrome* of the vector $\mathbf{R}$ with respect to the cyclic code with generator polynomial $g(x)$. ❒

Example 8.7. In Example 8.6 we saw that the generator polynomial for the $(7, 3)$ cyclic code of Example 8.2 is $g(x) = x^4+x^3+x^2+1$. The corresponding parity-check polynomial is then $h(x) = (x^7+1)/(x^4+x^3+x^2+1) = x^3 + x^2 + 1$. The eight codewords, as multiples of $g(x)$, are:

$$\begin{aligned}\mathbf{C}_0 &= 0 \cdot g(x) \\ \mathbf{C}_1 &= 1 \cdot g(x) \\ \mathbf{C}_2 &= x \cdot g(x) \\ \mathbf{C}_3 &= x^2 \cdot g(x) \\ \mathbf{C}_4 &= (1 + x) \cdot g(x) \\ \mathbf{C}_5 &= (1 + x^2) \cdot g(x) \\ \mathbf{C}_6 &= (x + x^2) \cdot g(x) \\ \mathbf{C}_7 &= (1 + x + x^2) \cdot g(x)\end{aligned}$$

The generator and parity-check matrices described in Corollary 1 are

$$G_1 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} g(x) \\ xg(x) \\ x^2 g(x) \end{bmatrix}$$

$$H_1 = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \tilde{h}(x) \\ x\tilde{h}(x) \\ x^2\tilde{h}(x) \\ x^3\tilde{h}(x) \end{bmatrix}$$

The generator and parity-check matrices described in Corollary 2 are

$$G_2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x^4 - x^4 \bmod g(x) \\ x^5 - x^5 \bmod g(x) \\ x^6 - x^6 \bmod g(x) \end{bmatrix}$$

$$H_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} = [1, x, x^2, x^3, x^4 \bmod g, x^5 \bmod g, x^6 \bmod g]$$

Note the appearance of a 3×3 identity matrix on the right side of G_2, and the 4×4 identity matrix on the left side of H_2. This is what makes these matrices "systematic." If we wish to obtain generator and parity-check matrices of the form $G = [I_k A]$ and $H = [-A^T I_{n-k}]$, as promised in Theorem 7.1, we can use the cyclic property of the code, cyclically shifting the rows of G_2 three places to the right, and those of H_2 three places to the right, thus obtaining

$$G_3 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$H_3 = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(For another way to describe G_3 and H_3, see Problem 8.9.) If we encode the vector $\mathbf{I} = [101]$ using G_1, the resulting codeword is $(1+x^2)(1+x^2+x^3+x^4) = 1+x^3+x^5+x^6 = [1001011]$. On the other hand, if we use G_2 instead, the codeword is $x^4(1+x^2) - [x^4(1+x^2)] \bmod (x^4+x^3+x^2+1) = x^6+x^4+x+1 = [1100101]$. The syndrome of the vector $\mathbf{R} = [1010011]$ with respect to H_1 is $[1101]$. (For an efficient way to make this computation, see Problem 8.10). On the other hand, if we use H_2 instead, we find that the "remainder" syndrome is $R(x) \bmod g(x) = (1 + x^2 + x^5 + x^6) \bmod (1 + x^2 + x^3 + x^4) = x^3 + x^2$, i.e., $\mathbf{S} = [0011]$. ❒

Example 8.8. In Example 8.6 we saw that the generator polynomial for the $(4, 2)$ $GF(3)$-cyclic code of Example 8.3 is $g(x) = x^2 + 2 = x^2 - 1$. Then the parity-check polynomial is $(x^4 - 1)/(x^2 - 1) = x^2 + 1$. The generator and parity-check matrices of Corollary 1 to Theorem 8.3 are then

$$G_1 = \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{bmatrix}$$

$$H_1 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

and

$$G_2 = \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

It is only a coincidence that $G_1 = G_2$ and $H_1 = H_2$ in this case. But see Problem 8.21. ❒

According to Theorem 8.3, the cyclic codes of length n over a given field F are in one-to-one correspondence with the monic divisors of $x^n - 1$ over F. Plainly then, in order to study cyclic codes over F it is useful to know how to factor $x^n - 1$ over F. Although we shall not make a systematic study of the factorization of $x^n - 1$ here, in Table 8.1. we give the factorization of $x^n - 1$ over $GF(2)$, for $1 \le n \le 31$. This table contains all the information necessary for a study of cyclic codes of length ≤ 31 over $GF(2)$.

n	$x^n + 1 =$
1	$(x+1)$
2	$(x+1)^2$
3	$(x+1)(x^2+x+1)$
4	$(x+1)^4$
5	$(x+1)(x^4+x^3+x^2+x+1)$
6	$(x+1)^2(x^2+x+1)^2$
7	$(x+1)(x^3+x+1)(x^3+x^2+1)$
8	$(x+1)^8$
9	$(x+1)(x^2+x+1)(x^6+x^3+1)$
10	$(x+1)^2(x^4+x^3+x^2+x+1)^2$
11	$(x+1)(x^{10}+x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1)$
12	$(x+1)^4(x^2+x+1)^4$
13	$(x+1)(x^{12}+x^{11}+x^{10}+x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1)$
14	$(x+1)^2(x^3+x+1)^2(x^3+x^2+1)^2$
15	$(x+1)(x^2+x+1)(x^4+x+1)(x^4+x^3+1)(x^4+x^3+x^2+x+1)$
16	$(x+1)^{16}$
17	$(x+1)(x^8+x^5+x^4+x^3+1)(x^8+x^7+x^6+x^4+x^2+x+1)$
18	$(x+1)^2(x^2+x+1)^2(x^6+x^3+1)^2$
19	$(x+1)(x^{18}+x^{17}+x^{16}+\cdots+x+1)$
20	$(x+1)^4(x^4+x^3+x^2+x+1)^4$
21	$(x+1)(x^2+x+1)(x^3+x+1)(x^3+x^2+1)(x^6+x^4+x^2+x+1)(x^6+x^5+x^4+x^2+1)$
22	$(x+1)^2(x^{10}+x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1)^2$
23	$(x+1)(x^{11}+x^9+x^7+x^6+x^5+x+1)(x^{11}+x^{10}+x^6+x^5+x^4+x^2+1)$
24	$(x+1)^8(x^2+x+1)^8$
25	$(x+1)(x^4+x^3+x^2+x+1)(x^{20}+x^{15}+x^{10}+x^5+1)$
26	$(x+1)^2(x^{12}+x^{11}+x^{10}+x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1)^2$
27	$(x+1)(x^2+x+1)(x^6+x^3+1)(x^{18}+x^9+1)$
28	$(x+1)^4(x^3+x+1)^4(x^3+x^2+1)^4$
29	$(x+1)(x^{28}+x^{27}+\cdots+x+1)$
30	$(x+1)^2(x^2+x+1)^2(x^4+x+1)^2(x^4+x^3+1)^2(x^4+x^3+x^2+x+1)^2$
31	$(x+1)(x^5+x^2+1)(x^5+x^3+1)(x^5+x^3+x^2+x+1)$ $\cdot(x^5+x^4+x^2+x+1)(x^5+x^4+x^3+x+1)(x^5+x^4+x^3+x^2+1)$

Table 8.1. Factorization of $x^n - 1 (= x^n + 1)$ into powers of irreducible polynomials over $GF(2)$, for $1 \le n \le 31$.

Example 8.9. Let's use Table 8.1 to list all possible binary cyclic codes of length 7. According to Table 8.1, $x^7 - 1$ factors into three distinct irreducible factors: $x^7 - 1 = (x+1)(x^3+x+1)(x^3+x^2+1)$, and so $x^7 - 1$ has $2^3 = 8$ distinct divisors $g(x)$. This yields the following list:

(n, k)	$g(x)$	comment
$(7, 7)$	1	no parity code
$(7, 6)$	$x+1$	overall parity-check code
$(7, 4)$	x^3+x+1	Hamming code
$(7, 4)$	x^3+x^2+1	Hamming code
$(7, 3)$	$(x+1)(x^3+x+1)$	Example 8.2 code
$(7, 3)$	$(x+1)(x^3+x^2+1)$	Example 8.2 code "reversed"
$(7, 1)$	$(x^3+x+1)(x^3+x^2+1)$	repetition code
$(7, 0)$	x^7+1	no information code

For example, we assert that the $(7,4)$ cyclic code with $g(x) = x^3 + x + 1$ is a Hamming code. To see why this is so, we note that the parity-check polynomial is $(x^7+1)/(x^3+x+1) = x^4 + x^2 + x + 1$. Thus by Corollary 1 to Theorem 8.3, a parity-check matrix for this code is

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

The columns of H are nonzero and distinct, and so by the definition in Section 7.4, this is indeed a Hamming code. We leave the investigation of the other "comments" as Problem 8.23.

❒

In Example 8.9, we saw that $g(x) = x^3 + x + 1$ generates a $(7,4)$ Hamming code. But according to Table 8.1, $g(x)$ divides not only $x^7 - 1$, but also $x^{14} - 1$; in fact, $x^3 + x + 1 \,|\, x^n - 1$ for any n which is a multiple of 7 (see Problem 8.24). Thus by Theorem 8.3(b), $g(x)$ generates a whole family of cyclic codes, with parameters $(7,4), (14,11), (21,18), \ldots$. However, all of these codes except the first contain a vector whose generating function is $x^7 - 1$, and so have minimum weight equal to 2. Hence these codes are not interesting for error correction; we shall call them *improper* cyclic codes. (See Problem 8.26. Also see Problem 8.27 for a discussion of cyclic codes for which the *parity-check* polynomial has period less than n.) In the rest of this chapter, when we refer to an (n,k) cyclic code with generator polynomial $g(x)$, we shall assume it is a *proper* cyclic code, i.e., one for which n is the *smallest* positive integer such that $g(x) \,|\, x^n - 1$. This integer is sometimes called the *period* of $g(x)$, since it is the period of the sequence $[x^i \bmod g(x)]_{i \geq 0}$. With this convention established, we are ready to continue our study of cyclic codes.

8.2. Shift-Register Encoders for Cyclic Codes.

Now that we know by Theorem 8.3 that every cyclic code is characterized by its generator polynomial, we can begin to see why cyclic codes are much easier to implement than arbitrary linear codes. In this section we shall show, in fact, that every cyclic code can be encoded with a simple finite-state machine called a **shift-register encoder**.

Recall that an encoding algorithm for an (n,k) linear code, cyclic or not, is a rule for mapping the set of length-k information sequences $(I_0, I_1, \ldots, I_{k-1})$ onto the set of length-n codewords $(C_0, C_1, \ldots, C_{n-1})$, or equivalently for mapping the information polynomials $I(x) = I_0 + I_1 x + \cdots + I_{k-1}x^{k-1}$ onto the set of code polynomials $C(x) = C_0 + C_1 x + \cdots + C_{n-1}x^{n-1}$. If the code is cyclic, Theorem 8.3 tells us that $C(x)$ is a codeword if and only if $C(x)$ is a multiple of $g(x)$, and so, as we saw in Corollary 1 to Theorem 8.3, one way to change an information polynomial $I(x)$ into a code polynomial is simply to multiply $I(x)$ by $g(x)$:

$$I(x) \rightarrow I(x)g(x). \tag{8.2}$$

It turns out that polynomial multiplication is easy to implement using something called *shift-register logic*, and we shall now make a brief study of this subject.

Figure 8.1 is an abstract representation of a machine capable of multiplying an *arbitrary* polynomial $I(x)$ by a *fixed* polynomial $g(x)$. Before proving that the circuit works, we had better explain its various components.

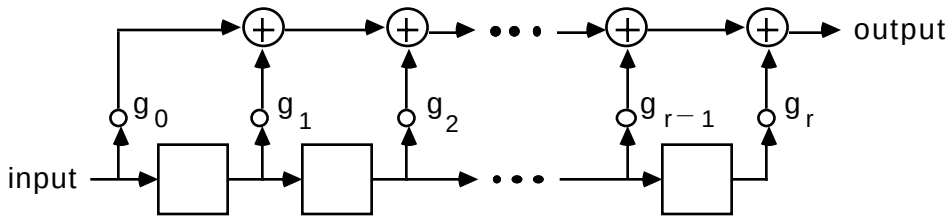
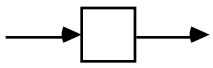


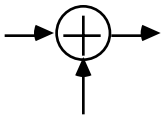
Figure 8.1. A shift register circuit for multiplying by $g(x) = g_0 + g_1 x + \cdots + g_r x^r$. Alternatively, this is an encoder for an (n,k) cyclic code with generator polynomial $g(x)$, with $r = n - k$.

The circuit in Figure 8.1 is build by connecting together three types of components: *flip-flops*, *adders*, and *constant multipliers*. The most important of these is the flip-flop, sometimes called a *delay* element:



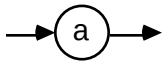
a flip flop

A flip-flop is a device which can store one element from the field F. Not pictured in our simplified circuit diagrams, but an important part of them, is an external clock which generates a timing signal ("tick") every t_0 seconds. When the clock ticks, the contents of the flip-flop are shifted out of the flip-flop in the direction of the arrow, through the circuit, until the next flip-flop is reached. Here the signal stops until the next tick. The contents of the shift register can be modified by the circuit between successive flip-flops, which is where the other two logic elements come in. The two-input *adder*, which looks like this:



an adder

is a device that computes the sum of its two input signals. (The circuit is always arranged in such a way that after every tick each input to the adder receives exactly one signal, so that the adder's output is always unambiguous.) Finally we come to the simplest circuit element, the *constant multiplier:*



constant multiplier

This device simply multiplies its input by the constant a, with no delay.

Now let's see if we can understand why the circuit of Figure 8.1 can be used for polynomial multiplication. We first write out in detail the formulas for the coefficients of the product $C(x) = C_0 + C_1 x + \cdots + C_{n-1}x^{n-1}$, where $C(x) = I(x)g(x)$:

$$\begin{aligned} C_0 &= I_0 g_0 \\ C_1 &= I_0 g_1 + I_1 g_0 \\ C_2 &= I_0 g_2 + I_1 g_1 + I_2 g_0 \\ &\vdots \\ C_j &= I_0 g_j + I_1 g_{j-1} + \cdots + I_j g_0 \\ &\vdots \\ C_{n-1} &= I_{k-1} g_r. \end{aligned} \tag{8.3}$$

The flip-flops of Fig. 8.1 are initially filled with 0's. We then feed in the sequence $I_0, \ldots, I_{k-1}$, followed by $r = n - k$ 0's, one bit every tick, to the shift register via the input arrow. Let us now study the behavior of

the circuit at each tick of the clock:

Tick 0:	Input: I_0
	Shift-register contents: $[0, 0, 0, \ldots, 0]$
	Output: $I_0 g_0$
Tick 1:	Input I_1
	Shift-register contents: $[I_0, 0, 0, \ldots, 0]$
	Output: $I_0 g_1 + I_1 g_0$
	$\vdots$
Tick j:	Input: I_j
	Shift-register contents: $[I_{j-1}, \cdots, I_1, I_0, \ldots, 0]$
	Output: $I_0 g_j + I_1 g_{j-1} + \ldots + I_j g_0$
	$\vdots$
Tick n-1:	Input: 0
	Shift-register contents: $[0, \ldots, 0, I_{k-1}]$
	Output: $I_{k-1} g_r$

Hence if the circuit in Figure 8.1 is initialized by placing r 0's in the flip-flops, and given the n-symbol input sequence $(I_0, I_1, \ldots, I_{k-1}, 0, \ldots, 0)$, the output sequence will indeed be $(C_0, C_1, \ldots, C_{n-1})$, where the C_j's are defined by Eq. (8.3), and so this circuit can be used as an encoder for the (n, k) cyclic code with generator polynomial $g(x)$.

`Comment.` The preceeding description is for a "generic" shift-register circuit, capable of performing arithmetic in an arbitrary field F. The required devices (flip-flop, adders, multipliers) are not "off-the-shelf" items, except when the field is $GF(2)$. In this case, however, these devices are very simple indeed:

flip-flop	=	D flip-flop
adder	=	`xor` gate
0-multiplier	=	no connection
1-multiplier	=	direct wire

Example 8.10. The generator polynomial for the $(7, 3)$ binary cyclic code of Example 8.2 is $g(x) = x^4 + x^3 + x^2 + 1$, as we saw in Example 8.6. The corresponding shift-register encoder is shown in Figure 8.2. If (I_0, I_1, I_2) are the three information bits to be encoded, the four flip-flops should be initialized to 0, and the seven bits $(I_0, I_1, I_2, 0, 0, 0, 0)$ should be used as inputs to the encoder. These seven input bits will then produce seven output bits $(C_0, C_1, C_2, C_3, C_4, C_5, C_6)$, which is the corresponding codeword. For example, if the information bits are $(1, 0, 0)$, the corresponding codeword will be the "impulse response" $(1, 0, 1, 1, 1, 0, 0)$, which is in fact the generator polynomial $1 + x^2 + x^3 + x^4$ (and also the codeword $\mathbf{C}_1$ from Example 8.2). ❒

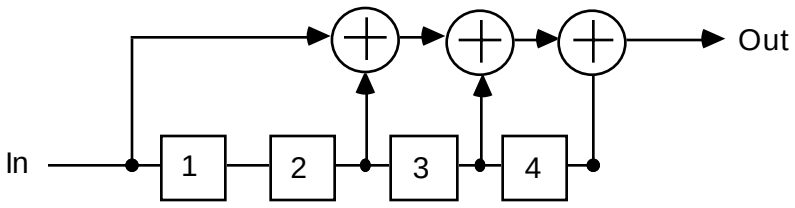
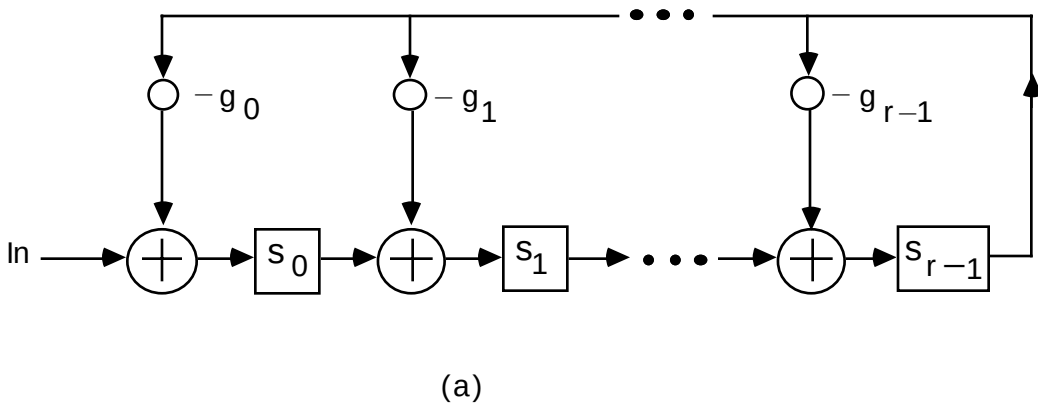


Figure 8.2. Nonsystematic encoder for the $(7, 3)$ cyclic code with generator polynomial $g(x) = 1 + x^2 + x^3 + x^4$.

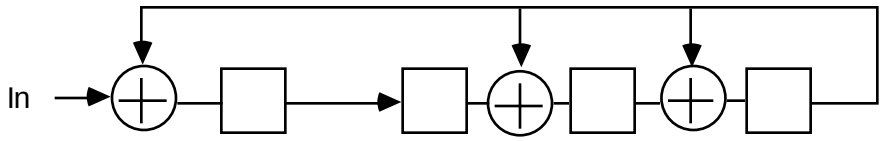
The encoders of Figures 8.1 and 8.2 could hardly be simpler, but they are unfortunately not *systematic*, i.e., the information bits $(I_0, I_1, \ldots, I_{k-1})$ do not appear unchanged in the corresponding codeword $(C_0, C_1, \ldots, C_{n-1})$. It is possible, however, to design a systematic shift-register encoder for any cyclic code which is only slightly more complex than the nonsystematic one. The idea is to use the result of Theorem 8.3, Corollary 2, which says that if $I(x)$ is an information polynomial, then

$$I(x) \to x^r I(x) - [x^r I(x)] \bmod g(x) \tag{8.4}$$

is a systematic encoding rule for a cyclic code with generator polynomial $g(x)$. The central component in such an encoder will be a "mod $g(x)$" circuit, where $g(x)$ is the code's generator polynomial. Figure 8.3(a) shows such a circuit, if $g(x) = x^r + g_{r-1}x^{r-1} + \cdots + g_0$.



(a)



(b)

Figure 8.3. (a). A general "mod $g(x)$" shift-register circuit, where $g(x) = x^r + g_{r-1}x^{r-1} + \cdots + g_0$. (b) A specific "mod $x^4 + x^3 + x^2 + 1$" shift-register, where all arithmetic is mod 2.

In order to see why the circuit of Figure 8.3(a) is able to perform a "mod $g(x)$" calculation, we focus on the shift-register contents $[s_0, s_1, \ldots, s_{r-1}]$, which we call the *state vector* of the machine, and on the corresponding generating function $S(x) = s_0 + s_1 x + \cdots + s_{r-1}x^{r-1}$, which we call the *state polynomial*. We begin with a lemma which explains how the state changes in response to an input.

`Lemma 3.` *If the circuit in Figure 8.3(a) has state polynomial $S(x)$, and the input is s, then the next state polynomial will be*

$$S'(x) = (s + xS(x)) \bmod g(x).$$

`Proof:` If the present state vector is $\mathbf{S} = [s_0, \ldots, s_{r-1}]$ and the input is s, then by our rules describing the building blocks of the circuit, the next state vector will be, by definition,

$$\mathbf{S}' = [s - g_0 s_{r-1}, s_0 - g_1 s_{r-1}, \ldots, s_{r-2} - g_{r-1} s_{r-1}] .$$

Hence the next state polynomial is

$$\begin{aligned} S'(x) &= s + s_0 x + \cdots + s_{r-2} x^{r-1} - s_{r-1}(g_0 + g_1 x + \ldots + g_{r-1} x^{r-1}) \\ &= s + xS(x) - s_{r-1} g(x). \end{aligned}$$

Thus $S'(x)$ is a polynomial of degree $< \deg g(x)$ such that $(s + xS(x)) - S'(x)$ is a multiple of $g(x)$, and so $S'(x) = (s + xS(x)) \bmod g(x)$, by Lemma 1 (iii). ❒

`Theorem 8.4.` *If the circuit of Figure 8.3(a) is initialized by setting $s_0 = s_1 = \ldots = s_{r-1} = 0$, and then given the input sequence $a_0, a_1, a_2, \ldots$, (a_0 is input at the 0th tick, a_1 at the first tick, etc.), then after the tth tick, the state polynomial will be*

$$S_t(x) = \sum_{j=0}^{t} a_j x^{t-j} \bmod g(x).$$

`Proof:` We use induction on t. For $t = 0, 1, \ldots, r-1$, the state vector is $\mathbf{S}_t = [a_t, a_{t-1}, \ldots, a_0, \overbrace{0, \ldots, 0}^{r-t-1}]$, and the statement of the theorem is a tautology. Assuming then that the theorem is true for $S_t(x)$, we consider $S_{t+1}(x)$:

$$\begin{aligned} S_{t+1}(x) &= a_{t+1} + xS_t(x) \bmod g(x) \qquad \text{(Lemma 3)} \\ &= a_{t+1} + x(\sum_{j=0}^{t} a_j x^{t-j} \bmod g(x)) \bmod g(x) \qquad \text{(induction hypothesis)} \\ &= a_{t+1} + (\sum_{j=0}^{t} a_j x^{t+1-j}) \bmod g(x) \qquad \text{(Lemma 1(iv))} \\ &= \sum_{j=0}^{t+1} a_j x^{t+1-j} \bmod g(x) \qquad \text{(Lemma 1(i) and (iii))}, \end{aligned}$$

as asserted. ❒

Theorem 8.4 explains why the circuit of Figure 8.3(a) is called a "mod $g(x)$ circuit." Our next goal is to use it, together with the encoding rule (8.4), to build a systematic shift-register encoder for a cyclic code with generator polynomial $g(x)$. The encoding rule (8.4) requires us to compute

$$[x^r I(x)] \bmod g(x) = \sum_{j=r}^{n-1} I_{j-r} x^j \bmod g(x).$$

According to Theorem 8.4, to compute this, we can give the mod $g(x)$ circuit the n-symbol input sequence $(I_{k-1}, I_{k-2}, \ldots, I_0, \overbrace{0, 0, \ldots, 0}^{r})$. However, since a systematic encoder could output the k information symbols unchanged on ticks 0 through $k-1$, it would like to start outputting the parity-checks, i.e., the coefficients of $[x^r I(x)] \bmod g(x)$, on the kth tick, but will have to wait for r more ticks before $[x^r I(x)] \bmod g(x)$ is ready. Fortunately, it is possible to avoid this r tick "down time" by using the circuit shown in Figure 8.4(a).

In the circuit of Figure 8.4(a), the input bits are fed into the *right* side of the shift-register, rather then the left, as in Figure 8.3. The result is that if the input stream is $a_0, a_1, \ldots$, after the tth tick, the shift-register's state polynomial will be $\sum_{j=0}^{t} a_j x^{r+t-j} \bmod g(x)$, rather than $\sum_{j=0}^{t} a_j x^{t-j} \bmod g(x)$. (To verify this assertion, do Problem 8.31). Thus if the encoder inputs the k information symbols $I_{k-1}, I_{k-2}, \ldots, I_0$ to the shift-register of Figure 8.4, the coefficients of $[x^r I(x)] \bmod g(x)$ will be ready, starting with the kth tick, as desired. A complete systematic encoder based on the shift-register of Figure 8.4(a) is shown in Figure 8.5(a). Note that as compared to the nonsystematic encoder of Figure 8.1, in the encoder of Figure 8.5(a), the information symbols are sent to the encoder in the *reverse* order, viz., $I_{k-1}, \ldots, I_0$, and the codeword components are also sent to the channel in the reverse order, viz., $C_{n-1}, C_{n-2}, \ldots, C_1, C_0$.

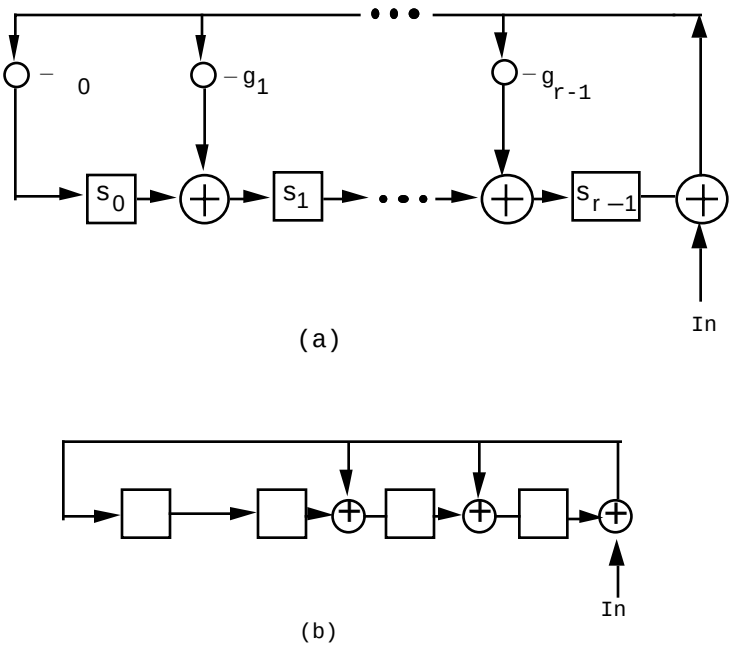


Figure 8.4. (a) A general shift register needed to build an n-tick systematic encoder for a cyclic code with generator polynomial $g(x) = g_0 + g_1 x + \cdots + g_r x^r$. (b) Special case $g(x) = x^4 + x^3 + x^2 + 1$.

Example 8.11. If we apply the general construction described in Figure 8.5(a) to the special case of $g(x) = x^4 + x^3 + x^2 + 1$, over the field $GF(2)$, we get the encoder depicted in Figure 8.5(b). If, for example, the information sequence $(I_2, I_1, I_0) = (1, 1, 0)$, followed by $(0, 0, 0, 0)$ is used as the input sequence to the encoder circuit of Figure 8.5(b), the output will be $(1, 1, 0, 1, 0, 0, 1)$, as detailded in the following table.

tick	input	SR contents	output
0	1	1011	1
1	1	0101	1
2	0	1001	0
3	0	0100	1
4	0	0010	0
5	0	0001	0
6	0	0000	1

Recalling that the components appear in reverse order, we see that this codeword is the same as codeword $\mathbf{C}_1 + \mathbf{C}_3$ in Example 8.2. ❒

We will conclude this section with a brief discussion of a third type of shift-register encoder for a cyclic code, which is based on the code's *parity-check polynomial*. Recall from Section 8.1 that if $\mathcal{C}$ is an (n, k) cyclic code with generator polynomial $g(x)$, its parity-check polynomial $h(x) = h_0 + h_1 x + \cdots + h_k x^k$ is defined as

$$h(x) = \frac{x^n - 1}{g(x)}. \tag{8.5}$$

Since every codeword $C(x)$ is by Theorem 8.3 a multiple of $g(x)$, it follows from (8.5) that $C(x)h(x)$ is a multiple of $g(x)h(x) = x^n - 1$, i.e.,

$$[C(x)h(x)]_n = 0. \tag{8.6}$$

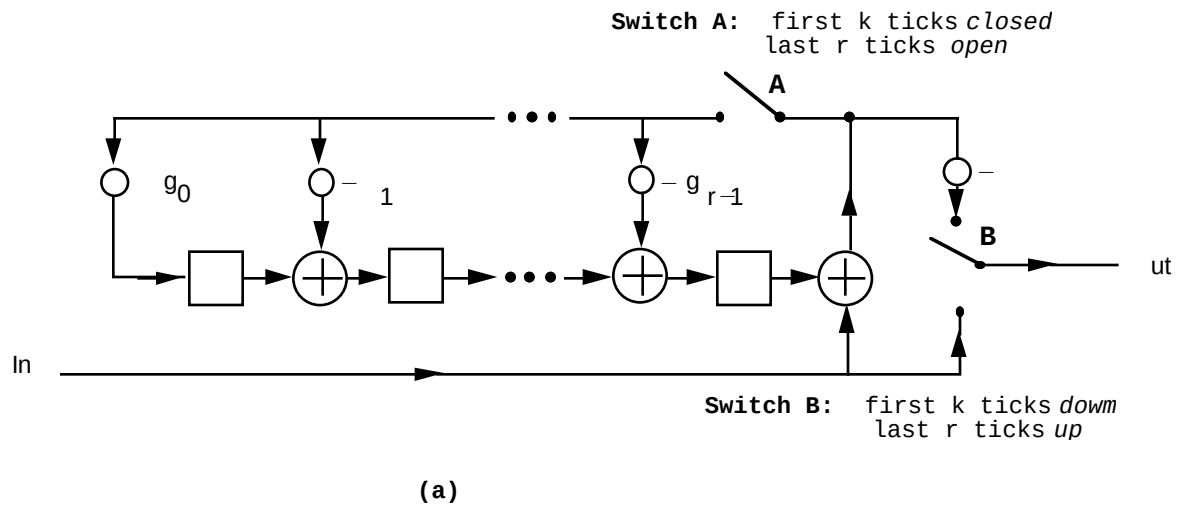


Figure 8.5. (a) An n-tick systematic encoder for a cyclic code with generator polynomial $g(x) = g_0 + g_1 x + \cdots + g_{r-1}x^{r-1} + x^r$. (b) Special case $g(x) = x^4 + x^3 + x^2 + 1$.

Equation (8.6) places strong conditions on the coefficients C_i of the codeword $C(x)$, as the following theorem shows.

`Theorem 8.5.` *For $i = 0, 1, \ldots, n-1$, we have*

$$\sum_{j=0}^{k} h_j C_{(i-j) \bmod n} = 0.$$

`Proof:` We have

$$\begin{aligned} h(x)C(x) &= \left(\sum_{j=0}^{k} h_j x^j\right)\left(\sum_{m=0}^{n-1} C_m x^m\right) \\ &= \sum_{j=0}^{k} h_j \left(\sum_{m=0}^{n-1} C_m x^{j+m}\right). \end{aligned}$$

If we reduce this expression $\bmod(x^n - 1)$, and use the fact that $x^i \bmod (x^n - 1) = x^{i \bmod n}$ (see Example 8.4), we find

$$[h(x)C(x)]_n = \sum_{j=0}^{k} h_j \left(\sum_{m=0}^{n-1} C_m x^{(j+m) \bmod n} \right). \tag{8.7}$$

Since by (8.6), $[h(x)C(x)]_n = 0$, it follows that each coefficient of $[h(x)C(n)]_n$ is zero as well. But from (8.7) it follows that for $i = 0, 1, \ldots, n-1$, the coefficient of x^i in $[h(x)C(n)]_n$ is

$$\sum_{j=0}^{k} h_j \cdot \text{coefficient of } x^i \text{ in } \left\{ \sum_{m=0}^{n-1} C_m x^{(j+m) \bmod n} \right\}.$$

If $m \in \{0, 1, \ldots, n-1\}$, the exponent $(j+m) \bmod n$ will equal i if and only if $m = (i-j) \bmod n$ (see Problem 8.4), and so the coefficient of x^i in $[h(x)C(x)]_n$ is

$$\sum_{j=0}^{k} h_j C_{(i-j) \bmod n},$$

which must therefore be zero. ❒

`Corollary.` If $h_0 = 1$, then for $i = k, k+1, \ldots, n-1$,

$$C_i = -\sum_{j=1}^{k} h_j C_{i-j}.$$

`Proof:` If $k \le i \le n-1$, then for $j = 0, 1 \ldots, k$ we have $(i-j) \bmod n = i - j$, and Theorem 8.5 becomes

$$\sum_{j=0}^{k} h_j C_{i-j} = 0.$$

If $h_0 = 1$, this last equation can be rearranged to give the result stated. ❒

The Corollary to Theorem 8.5 says that once the first k components $C_0, C_1, \ldots, C_{k-1}$ of a codeword are known, the remaining r components $C_k, C_{k+1}, \ldots, C_{n-1}$ can be computed by *linear recursion*, i.e. each new component is a fixed linear combination of the previous k components. This fact leads immediately to a shift-register encoder, because it is easy to design a shift register circuit that can implement a given linear recursion; see Figure 8.6.

Figure 8.6(a) shows a k-stage shift register circuit capable of generating any sequence $(S_0, S_1, S_2, \ldots)$ that satisfies the linear recursion $S_t = -\sum_{j=1}^{k} h_j S_{t-j}$, where $h_1, h_2, \ldots, h_k$ are fixed constants. The circuit must be initalized by loading the k flip-flops with the k initial values $S_0, S_1, \ldots, S_{k-1}$ (with S_0 occupying the rightmost flip-flop). Then, for $t \ge k$, if the shift-register contents are $(S_{t-1}, \ldots, S_{t-k})$, after the next tick, the shift-register contents will be $(S_t, S_{t-1} \cdots S_{t-k+1})$. Figure 8.6(b) shows the special case $h(x) = x^3 + x^2 + 1$, corresponding to the recursion $S_t = S_{t-2} + S_{t-3}$, for $t \ge 3$, over the binary field $GF(2)$.

It should be clear how to use the shift register circuit of Figure 8.6(a) to build a systematic encoder for a cyclic code with parity-check polynomial $h(x) = 1 + h_1 x + \cdots + h_k x^k$ (see Figure 8.7(a)). During the first k ticks the switch is in the "down" position and the k information symbols are simultaneously sent out to the channel, and loaded into the k-stage shift register. Then the switch is put in the "up" position, and during the remaining r ticks, the shift-register circuit calculates the remaining r codeword components via the recursion $C_i = -\sum_{j=1}^{k} h_j C_{i-j}$. Notice that the systematic "$h(x)$ encoder" of Figure 8.7 differs from the systematic "$g(x)$ encoder" of Figure 8.5 in two important ways: In the $g(x)$ encoder, the information symbols $(I_0, I_1, \ldots, I_{k-1})$ occupy codeword components $C_r, C_{r+1}, \ldots, C_{n-1}r$, and the codeword symbols are sent to the channel in the reverse order $C_{n-1}, C_{n-2}, \ldots, C_0$, whereas in the $h(x)$ encoder the information symbols occupy codeword components $C_0, C_1, \ldots, C_{k-1}$ and the codeword symbols are sent to the channel in the natural order $C_0, C_1, \ldots, C_{n-1}$.

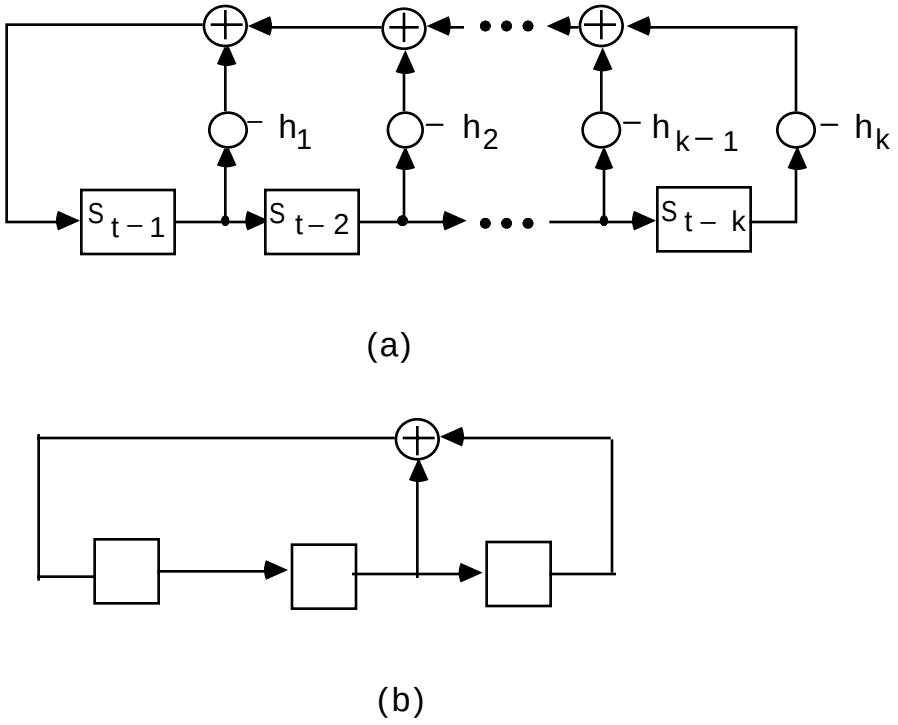


Figure 8.6. (a) A shift-register circuit that implements the kth order linear recursion $S_t = -\sum_{j=1}^{k} h_j S_{t-j}$. (b) The special case $S_t = S_{t-2} + S_{t-3}$, corresponding to $h(x) = 1 + x^2 + x^3$.

Example 8.12. If we apply the general construction of Figure 8.7(a) to the particular $(7, 3)$ binary cyclic code with $g(x) = x^4 + x^3 + x^2 + 1$, and $h(x) = x^3 + x^2 + 1$, we get the encoder shown in Figure 8.7(b). Thus for example, if the information symbols are $(I_0, I_1, I_2) = (1, 1, 0)$, the encoder of Figure 8.7(b) implements the recursion $C_i = C_{i-2} + C_{i-3}$ and produces the codeword $(1, 1, 0, 0, 1, 0, 1)$, which is codeword $\mathbf{C}_1 + \mathbf{C}_2 + \mathbf{C}_3$ in Example 8.2. Notice also that the $g(x)$ encoder for this code, illustrated in Figure 8.5b needs four flip-flops and 3 mod-2 adders whereas the $h(x)$-encoder in Figure 8.7(b) only needs three flip-flops and one mod-2 adder. As a general rule the $h(x)$ encoder will be simpler when $k < r$, i.e. $k < n/2$. ❒

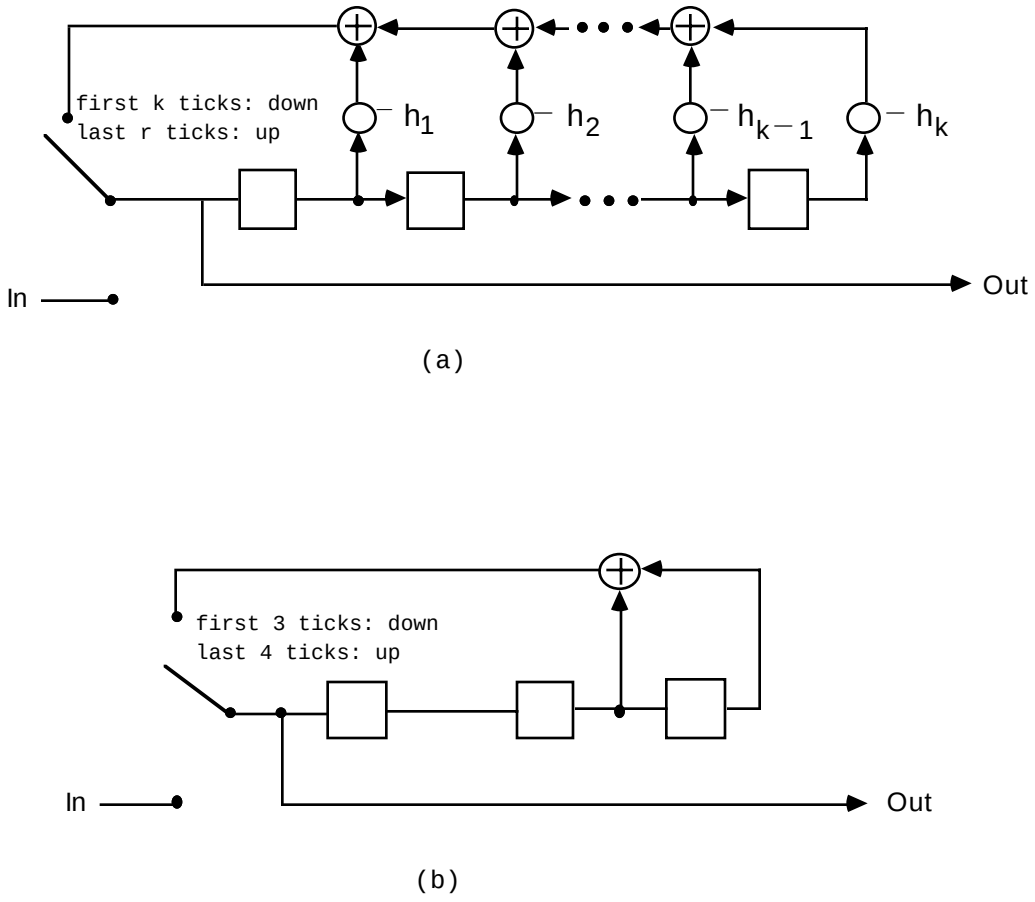


Figure 8.7. (a) A systematic shift-register encoder for an (n, k) cyclic code with parity-check polynomial $h(x) = 1 + h_1 x + \cdots + h_k x^k$. (b) Special case $h(x) = 1 + x^2 + x^3$.

8.3. Cyclic Hamming Codes.†

In Section 7.4 we defined a binary Hamming code of length $2^m - 1$ to be any linear code whose parity-check matrix has as columns the $2^m - 1$ nonzero binary vectors of length m, *arranged in any order*. For example, the following parity-check matrix defines a $(7, 4)$ Hamming code (Cf. the matrix H_3 in Section 7.4):

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

There are of course $(2^m - 1)!$ ways to order these columns, and although any one of these orderings produces a perfect single-error correcting code, some orderings are better than others from an implementational standpoint. In fact, we shall see in this section that it is possible to choose an ordering that produces a

† This section assumes the reader to be familiar with the theory of finite fields, and may be omitted on first reading.

cyclic version of the Hamming code, which leads to a simple shift-register implementation of both the encoder and the decoder. To describe the appropriate ordering, we must assume the reader to be familiar with the finite field $GF(2^m)$, in which each element is represented by a binary vector of length m. (In Appendix C we give a summary of the needed facts.)

To obtain a cyclic Hamming code of length $2^m - 1$, we begin with a primitive root $\alpha \in GF(2^m)$ and use it to define a linear code $\mathcal{C}$ as follows. A vector $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1})$ * with components in $GF(2)$ is in $\mathcal{C}$ if and only if

$$C_0 + C_1\alpha + \cdots + C_{n-1}\alpha^{n-1} = 0. \tag{8.8}$$

Alternatively, $\mathcal{C}$ is the binary linear code defined by the $1 \times n$ parity-check matrix

$$H = [\,1 \quad \alpha \quad \alpha^2 \quad \cdots \quad \alpha^{n-1}\,]. \tag{8.9}$$

We shall soon prove that the code defined by (8.8) or (8.9) is a cyclic Hamming code. For now, the important thing to notice about this definition is that the codeword components are in the binary field $GF(2)$ but the entries in the parity-check matrix are in the extension field $GF(2^m)$. There are, as we shall see, many advantages to this "two-field" approach. Still, it is possible to define the same code via a more conventional (and more complicated) binary parity-check matrix, by replacing each of the powers of α in H by its representation as an m-bit column vector. For Example, with $m = 3$, if α is a primitive root in $GF(2^3)$ satisfying $\alpha^3 = \alpha + 1$, then Table 8.2 shows the powers of α represented as three-dimensional vectors.

i	α^i
0	001
1	010
2	100
3	011
4	110
5	111
6	101

Table 8.2. The powers of α in $GF(8)$, where $\alpha^3 = \alpha + 1$.

Thus the 1×7 $GF(8)$-matrix H in (8.9) defines the same binary code as the following 3×7 binary matrix H':

$$H' = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

The following theorem is the main result of this section.

`Theorem 8.6.` *The code defined above (i.e., by (8.8) or (8.9)) is an $(n, n-m)$ binary cyclic code with generator polynomial $g(x)$, the minimal polynomial of α. Furthermore, its minimum distance is 3, so that it is a Hamming code.*

`Proof:` It is clear that the code is a binary linear code of length n. We need to show that it is cyclic, and that $g(x)$ is the generator polynomial. To show that it is cyclic, note that if we multiply Equation (8.8) by α, and use the fact that $\alpha^n = 1$, we get

$$C_{n-1} + C_0\alpha + \cdots + C_{n-2}\alpha^{n-2} = 0.$$

Thus if $\mathbf{C}$ satisfies (8.8), so does $\mathbf{C}^R$, and so the code is cyclic. To show that $g(x)$ is the generator polynomial, we note that (8.8) is equivalent to $C(\alpha) = 0$, where $C(x) = C_0 + C_1x + \cdots + C_{n-1}x^{n-1}$ is the code polynomial,

* Here and hereafter in this section, we adopt the convention that $n = 2^m - 1$.

and observe that this is true if and only if $C(x)$ is a multiple of α's minimal polynomial $g(x)$. Thus the code consists of all polynomials of degree $n-1$ or less which are multiples of $g(x)$, and hence by Theorem 8.3, $g(x)$ is the code's generator polynomial.

To complete the proof, we need to show that the code's minimum distance $d_{\min}$ is 3. Since the code is linear, $d_{\min} = w_{\min}$. We use the definition (8.8); if there were a codeword of weight 1, then $\alpha^i = 0$ for some i, which is impossible. Similarly, a word of weight 2 could exist if and only if $\alpha^i + \alpha^j = 0$ for some i and j with $0 \le i < j \le n-1$. Dividing by α^i, this becomes $1 + \alpha^{j-i} = 0$, which is impossible, because the smallest positive power of α equal to 1 is α^n. Finally we note that there are many words of weight 3; for example, if $1 + \alpha = \alpha^j$, then there is a codeword of weight 3 with nonzero components C_0, C_1, and C_j. ❒

Now that we know that there are cyclic Hamming codes, it follows from the results in Section 8.2 that it is possible to build simple shift-register encoders for Hamming codes, based on the generator polynomial. Since the generator polynomial for a Hamming code must be a primitive polynomial, in order to implement a cyclic Hamming code of length $2^m - 1$, it is necessary to have a primitive polynomial of degree m. In Table 8.3 we list one primitive polynomial of degree m, for $1 \le m \le 12$. For a given value of m, there will in general be many primitivie polynomials of degree m, but the ones in Table 8.3 are chosen to have the fewest possible nonzero coefficients, which will lead to shift-register encoders with the fewest possible mod-2 adders.

m	primitive polynomial of degree m
1	$x + 1$
2	$x^2 + x + 1$
3	$x^3 + x + 1$
4	$x^4 + x + 1$
5	$x^5 + x^2 + 1$
6	$x^6 + x + 1$
7	$x^7 + x + 1$
8	$x^8 + x^7 + x^2 + x + 1$
9	$x^9 + x^4 + 1$
10	$x^{10} + x^3 + 1$
11	$x^{11} + x^2 + 1$
12	$x^{12} + x^6 + x^4 + x + 1$

Table 8.3. Some primitive polynomials—possible generator polynomials for Hamming codes.

More important than the encoding simplicity of Hamming codes, however, is the fact it is also possible to build simple *decoders* for them. We illustrate such a decoder for the $(7, 4)$ Hamming code with generator polynomial $g(x) = x^3 + x + 1$ in Figure 8.8. The decoder consists of three main parts, two shift register circuits, and an `AND` gate. The upper shift register is a "mod $g(x)$" circuit, and the lower shift register is a "mod $x^n + 1$" circuit. The `AND` gate outputs a 1 if the upper shift-register contains the pattern $10 \cdots 0$, and otherwise it outputs 0. We will assume that the transmitted codeword is $C(x) = (C_0, C_1, \ldots, C_{n-1})$ and the received word is $R(x) = (R_0, R_1, \ldots, R_{n-1})$, with $R(x) = C(x) + E(x)$, where $E(x) = (E_0, E_1, \ldots, E_{n-1})$ is the error pattern.

In the decoding circuit of Figure 8.8, the noisy codeword $(R_0, R_1, \ldots, R_{n-1})$ is clocked in from the left, in the reverse order (i.e., R_{n-1} goes first), so that (by Theorem 8.4) after n ticks the upper shift register contains $R(x)$ mod $g(x)$, and the lower shift-register contains $R(x)$ mod x^n+1, i.e., $(R_0, R_1, \ldots, R_{n-1})$. Since $R(x) = C(x) + E(x)$, and $C(x)$ mod $g(x) = 0$, since $C(x)$ is a codeword, in fact the upper shift register will contain $E(x)$ mod $g(x)$. If there are no errors, then $E(x) = 0$ and the upper shift register will contain all zeros. If there is one error in the eth position then $E(x) = x^e$, where $0 \le e \le n-1$.

At this point, switch A is opened, switch B is closed, and the two shift registers run autonomously for n more ticks. We shall call these n ticks the *decoding cycle*. At the end of the decoding cycle, if there is at

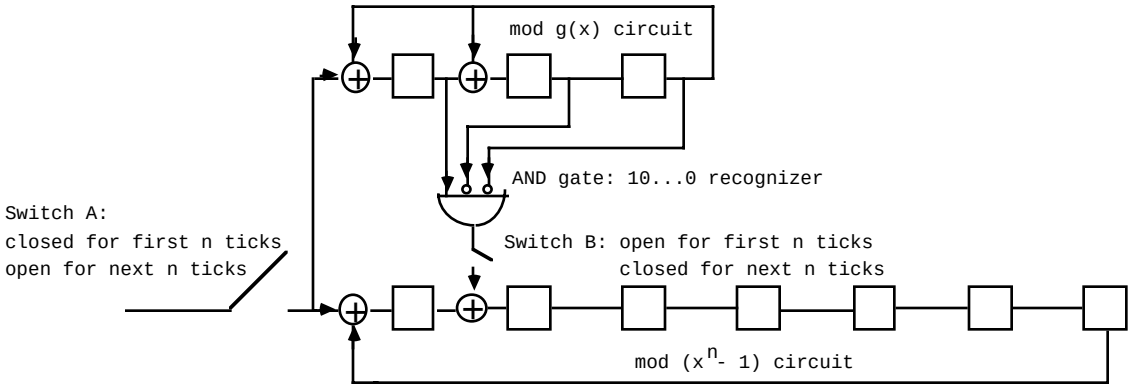


Figure 8.8. A decoding circuit for the $(7, 4)$ Hamming code with generator polynomial $g(x) = x^3 + x + 1$.

most one error, the original codeword $C(x)$ will appear in the lower shift register. Let us see why this is so.

After the tth tick of the decoding cycle, the contents of the upper shift will be $x^t E(x) \bmod g(x)$ and the lower shift register will contain $x^t R(x) \bmod x^n + 1$, i.e. the tth right cyclic shift $(R_{n-t}, \ldots, R_{n-t-1})$ of the received word. The two shift registers are connected via switch B by the `AND` gate, which outputs a 1 if and only if the upper shift register contains $10 \cdots 0$, i.e., iff $x^t E(x) \bmod g(x) = 1$. If there are no errors, $E(x) = 0$ and so the `AND` gate will never be triggered, so that after the decoding cycle, the lower shift register will contain the received word, with no changes. However, if there is one error, in the eth position, so that $E(x) = x^e$, then after $(n - e) \bmod n$ ticks of the decoding cycle, the upper shift register will contain $x^{n-e} x^e \bmod g(x) = x^n \bmod g(x) = 1$, since $g(x)$ is a divisor of $x^n - 1$. At this point the lower shift register will contain $(R_e, R_{e+1}, \ldots, R_{e-1})$ and on the next tick the `AND` gate will output a 1 and complement the erroneous component R_e of the received word, i.e., the error will be corrected. Then after $(e - 1) \bmod n$ further ticks, the received word will have completed its circular journey around the lower shift register, and will appear, with its error corrected, in the lower shift register.

In Section 8.5, we will see how to generalize the circuit of Figure 8.8 in order to build a decoder for a cyclic *burst* error-correcting code. The underlying theory for burst error correction will be presented in Section 8.4. (Incidentally, the *dual* codes of cyclic Hamming codes are also interesting and important. They are covered briefly in Problem 8.41).

8.4. Burst Error Correction

On many channels of practical importance, errors, when they occur, tend to occur in *bursts*. Physically, a burst of errors occurs when for some reason the channel noise severity increases for a brief time, and then returns to normal. In this section we will see how cyclic codes can be used to detect and correct error bursts.

We begin with a purely mathematical definition of an error burst. If a codeword $\mathbf{C}$ is transmitted, and is received as $\mathbf{R} = \mathbf{C} + \mathbf{E}$, then the error vector $\mathbf{E}$ is called a *burst of length b* if the nonzero components of $\mathbf{E}$ are confined to b consecutive components. For example $\mathbf{E} = (010000110)$ is a burst of length 7:

$$\begin{array}{cccccccccc} & * & * & * & * & * & * & * & \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

(In the display, the $*$'s mark the error burst.) Since we will be correcting bursts of errors with cyclic codes, for technical reasons we also need to define a *cyclic burst*. An error vector $\mathbf{E}$ is called a *cyclic burst of length b* if its nonzero components are confined to b cyclically consecutive components. For example, the error vector $\mathbf{E} = (010000110)$ cited above as a burst of length 7 is also a cyclic burst of length 5:

$$\begin{array}{cccccccccc} * & * & & & & & * & * & * & \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

In the rest of this section, "burst" will always mean "cyclic burst".

It is useful to have a compact description of a burst error vector, and so we introduce the notion of the *pattern* and *location* of a burst. If $\mathbf{E}$ is a nonzero burst error vector, its *burst pattern* is the string of symbols beginning with the first nonzero symbol in $\mathbf{E}$ and ending with the last nonzero symbol. The burst's *location* is the index of the first nonzero symbol in the burst. For example, the vector $\mathbf{E} = (010000110)$, viewed as a burst of length 7, has burst pattern 1000011 and burst location 1 (assuming the components are numbered $0, 1, \ldots, 8$). Unfortunately, the "burst pattern-burst location" description of most vectors $\mathbf{E}$ isn't unique, since any nonzero symbol in $\mathbf{E}$ can be taken as the first symbol in a cyclic burst. Thus an error vector of weight w will have w burst descriptions. For example, $\mathbf{E} = (010000110)$ has three burst descriptions:

pattern	*location*
1000011	1
11001	6
100100001	7

This ambiguity is annoying but usually not serious, since the correctable bursts that occur in practice tend to be quite short, and a short burst can have only one short description, as the following theorem shows.

`Theorem 8.7.` *Suppose* $\mathbf{E}$ *is an error vector of length* n *with two burst descriptions* $(\text{pattern}_1, \text{location}_1)$ *and* $(\text{pattern}_2, \text{location}_2)$. *If* $\text{length}(\text{pattern}_1) + \text{length}(\text{pattern}_2) \leq n+1$, *then the two descriptions are identical, i.e.,* $\text{pattern}_1 = \text{pattern}_2$ *and* $\text{location}_1 = \text{location}_2$.

`Proof:` As noted above, if $\mathbf{E}$ has weight w, then $\mathbf{E}$ has exactly w different burst descriptions. If $w = 0$ or 1, therefore, there is nothing to prove, so we assume $w \geq 2$.

For a given burst description of $\mathbf{E}$, the pattern will contain all of $\mathbf{E}$'s nonzero components, and so the components of $\mathbf{E}$ *not* included in the pattern will form a cyclic run of 0's, beginning just after the last nonzero component in the pattern, and continuing until just before the first nonzero component in the pattern. The set of indices corresponding to this run of $0's$ we call the *zero run* associated with the given burst description. For example, for $\mathbf{E} = (010000110)$, as we saw above, there are three burst descriptions. In the first of these descriptions, i.e., `100011`, the pattern begins at position 1 and ends at position 7. Thus the zero run associated with this burst description is `(0,8)`. Altogether there are three zero runs, one for each of the burst descriptions of $\mathbf{E}$:

pattern	*location*	*zero run*
1000011	1	(8,0)
11001	6	(2,3,4,5)
100100001	7	none

(In the last burst description, the zero run is empty, which we indicate by the word "none.") Plainly the zero runs associated with different burst descriptions are disjoint, and the total length of all of the zero runs is $n - w$, where w is the weight of $\mathbf{E}$. To prove the theorem, we note that if the two burst descriptions

$$(\text{pattern}_1, \text{location}_1) \quad \text{and} \quad (\text{pattern}_2, \text{location}_2)$$

of $\mathbf{E}$ are different, then since their zero runs are disjoint they account for $(n - \text{length}(\text{pattern}_1)) + (n - \text{length}(\text{pattern}_2))$ zeros in $\mathbf{E}$. But since $\text{length}(\text{pattern}_1) + \text{length}(\text{pattern}_2) \leq n+1$, this number is $\geq n-1$, which contradicts the fact that $\mathbf{E}$ has weight ≥ 2. Thus the burst descriptions must be identical. ❒

`Corollary.` *An error vector* $\mathbf{E}$ *can have at most one description as a burst of length* $\leq (n+1)/2$.

`Proof:` Two distinct descriptions of length $\leq (n+1)/2$ would contradict Theorem 8.7. ❒

With the help of Theorem 8.7, we can now count the number of burst patterns of a given length.

`Theorem 8.8.` *Over a two-letter alphabet, there are exactly* $n2^{b-1} + 1$ *vectors of length* n *which are bursts of length* $\leq b$, *provided* $1 \leq b \leq (n+1)/2$.

`Proof:` By the Corollary to Theorem 8.7, if $b \leq (n+1)/2$, a nonzero burst of length b has a unique description (as a burst of length $\leq b$). There are n possibilities for the location. The pattern must start with a 1 and have length $\leq b$, which means that the possible patterns are in one-to-one correspondence with the 2^{b-1} binary strings of length b which begin with a 1. Thus there are 2^{b-1} possible patterns, and so a total of $n \cdot 2^{b-1}$ nonzero bursts of length $\leq b$. Adding 1 to this number to account for the all-zero burst, we get $n2^{b-1} + 1$, as asserted. ❒

The next two theorems give useful bounds on the size of codes which correct burst errors. For simplicity, we shall call a code capable of correcting all burst error patterns of length $\leq b$ a *burst-b error correcting code*.

`Theorem 8.9.` *(The Hamming Bound for burst error correction.) If* $1 \leq b \leq (n+1)/2$, *a binary burst-b error correcting code has at most* $2^n/(n2^{b-1}+1)$ *codewords.*

`Proof:` By Theorem 8.8 there are $n2^{b-1} + 1$ burst error patterns of length $\leq b$. If there are M codewords, there are then $M(n2^{b-1}+1)$ words which differ from a codeword by a burst of length $\leq b$. These words must all be distinct, and so $M(n2^{b-1}+1) \leq 2^n$. ❒

`Corollary (The Abramson Bounds).` *If* $1 \leq b \leq (n+1)/2$, *a binary linear* (n,k) *linear burst-b error correcting code must satisfy*

$$n \leq 2^{r-b+1} - 1, \qquad \textit{(strong Abramson bound)}$$

where $r = n - k$ *is the code's redundancy. An alternative formulation is*

$$r \geq \lceil \log_2(n+1) \rceil + (b-1). \qquad \textit{(weak Abramson bound)}$$

`Proof:` For a linear (n,k) code there are $M = 2^k$ codewords, and so by Theorem 8.9, $2^k \leq 2^n/(n2^{b-1}+1)$. Rearranging this, we get $n \leq 2^{r-b+1} - 2^{-b+1}$. Since n must be an integer, this bound can be improved to $n \leq 2^{r-b+1} - 1$, the strong Abramson bound. Rearranging *this* to obtain a bound on r, we get the weak Abramson bound. ❒

`Theorem 8.10.` *If* $b \leq n/2$, *a binary burst-b error correcting code has at most* 2^{n-2b} *codewords.*

`Proof:` If $M > 2^{n-2b}$, then by the Pigeon-Hole Principle there must be two distinct codewords which agree in their first $n - 2b$ coordinates. These two codewords can then be represented schematically as follows:

$$X = \overbrace{* \ * \ * \ * \ * \ * \ * \ * \ *}^{n-2b} \overbrace{A \ A \ A \ A \ A \ A}^{2b}$$
$$Y = * \ * \ * \ * \ * \ * \ * \ * \ B \ B \ B \ B \ B \ B$$

where "$*$" denotes agreement, and the A's and B's are arbitrary. But then the word

$$Z = * \ * \ * \ * \ * \ * \ * \ * \ A \ A \ A \ B \ B \ B$$

differs from both X and Y by a burst of length $\leq b$, a contradiction. ❒

`Corollary (The Reiger Bound).` *If $0 \leq b \leq n/2$, a binary (n, k) linear burst-b error-correcting code must satisfy*

$$r \geq 2b$$

where $r = n - k$ is the code's redundancy.

`Proof:` The number of codewords in an (n, k) binary linear code is 2^k, which by Theorem 8.10 must be $\leq 2^{n-2b}$. This is equivalent to the statement of the Corollary. ❒

We are now in a position to discuss a series of examples of burst error-correcting codes. In each case, the code will be cyclic and meet either the strong Abramson or Reiger Bound (which apply all linear codes, not just cyclic codes.) In this discussion, when we say that a particular bound (either the Abramson bound or the Reiger bound) is *tight*, we mean that there exists a code whose redundancy is equal to the value of the bound. If no such code exists, we will say that the bound is *loose*.

Example 8.13. The $(n, 1)$ binary repetition code with $g(x) = x^{n-1} + x^{n-2} + \cdots + x + 1$, where n is odd, can correct all error patterns of weight $\leq (n-1)/2$, and so is a burst-$(n-1)/2$ error correcting code. Since $r = n - 1$, the Reiger Bound is tight. ❒

Example 8.14. The (n, n) code, consisting of all possible codewords of length n is a cyclic code with $g(x) = 1$. It is (trivally) a $b = 0$ burst error-correcting code, and since $r = 0$ too, the Reiger bound is again tight. ❒

Example 8.15. Any binary Hamming code, with $n = 2^m - 1$, $r = m$, and $g(x)=$ a primitive polynomial of degree m, is a $b = 1$ burst error correcting code. (Any error vector of weight 1 is *ipso facto* a burst of length 1.) The strong Abramson bound is tight for all these codes. ❒

Example 8.16.† Any cyclic Hamming code from which the codewords of odd weight have been removed is a $b = 2$ burst error correcting code called an *Abramson Code*. These codes are cyclic codes with generator polynomials of the form $g(x) = (x + 1)p(x)$, where $p(x)$ is a primitive polynomial. (See Problem 8.55). The smallest Abramson code is the $(7, 3)$ cyclic code with $g(x) = (x + 1)(x^3 + x + 1)$ and parity-check matrix

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix},$$

where α is a primitive root satisfying the equation $\alpha^3 + \alpha + 1 = 0$ in GF(8). To verify that this code is indeed a $b = 2$ burst error connecting code, we need to check that the syndromes of all bursts of length $b \leq 2$ are distinct. For $b = 0$ (the all zeros error pattern), the syndrome is $\binom{0}{0}$. For $b = 1$, a burst with description $(1, i)$ has syndrome $\binom{\alpha^i}{1}$. For $b = 2$, a burst with description $(11, i)$ has syndrome $\binom{\alpha^i(\alpha+1)}{0}$. These $1 + 2n$ syndromes are all distinct, and so the code is indeed a $b = 2$ burst error correcting code. Note finally that if $g(x) = (x + 1)p(x)$, where $p(x)$ is a primitive polynomial of degree m, then $n = 2^m - 1$, $r = m + 1$, and $b = 2$, so that the strong Abramson Bound is tight. (For $m = 3$, i.e. the $(7, 3)$, $b = 2$ code, the Reiger Bound is tight, too, but for all larger values of m the Reiger Bound is loose.) ❒

Example 8.17.† The $(15, 9)$ binary cyclic code with generator polynomial $g(x) = (x^4 + x + 1)(x^2 + x + 1) = x^6 + x^5 + x^4 + x^3 + 1$ turns out to be a $b = 3$ burst error-correcting code, for which, therefore, both the strong Abramson and Reiger bounds are tight. To prove that this is so, we use the parity-check matrix

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{14} \\ 1 & \omega & \omega^2 & \cdots & \omega^{14} \end{pmatrix},$$

where α is a primitive root in $GF(16)$ satisfying $\alpha^4 + \alpha + 1 = 0$, and ω is an element of order three in $GF(16)$ satisfying $\omega^2 + \omega + 1 = 0$, and check that the syndromes of all error bursts of length $b \leq 3$ are distinct.

The all-zero pattern has syndrome $\binom{0}{0}$. For $b = 1$, a burst with description $(1, i)$ has syndrome $\binom{\alpha^i}{\omega^i}$. For $b = 2$, a burst with description $(11, i)$ has syndrome $\binom{\alpha^i(1+\alpha)}{\omega^i(1+\omega)} = \binom{\alpha^{i+4}}{\omega^{i+2}}$, since $1 + \alpha = \alpha^4$ and $1 + \omega = \omega^2$. For $b = 3$, there are two possible patterns, viz. 101 and 111. A burst with description $(101, i)$ has syndrome $\binom{\alpha^i(1+\alpha^2)}{\omega^i(1+\omega^2)} = \binom{\alpha^{i+8}}{\omega^{i+1}}$, since $1 + \alpha^2 = \alpha^8$ and $1 + \omega^2 = \omega$, and a burst with description $(111, i)$ has syndrome $\binom{\alpha^i(1+\alpha+\alpha^2)}{\omega^i(1+\omega+\omega^2)} = \binom{\alpha^{i+10}}{0}$, since $1+\alpha+\alpha^2 = \alpha^{10}$ and $1+\omega+\omega^2 = 0$. Plainly the all-zero pattern and the pattern 111 cannot be confused with anything else, because of the "0" in the second component of the syndrome. To distinguish between the patterns 1, 11, and 101, we need to look a bit deeper. For any one of these three patterns, the syndrome will be of the form $\binom{\alpha^s}{\omega^t}$; to distinguish between them, we look at $(s - t)$ mod 3. If the pattern is 1, then $s = i$, $t = i$, and $(s - t)$ mod $3 = 0$; if the pattern is 11, then $s = i + 4$, $t = i + 2$, and $(s - t)$ mod $3 = 2$; and if the pattern is 101, then $s = i + 8$, $t = i + 1$, and $(s - t)$ mod $3 = 1$. Thus the 61 syndromes of the bursts of length ≤ 3 are distinct, and the following table summarizes the relationship between syndromes and burst descriptions.

syndrome	*burst description*	
$\binom{0}{0}$	$\emptyset$	
$\binom{\alpha^s}{0}$	$(111, s - 10 \bmod 15)$	
$\binom{\alpha^s}{\omega^t}$	$(1, s)$	if $(s - t) \bmod 3 = 0$
	$(101, s - 8 \bmod 15)$	if $(s - t) \bmod 3 = 1$
	$(11, s - 4 \bmod 15)$	if $(s - t) \bmod 3 = 2$.

For example, the syndrome $\binom{\alpha^{11}}{1}$ has $s = 11$, $t = 0$, and so corresponds to the burst description $(11, 7)$, i.e., the error pattern is (0000000000011000). ❒

The mathematical details of Example 8.17 are somewhat intricate, and do not easily generalize. (But see Problem 8.63). Nevertheless, researchers over the years have succeeded in finding a large number of cyclic burst error correcting codes which meet the strong Abramson bound; these are usually called *optimium* burst error-correcting codes. In Table 8.3, which follows, we list a number of such codes.

The next example illustrates the important *interleaving* technique, which is a simple way of boosting the burst error connecting ability of a code. (See also Problem 7.32).

Example 8.18. Consider again the $(7, 3)$ $b = 2$ Abramson code with $g(x) = x^4+x^3+x^2+1$ (see Example 8.16). Let **A**, **B**, and **C** be any three words from this code, which we can display as a 3×7 array:

$$\begin{array}{c} A_0\ A_1\ A_2\ A_3\ A_4\ A_5\ A_6 \\ B_0\ B_1\ B_2\ B_3\ B_4\ B_5\ B_6 \\ C_0\ C_1\ C_2\ C_3\ C_4\ C_5\ C_6. \end{array}$$

The following length 21 vector, which is built by reading the above array by columns, is called the *interleaving* of **A**, **B**, and **C**:

$$A_0\ B_0\ C_0\ A_1\ B_1\ C_1\ A_2\ B_2\ C_2\ A_3\ B_3\ C_3\ A_4\ B_4\ C_4\ A_5\ B_5\ C_5\ A_6\ B_6\ C_6 \quad .$$

generator polynomial	(n,k)	b
$(x^3+x+1)(x+1)$	$(7,3)$	2
$(x^4+x+1)(x+1)$	$(15,10)$	2
$(x^4+x+1)(x^2+x+1)$	$(15,9)$	3
$(x^5+x^2+1)(x+1)$	$(31,25)$	2
$(x^6+x+1)(x+1)$	$(63,56)$	2
$(x^6+x+1)(x^2+x+1)$	$(63,55)$	3
$(x^7+x+1)(x+1)$	$(127,119)$	2
$(x^8+x^7+x^2+x+1)(x+1)$	$(255,246)$	2
$(x^8+x^7+x^2+x+1)(x^2+x+1)$	$(255,245)$	3
$(x^9+x^7+x^6+x^3+x^2+x+1)(x+1)$	$(511,501)$	2
$(x^9+x^7+x^6+x^3+x^2+x+1)(x^3+x+1)$	$(511,499)$	4
$(x^{10}+x^5+x^3+x^2+1)(x+1)$	$(1023,1012)$	2
$(x^{10}+x^5+x^3+x^2+1)(x^2+x+1)$	$(1023,1011)$	3
$(x^{10}+x^5+x^3+x^2+1)(x^2+x+1)(x+1)$	$(1023,1010)$	4
$(x^{11}+x+1)(x+1)$	$(2047,2035)$	2
$(x^{12}+x^{11}+x^9+x^8+x^7+x^5+x^2+x+1)(x+1)$	$(4095,4082)$	2
$(x^{12}+x^{11}+x^9+x^8+x^7+x^5+x^2+x+1)(x^2+x+1)$	$(4095,4081)$	3
$(x^{12}+x^{11}+x^9+x^8+x^7+x^5+x^2+x+1)(x^2+x+1)(x+1)$	$(4095,4080)$	4
$(x^{15}+x^{13}+x^{10}+x^9+x^5+x^3+x^2+x+1)(x^3+x+1)(x+1)$	$(32767,32748)$	5

Table 8.3. Some cyclic burst-error correcting codes that meet the strong Abramson bound, i.e., satisfy $n = 2^{r-b+1} - 1$.

Let us suppose that this long codeword is transmitted over a bursty channel and suffers a burst of length 6, indicated by $*$'s:

$$A_0\ B_0\ C_0\ A_1\ B_1\ C_1\ A_2\ B_2\ C_2\ A_3\ *\ *\ *\ *\ *\ *\ B_5\ C_5\ A_6\ B_6\ C_6\ .$$

It is possible to correct this burst of errors, simply by "de-interleaving" this long codeword into its component codewords, because after de-interleaving, no one of the three codewords will have suffered a burst larger than two:

$$\begin{array}{l} A_0\ A_1\ A_2\ A_3\ *\ *\ A_6 \\ B_0\ B_1\ B_2\ *\ *\ B_5\ B_6 \\ C_0\ C_1\ C_2\ *\ *\ C_5\ C_6. \end{array}$$

The code consisting of all possible interleavings of three codewords from the $(7,3)$ Abramson code is called the *depth-3 interleaving* of the original code. It is a $(21,9)$ linear code; and the argument above shows that it is in fact a $b = 6$ burst-error-correcting code. More generally, for any positive integer j, the depth-j interleaving of the $(7,3)$ Abramson code is a $(7j, 3j)$ $b = 2j$ burst-error-correcting code. Note that each one of these codes satisfies the Reiger bound with equality (since $r = 4j$ and $b = 2j$), so that in a certain sense, no loss of efficiency results when a code is interleaved. ❒

A straightforward generalization of the argument given in Example 8.18 leads to the following important theorem.

`Theorem 8.11.` *If $\mathcal{C}$ is an (n, k) linear burst-b error-correcting code, then the depth-j interleaving of $\mathcal{C}$ is a (nj, kj) burst-bj error-correcting code.*

It is not obvious, but it is nevertheless true, that if we interleave a *cyclic* code to depth j, the resulting code is also cyclic. The next theorem spells this out.

`Theorem 8.12.` *If $\mathcal{C}$ is an (n, k) cyclic code with generator polynomial $g(x)$, then the depth-j interleaving of $\mathcal{C}$ is an (nj, kj) cyclic code with generator polynomial $g(x^j)$.*

`Proof:` To prove the theorem, we introduce the symbol "$\wr$" to denote the interleaving operation. Thus the vector obtained by interleaving the j codewords $\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_{j-1}$ is denoted by $\mathbf{C}_0 \wr \mathbf{C}_1 \wr \cdots \wr \mathbf{C}_{j-1}$. An easy lemma, whose proof we leave as Problem 8.65, shows how to compute the right cyclic shift of an interleaved codeword:

$$[\mathbf{C}_0 \wr \mathbf{C}_1 \wr \cdots \wr \mathbf{C}_{j-1}]^R = [\mathbf{C}_{j-1}^R \wr \mathbf{C}_0 \wr \cdots \wr \mathbf{C}_{j-2}]. \tag{8.10}$$

Equation (8.10) shows that the depth-j interleaving of a cyclic code is cyclic, since if $\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_{j-1}$ are words from a fixed cyclic code, then so are $\mathbf{C}_{j-1}^R, \mathbf{C}_0, \ldots, \mathbf{C}_{j-2}$. Since the redundancy of the interleaved code is rj, its generator polynomial will be its unique monic polynomial of degree rj. But if $g(x)$ is the generator polynomial for the original code, then the interleaved codeword $[g(x) \wr 0 \wr \cdots \wr 0]$, which is the polynomial $g(x^j)$, has degree rj, and so $g(x^j)$ must be the generator polynomial for the interleaved code. ❒

Example 8.19. Starting with the $(7, 3)$ $b = 2$ Abramson code with generator polynomial $g(x) = x^4+x^3+x^2+1$, and using the interleaving technique, we can produce an infinite family of cyclic burst error correcting codes, viz. the $(7j, 3j)$ $b = 2j$ codes, with generator polynomials $g_j(x) = x^{4j} + x^{3j} + x^{2j} + 1$. Similarly, starting with the cyclic $(15, 9)$ $b = 3$ code of Example 8.17, we obtain another infinite family, viz. the $(15j, 9j)$ $b = 3j$ cyclic codes with generator polynomials $g_j(x) = x^{6j} + x^{5j} + x^{4j} + x^{3j} + 1$. Note that every code in each of the families meets the Reiger bound (Corollary to Theorem 8.11). ❒

Example 8.20. In the following table, we consider binary cyclic $n = 15$ burst error-correcting codes, for $b = 1, 2, \ldots, 7$.

b	r_A	r_R	$g(x)$	comment
1	*4*	2	$x^4 + x + 1$	Hamming code
2	*5*	4	$(x^4 + x + 1)(x + 1)$	Abramson code (Example 8.16)
3	*6*	*6*	$(x^4 + x + 1)(x^2 + x + 1)$	Example 8.17
4	7	*8*	$(x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)$	Problem 8.52
5	8	*10*	$(x^4 + x + 1)(x^2 + x + 1)(x^4 + x^3 + 1)$ $= x^{10} + x^5 + 1$	$(3, 1)$ interleaved $\times 5$
6	9	*12*	$(x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^4 + x^3 + 1)$ $= x^{12} + x^9 + x^6 + x^3 + 1$	Problem 8.54
7	10	*14*	$x^{14} + x^{13} + \cdots + x + 1$	Repetition code

For each such value of b, we list the lower bounds on the needed redundancy using the (weak) Abramson bound (r_A) and the Reiger bound (r_R). By chance, turns out that in every case there is a cyclic code whose redundancy is $\max(r_A, r_R)$, and we have listed the generator polynomial $g(x)$ for such a code in each case. The cases $b = 1, 2, 3$ we have already considered; the case $b = 4$ we leave as Problem 8.58. To obtain the $b = 5$ code, we interleave the $(3, 1)$ repetition code with $g(x) = x^2 + x + 1$ to depth 3, using Theorem 8.12. The case $b = 6$ is left as Problem 8.60. Finally, the case $b = 7$ is simply the $n = 15$ repetition code. ❒

If we combine the codes found in Table 8.3 (or in more extensive tables which have been found by elaborate algebraic or *ad hoc* computer methods) with the interleaving technique, we can produce many

good burst-error correcting cyclic codes, and indeed some of the burst-error correcting codes used in practice have been constructed this way. However, there is another, quite different, general approach to designing burst-error correcting codes which has had great success as well, called the *Fire Code* method, which we shall now study.

Fire codes differ in a number of ways from the other cyclic burst-error correcting codes we have studied. The most important difference is their guaranteed ability to *detect* many error bursts that are too long to *correct*. Indeed, Fire codes are *strong* burst-error correcting codes in the sense of the following definition.

`Definition.` An (n, k) cyclic code with generator polynomial $g(x)$ is said to be a *strong* burst-b error-correcting code, if, whenever $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are burst error vectors with the same syndrome, and with burst descriptions (pattern$_1$, location$_1$) and (pattern$_2$, location$_2$) such that length(pattern$_1$) + length(pattern$_2$) $\leq 2b$, then $\mathbf{Z}_1 = \mathbf{Z}_2$. ❒

The following theorem shows that strong burst-error-correcting codes are capable of simultaneous burst error correction and detection.

`Theorem 8.13.` *If $\mathcal{C}$ is a strong burst-b error-correcting code, then for any pair of nonnegative integers b_1 and b_2 such that $b_1 \leq b_2$ and $b_1 + b_2 = 2b$, it is possible to design a decoder for $\mathcal{C}$ that will correct all bursts of length $\leq b_1$, while at the same time detecting all burst of length $\leq b_2$.*

`Proof:` This follows from Theorem 7.4 and the definition of strong burst correction. To see this, let $\mathcal{E}$ denote the set of error bursts of length $\leq b_1$, and let $\mathcal{F}$ denote the set of error bursts of length $\leq b_2$. Then if $\mathbf{Z}_1 \in \mathcal{E}$ and $\mathbf{Z}_2 \in \mathcal{F}$, we know that $\mathbf{Z}_1$ and $\mathbf{Z}_2$ have burst descriptions such that length(pattern$_1$) $\leq b_1$ and length(pattern$_2$) $\leq b_2$. But $b_1 + b_2 = 2b$, and so by the definition of a strong burst-b error correcting code, $\mathbf{Z}_1$ and $\mathbf{Z}_2$ have different syndromes. Thus by Theorem 7.4, $\mathcal{C}$ has the advertised capability. ❒

Example 8.21.† Most burst-b error-correcting codes, including all of the codes in Table 8.3, are *not* strong. For example, consider the $(7, 3)$ $b = 2$ Abramson code. Its parity-check matrix can be taken to be

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix},$$

where $\alpha^3 + \alpha + 1 = 0$ in GF(8), as we saw in Example 8.16. This code corrects all bursts of length ≤ 2; but it cannot correct all bursts of length ≤ 1 while detecting all bursts of length ≤ 3, since e.g. the error patterns (1110000) and (0000010) have the same syndrome, viz. $\binom{\alpha^5}{1}$. (See Problem 8.68). ❒

Example 8.22. As a degenerate example of a strong burst-b error correcting code, consider the $(n, 0)$ "no information" cyclic code with generator polynomial $x^n - 1$. This code has only one codeword (the all zeros codeword) and so it is not useful for transmitting information; nevertheless, all error patterns have distinct syndromes, and so it is a strong burst-n error-correcting code. (This code will be used as a building block for the Fire codes to be defined in the Corollary to Theorem 8.14, which follows.) Similarly, the $(n, 1)$ binary repetition code with $g(x) = (x^n + 1)/(x + 1)$ is a strong burst-b error correcting code with $b = (n - 1)/2$, if n is odd. (See Problem 8.66). ❒

The following theorem gives a general construction for cyclic burst-b error correcting codes, both strong and weak.

`Theorem 8.14.` *(The Fire Construction). Suppose $g_1(x)$ is the generator polynomial for an (n_1, k_1) cyclic code which is a (strong) burst-b_1 error correcting code, $g_2(x)$ is the generator polynomial for an (n_2, k_2) cyclic code, and all of $g_2(x)$'s irreducible factors have degree $\geq m$, and finally that $g_1(x)$ and $g_2(x)$ are relatively prime. Then $g(x) = g_1(x)g_2(x)$ is the generator polynomial for an (n, k) cyclic code which is a (strong) burst-b error-correcting code, where:*

$$\begin{aligned} n &= \mathrm{lcm}(n_1, n_2) \\ k &= n - \deg(g_1) - \deg(g_2) \\ b &= \min(b_1, m, (n_1 + 1)/2). \end{aligned}$$

`Proof:` By Theorem 8.3a, $g_1(x) \,|\, x^{n_1}-1$ and $g_2(x) \,|\, x^{n_2}-1$. Thus since both $x^{n_1}-1$ and $x^{n_2}-1$ divide x^n-1, where $n = \mathrm{lcm}(n_1, n_2)$, and since $g_1(x)$ and $g_2(x)$ are relatively prime, it follows that $g(x) = g_1(x)g_2(x) \,|\, x^n - 1$, and so by Theorem 8.3 b, $g(x)$ generates an (n, k) cyclic code, where $k = n-\deg(g) = n-\deg(g_1)-\deg(g_2)$. This much is easy. The heart of the proof deals with the assertion about the burst-error correcting capability of the code. We will assume that the code generated by $g_1(x)$ is strong, and show that the resulting longer code is also strong. The proof for a "weak" $g_1(x)$ is left as a Problem 8.69.

Thus let $\mathbf{Z}_1$ and $\mathbf{Z}_2$ be two error vectors of length n with the same syndrome relative to the cyclic code with generator polynomial $g(x)$, and with burst descriptions (P_1,i) and (P_2,j) such that $\mathrm{length}(P_1) + \mathrm{length}(P_2) \le 2b$. We need to show that $\mathbf{Z}_1 = \mathbf{Z}_2$.

Let's denote the generating functions for the two burst *patterns* by $P_1(x)$ and $P_2(x)$. It follows that the generating functions for the error vectors $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are $[x^iP_1(x)]_n$ and $[x^jP_2(x)]_n$, respectively, and so the remainder syndromes are $[x^iP_1(x)]_n \bmod g(x)$ and $[x^jP_2(x)]_n \bmod g(x)$. But since $g(x) \,|\, x^n - 1$, Lemma 1(v) implies that the two syndromes can in fact be written as

$$S_1(x) = x^iP_1(x) \bmod g(x)$$
$$S_2(x) = x^jP_2(x) \bmod g(x).$$

Since $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are assumed to have the same syndrome, we therefore have

$$x^iP_1(x) \equiv x^jP_2(x) \pmod{g(x)}. \tag{8.11}$$

Since $g(x) = g_1(x)g_2(x)$, then also

$$x^iP_1(x) \equiv x^jP_2(x) \pmod{g_1(x)}.$$

But $\mathrm{length}(P_1) + \mathrm{length}(P_2) \le 2b$ by assumption, and $2b \le 2b_1$ by the definition of b. Thus since $g_1(x)$ generates a strong burst-b_1 error-correcting code, it follows that from the viewpoint of the cyclic code generated by $g_1(x)$, the error vectors $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are identical, i.e.

$$x^iP_1(x) \equiv x^jP_2(x) \pmod{x^{n_1} - 1}.$$

But $\mathrm{length}(P_1) + \mathrm{length}(P_2) \le 2b \le n_1 + 1$, and so by Theorem 8.7,

$$P_1 = P_2 \tag{8.12}$$
$$i \equiv j \pmod{n_1}. \tag{8.13}$$

In view of this, (8.11) implies

$$(x^i - x^j)P_1(x) \equiv 0 \pmod{g_2(x)}. \tag{8.14}$$

But $2 \cdot \mathrm{length}(P_1) \le 2b$, and so $P_1(x)$ has degree $\le b-1$, which is less than m, the degree of the smallest-degree divisor of $g_2(x)$. Thus $P_1(x)$ and $g_2(x)$ are relatively prime, and so we can cancel $P_1(x)$ from (8.14), obtaining

$$x^i - x^j \equiv 0 \pmod{g_2(x)}. \tag{8.15}$$

But since the sequence $x^t \bmod g(x)$, has period n_2, (8.15) implies $i \equiv j \pmod{n_2}$. Combining this fact with (8.13), and using the fact that $n = \mathrm{lcm}(n_1, n_2)$, we have $i \equiv j \pmod{n}$; but since i and j both lie in the range $0, 1, \ldots, n-1$, it follows that $i = j$. Since we already showed that $P_1 = P_2$, it follows that the error vectors $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are identical, and this completes the proof. ❒

`Corollary (The classical Fire codes).` Let $g(x) = (x^{2b-1} - 1)f(x)$, where $f(x)$ is an irreducible polynomial which is not a divisor of $x^{2b-1} - 1$, of degree $m \ge b$ and period n_0. Then $g(x)$ is the generator polynomial for an $(n, n-2b+1-m)$ cyclic code which is a strong burst-b corrector, where $n = \mathrm{lcm}(2b-1, n_0)$. This code is called a *Fire code*, in honor of Philip Fire, its discoverer.

`Proof:` This follows immediately from Theorem 8.14, by taking $g_1(x) = (x^{2b-1} - 1)$ and $g_2(x) = f(x)$ (In Example 8.22 we saw that the $g_1(x)$ code is a strong burst-b corrector). ❒

Example 8.23. According to the Corollary to Theorem 8.14, the binary cyclic code with $g(x) = (x^3+1)(x^3+x+1) = x^6+x^4+x+1$ generates a $(21, 15)$ strong $b = 2$ burst error correcting Fire code. Thus it is possible to devise a decoder for this code that corrects all bursts of length ≤ 2; or to correct all bursts of length ≤ 1 and detect all bursts of length ≤ 3; or to detect all bursts of length ≤ 4. Note that this code meets neither the strong Abramson bound nor the Reiger bound; on the other hand, codes that do meet these bounds are apparently never strong burst correctors. Also note that if we take $g_1(x) = x^4+x^3+x^2+1$, which generates the (weak) $(7, 3)$ $b = 2$ Abramson code, and $g_2(x) = x^2+x+1$, Theorem 8.14 implies that $g(x) = g_1(x)g_2(x)$ generates a (weak) $(21, 15)$ $b = 2$ burst error correcting code. But $g_1(x)g_2(x) = x^6+x^4+x+1$, and we have already seen that this polynomial is strong! ❒

Example 8.24. The polynomial $P_{35}(x) = x^{35}+x^{23}+x^8+x^2+1$ is a primitive binary polynomial of degree 35, and so by the Corollary to Theorem 8.14, $g(x) = (x^{13}+1)P_{35}(x) = x^{48}+x^{36}+x^{35}+x^{23}+x^{21}+x^{15}+x^{13}+x^8+x^2+1$ generates a strong cyclic $(13(2^{35}-1), 13(2^{35}-1)-48) = (446676598771, 446676598723)$ $b = 7$ Fire code. This particular Fire code is quite famous, because IBM has used a "shortened" version of it in many of its disk drives. What is a shortened cyclic code? In general, if $g(x)$ has degree r and generates a $(n, n-r)$ cyclic code, then for any $n_0 \leq n$, $g(x)$ also generates a (n_0, n_0-r) shortened cyclic code. This code consists of all vectors $\mathbf{C}$ of length n_0 whose generating function $C(x)$ is a multiple of $g(x)$. In the case of the IBM Fire code, $n_0 = 152552$, so the code used is actually a $(152552, 152504)$ shortened cyclic $b = 7$ strong burst error correcting code. The IBM decoder is very conservative and only attempts to correct bursts of length ≤ 4, and so, since the code is strong, it thereby gains the ability to detect all bursts of length ≤ 10. In fact, however, because the code has been so drastically shortened, a computer-aided calculation shows that the IBM decoder will actually detect all burst error patterns of length ≤ 26, and all but an infinitesmal fraction of longer bursts. ❒

We conclude this section with a short table listing some useful binary Fire codes. For each of these codes, $g(x) = (x^{2b-1}+1)p_b(x)$, where $p_b(x)$ is a primitive polynomial of degree b. The redundancy is thus $r = 3b-1$, although the redundancy required by the weak Abramson and Reiger bounds is typically considerably smaller than this. This "extra" redundancy is apparently the price that must be paid in order that the codes be strong burst correctors.

(n, k)	b	generator polynomial
$(35, 27)$	3	$(x^5+1)(x^3+x+1) = x^8+x^6+x^5+x^3+x+1$
$(105, 94)$	4	$(x^7+1)(x^4+x+1) = x^{11}+x^8+x^7+x^4+x+1$
$(279, 265)$	5	$(x^9+1)(x^5+x^2+1) = x^{14}+x^{11}+x^9+x^5+x^2+1$
$(693, 676)$	6	$(x^{11}+1)(x^6+x+1) = x^{17}+x^{12}+x^{11}+x^6+x+1$
$(1651, 1631)$	7	$(x^{13}+1)(x^7+x+1) = x^{20}+x^{14}+x^{13}+x^7+x+1$
$(255, 232)$	8	$(x^{15}+1)(x^8+x^4+x^3+x^2+1) = x^{23}+x^{19}+x^{18}+x^{17}+x^{15}+x^8+x^4+x^3+x^2+1$
$(8687, 8661)$	9	$(x^{17}+1)(x^9+x^4+1) = x^{26}+x^{21}+x^{17}+x^9+x^4+1$
$(19437, 19408)$	10	$(x^{19}+1)(x^{10}+x^3+1) = x^{29}+x^{22}+x^{19}+x^{10}+x^3+1$

Table 8.4. Some Fire Codes.

8.5 Decoding Burst Error Correcting Cyclic Codes

In Section 8.2 we saw how to design a shift-register encoder for an arbitrary cyclic code. As a rule it is much harder to design a corresponding *decoder*, but in the important special case of burst error correcting cyclic codes, there is a simple decoding algorithm called the *burst-trapping algorithm* which naturally lends itself to shift-register implementation. In this section we will see how burst-trapping works.

Here is the underlying idea. Suppose that $g(x)$ generates an (n, k) cyclic code $\mathcal{C}$, and that a transmitted codeword $C(x)$ is received as $R(x)$, where

$$R(x) = C(x) + E(x),$$

$E(x)$ being the error pattern. If the decoder computes the remainder syndrome $S(x)$ defined by

$$S(x) = R(x) \bmod g(x),$$

then the vector obtained by subtracting $S(x)$ from $R(x)$, viz.

$$\hat{C}(x) = R(x) - S(x), \tag{8.16}$$

is guaranteed to be a codeword. This is because $\hat{C}(x) \bmod g(x) = R(x) \bmod g(x) - S(x) \bmod g(x) = 0$. Thus if $\mathcal{C}$ can correct all errors in the set $\mathcal{E}$, and $S(x)$ lies in $\mathcal{E}$, then the decoder can safely assert that $\hat{C}(x)$ is the actual transmitted codeword, since no other codeword can differ from $R(x)$ by an error pattern in $\mathcal{E}$. This much is true for any cyclic code. Let's now consider the special case of a cyclic *burst* error correcting code.

Suppose then that $\mathcal{C}$ is a burst-b error correcting code, and that the syndrome $S(x)$ satisfies the two conditions

$$\begin{aligned} S(0) &\neq 0 \\ \deg(S(x)) &\leq b - 1. \end{aligned} \tag{8.17}$$

What this means is that the *syndrome itself* is a (left-justified) burst of length $\leq b$, and so by the above discussion the decoder can safely assert that $\hat{C}(x)$, as defined in (8.16), is the actual transmitted codeword. Decoding is therefore easy if (8.17) is satisfied. Unfortunately, however, this only happens if the error vector is a burst located at position 0.

Suprisingly, however, this simple idea can be made to work even if the burst is located in a position *other* than 0, by taking advantage of the "cyclicness" of the code. Here's how: if the error vector $E(x)$ is a nonzero burst of length $\leq b$, then $E(x)$ has a unique burst description of the form $(P(x), i_0)$ with $P(0) \neq 0$ and $\deg(P) \leq b - 1$. Then $E(x) = [x^{i_0} P(x)]_n$ and the remainder syndrome is

$$S(x) = [x^{i_0} P(x)] \bmod g(x),$$

where $g(x)$ is the code's generator polynomial. This situation is depicted in Figure 8.9a. As we already noted, if the burst is located in position 0, i.e., if $i_0 = 0$, then $S(x) = P(x)$ and the error burst can be corrected immediately. If $i_0 \neq 0$, however, it is possible to shift $E(x)$ cyclically to the right until the burst pattern $P(x)$ *is* located, or "trapped", in position 0 as shown in Figure 8.9b. The number of cyclic shifts required to do this is the unique integer j_0 in the range $0 \leq j_0 \leq n - 1$ such that $i_0 + j_0 \equiv 0 (\mathrm{mod} n)$, which is

$$j_0 = (-i_0) \bmod n.$$

If now $R_{j_0}(x)$ denotes the j_0th cyclic shift of $R(x)$, we have

$$R_{j_0}(x) = C_{j_0}(x) + E_{j_0}(x),$$

where $C_{j_0}(x)$ and $E_{j_0}(x)$ are the j_0th cyclic shifts of $C(x)$ and $E(x)$, respectively. Now $C_{j_0}(x)$ is a codeword since the code is cyclic, and $E_{j_0}(x) = P(x)$ by the definition of j_0, and so if $S_{j_0}(x)$ denotes the remainder syndrome of $R_{j_0}(x)$, we have

$$\begin{aligned} S_{j_0}(x) &= R_{j_0}(x) \bmod g(x) \\ &= (C_{j_0}(x) + E_{j_0}(x)) \bmod g(x) \\ &= P(x). \end{aligned}$$

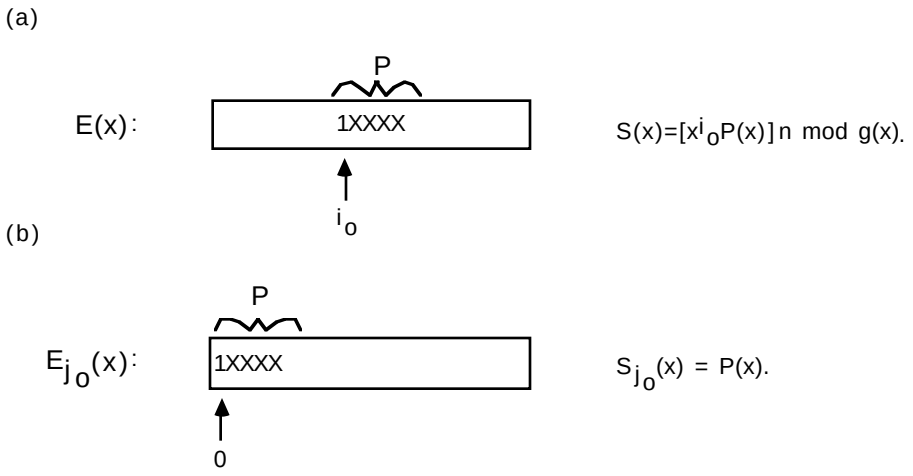


Figure 8.9. (a). The burst error vector $E(x) = [x^{i_0}P(x)]_n$.
(b) After shifting $\mathbf{E}(x)$ j_0 units to the right, where
$j_0 = (n - i_0) \bmod n$, the error pattern is "trapped"
at location 0, where the corresponding syndrome
is $P(x)$, a left-justified burst of length $\leq b - 1$.

This means that $S_{j_0}(x)$ satisfies the conditions (8.17), and so the decoder can safely assert that $\hat{C}_{j_0}(x)$, defined by

$$\hat{C}_{j_0}(x) = R_{j_0}(x) - S_{j_0}(x)$$

is the j_0 th cyclic shift of the actual transmitted codeword.

It follows that if the decoder successively computes $S_0(x), S_1(x), \ldots$, and tests each of these polynomials for the conditions (8.17), eventually the burst error will be "trapped," and it can be corrected. Indeed, the burst error *pattern* will be given by $S_{j_0}(x)$, and the burst error *location* will be given by the formula $i_0 = (-j_0) \bmod n$, where j_0 is the number of shifts required to trap the error. At this point, we could design a fairly simple decoder using these ideas. However, before doing so, we wish to note that the calculation of the successive $S_j(x)$'s can be considerably simplified by using the following result.

`Theorem 8.15.` *(Meggitt's Lemma).: For $j \geq 0$ define*

$$S_j(x) = [x^j R(x)]_n \bmod g(x),$$

i.e., $S_j(x)$ is the remainder syndrome of the jth cyclic shift of $R(x)$. Then for $j \geq 0$,

$$S_{j+1}(x) = [xS_j(x)] \bmod g(x)$$

`Proof:` First note that by Lemma 1(v),

$$S_j(x) = [x^j R(x)] \bmod g(x),$$

since $g(x) \,|\, x^n - 1$. Then

$$\begin{aligned}
[xS_j(x)] \bmod g(x) &= [x([x^j R(x)] \bmod g(x))] \bmod g(x) \\
&= [x^{j+1} R(x)] \bmod g(x) \qquad \text{by Lemma 1(iv)} \\
&= S_{j+1}(x).
\end{aligned}$$

❒

Example 8.25. We illustrate these ideas using the $(7,3)$ $b=2$ Abramson code with $g(x)=x^4+x^3+x^2+1$. Suppose the received vector is $\mathbf{R}=[1010011]$, i.e., $R(x)=x^6+x^5+x^2+1$. Then $S_0(x)=R(x) \bmod g(x) = (x^6+x^5+x^2+1) \bmod (x^4+x^3+x^2+1) = x^3+x^2$. Using Meggitt's Lemma, we successively complete $S_1(x), S_2(x)$, etc.:

$$\begin{aligned} S_1(x) &= [xS_0(x)] \bmod g(x) \\ &= (x^4+x^3) \bmod (x^4+x^3+x^2+1) \\ &= x^2+1. \end{aligned}$$

Similarly,

$$\begin{aligned} S_2(x) &= x^3+x \\ S_3(x) &= (x^4+x^2) \bmod g(x) = x^3+1 \\ S_4(x) &= (x^4+x) \bmod g(x) = x^3+x^2+x+1 \\ S_5(x) &= (x^4+x^3+x^2+x) \bmod g(x) = x+1. \end{aligned}$$

Now we stop, since $S_5(x)$ satisfies the conditions (8.17), and conclude that the burst error *pattern* is 11, and the burst error *location* is $(-5) \bmod 7 = 2$. Thus the error vector is $E=[0011000]$, and the corrected codeword is $\mathbf{R}+\mathbf{E}=[1001011]$. ❒

In Figure 8.10 we display a complete decoding algorithm for a burst-b error-correcting cyclic code, including the simplification afforded by Meggitt's Lemma. In this algorithm, the syndrome $S_0(x)$ is first computed at line 3. The `for` loop at lines 4-8 successively tests the shifted syndrome $S_0(x)$, $S_1(x), \ldots, S_{n-1}(x)$ for the condition (8.17). If this condition is satisfied, the burst of errors is corrected at line 6. Then the next cyclic shift $R_{j+1}(x)$ of the received word is completed at line 7, and the next syndrome $S_{j+1}(x)$ is computed (using Meggitt's lemma) at line 8. After n cyclic shifts, the original received sector, minus the burst error pattern, is output at line 9. (The algorithm also works if there are *no* errors; see Problem 8.75).

```
    /* Burst Error Trapping Decoding Algorithm */
 {
1.     input R(x);
2.     R_0(x) ← R(x);
3.     S_0(x) ← R_0(x) mod g(x);
4.     for(j = 0 to n − 1) {
5.          if (S_j(0) ≠ 0 and deg S_j(x) ≤ b − 1)
6.            R_j(x) ← R_j(x) − S_j(x);
7.          R_{j+1}(x) ← [xR_j(x)] mod x^n − 1;
8.          S_{j+1}(x) ← [xS_j(x)] mod g(x);
       }
9.     output R_n(x);
 }
```

Figure 8.10 "Full-Cycle-Clock-Around" Burst Trapping Decoding Algorithm for Cyclic Burst Error Correction.

The algorithm described in Figure 8.10 lends itself to implementation by shift-register logic. We illustrate this in Figures 8.11 and 8.12, for two different burst-error correcting cyclic codes, the $(7,3)$ $b=2$ Abramsom code of Example (8.16) and the $(15,9)$ $b=3$ code of Example 8.17.

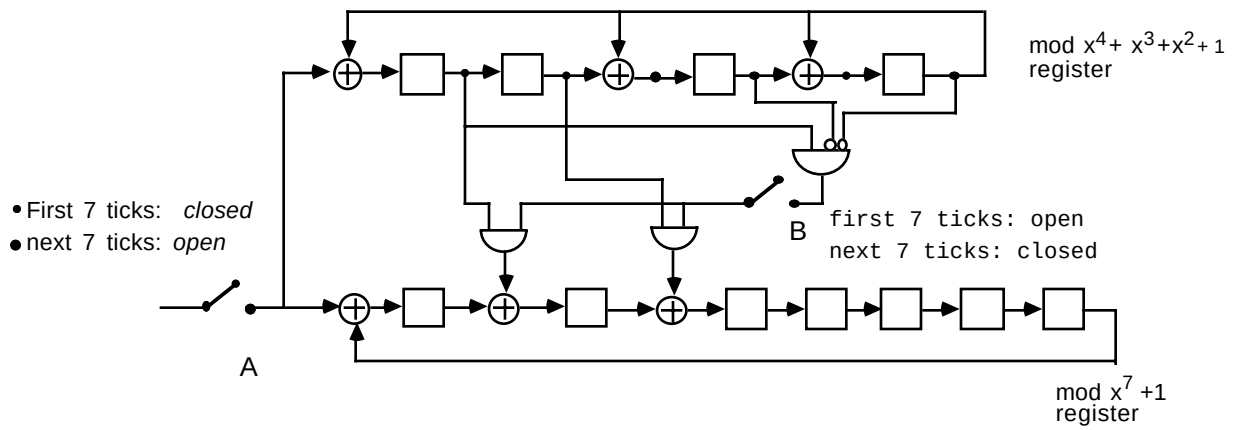


Figure 8.11. Complete decoding of circuit for the $(7, 3)$ $b = 2$ Abramson code with $g(x) = x^4 + x^3 + x^2 + 1$ (see Example 8.16). After 14 ticks, the decoding is complete, and the decoded word appears in the lower register.

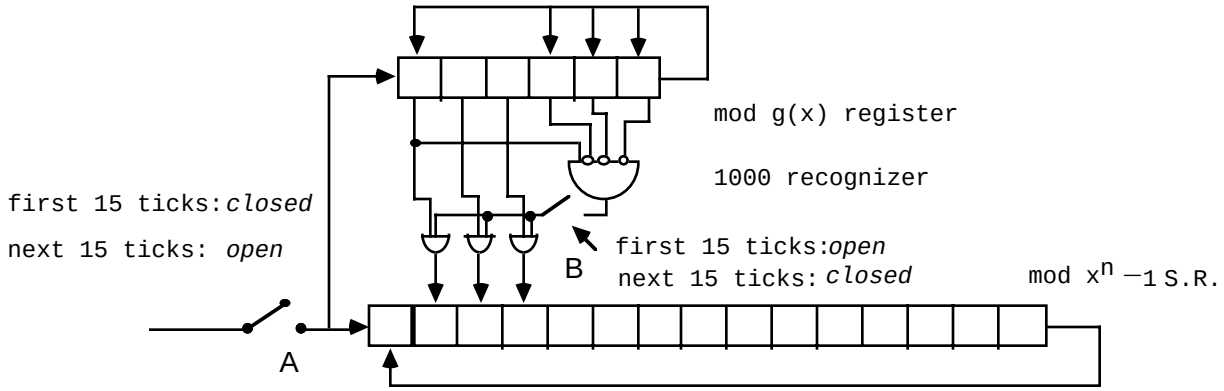


Figure 8.12. Simplified diagram for a complete decoding circuit for the $(15, 9)$ $b = 3$ cyclic code with $g(x) = x^6 + x^5 + x^4 + x^3 + 1$ (see Example 8.17). After after 30 ticks, the decoding is complete, and the decoded word appears in the lower register.

In the decoding circuits in Figures 8.11 and 8.12, there are three main components: a mod $g(x)$ shift register, a mod $x^n - 1$ shift register, and a "$100 \cdots 0$ recognizer" circuit correcting the two shift registers. Initially, each flip-flop contains 0, switch A is in the *closed* position, and switch B is in the *open* position. The symbols of the received vector are then clocked in from the left in the order $R_{n-1}, R_{n-2}, \ldots, R_0$. After n ticks, the upper shift register will contain $S_0(x) = R(x) \bmod g(x)$ (by Theorem 8.4) and the lower shift will contain $R(x)$. Next, switch A is *opened*, and switch B is *closed*, thus allowing the upper shift register

to communicate to the lower shift register via the $100\cdots0$ recognizer, and the decoder operates for n more ticks. We call these n ticks the *decoding cycle*. After the jth tick of the decoding cycle, the upper shift register will contain $[x^j S_0(x)] \bmod g(x) = S_j(x)$, and the lower register will contain $[x^j R(x)]_n = R_j(x)$. If conditions (8.17) are satisfied, then the leftmost flip-flop of the upper register will contain a 1, and the rightmost $r-b$ flip-flop will all contain 0's. This will cause the $100\cdots0$ recognizer to output a 1, and the trapped error pattern $S_j(x)$ will then be added into $R_j(x)$ at the next tick. After the n ticks of the decoding cycle have been completed, the symbols in $R(x)$ will have made a complete circuit around the $\bmod(x^n-1)$ register, and he returned to their original position, with the error burst corrected.

In general, circuits like these in Figures 8.11 and 8.12 are called "full-period clock-around" decoders. They require exactly $2n$ clock ticks to correct any burst of length $\leq b$, with hardware complexity that is $O(n)$. They are in widespread use in practice. However, for drastically shortened cyclic burst error correcting codes, (e.g. Example 8.24), a modification of this circuit may be required–see Problem 8.77.

8.6. Problems for Chapter 8.

8.1. Show that the number of k-dimensional subspaces of an n-dimensional vector space over the field $GF(q)$ is exactly

$$\frac{(q^n-1)(q^{n-1}-1)\cdots(q^{n-k+1}-1)}{(q^k-1)(q^{k-1}-1)\cdots(q-1)},$$

and use this formula to verify that there are exactly 11811 binary $(7,3)$ linear codes.

8.2. Calculate the following:
(a) $10^{10} \bmod 12$
(b) $x^{10^{10}} \bmod (x^{12}-1)$
(c) $(x^{15}-1) \bmod x^{10}-1$.

8.3. Is the "mod" operation associative and/or commutative? That is, are the identities

$$(P \bmod Q) \bmod R = P \bmod (Q \bmod R)$$
$$P \bmod Q = Q \bmod P$$

true in general?

8.4. Show that if $m \in \{0,1,\ldots,n-1\}$, then $(j+m) \bmod n = i$ if and only if $m = (i-j) \bmod n$.

8.5. (a) Prove Lemma 1, part (i).
(b) Prove Lemma 1, part (ii).
(c) Prove Lemma 1, part (iii).
(d) Prove Lemma 1, part (iv).
(e) Prove Lemma 1, part (v).

8.6. Give a formal proof that if $\mathcal{C}$ is a cyclic code, and if $C(x) \in \mathcal{C}$, then for all $i \geq 1$, $x^i C(x) \bmod x^n-1 \in \mathcal{C}$. Hint: Use Lemma 1(iv).]

8.7. Find the G_1 and H_1 matrices for the $(8,4)$ cyclic code over $GF(3)$ with generator polynomial $g(x) = (x^2+1)(x^2+x+1)$. (See Corollary 1 to Theorem 8.3.)

8.8. Find the G_2 and H_2 matrices for the $(8,4)$ cyclic code over $GF(3)$ with generator polynomial $g(x) = (x^2-1)(x^2+x+1)$. (See Corollary 2 to Theorem 8.3.)

8.9. (a) Show that the following matrices are generator and parity-check matrices for an (n,k) cyclic code with parity-check polynomial $h(x)$, (and "reversed" parity-check polynomial $\tilde{h}(x)$):

$$G_3 = [x^j \tilde{h}(x)]_{j=0}^{n-1} \qquad \text{(columns)}$$
$$H_3 = [x^{i+k} - x^{i+k}\tilde{h}(x)]_{i=0}^{r-1} \qquad \text{(rows.)}$$

(b) Use this result to find generator and parity-check matrices for the $(7, 3)$ cyclic code with $g(x) = x^4 + x^3 + x^2 + 1$.

8.10. This problem concerns the matrix H_1 of Corollary 1 to Theorem 8.3.

(a) If the syndrome $\mathbf{S}$ of the vector $\mathbf{R} = (R_0, R_1, \ldots, R_{n-1})$ is calculated as $\mathbf{S}^T = H_1\mathbf{R}^T$, show that the generating functions $R(x) = R_0 + R_1x + \cdots + R_{n-1}x^{n-1}$ and $S(x) = S_0 + S_1x + \cdots + S_{r-1}x^{r-1}$ are related by

$$S(x) = \frac{[R(x)h(x)] \bmod x^n - [R(x)h(x)] \bmod x^k}{x^k},$$

(which is a just clumsy way of saying that $(S_0, S_1, \ldots, S_{r-1})$ are the coefficients of $x^k, x^{k+1}, \ldots, x^{n-1}$ in the product $R(x)h(x)$.)

(b) Using the result of part (a) find the syndrome of the vector $\mathbf{R} = [1001011]$ with respect to the "H_1" matrix for the cyclic code with $g(x) = x^4 + x^3 + x^2 + 1$. (Cf. Example 8.7).

8.11. Show that the dual code of a cyclic code is also a cyclic code. If an (n, k) cyclic code has generator polynomial $g(x)$, and parity-check polynomial $h(x)$, what are the generator and parity-check polynomials of the dual code? [Hint: Refer to Corollary 1 of Theorem 8.3.] Illustrate your results by finding the generator and parity-check polynomials for the dual code for the cyclic code of Examples 8.2 and 8.4.

8.12. Write each of the 9 codewords in the $(4, 2)$ cyclic code of Example 8.3 as a multiple of the generator polynomial $g(x) = x^2 + 2$.

8.13. This problem asks you to count the number of binary block codes codes of length 7 with 16 codewords, with various restrictions.

(a) How many such codes, total, are there?
(b) How many such *linear* codes?
(c) How many such *linear* codes, with $d_{\min} = 3$?
(d) How many such *cyclic* codes?
(e) How many such *cyclic* codes, with $d_{\min} = 3$?

8.14. Consider the $(7, 4)$ cyclic Hamming code with $g(x) = x^3 + x + 1$, as used to correct *erasures*. Since its minimum distance is 3, we know that it will correct all patterns of 2 or fewer erasures. However, it will also correct some patterns of *three* erasures. For example, the codeword $g(x) = [1101000]$ with erasures in positions 1, 3, and 6 is $\mathbf{R} = [1 * 0 * 00*]$. This pattern of three erasures is correctable, because of the 16 codewords in the code, only [1101000] agrees with $\mathbf{R}$ in the unerased positions. However, not all patterns of three erasures are correctable: e.g., $[**0*000]$ could be either [1101000] or [0000000]. Here is the problem: of the $\binom{7}{3} = 35$ possible patterns of three erasures, how many are correctable, and how many are not?

8.15. How many binary cyclic codes of length n are there, for $1 \le n \le 20$? How many of these are improper?

8.16. How many binary cyclic codes of length 63 are there? How many are improper?

8.17. For which values of n in the range $3 \le n \le 20$ are there exactly 4 cyclic codes of length n over F_2?

8.18. What fraction of linear codes of length 4 over F_2 are cyclic?

8.19. Find a formula for the number of cyclic codes of length 2^m over F_2.

8.20. Over the field $GF(3)$, the polynomial $x^8 - 1$ factors as $x^8 - 1 = (x + 1)(x + 2)(x^2 + 1)(x^2 + x + 1)(x^2 + 2x + 1)$. For each k in the range $0 \le k \le 8$, determine the number of $(8, k)$ cyclic codes over $GF(3)$.

8.21. In Example 8.8, we saw that for the $(4, 2)$ cyclic code over F_3 with $g(x) = x^2 - 1$, the generator and parity-check matrices of Corollaries 1 and 2 of Theorem 8.3 are equal, i.e., $G_1 = G_2$ and $H_1 = H_2$. Can you find a general class of cyclic codes for which this property holds?

8.22. Suppose that over a given field F, the polynomial $x^n - 1$ factors as

$$x^n - 1 = P_1(x)^{e_1} P_2(x)^{e_2} \cdots P_M(x)^{e_M},$$

where the $P_i(x)$'s are distinct irreducible polynomials. In terms of $e_1, e_2, \ldots, e_M$, how many cyclic codes of length n over F are there?

8.23. Explain each of the "comments" in the table in Example 8.9.

8.24. Prove that over the field F_2, $x^3 + x + 1 \mid x^{7m} - 1$, for all $m \geq 1$.

8.25. There are eight fourth-degree binary polynomials of the form $g(x) = x^4 + g_3x^3 + g_2x^2 + g_1x + 1$. For each of these polynomials answer the following:
(a) What is the period of $g(x)$?
(b) If n is the period you found in part (a), What is the k and d for the corresponding cyclic code?

8.26. (K. Sivarajan) Show that a binary cyclic code is a capable of correcting a single error, i.e., $d_{\min} \geq 3$, if and only if it is a proper cyclic code.

8.27. At the end of Section 8.1, we discussed briefly *improper* cyclic codes, i.e., those for which the period of the generator polynomial $g(x)$ is less than n. In this problem, we will investigate cyclic codes for which the *parity-check* polynomial has period less than n. Thus let C be an (n, k) cyclic code, and let $x^n - 1 = g(x)h(x)$, where $g(x)$ is the generator polynomial and $h(x)$ is the parity-check polynomial, for C. Suppose further that the period of $h(x)$ is $n_0 < n$, and that C_0 is the (n_0, k) cyclic code with generator polynomial $g(x) = (x^{n_0} - 1)/h(x)$.
(a) What is the parity-check polynomial for C_0?
(b) If d_0 is the minimum distance of C_0, what is the minimum distance of C?
(c) Show that $C = (n/n_0)C_0$, where "jC_0" denotes the code of length n obtained from C_0 by repeating each codeword j times.

8.28. For each j in the range $1 \leq j \leq 16$, find the length, dimension, and minimum distance of the (proper) binary cyclic code with generator polynomial $(x + 1)^j$. (Hint: The result of the previous problem will be helpful.)

8.29. If $g(x)$ has period n and degree r, then for any $m \geq 1$, $g(x) \mid x^{mn} - 1$ and so by Theorem 8.3b, $g(x)$ generates an $(nm, nm - r)$ cyclic code. (For $m \geq 2$, this code is improper.)
(a) Show that a vector $C = [C_0, C_1, \ldots, C_{nm-1}]$ is in this code if and only if $\sum_{i=0}^{nm-1} C_i x^{i \bmod n} \equiv 0 \pmod{g(x)}$.
(b) Show that the condition in part (a) is equivalent to saying that

$$[C_0, C_1, \ldots, C_{n-1}] + [C_n, C_{n+1}, \ldots, C_{2n-1}] + \cdots + [C_{(m-1)n}, C_{(m-1)n+1}, \ldots, C_{mn-1}]$$

is in the $(n, n - r)$ proper cyclic code generated by $g(x)$.
(c) How many words of weight 2 are in this code?

8.30. In Figure 8.3 we see a "mod $g(x)$" circuit for a *monic* polynomial $g(x)$. Explain how to modify this circuit for a non-monic polynomial $g(x) = g_0 + g_1x + \cdots + g_rx^r$, with $g_r \neq 0, 1$.

8.31. (a) Show that for the circuit of Figure 8.4(a), the relationship between the present state polynomial $S(x)$ and the next state polynomial $S'(x)$ is

$$S'(x) = (xS(x) + sx^r) \bmod g(x),$$

where s is the input signal.

(b) Use the result of part (a) to show that if the circuit of Figure 8.4(a) is initialized with $s_0 = s_1 = \cdots = s_{r-1} = 0$, and then given the input sequence $a_0, a_1, \ldots$, then after the tth tick the state polynomial will be

$$S_t(x) = \sum_{j=0}^{t} a_j x^{r+t-j} \bmod g(x).$$

8.32. Consider the polynomial $g(x) = x^3 + 2x^2 + 2$ over the three-element field $GF(3)$.
(a) What is the period of this polynomial?
(b) If n denotes the period you found in part(a), find a parity-check matrix for the cyclic code of length n generated by $g(x)$.
(c) Does this code have any words of weight 2? If not, explain why not. If so, explicitly list all such words.
(d) Draw a systematic shift-register encoder for this code.

8.33. Design three different shift-register encoders for the $(15, 7)$ binary cyclic code with $g(x) = x^8 + x^7 + x^6 + x^4 + 1$.

8.34. This problem concerns the $(31, 26)$ cyclic Hamming code with $g(x) = x^5 + x^2 + 1$.
(a) Find a systematic (i.e., of the form $[A|I_5]$) parity-check matrix for this code.
(b) Design a decoding circuit for this code.
(c) Use your decoder from part (b) to decode the received word

$$\mathbf{R} = [11111111111100000000000000000000].$$

8.35. Find a formula for the number of words of weight 3 in the $(2^m - 1, 2^m - m - 1)$ cyclic Hamming code defined by the parity-check matrix (8.9).

8.36. Let $g(x)$ be a polynomial of degree r over a finite field with q elements, such that $g(0) \neq 0$.
(a) Show that the period of $g(x)$ is $\leq q^r - 1$.
(b) Show that if the period of $g(x)$ is $q^r - 1$, then $g(x)$ is necessarily irreducible. [Note: An ireducible polynomial of degree r and period $q^r - 1$ is called a *primitive* polynomial.]

8.37. Consider the $(15, 11)$ binary cyclic Hamming code with generator polynomial $g(x) = x^4 + x + 1$.
(a) Write out a 4×15 binary parity-check matrix.
(b) Write out a 1×15 F_{16} parity-check matrix.

8.38. Prove that if α is a primitive root in the field $GF(2^m)$ with minimal polynomial $g(x)$, and if the expansion of the power α^j is

$$\alpha^j = \sum_{i=0}^{m-1} h_{i,j}\alpha^i \qquad \text{for } j = 0, 1, \ldots, n-1,$$

then the $m \times 2^m - 1$ matrix $H = (h_{i,j})$ for $i = 0, 1, \ldots, m-1$, $j = 0, 1, \ldots, 2^m - 2$ is the parity-check matrix for a cyclic Hamming code with polynomial $g(x)$.

8.39. Let $g(x)$ be a binary primitive polynomial of degree m, and let H be a $m \times 2^m - 1$ binary matrix which is a parity-check matrix for a $(2^m - 1, 2^m - m - 1)$ Hamming code. How many of the $(2^m - 1)!$ possible permutations of the columns of H yield a parity-check matrix for the *cyclic* Hamming code with $g(x)$ as the generator polynomial?

8.40. In Table 8.3, except for the $m = 1$ entry, there are no polynomials of even weight. Explain why this is so, i.e., why no polynomial of even weight can ever be primitive.

8.41. (Dual codes to cyclic Hamming codes—also called *cyclic simplex codes*, or *maximal-length shift register codes*). Let $\mathcal{C}_k$ be an (n, k) (proper) cyclic code whose parity check polynomial $h(x)$ is a primitive polynomial of degree k, and let G_1 be the generator matrix for $\mathcal{C}_k$ described in Corollary 1 to Theorem 8.3.

(a) Show that $n = 2^k - 1$.
(b) Describe an efficient shift-register encoder for $\mathcal{C}_k$.
(c) Show that the set of nonzero codewords of $\mathcal{C}$ is exactly equal to the set of cyclic shifts of the first row of G_1. Illustrate this result for $\mathcal{C}_4$ using the primitive polynomial $x^4 + x + 1$ from Table 8.3.
(d) Find the weight enumerator for $\mathcal{C}_k$.

8.42. Find a formula for the number of *ordinary* (i.e., not cyclic) bursts of length b, over a two-letter alphabet (cf. Theorem 8.8.), for $1 \leq b \leq n$.

8.43. Generalize Theorem 8.8 to a q-letter alphabet.

8.44. Extend Theorem 8.8 to cover the case $b > (n + 1)/2$.

8.45. Generalize Theorem 8.9 and its Corollaries to a q-letter alphabet.

8.46. Generalize Theorem 8.10 and its Corollary to cover the case of a q-letter alphabet.

8.47. Why is the case $b = 0$ not covered in Theorem 8.9?

8.48. Why do you suppose the two versions of the Abramson bound are called "strong" and "weak"?

8.49. Are the Abramson bounds (Corollary to Theorem 8.9) ever tight for the binary repetition codes?

8.50. Is the Reiger bound ever tight for the Abramson codes?

8.51. In some burst-error-correcting applications, it is sufficient to correct bursts that occur only in certain special locations. The most important example of this is what is called *phased burst error correction.* In phased burst error correction, the block length n is a multiple of b, and bursts of length b may occur only at locations which are multiples of b. A code capable of correcting all phased bursts of length b is called a *phased* burst-b error correcting code. For example, a phased burst-3 error-correcting code of length 12 must be able to correct any burst of length 3 which occurs at locations 0, 3, 6, or 9.

(a) Show that Theorem 8.10, and hence the Reiger bound, applies to phased burst error-correcting codes.
(b) Show that if a code is capable of correcting t phased bursts of length b, then $r \geq 2tb$.

8.52. Consider a $(20, 12)$ binary linear code in which the parity-checks are described by the following array:

$$\begin{bmatrix} C_0 & C_4 & C_8 & C_{12} & C_{16} \\ C_{17} & C_1 & C_5 & C_9 & C_{13} \\ C_{14} & C_{18} & C_2 & C_6 & C_{10} \\ C_{11} & C_{15} & C_{19} & C_3 & C_7 \end{bmatrix}$$

In a given codeword $[C_0, C_1, \ldots, C_{19}]$, the components occupying the upper left 3×4 portion of the array carry the information, and the components forming the right column and bottom row of the array are parity positions. The parity check rules are that the sum of the components in each row and column of the array is zero. Thus for example, $C_0 + C_4 + C_8 + C_{12} + C_{16} = 0$ (first row sum), and $C_4 + C_1 + C_{18} + C_{15} = 0$ (second column sum).

(a) Show that this code can correct all phased bursts of length 4.
(b) Show that the corresponding 3×4 array code is *not* capable of correcting all phased burst of length 3.
(c) Generalize, and show that a $b \times (b+1)$ array code can correct all phased bursts of length b, if and only if $b + 1$ is a prime number.

8.53. (Burton codes.)

(a) Let $g(x) = (x^b - 1)p_b(x)$, where $p_b(x)$ is an irreducible polynomial of degree b and period n_1. Show that $g(x)$ generates an $(n, n-2b)$ cyclic code with $n = \text{lcm}(n_1, b)$ which is capable of correcting all *b-phased bursts* of length b. (See previous problem.)

(b) Consider the sequence of $(nj, (n-2b)j)$ cyclic codes obtained by interleaving the codes in part (a), and let b_j denote the corresponding (ordinary, i.e., unphased) burst error-correcting capability. Show that $\lim_{j\to\infty} \frac{b_j}{2bj} = \frac{1}{2}$, so that these code asymptotically meet the Reiger bound.

8.54. Consider a binary cyclic $b = 3$ burst error correcting code of length 31.

(a) Use the Abramson and the Reiger bounds to estimate the minimum needed redundancy.

(b) Do you think there is such a code with the redundancy predicted in part (a)? Explain fully.

8.55. Show that if $g(x)$ is the generator polynomial for a an (n, k) cyclic code, and if $(x + 1)$ is not a divisor of $g(x)$, then $g'(x) = (x+1)g(x)$ is the generator polynomial for an $(n, k-1)$ code, which equals the original code minus its words of odd weight.

8.56. Decode the following noisy codewords from the $(7, 3)$ Abramson code of Example 8.16:

(a) [0101010]
(b) [1111111]
(c) [1110010]
(d) [1101000]
(e) [1011111]

8.57. Decode the following noisy codewords from the $(15, 9)$ $b = 3$ burst error-correcting code of Example 8.17:

(a) [000001001111000]
(b) [101110001100000]
(c) [111001000001111]
(d) [110100000101010]
(e) [111100000000000]

8.58. Verify the assertion, made in Example 8.20, that the polynomial $g(x) = (x^4+x+1)(x^4+x^3+x^2+x+1)$ generates a $(15, 7)$ $b = 4$ burst error correcting cyclic code.

8.59. Do either of the following two polynomials generate a $(15, 7)$ $b = 4$ burst error correcting cyclic code?

(a) $g(x) = (x^4 + x^3 + 1)(x^4 + x^3 + x^2 + x + 1)$
(b) $g(x) = (x^4 + x + 1)(x^4 + x^3 + 1)$.

8.60. Verify the assertion, made in Example 8.20, that the polynomial $g(x) = x^{12} + x^9 + x^6 + x^3 + 1$ generates a $(15, 3)$ $b = 6$ burst error correcting cyclic code.

8.61. In the $(15, 9)$ code of Example 8.17, 61 of the 64 cosets are accounted for by the bursts of length ≤ 3. Describe the syndromes of the missing three cosets. What is the length of the shortest burst pattern in each of these cosets?

8.62. In Example 8.17, we saw that $g(x) = (x^4 + x + 1)(x^2 + x + 1)$ generates a $(15, 9)$ $b = 3$ burst error-correcting cyclic code. Does $g'(x) = (x^4 + x^3 + 1)(x^2 + x + 1)$ work as well? How about $g''(x) = (x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)$?

8.63. Generalize Example 8.17 by showing that the $(2^m - 1, 2^m - m - 3)$ binary cyclic code with generator polynomial $g(x) = p(x)(x^2 + x + 1)$, where $p(x)$ is a primitive polynomial of degree m such that $1 + x \equiv x^a \pmod{p(x)}$, where $a \bmod 3 \neq 2$, is a $b = 3$ burst error correcting code that meets the Abramson bound.

8.64. If the code $\mathcal{C}$ meets the Abramson bound, will the depth-j interleaving of $\mathcal{C}$ also meet the Abramson bound? If so, give a proof. If not, give a counterexample.

8.65. Prove the result (8.10) about the right cyclic shift of an interleaved vector.

8.66. Prove that the $(n, 1)$ binary repetition code with $g(x) = (x^n + 1)/(x + 1)$ is a strong burst-b error correcting code with $b = (n - 1)/2$, if n is odd.

8.67. Give a formal proof of Theorem 8.11.

8.68. (See Examples 8.11 and 8.21.) For the $(7, 3)$ Abramsom code, show that every burst with pattern 111 has the same syndrome as some burst with pattern 1.

8.69. Prove Theorem 8.14 in the case that the code generated by $g_1(x)$ is a weak (i.e. not strong) burst-b error-correcting code.

8.70. As mentioned in Example 8.24, if $g(x)$ is a polynomial of degree r and period n, and if $n_0 < n$, the $(n_0, n_0 - r)$ *shortened* cyclic code with generator polynomial $g(x)$ consists of all vectors $[C_0, C_1, \ldots, C_{n_0-1}]$ whose generating function $C_0 + C_1 x + \cdots + C_{n_0-1} x^{n_0-1}$ is a multiple of $g(x)$.

(a) Show that this is a linear code.

(b) If $g(x) = x^6 + x^4 + x + 1$ (see Example 8.23), find generator and parity-check matrices for the $(16, 10)$ shortened cyclic code generated by $g(x)$.

8.71. Why is there no entry for $b = 2$ in Table 8.4?

8.72. For each of the n's and b's in Table 8.4, estimate the redundancy needed, using the Abramson and Reiger bounds, and compare this value to the actual redundancy of the given Fire code.

8.73. Consider the $(35, 27)$ $b = 3$ Fire code from Table 8.4. Decode the following received words, using a "$b_1 = 3, b_2 = 3$" decoder (see Theorem 8.13)

$$\mathbf{R}_1 = [11010110111010110111010110100000111]$$
$$\mathbf{R}_2 = [01001110101101000001101011011010110]$$

Now assume a "$b_1 = 2, b_2 = 4$" decoder, and decode the same two received words.

8.74. Explain in detail how you would modify a burst-trapping decoding circuit for a burst b-error correcting Fire code, so that it is capable of *correcting* all bursts of length $\leq b_1$, and also *detecting* all bursts of length between $b_1 + 1$ and $2b - b_1$ (inclusive). (Here b_1 is an integer satisfying $1 \leq b_1 < b$.) Give a sketch for your modified decoder.

8.75. Show that if *no* errors occur, the algorithm in Figure 8.10 will still work.

8.76. Explain how to modify the algorithm of Figure 8.10 so that a burst of length $\geq b + 1$ which does not have the same syndrome as a burst of length $\leq b$ will be detected.

8.77. The decoding algorithm described in Figure 8.10 is for an $(n, n - r)$ burst-error correcting cyclic code with generator polynomial $g(x)$. It requires just n executions of the `for` loop at lines 4–8 to complete the decoding of one length n codeword. It will also work for a *shortened* $(n_0, n_0 - r)$ cyclic code with the same generator polynomial, but it still requires n, rather than n_0, executions of the `for` loop. If the code has been drastically shortened, this is unacceptable. For example, the IBM code described in Example 8.24 has $n_0 = 152552$ but $n = 446676598771$, and a decoder that required 446676598771 steps to decode one word of length 152552 would be useless. Fortunately, there is a simple modification of the decoding algorithm

that allows the decoding of the shortened code in just n_0 steps. It is described in the following pseudocode listing. (Note the changes from the Figure 8.10 algorithm at lines 7 and 8.)

```
    /* Burst Error Trapping Decoding Algorithm for Shortened Cyclic Codes */
 {
1.     input R(x);
2.     R_0(x) ← R(x);
3.     S_0(x) ← R_0(x) mod g(x);
4.     for(j = 0 to n_0 − 1) {
5.          if (S_j(0) ≠ 0 and deg S_j(x) ≤ b − 1)
6.            R_j(x) ← R_j(x) − S_j(x);
7.          R_{j+1}(x) ← [x^{−1} R_j(x)] mod x^{n_0} − 1;
8.          S_{j+1}(x) ← [x^{−1} S_j(x)] mod g(x);
       }
9.     output R_{n_0}(x);
 }
```

(a) Explain why this algorithm works.

(b) Consider the $(30, 16)$ *shortened* Fire code with $g(x) = (x^5 + x^2 + 1)(x^9 + 1)$, and use the given algorithm to decode the following received word:

$$\mathbf{R} = [111101001110100011011011110010].$$

(c) Describe a shift-register implememtation of this algorithm. [Hint: You will require both a "$S(x) \rightarrow xS(x) \bmod g(x)$" shift register and a "$S(x) \rightarrow x^{-1}S(x) \bmod g(x)$" shift register.]