5.3 BOUNDS ON MINIMUM HAMMING DISTANCE FOR BLOCK CODES

We have witnessed the key role of Hamming distance between codewords and the special importance of the minimum distance $d_{\min}$. The central problem of block coding theory has been the description of codes with large (or largest) $d_{\min}$ for a given n, k, and q. For small codes, the best codes can be found by exhaustive search of generator matrices if necessary. The range of parameters for which this exhaustive search can be performed is rather limited, however, and in general we must settle for upper and/or lower bounds on $d_{\min}$. In some cases these will be equal, implying we have described the error-correcting power without actually constructing the code.

Tables of bounds on $d_{\min}$ are provided by MacWilliams and Sloane [3], Helgert and Stinaff [10], and Verhoeff [11]. In Table 5.2 we include an abbreviated version of the table of tightest known bounds on $d_{\min}$ for linear binary codes listed in [11]; the reader may find it instructive to attempt constructions of some of the smaller codes by finding appropriate systematic-form generator matrices whose row sums give the stated minimum weight. Equivalently, we may attempt construction of parity check matrices of a certain size such that all sets of $d_{\min} - 1$ columns are linearly independent. There are numerous open questions; it is intriguing to see in Table 5.2 that for $n = 25$ and $k = 16$ the question is still open as to whether the largest attainable is $d_{\min} = 4$ or 5.

In this section, we will develop some of the important general bounds. These bounds may be used in assessing the possible existence of a code having certain parameters, or in establishing how good a known code is against theoretical limits. In addition, asymptotic results on code performance for large block length follow readily from these bounds. We begin with three upper bounds on $d_{\min}$, referred to as the Hamming bound, the Singleton bound, and the Plotkin bound.

5.3.1 Hamming (Sphere-packing) Bound

Consider (n, k) codes over GF(q). We shall visualize the codewords as points in the n-dimensional space of n-tuples over GF(q) and imagine centering a "ball" of radius t Hamming units at each of the q^k codewords in this space, as schematically shown in Figure 5.3.1. These balls will be regarded as minimum distance decoding regions for the various codewords. We ask for the largest t such that it is possible that balls do not intersect (even share points) so that a code will be t error correcting. The answer follows from a simple volumetric calculation and provides an indirect answer for the largest possible $d_{\min}$, since $t = \lfloor (d_{\min} - 1)/2 \rfloor$.

A ball of radius t includes all vectors at Hamming distance $0, 1, \ldots, t$ from a reference vector, and the number of such vectors is called the n-dimensional *volume*:

$$V_t^{(n)} = \sum_{j=0}^{t} C_j^n (q-1)^j. \tag{5.3.1}$$

The balls cannot be disjoint unless the total volume of n-space is at least as large as the volume contained in q^k decoding regions. Thus, it is *necessary* for a code to be t error

TABLE 5.2 LISTING OF LARGEST ACHIEVABLE d_{min} VERSUS n AND k FOR LINEAR BINARY CODES

$n \mid k$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	1																	
3	2	1																
4	2	2	1															
5	3	2	2	1														
6	4	3	2	2	1													
7	4	4	3	2	2	1												
8	5	4	4	2	2	2	1											
9	6	4	4	3	2	2	2	1										
10	6	5	4	4	3	2	2	2	1									
11	7	6	5	4	4	3	2	2	2	1								
12	8	6	6	4	4	4	3	2	2	2	1							
13	8	7	6	5	4	4	4	3	2	2	2	1						
14	9	8	7	6	5	4	4	4	3	2	2	2	1					
15	10	8	8	7	6	5	4	4	4	3	2	2	2	1				
16	10	8	8	8	6	6	5	4	4	4	2	2	2	2	1			
17	11	9	8	8	7	6	6	5	4	4	3	2	2	2	2	1		
18	12	10	8	8	8	7	6	6	4	4	4	3	2	2	2	2	1	
19	12	10	9	8	8	8	7	6	5	4	4	4	3	2	2	2	2	1
20	13	11	10	9	8	8	8	7	6	5	4	4	4	3	2	2	2	2
21	14	12	10	10	8	8	8	8	7	6	5	4	4	4	3	2	2	2
22	14	12	11	10	9	8	8	8	8	7	6	5	4	4	4	3	2	2
23	15	12	12	11	10	9	8	8	8	8	7	6	5	4	4	4	3	2
24	16	13	12	12	10	10	8–9	8	8	8	8	6	6	4–5	4	4	4	3
25	16	14	12	12	11	10	9–10	8–9	8	8	8	6	6	5–6	4–5	4	4	4

From Verhoeff [11].

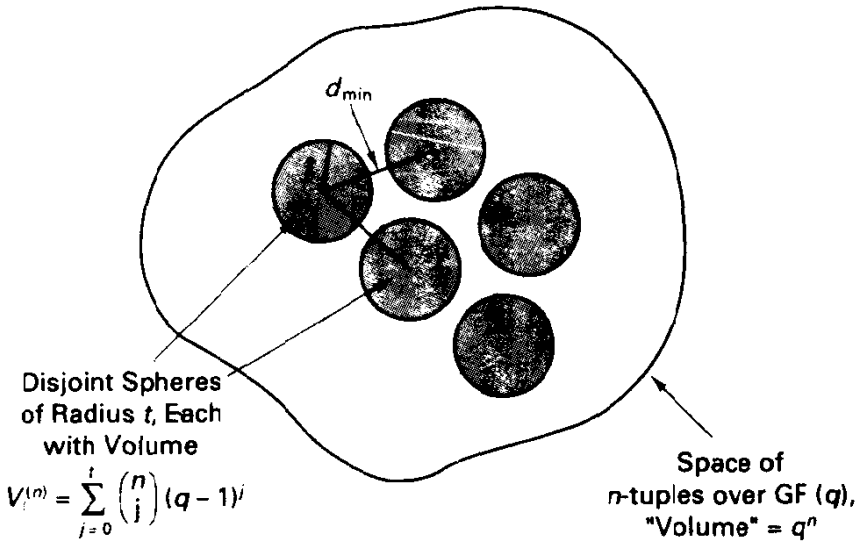


Figure 5.3.1 Illustration of Hamming or sphere-packing bound.

correcting that

$$q^k \sum_{j=0}^{t} C_j^n (q-1)^j \leq q^n, \tag{5.3.2a}$$

or, equivalently,

$$q^{n-k} \geq \sum_{j=0}^{t} C_j^n (q-1)^j. \tag{5.3.2b}$$

Still another form of the bound is

$$n - k \geq \log_q \sum_{j=0}^{t} C_j^n (q-1)^j. \tag{5.3.3}$$

Notice that this argument does not involve any structural properties of the code, but only the code size and block length, so the bound holds for nonlinear (n, k) codes as well.

Example 5.11 Application of the Hamming Bound

With $n = 15$, $k = 8$, and $q = 2$, we might ask for an upper bound on achievable values of t. Substitution in (5.3.2) yields

$$\sum_{j=0}^{t=2} C_j^{15} = 121 \leq 2^{(15-8)} = 128, \tag{5.3.4}$$

so $t = 2$ is allowed by the Hamming bound. A similar check will show, however, that $t = 3$ is impossible. In fact, a (15, 8) double-error-correcting *nonlinear* code exists and has been described by Nordstrom and Robinson [12]. This code and its generalizations provide the most notable examples where nonlinear codes offer any advantage over linear codes. From Table 5.2, the best linear code with $n = 15$ and $t = 2$ is a BCH code having $k = 7$. That linear code thereby provides only half as dense a packing of spheres in 15-dimensional space as the nonlinear code.

Perfect and quasi-perfect codes

The case of equality in (5.3.2) presents interesting possibilities, for it suggests that the total volume is exactly divisible among q^k balls of radius t, leaving no interstitial space. Of course, having exactly the correct total volume does not ensure that such volume can be divided into radius-t balls centered about each code point without overlap. If, however, an (n, k) code can be demonstrated to have decision zones of radius t, with equality in (5.3.2), then the code is termed a ***perfect*** code. In a perfect code, *every* received n-tuple is within t Hamming units of one and only one codeword. This is obviously a special partitioning of n-dimensional space, and such codes are also called *close packed.*

Perfect codes are in fact very rare. The $(n, 1)$ repetition codes of Example 5.7 are perfect codes for n odd and any q, with $t = (n-1)/2$. When $t = 1$, equality is obtained in (5.3.2) for q-ary codes when $1 + n(q-1) = q^{n-k}$, or if $n = (q^{n-k} - 1)/(q-1)$. Since this last equation has integer-valued solutions for any q and $n-k$, perfect $t = 1$ codes then might exist with these lengths. For example, with $q = 16$, setting $n - k = 3$ points to the possibility of a perfect $t = 1$ code with $n = (16^3 - 1)/15 = 273$, so the code would be a $(273, 270)$ code over GF(16). An infinite family of such codes (Hamming codes) does in fact exist, defined by the parity check matrix construction as described in Section 5.2.5. They may also be described as cyclic codes, as demonstrated in Section 5.4.

Beyond these examples of perfect codes, there are surprisingly only two others: a $(23, 12)$ $t = 3$ binary code and a $(11, 6)$ ternary $(q = 3)$ code with $t = 2$, both due to Golay [13]. Others have been conjectured; for example $(n, k, q) = (90, 78, 2)$ provides equality in (5.3.2) with $t = 2$, but such a perfect code does not exist. Tietavainen [14] has put the issue of other perfect codes over arbitrary finite fields to rest in the negative. The paucity of perfect codes should not be regarded as discouraging, however, for such elegant mathematical objects are not essential to good performance. We will settle simply for good spatial packings.

The next best arrangement is a ***quasi-perfect*** code, of which there are many. A code is t error correcting quasi-perfect if balls of radius t located about each code point are distinct, and all remaining points in the space of n-tuples are at most $t + 1$ units from a code point. In terms of the standard array for the code, this implies that the coset leaders include all error patterns up through weight t, some patterns of weight $t + 1$, and none of larger weight. This is clearly a good packing of codewords in space, if not perfect, and can be shown to yield a code with minimum error probability for a specified n, k, and q. An example is provided by the $(6, 3)$ binary code studied in Example 5.8.

5.3.2 Singleton Bound (15): $d_{\min} \leq n - k + 1$

This upper bound on distance for linear codes is easy to demonstrate, but is usually not attainable. Recall that the minimum distance of a linear code is equal to the minimum nonzero weight of a codeword. A nonzero codeword may have as few as one nonzero symbol in the k message positions of a codeword, and the largest possible weight of the remaining $n - k$ parity symbols is obviously $n - k$. Thus, the minimum weight, and hence $d_{\min}$, cannot exceed $n - k + 1$. Codes that achieve the upper bound are said to be

maximum distance separable (MDS), or simply ***maximum***, and the most notable codes to do so are the Reed–Solomon (RS) codes. These will be discussed in cyclic form in Section 5.4. The only *binary* MDS codes are the $(n, 1)$ repetition codes with $d_{\min} = n$ and their duals, the $(n, n-1)$ codes with $d_{\min} = 2$. The (5, 3) code over GF(4) presented in Example 5.9 is an MDS code, since $d_{\min} = n - k + 1 = 3$.

An important implication of the Singleton bound is provided by normalizing $d_{\min}$ by block length n, in which case we have that $d_{\min}/n \leq (n-k+1)/n = (1-R) + 1/n$. For long-block-length codes, we find that $d_{\min}/n$ essentially cannot exceed $1 - R$. If we realize that the guaranteed error-correcting capability t is upper-bounded by $d_{\min}/2$, then we find that the guaranteed fractional error-correcting capability, t/n, of a linear code is upper-bounded (asymptotically for large n) by one-half the code redundancy $(1 - R)/2$. Long $R = \frac{1}{2}$ codes, over any alphabet, therefore cannot hope to correct more than 25% of the incorrect symbols in a codeword.

Another interesting interpretation of the Singleton bound is that t is essentially upper bounded for long codes by $(n - k)/2$, which is to say that *at least* two parity symbols are required for every unit of guaranteed error-correcting capability.

5.3.3 Plotkin Bound (16)

In a linear (n, k) code over GF(q), in every column of a table of the q^k codewords, each symbol in GF(q) appears exactly q^{k-1} times (provided none of the columns of the generator matrix is all zero). For example, note that every column in the code listing of Figure 5.2.2 has exactly eight 1's and eight 0's. The total weight of all codewords is thus $n(q-1)q^{k-1}$. Furthermore, since there are $q^k - 1$ nonzero codewords in the code, the average codeword weight is

$$w_{\text{ave}} = \frac{n(q-1)q^{k-1}}{q^k - 1}. \tag{5.3.5}$$

The minimum weight, equivalent to $d_{\min}$, must be no larger than the average weight. Thus, we have that for q-ary codes

$$d_{\min} \leq \frac{n(q-1)q^{k-1}}{q^k - 1}, \tag{5.3.6}$$

which provides another upper bound on $d_{\min}$ in terms of n, k, and q.

Upon dividing both sides of (5.3.6) by n, we have that

$$\frac{d_{\min}}{n} \leq \frac{(q-1)q^{k-1}}{q^k - 1}. \tag{5.3.7}$$

In the limit of large k, we find that minimum distance as a fraction of block length is essentially upper-bounded by $(q-1)/q$, independent of rate R. This bound thus is usually very loose, especially at high rates and for large q.

Other upper bounds on minimum distance, tighter than the three bounds presented here but more cumbersome to formulate, have been obtained by Elias [17] and McEliece et al. [18]. These are discussed in [1] and [5]. The exercises also address an upper bound on $d_{\min}$ due to Griesmer.

Next, we turn to the opposite problem, and for a given set of code parameters, seek existence results, that is, define the parameter range within which codes are guaranteed to exist. We can think of this process as establishing lower bounds on $d_{\min}$ for a given n, k, and q. We will develop similar bounds due to Gilbert [19] and Varshamov [20], which, although distinct, give the same asymptotic (large block length) guarantees for codes.

5.3.4 Gilbert Bound

Now suppose we are given a block length n and target distance d, with $2 \leq d \leq n$, and we wish to establish a guaranteed dimension k. The development is similar to that of the Hamming bound, in essence a geometric argument involving balls in n-dimensional space. We will formulate the proof for linear codes, although the result holds for nonlinear codes with equal rate.

We try to construct a generator matrix $\mathbf{G}$ with eventual rank k, for which all (nonzero) linear combinations of rows have weight d or larger. We begin by removing from the space of n-tuples over GF(q) all vectors within $d-1$ Hamming units of the $\mathbf{0}$ vector. There are $V_{d-1}^{(n)}$ vectors in this ball. For the first row, $\mathbf{g}_1$, of the generator matrix, we select any remaining n-tuple. Note that it and all nonzero multiples of this vector have weight at least d. Surrounding each vector of the form $u_1\mathbf{g}_1$, we further remove all vectors within a radius $d-1$. From the remaining n-tuples, if any, select $\mathbf{g}_2$. We know that $u_1\mathbf{g}_1 + u_2\mathbf{g}_2 \neq \mathbf{0}$, by construction, so we have a two-dimensional linear code at this stage. Furthermore, all linear combinations of these two rows, where u_1 and u_2 are not both zero, will have weight at least d. Now around each of the q^2 vectors in this code, remove all $V_{d-1}^{(n)}$ vectors within radius $d-1$. If any remain, pick one as $\mathbf{g}_3$ and continue the process until all the q^n vectors are consumed by one of the balls of radius of $d-1$. This sequence is illustrated in Figure 5.3.2.

We note that when $\mathbf{g}_j$ is selected we have already removed q^{j-1} balls; in the worst case, these balls will be disjoint, but in general some overlap exists. Nonetheless, if k satisfies

$$q^{k-1}V_{d-1}^{(n)} < q^n, \tag{5.3.8}$$

then an (n, k) linear code over GF(q) with $d_{\min} \geq d$ is guaranteed to exist. [The possibility of significant sphere overlap allows codes to have possibly larger dimension than (5.3.8) might suggest.]

Rewriting (5.3.8) gives

$$\frac{1}{q}\sum_{j=0}^{d-1} C_j^n (q-1)^j < q^{n-k} \tag{5.3.9}$$

as the *sufficient condition* for existence of an (n, k) code with $d_{\min} = d$.

Example 5.12 Application of the Gilbert Bound

Suppose we seek a single-error-correcting binary code of block length 10 and wish to know what values of k are guaranteed. We take $n = 10$, $d = 3$, and $q = 2$, and the sufficient condition (5.3.9) becomes

$$\frac{1}{2}\sum_{j=0}^{2} C_j^{10} < 2^{10-k} \tag{5.3.10}$$

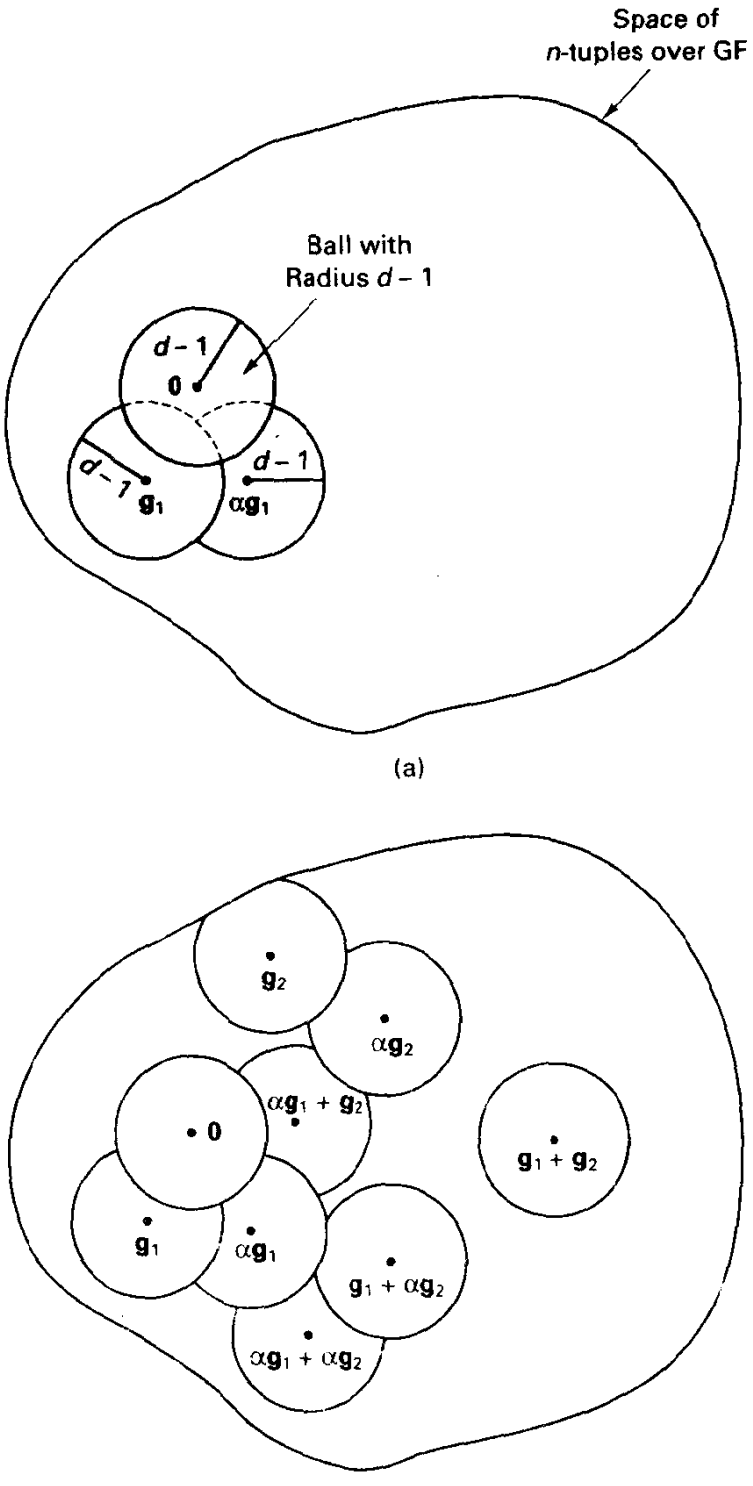


Figure 5.3.2 Gilbert bound argument with $q = 3$. (a) Prior to selection of $\mathbf{g}_2$; q balls removed; (b) prior to selection of $\mathbf{g}_3$; q^2 balls removed.

The left-hand side is 28, which implies (at least) that a $k = 5$ code exists with $d_{\min} \geq 3$, and hence with $t = 1$. On the other hand, the Hamming bound with $t = 1$ requires that $k \leq 6$. In fact, a (10, 6) code is achievable with $d_{\min} = 3$, as shown in Table 5.2.

5.3.5 Varshamov Bound

A slightly different existence statement is due to Varshamov [20], based on construction of the parity check matrix **H**. For this bound, we first fix the number of parity symbols $n - k = r$. (This is equivalent to fixing the dimension of the dual code.) We also adopt some target distance $2 \leq d \leq r + 1$. From Section 5.2, we know that, if we can construct an $n - k$ by n parity check matrix over GF(q) such that every set of $d - 1$ columns is linearly independent, then we have described an (n, k) code with $d_{\min} \geq d$.

We form the first r columns of **H** as the r by r identity matrix. Clearly, these columns are linearly independent. For the $(r + 1)$st column, we use any nonzero r-tuple that is not a linear combination of any $d-2$ previous columns. There are $\sum_{j=1}^{d-2} C_j^r (q-1)^j$ such linear combinations of the first r columns. In general, these are not all distinct, but, in any case, if this number of linear combinations is less than the number of nonzero $(n-k)$-tuples, then an $(r+1)$st column can be added, still maintaining linear independence of $d - 1$ columns.

We can pursue this extension to add at least an nth column if

$$\sum_{j=1}^{d-2} C_j^{n-1} (q-1)^j < q^{n-k} - 1 \tag{5.3.11}$$

or, equivalently, if

$$\sum_{j=0}^{d-2} C_j^{k+r-1} (q-1)^j < q^r. \tag{5.3.12}$$

Equation (5.3.12) is a sufficient condition for the existence of an $(n, n - r)$ linear code, with $d_{\min} \geq d$. As with the Gilbert bound proof, nondistinctness of linear combinations of columns generally means that the number of r-tuples is not exhausted so quickly, and achievable values for n, given $n-k$ and d, are perhaps larger than guaranteed by (5.3.12). It is worth noting that the bound just presented bears strong resemblance to the form of the Hamming bound, (5.3.2b), with an important difference on the range of the sum.

Example 5.13 Applications of the Varshamov Bound

Suppose that we pick $r = n - k = 5$, $d = 3$, and $q = 2$. The condition (5.3.12) is $1 + (k + 4) < 32$, implying that $k = 26$ is achievable. Thus, a (31, 26) code exists with $d_{\min} \geq 3$. Such a code is the Hamming code with these parameters and $d_{\min} = 3$.

As a second case, let us choose $r = n - k = 17$ and $d = 7$, with $q = 2$. Essentially, the question is "How large a value of k is guaranteed for codes with triple-error-correcting capability and that use 17 parity bits?" The sufficient condition becomes

$$\sum_{j=0}^{5} C_j^{k+16} < 2^{17} = 131{,}072. \tag{5.3.13}$$

Solution shows that the largest k is 12, so a (29, 12) code is known to exist with $d_{\min} \geq 7$. It will be seen in Section 5.4 that a (31, 16) binary BCH code has $d_{\min} = 7$, and this code

could be shortened to (29, 14) with the same minimum distance (see Section 5.6). Thus, we can actually construct a code with $n = 29$ having higher rate than guaranteed by the Varshamov bound. Verhoeff's complete tables [11] show in fact that a (29, 14) code with $d_{\min} = 8$ exists.

The Varshamov and Gilbert bounds are often used interchangeably, but in fact they are different statements. The former fixes $n - k$ and specifies a guaranteed value for n, and hence k. The Gilbert bound on the other hand fixes n and determines the guaranteed k. Both bounds predict the same asymptotic story. Before looking at the asymptotic significance of these bounds, we note that the issue of optimal design is still a surprisingly open question in selected cases. Verhoeff [11] observes that the current best upper and lower bounds to $d_{\min}$ for the binary (127, 33) case are 48 and 32, respectively!

5.3.6 Asymptotic Forms of the Varshamov–Gilbert and Hamming Bounds

The bounds just studied can be applied for any specific n, k, and q. It is also illuminating to study their implications for large block length n. It may be shown [2] that the volume $V_t^{(n)}$ of a radius-t sphere in n-space is bounded by

$$\frac{t}{2n^{1/2}} q^{nh_q(t/n)} < V_t^{(n)} \leq q^{nh_q(t/n)}, \tag{5.3.14}$$

where $h_q(x)$ is the q-ary version of the binary entropy function:

$$h_q(x) = x \log_q(q-1) - x \log_q x - (1-x) \log_q (1-x), \qquad 0 \leq x \leq \frac{q-1}{q}. \tag{5.3.15}$$

Use of this result in the statement of the Gilbert bound (5.3.10), followed by letting n become large, shows that q-ary codes exist with

$$\frac{d_{\min}}{n} \geq h_q^{-1}\left(1 - \frac{k}{n}\right) = h_q^{-1}(1 - R), \tag{5.3.16}$$

where $h_q^{-1}(x)$ is the inverse of the entropy function. Thus, binary rate $\frac{1}{2}$ block codes exist for which the ratio of d_{min} to block length exceeds about 0.11, or for which the ratio of guaranteed error-correcting capability t to block length n exceeds 0.055, as n and k become large. More importantly, the Varshamov–Gilbert bound establishes that codes exist with fixed rate whose ratio of minimum distance to block length remains bounded away from zero as block length increases. On the contrary, there are only a few known constructions of codes for which the ratio of distance to block length does not diminish to zero as the length grows;[14] in particular, BCH codes, the best known family of block codes, do not satisfy the Varshamov–Gilbert bound for large n and, even worse, the ratio of $d_{\min}$ to block length goes to zero as n increases [28].

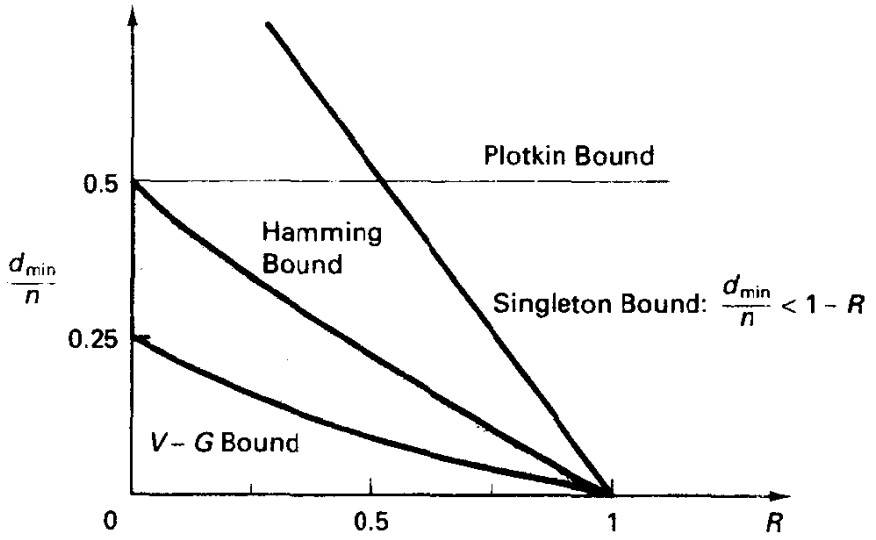
[14]The Varshamov–Gilbert constructions show in principle how to generate codes meeting the bound for large n, but these are not likely to have any appealing implementation properties, and will not constitute a family of code designs.

Similar arguments show that the asymptotic form of the Hamming upper bound is

$$\frac{t}{n} < \frac{d_{\min}}{2n} < h_q^{-1}(1 - R). \tag{5.3.17}$$

Thus, the asymptotic forms of the upper and lower bounds bracket the attainable distance to within a factor of 2, as a function of rate R and block length n.

In Figure 5.3.3 we plot the asymptotic forms of the Hamming, Singleton, and Plotkin upper bounds on $d_{\min}$, as well as the Varshamov–Gilbert lower bound, normalized to block length for binary codes of rate R. Such plots clearly delineate the possible and forbidden regions for long codes of various code rates.



Hamming Bound: $\frac{d_{\min}}{2n} \leq h_2^{-1}(1 - R)$

V – G Bound: $\frac{d_{\min}}{n} \geq h_2^{-1}(1 - R)$

Figure 5.3.3 Asymptotic upper and lower bounds on normalized $d_{\min}$ versus R for binary codes. Shaded region is existence region, defined by Hamming upper bound and V–G lower bounds.

5.3.7 Channel Capacity and the Coding Theorem Revisited

It is worthwhile revisiting the issue of channel capacity in the context of the bounds on the capability of block codes we have just developed. The channel capacity for a q-ary uniform channel, as developed in Chapter 2, is

$$C = 1 + P_s \log_q(q-1) - P_s \log_q P_s - (1 - P_s)\log_q(1 - P_s) \quad q\text{-ary units/symbol}, \tag{5.3.18}$$

where P_s is the symbol error probability. Using the notation of (5.3.15), we have then that

$$C = 1 - h_q(P_s). \tag{5.3.19}$$

In the case of a binary symmetric channel with $\epsilon = 0.11$, the channel capacity is $\frac{1}{2}$ bit per channel use. The converse to the coding theorem produced in Chapter 2 showed that no coding scheme having rate $R > \frac{1}{2}$ can have arbitrarily small probability of error.

Consider n uses of a BSC to transmit an (n, k) binary codeword. If n is large, the law of large numbers holds that the number of errors occurring within a block is within a small interval near $n\epsilon$, whose probability increases as block length grows. For arbitrarily reliable communication to ensue, it is clear that the code must be capable of correcting at least $n\epsilon$ errors per block, or that the fractional error-correcting capability t/n must be at least ϵ. However, the Hamming bound in asymptotic form shows that the code rate R cannot exceed $1 - h_2(t/n) = 1 - h_2(\epsilon)$, or that we must have $R < C$ for reliable communication. Thus, the Hamming bound provides an alternative demonstration of the converse to the coding theorem for the special case of the binary symmetric channel; this is easily extended to the M-ary uniform channel.

The positive side of the coding theorem is not so well supported here, however. The Varshamov–Gilbert bound guarantees only that long binary codes of rate $R = 0.22$ exist with distance $0.22n$, or with fractional error-correcting capability of 0.11. Given the existence of reliable codes (indeed linear codes, although we did not prove this) that operate with rate arbitrarily near capacity, we must surmise that either (1) the Varshamov–Gilbert asymptotic guarantee is pessimistic in stating the achievable minimum distance of long block codes, or (2) reliable communication at rates near capacity is somehow accomplished with decoders capable of correcting many error patterns with more errors than the number guaranteed correctable.

5.4 CYCLIC CODES

Cyclic codes are linear block codes with an important additional property: ***a cyclic, or end-around, shift of any codeword is also a codeword.*** In the language of algebra, the codewords constitute a group under the cyclic shift operation. This additional structure permits further simplification in encoding and decoding, relative to that of general linear codes, and virtually all block codes employed in modern practice are cyclic codes or closely related to one.

To make our definition clearer by examples, we list three $(3, 2)$ codes in Figure 5.4.1. The first is cyclic, while the second is not, failing the cyclic shift test, although it is a linear code. The third code meets the cyclic shift requirement, but because the code is not linear, we shall not include it in the class of cyclic codes. The definition extends as well to nonbinary codes, and the reader can verify that the code defined over GF(4) in (5.2.5) is also cyclic.

5.4.1 Structure of Cyclic Codes

In discussing cyclic codes it is convenient to use polynomial representation for the codewords and for encoding and decoding operations, since shifting of a codeword

000	000	111
110	100	100
011	011	010
101	111	001
(a) Cyclic	(b) Linear, Noncyclic	(c) Nonlinear

Figure 5.4.1 Three (3, 2) block codes.

is just equivalent to a modification of the exponents of a polynomial. Specifically, if $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ denotes a codeword with elements in GF(q), we associate with it a polynomial over GF(q) of degree at most $n - 1$:

$$x(D) = x_0 + x_1 D^1 + x_2 D^2 + \cdots + x_{n-1} D^{n-1}. \tag{5.4.1}$$

Note the coefficients of the polynomial are in GF(q). Now consider a one-position right-cyclic shift of $\mathbf{x}$, producing $\mathbf{x}^{(1)} = (x_{n-1}, x_0, x_1, \ldots, x_{n-2})$. The associated polynomial would be

$$x^{(1)}(D) = x_{n-1} + x_0 D^1 + x_1 D^2 + \ldots + x_{n-2} D^{n-1}, \tag{5.4.2}$$

which is another polynomial of degree at most $n - 1$. Note that the powers found in (5.4.1) have merely been increased by one, modulo n.

The two polynomials $x(D)$ and $x^{(1)}(D)$ are related by

$$x^{(1)}(D) = Dx(D) \bmod (D^n - 1). \tag{5.4.3}$$

This follows since

$$\begin{aligned} Dx(D) &= x_0 D + \cdots + x_{n-1} D^n \\ &= x_{n-1} + x_0 D + x_1 D^2 + \cdots + x_{n-1} D^n \cdots - x_{n-1} \\ &= x_{n-1} + x_0 D + \cdots + x_{n-2} D^{n-1} + x_{n-1}(D^n - 1), \end{aligned} \tag{5.4.4}$$

and upon division by $D^n - 1$ we have $x^{(1)}(D)$ in (5.4.2) as remainder. This argument generalizes for any amount of shifting, and in general $D^j x(D) \bmod (D^n - 1)$ is the code polynomial corresponding to a right-cyclic shift of any codeword $\mathbf{x}$ by j positions.

Now suppose that we are given a particular cyclic (n, k) code over GF(q). We define the ***generator polynomial*** $g(D)$ of the cyclic code as the monic polynomial of minimum degree among the set of nonzero codeword polynomials. We suppose that the degree of this polynomial is $r \leq n - 1$, and write

$$g(D) = g_0 + g_1 D^1 + \cdots + g_r D^r, \tag{5.4.5}$$

where again the coefficients are members of GF(q). There is a unique choice for this polynomial, for if two distinct monic polynomials of degree r existed in the code, these would produce, upon differencing, another code polynomial of lesser degree, yielding a contradiction to the minimality of the degree of the two original polynomials.

Two fundamental properties of cyclic codes are:

Property 1. $x(D)$ is a code polynomial if and only if $x(D) = u(D)g(D)$, where $u(D)$ is of degree $n - 1 - r$ or less.

Property 2. $g(D)$ generates an (n, k) cyclic code if and only if $g(D)$ is of degree $n-k$ and is a factor of D^n-1.

These properties stipulate that all codewords in an (n, k) cyclic code are multiples of a single underlying polynomial, $g(D)$, and that this $g(D)$ must have special degree and be a factor of D^n-1.

To prove property 1, we note that $u_0 g(D)$ is a code polynomial of the code for any u_0 in GF(q). Furthermore, $u_j D^j g(D)$ is a distinct codeword for all $j \le n-1-r$, since these polynomials are just scalar multiples of right shifts of the polynomial $g(D)$. By linearity of the code, all linear combinations of $g(D)$ and its j-place shifts, $j \le n-1-r$, are also codewords, so

$$x(D) = (u_0 + u_1 D + \cdots + u_{n-1-r} D^{n-1-r}) g(D) \tag{5.4.6}$$

is a codeword. We now show that *all* code polynomials have $g(D)$ as a common factor. By Euclid's division algorithm for polynomials, $x(D) = a(D)g(D) + b(D)$, where $a(D)$ and $b(D)$ are the unique quotient and remainder polynomials, respectively, upon dividing $x(D)$ by $g(D)$. Since $g(D)$ has degree r, we know that $b(D)$ will have degree less than r and that the quotient polynomial $a(D)$ will have degree $\le n-1-r$. But since $x(D)$ and $a(D)g(D)$ are code polynomials [the latter fact was just demonstrated in (5.4.6)], linearity implies that $b(D) = x(D) - a(D)g(D)$ must be a codeword. However, unless $b(D)$ is the zero polynomial, we would have a code polynomial with degree less than r, forming a contradiction of the minimal degree of $g(D)$. Thus, $x(D)$ is a codeword if and only if $x(D) = u(D)g(D)$, where $\deg\{u(D)\} \le n-1-r$. There are clearly q^{n-r} such polynomials, and $u(D)$ could be designated the message polynomial.

To verify property 2, we assume that the code is cyclic with generator polynomial $g(D)$, and recall that $g(D)$ is monic and degree r. Therefore, upon multiplication of $g(D)$ by D^{n-r}, the highest-order term is D^n, and we could express the result using Euclid's division algorithm as

$$D^{n-r} g(D) = 1 \cdot (D^n - 1) + b(D), \tag{5.4.7}$$

where we have defined the remainder in (5.4.7) as

$$b(D) = D^{n-r} g(D) \bmod (D^n - 1). \tag{5.4.8}$$

This remainder has degree less than n. By recalling (5.4.3), we see that $b(D)$ defined by (5.4.8) is a cyclic shift of $g(D)$ and is therefore a codeword containing $g(D)$ as a factor. Thus, rewriting (5.4.7) yields

$$\begin{aligned} D^n - 1 &= D^{n-r} g(D) - b(D) \\ &= D^{n-r} g(D) - u(D)g(D) = g(D) \cdot (D^{n-r} - u(D)), \end{aligned} \tag{5.4.9}$$

proving that $D^n - 1$ must contain $g(D)$ as a factor.

Next suppose that $g(D)$ has degree $n-k$ and is a factor of D^n-1. We wish to argue that the code formed by linear combinations of $g(D)$ and its first $k-1$ shifts, $Dg(D), D^2 g(D), \ldots, D^{k-1} g(D)$, is a linear cyclic code. The code is linear by definition and has dimension k, for the vectors associated with $g(D)$ and its first $k-1$ shifts are linearly independent. Suppose that $x(D)$ corresponds to a code polynomial formed as

described; that is,

$$x(D) = (u_0 + \cdots + u_{k-1}D^{k-1})g(D) = u(D) \cdot g(D).$$

Then a right shift of this polynomial is expressed as

$$Dx(D) = x_{n-1}(D^n - 1) + x_{n-2}D^{n-2} + \cdots + x_0 D + x_{n-1}. \tag{5.4.10}$$

Because $x(D)$ and $D^n - 1$ are divisible by $g(D)$, it follows that $x_{n-2}D^{n-1} + x_{n-3}D^{n-2} + \cdots + x_0 D + x_{n-1}$ is also divisible by $g(D)$. Extending this to all shifts, we have that any cyclic shift of a codeword formed as in the preceding, is also expressible as a linear combination of $g(D)$ and its $k-1$ shifts. This shows that the code is cyclic and completes the proof of property 2.

The generator polynomial plays the same fundamental role for cyclic codes that the generator matrix G does for general linear codes: everything we could seek to know about the code is embodied in $g(D)$. On the other hand, whereas we can fashion a general linear code for any (n, k) simply by specifying $\mathbf{G} = [\mathbf{I}, \mathbf{P}]$, we are restricted to certain (n, k) combinations for cyclic codes because of the requirement that $g(D)$ must be a polynomial with degree $n - k$ factoring $D^n - 1$. Although factorizations of $D^n - 1$ exist for all n, we cannot, in general, obtain factors of any desired degree $n - k$. [Try factoring $D^5 - 1$ over GF(2) with third-degree polynomials to produce a (5, 2) cyclic binary code.] However, the set of cyclic codes is suitably rich, and with the code-modification techniques of Section 5.6, we can achieve any design of interest, albeit with a code that is not strictly cyclic.

An important observation about $g(D)$ is that if its degree is r there are r roots of the polynomial, that is, solutions α_j to $g(\alpha_j) = 0$. Furthermore, since $g(D)h(D) = D^n - 1$ for some $h(D)$, the r roots of $g(D)$ are nth roots of unity, that is, $\alpha_j^n = 1$. These roots are typically found in some extension field of GF(q), the coefficient field for $g(D)$; but it is important to note that $g(D)$, and hence an (n, k) cyclic code, is completely specified by the $r = n - k$ roots of $g(D)$. Descriptions we shall soon encounter for the most popular codes involve specification of the root set for the code or, equivalently, the roots of $g(D)$.

The connection between cyclic codes and more general linear codes is further enhanced by forming a generator matrix of a cyclic code. As proved earlier, the code polynomials are linear combinations of $g(D), Dg(D), D^2g(D), \ldots, D^{k-1}g(D)$. Equating polynomials with vectors reveals that any codeword is a linear combination of $\mathbf{g} = (g_0, g_1, \ldots, g_{n-k})$ and its $k - 1$ right shifts. Thus, one expression of G is the banded matrix

$$\mathbf{G} = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & 0 & 0 \\ 0 & g_0 & g_1 & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdots & g_{n-k} & 0 \\ 0 & 0 & 0 & \cdots & g_{n-k-1} & g_{n-k} \end{bmatrix}. \tag{5.4.11}$$

The form of the generator matrix given in (5.4.11) is not in systematic form, but as with any linear code, an equivalent code in systematic form can be produced. This is described in the next section.

The "*cofactor*" of $g(D)$,

$$h(D) = \frac{D^n - 1}{g(D)} = h_0 + h_1 D + \cdots + h_k D^k \tag{5.4.12a}$$

is called the ***parity check polynomial*** of a cyclic code, because it provides the entries in the parity check matrix for the cyclic code, as we shall see shortly. Notice also that multiplication of the codeword $x(D) = u(D)g(D)$ by $h(D)$ produces $u(D)(D^n - 1)$. This polynomial contains two unambiguous copies of the polynomial $u(D)$, since the degree of $u(D)$ is at most $k - 1 < n - 1$. This provides a simple scheme for message recovery when the received sequence is correct, but this method of decoding lacks any error-correcting power.

Because $h(D)$ has degree k and divides $D^n - 1$, it generates an $(n, n - k)$ cyclic code as well, and this code is essentially dual to the original code. [Actually, we need to form the dual code using the reversed form of the polynomial $h(D)$, made monic, as described later.]

The cyclic code generated by $g(D)$ has a parity check matrix $\mathbf{H}$ obtained as follows. Consider again multiplying any code polynomial $x(D)$ by $h(D)$:

$$y(D) = x(D)h(D) = a(D)g(D)h(D) = (D^n - 1)a(D). \tag{5.4.13}$$

Since the degree of $a(D)$ is $k - 1$ or less, we have that the $n - k$ coefficients of the terms $D^k, D^{k+1}, \ldots, D^{n-1}$ in the product $y(D) = x(D)h(D)$ must vanish. These coefficients may be interpreted as being produced by a convolution of the codeword sequence with the sequence $h_0, h_1, \ldots, h_{k-1}$, prefixed and appended with zeros. We thus require that

$$y_m = \sum_{j=0}^{n-1} x_j h_{m-j} = 0, \qquad m = k, k+1, \ldots, n-1. \tag{5.4.14}$$

This set of equations may be expressed in matrix form by $\mathbf{y} = \mathbf{x}\mathbf{H}^T = \mathbf{0}$, where $\mathbf{H}$ is the parity check matrix of the code. Use of (5.4.14) implies that the parity check matrix can be expressed as

$$\mathbf{H} = \begin{bmatrix} h_k & h_{k-1} & \cdots & 0 \\ 0 & h_k & \cdots & h_0 \\ 0 & 0 & \cdots & h_1 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & h_k \end{bmatrix}, \tag{5.4.15}$$

where h_i are the coefficients of the parity check polynomial $h(D)$. This $\mathbf{H}$ matrix is the generator matrix for the code that is the dual of the code produced by $\mathbf{G}$ of (5.4.11). While the matrix has the same general structure, note that the coefficients of $h(D)$ are entered in reverse order relative to that shown in (5.4.11). This is equivalent to saying that the cyclic code that is the dual of that generated by $g(D)$ is produced by

$$\tilde{h}(D) = h_k + h_{k-1} D^1 + \cdots + h_1 D^{k-1} + h_0 D^k \tag{5.4.12b}$$

which is often called the ***reciprocal polynomial***. If the reciprocal polynomial is not monic, we divide all coefficients by h_0 to make it so, thus making the dual code generator polynomial monic.

We now turn to several examples of cyclic codes to clarify this discussion.

Example 5.14 Binary Cyclic Codes of Length $n = 7$

$D^7 - 1$ may be factored over GF(2) as $(D+1)(D^3+D+1)(D^3+D^2+1)$, as multiplication will confirm. We can therefore obtain the following four cyclic codes by combining different factors into $g(D)$:

$$\begin{aligned} k = 1: \quad & g(D) = (D^3 + D + 1)(D^3 + D^2 + 1) \\ & \quad = D^6 + D^5 + D^4 + D^3 + D^2 + D^1 + 1 \\ k = 3: \quad & g(D) = (D + 1)(D^3 + D^2 + 1) = D^4 + D^2 + D + 1 \\ k = 4: \quad & g(D) = D^3 + D + 1 \\ k = 6: \quad & g(D) = D + 1 \end{aligned} \tag{5.4.16}$$

The first generator polynomial produces a (7, 1) repetition code, and the last, its dual code, a (7, 6) single parity bit code. (Writing out the generator matrices and/or parity check matrices will readily confirm this.) The second and third codes are also duals, the third being equivalent to the (7, 4) Hamming code. It is worthwhile experimenting with one of these codes, say that with $k = 3$, by forming all code polynomials according to $x(D) = u(D)g(D)$ and verifying the cyclic property.

To further consider the possibilities for cyclic codes, we list all binary cyclic codes of block lengths 10 and 15 in Figures 5.4.2a and 5.4.2b, respectively. For each code, we list the generator polynomial in binary form, the minimum distance, and the complete weight spectrum. The codes are found by a simple program that tests divisor polynomials of degree $n - k$ until zero remainder is found. The weight spectra are then determined by exhaustion; that is, all codewords are computed and their Hamming weight determined. (This rapidly becomes too time consuming for a minicomputer.) Also, we have not listed codes for which the generator (in binary) is a reversal of a listed code. Such codes have equivalent weight spectra.

Notice that for a given (n, k) pair, more than one choice of $g(D)$ may be available, and these can produce quite different codes. Thus, merely finding a $g(D)$ of proper order that divides $D^n - 1$ does not end the search for the best code. Some cyclic codes are simply bad, but the best ones are essentially as good as any that can be constructed. An example to be of interest shortly is the case of $(n, k) = (15, 7)$. The first such code

k	g	d_{min}	$w=0$	1	2	3	4	5	6	7	8	9	10
9	11	2	1	0	45	0	210	0	210	0	45	0	1
8	101	2	1	0	20	0	110	0	100	0	25	0	0
6	11111	2	1	0	5	0	10	32	10	0	5	0	1
5	100001	2	1	0	5	0	10	0	10	0	5	0	1
4	1100011	4	1	0	0	0	10	0	0	0	5	0	0
2	101010101	5	1	0	0	0	0	2	0	0	0	0	1
1	1111111111	10	1	0	0	0	0	0	0	0	0	0	1

Figure 5.4.2a Binary cyclic codes with $n = 10$.

k	g	d_{min}	$w=0$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
14	11	2	1	0	105	0	1365	0	5005	0	6435	0	3003	0	455	0	15	0	a
13	111	2	1	0	30	125	315	750	1300	1575	1575	1300	750	315	125	30	0	1	
12	1001	2	1	0	30	0	315	0	1300	0	1575	0	750	0	125	0	0	0	
11	10011	3	1	0	0	35	105	168	280	435	435	280	168	105	35	0	0	1	b, c
	11111	2	1	0	15	0	90	243	270	405	405	270	243	90	0	15	0	1	
10	110101	4	1	0	0	0	105	0	280	0	435	0	168	0	35	0	0	0	
	100001	2	1	0	15	0	90	0	270	0	405	0	243	0	0	0	0	0	
9	1011101	4	1	0	0	0	30	60	60	105	105	60	60	30	0	0	0	1	
	1111001	3	1	0	0	5	15	60	100	75	75	100	60	15	5	0	0	1	
8	11010001	4	1	0	0	0	15	0	100	0	75	0	60	0	5	0	0	0	
	11100111	4	1	0	0	0	30	0	60	0	105	0	60	0	0	0	0	0	
7	111010001	5	1	0	0	0	0	18	30	15	15	30	18	0	0	0	0	1	c
	110111011	3	1	1	0	0	5	0	3	25	30	30	25	3	0	5	0	0	
6	1011001101	6	1	0	0	0	0	0	25	0	30	0	3	0	5	0	0	0	
	1100111001	6	1	0	0	0	0	0	30	0	15	0	18	0	0	0	0	0	
5	10100110111	7	1	0	0	0	0	0	0	15	15	0	0	0	0	0	0	1	c
	10000100001	3	1	0	0	5	0	0	10	0	0	10	0	0	5	0	0	1	
4	111101011001	8	1	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	d
	110001100011	6	1	0	0	0	0	0	10	0	0	0	0	0	5	0	0	0	
3	100100100100 1	5	1	0	0	0	0	3	0	0	0	0	3	0	0	0	0	1	
2	11011011011011	10	1	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	
1	111111111111111	15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	

(a) Single-parity bit code
(b) Hamming code
(c) BCH code
(d) Maximal-length sequence code, dual of Hamming code

Figure 5.4.2b Binary cyclic codes with $n = 15$.

listed in Figure 5.4.2b has $d_{min} = 5$, while the second cyclic code has $d_{min} = 3$. Clearly, the former is preferred. Also, we note from Figure 5.4.2a that the block length 10 codes are remarkably weak. It is known, for example, that a linear (but not cyclic) (10, 5) code has $d_{min} = 4$. This weakness extends to all even-length cyclic codes. Finally, we observe that there are no (10, 7) or (10, 3) cyclic codes. The reason that $(15, k)$ cyclic codes exist for all k is that $D^{15} - 1$ has polynomial factors of degrees 4, 2, and 1, and these may be combined as in Example 5.14 to yield a $g(D)$ of any desired degree.

Example 5.15 $n = 23$ **Code over GF(2): Binary Golay Code**

$D^{23} - 1$ factors over the binary field as

$$D^{23} - 1 = (D^{11} + D^{10} + D^6 + D^5 + D^4 + D^2 + 1) \cdot (D^{11} + D^9 + D^7 + D^6 + D^5 + D + 1) \cdot (D + 1), \tag{5.4.17}$$

which may be verified by polynomial multiplication. Either of the degree-11 polynomials may be taken as the generator polynomial for a cyclic (23, 12) binary code, one of the most celebrated and highly studied objects in coding and combinatorics, the perfect (23, 12) Golay code [13], with $d_{min} = 7$. Figure 5.4.3 lists the weight spectrum of the (23, 12) binary Golay code; although not obvious, either choice of generator polynomial yields the same code and weight spectrum.

w	A_w
0	1
7	253
8	506
11	1288
12	1288
15	506
16	253
23	1
	4096

Figure 5.4.3 Weight spectrum for (23, 12) binary Golay code.

Another perfect code found by Golay is described in the next example.

Example 5.16 $n = 11$ **Cyclic Code over GF(3): Ternary Golay Code**

The polynomial $g(D) = D^5 + D^4 + 2D^3 + D^2 + 2$ over GF(3) is a factor of $D^{11} - 1$, with coefficient arithmetic performed modulo 3. Thus, $g(D)$ generates an (11, 6) ternary cyclic code. Since the weight of $g(D)$ is 5, d_{min} can be no greater than 5, and it may be shown that it is indeed 5. We may verify the necessary condition for perfectness to occur: equality in the Hamming bound (5.3.2) with $q = 3$, $t = 2$, $n = 11$, and $k = 6$, although this is not sufficient in itself to claim that such a perfect code exists.

Further tabulation of binary cyclic codes is provided in [21] and [22] for lengths up to $n = 99$, including the minimum distance and descriptions of the generator polynomials.

5.4.2 Encoding of Cyclic Codes

Property 1 of cyclic codes suggests one implementation of an encoder: performing the computation of $x(D) = u(D)g(D)$. Such an encoding system is shown in Figure 5.4.4 and is nothing more than a digital transversal filter over GF(q) that convolves the information sequence $u_{k-1}, \ldots, u_0$ with the impulse response $g_{n-k}, g_{n-k-1}, \ldots, g_0$. We can easily check that the circuit performs the same computations as longhand multiplication of the polynomials $u(D)$ and $g(D)$.

CAUTION

At this point considerable confusion may arise over the ordering of the information sequence and the corresponding code sequence. Our convention in Figure 5.4.4 is that in $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ the rightmost (largest index) symbol is the first symbol to enter the encoder, subscript order notwithstanding.

The codewords produced by this implementation of the encoder are nonsystematic; as may be seen from (5.4.6) or from the diagram of Figure 5.4.4, the information symbols do not appear explicitly in the code stream. However, we know that this linear code has an equivalent systematic counterpart; production of systematic-form codewords is convenient and simple as well. Consider the polynomial $D^{n-k}u(D)$. By Euclid's division theorem we can express this in quotient/divisor/remainder form as

$$\begin{aligned} D^{n-k}u(D) &= a(D)g(D) + [D^{n-k}u(D)] \bmod g(D) \\ &= a(D)g(D) + p(D). \end{aligned} \tag{5.4.18a}$$

Thus, rearranging (5.4.18a) reveals that the polynomial

$$x(D) = D^{n-k}u(D) - [D^{n-k}u(D)] \bmod g(D) \tag{5.4.18b}$$

is a multiple of $g(D)$, hence a member of the cyclic code. Also, because $u(D)$ has degree at most $k-1$, then $D^{n-k}u(D)$ has maximal exponent $n-1$ and minimal exponent $n-k$. Since the remainder of the division by $g(D)$ [designated previously by $p(D)$ for parity polynomial] has degree as most $n-k-1$, the two polynomials on the right-hand side in (5.4.18b) have exponent ranges that do not overlap, and the polynomial $u(D)$ appears distinctly in the highest-order k positions of $x(D)$. In n-tuple terms, $\mathbf{u}$ appears in the high-index k positions of $\mathbf{x}$, followed by the $n-k$ parity symbols $\mathbf{p}$.

Thus, to implement systematic encoding we need a means of dividing $D^{n-k}u(D)$ by $g(D)$, producing the remainder, as indicated in (5.4.18). First, we argue that the circuit of Figure 5.4.5 computes the remainder of a general polynomial $r(D)$ upon division by $g(D)$, based on equivalence with the operations of longhand division. In dividing $r(D)$ by $g(D)$, the first coefficient in the quotient polynomial is $r_{n-1}g_{n-k}^{-1}$. In long division, we proceed to form the product of this first coefficient and $g(D)$ (the divisor), and then subtract from the original dividend. This leaves us with a result whose degree is at most $n-2$. The next coefficient of the quotient polynomial is thus $(r_{n-2} - r_{n-1}g_{n-k}^{-1}g_{n-k-1})g_{n-k}^{-1}$. We multiply this by $g(D)$, subtract, and so on. Thus, in Figure 5.4.5 the proper quotient coefficients appear sequentially at point A, and the feedback into the register is providing

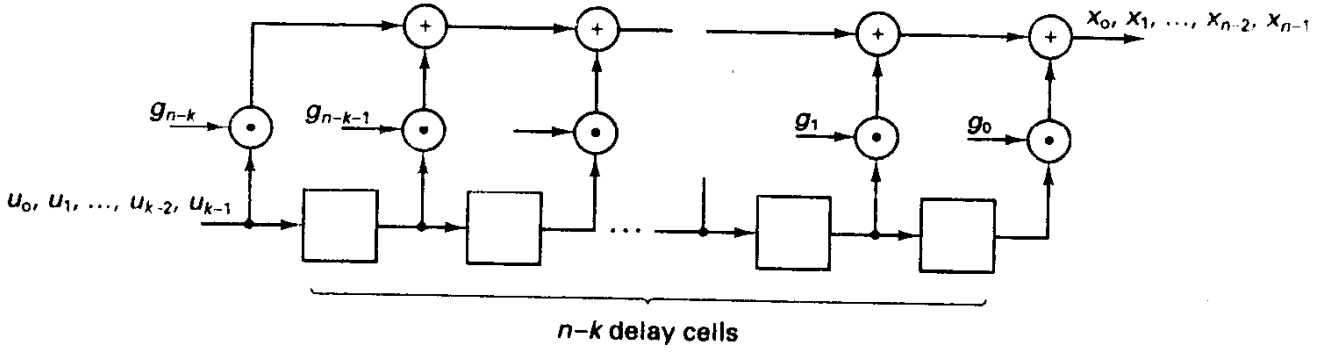


Figure 5.4.4 Nonsystematic encoder for cyclic (or polynomial) code. Adders, multipliers, and delay cells are q-ary.

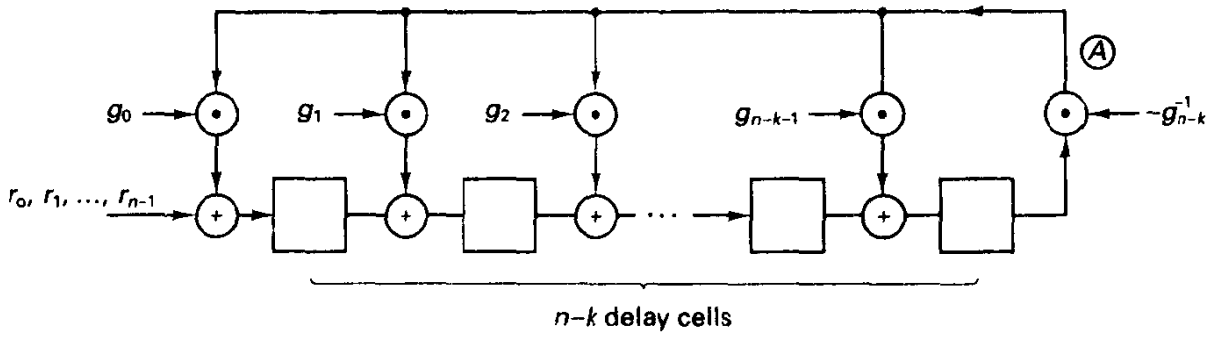


Figure 5.4.5 Circuit for dividing $r(D)$ by $g(D)$. After n cycles, coefficients of remainder polynomial reside in register. Register is initially zeroed.

the coefficients in the polynomial to be subtracted from the previous residual. After n cycles the register holds the coefficients of the remainder polynomial. References [1] and [4] develop further details and alternative circuit realizations for polynomial arithmetic. [The reader is cautioned that many texts develop this implementation for binary codes, in which case subtraction and addition are equivalent, and multiplication by g_j is trivial, so the circuit appears slightly simpler in that case. In any field $g_{n-k}^{-1} = 1$, because $g(D)$ is monic, and for fields of characteristic 2, addition is equivalent to subtraction.]

To apply this procedure to systematic encoding, recall that we need to compute the remainder of $D^{n-k}u(D)$ upon division by $g(D)$ and append this to the message, represented by $D^{n-k}u(D)$. This is readily accomplished with the circuit of Figure 5.4.6a. Premultiplication by D^{n-k} amounts to moving the injection point $n-k$ stages to the right, that is, to the right-hand end of the register, and allows that the remainder be available after k clock cycles. (Figure 5.4.6b illustrates the equivalence with long division.) The encoding cycle begins with the register contents zeroed and the feedback switch closed. Information symbols are clocked into the circuit, each time sending these to the channel or buffer and updating the register contents. After k shifts, the feedback path is disabled, and the register contents, which are the coefficients of $p(D)$ in (5.4.18b), are clocked out to be appended to the information symbols. If we are interfacing to a synchronous channel, a buffering and rate conversion by n/k must also take place.

Example 5.17 Encoder for $g(D) = D^3 + D + 1$

The systematic-form encoder is shown in Figure 5.4.7. Note that for a binary code the GF(q) multipliers are unnecessary, and subtraction is equivalent to addition. Thus, we merely form a feedback connection when $g_j = 1$. Also, the modulo-2 adders are simple 1-bit exclusive-or gates.

To encode the message $(u_0, u_1, u_2, u_3) = (1, 0, 1, 0)$, the successive states of the register are (000), (000), (110), (011), and (001), and the entire codeword is $\mathbf{x} = (0, 0, 1, 1, 0, 1, 0)$. The code so generated is equivalent to that of the (7, 4) Hamming code studied earlier, although the codewords form a different set than listed in Figure 5.2.2.

Example 5.18 CCITT Cyclic Redundancy Check Code

The binary polynomial $g(D) = D^{16} + D^{12} + D^5 + 1$ has been selected by the international standards organization CCITT as a ***cyclic redundancy check (CRC)*** code generator for

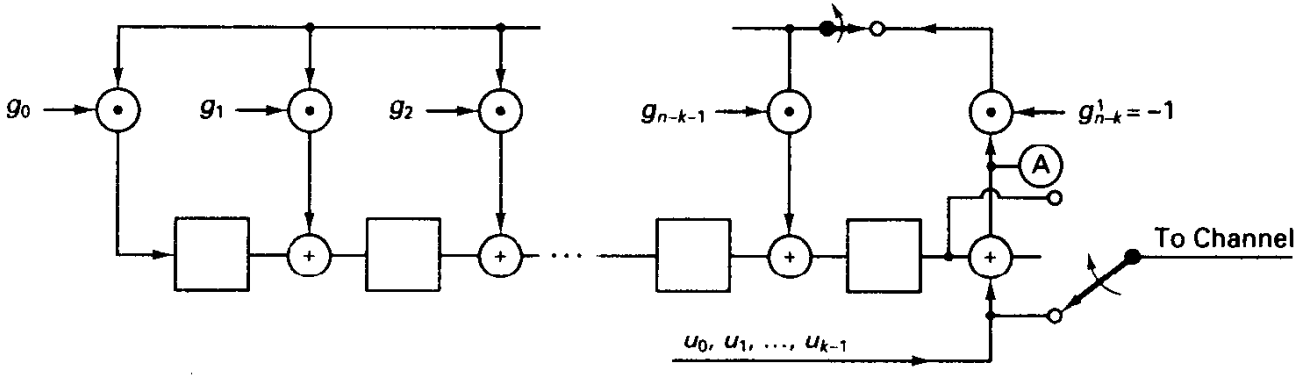


Figure 5.4.6a Systematic-form encoder for cyclic code. Register initially zeroed; after k cycles, feedback gate opens and output multiplexer switches to upper position to append parity symbols.

$$-u_{k-1}g_{n-k}^{-1}D^{k-1} - (u_{k-2} - u_{k-1}g_{n-k}^{-1}g_{n-k-1})g_{n-k}^{-1} + \cdots$$

$$\frac{D^{n-k}u(D)}{g(D)} \Longleftrightarrow g_{n-k}D^{n-k} + \cdots + g_0 \;\Big|\; u_{k-1}D^{n-1} + u_{k-2}D^{n-2} + \cdots + u_0D^{n-k}$$

$$-u_{k-1}D^{n-1} - (u_{k-1}g_{n-k}^{-1}g_{n-k-1}D^{n-2}) - \cdots$$

$$(u_{k-2} - u_{k-1}g_{n-k}^{-1}g_{n-k-1})\,D^{n-2} - \cdots$$

$$-(u_{k-2} - u_{k-1}g_{n-k}^{-1}g_{n-k-1})\,D^{n-2} - \cdots$$

Figure 5.4.6b Long-division and relationship to Figure 5.4.6a. Quotient coefficients appear at A in Figure 5.4.6a; remainders appear in $n-k$ cell register.

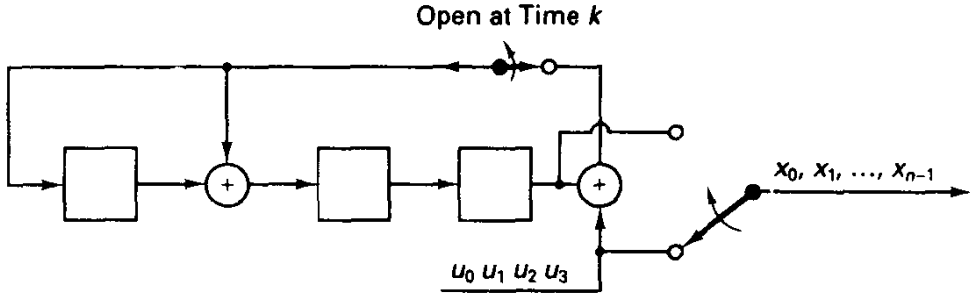


Figure 5.4.7 Systematic encoder for (7, 4) Hamming code with $g(D) = D^3 + D + 1$.

certain binary data communication protocols, for example, in packet communication using the X.25 protocol. The codewords are formed as by a k-bit information sequence followed by a check pattern of 16 bits (two 8-bit bytes) at the end of the message, computed according to (5.4.18). Systematic encoding can be accomplished with the circuit of Figure 5.4.8 and is available commercially in integrated circuit form. Although the encoder can be used with any packet length k, producing a $(k + 16, k)$ code, the resultant set of codewords is not cyclic in general [since $g(D)$ usually does not factor $D^{k+16} - 1$], but is simply called a ***polynomial code***. For the normal application of this code (error detection only), this is not a substantial issue.

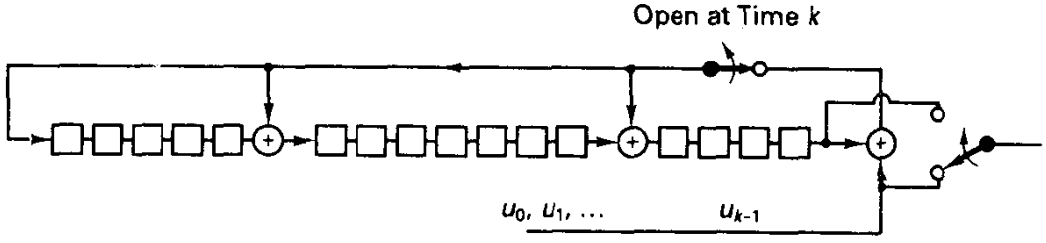


Figure 5.4.8 Systematic encoder for CCITT polynomial code with $g(D) = D^{16} + D^{12} + D^5 + 1$.

We are now ready to present the most popular cyclic codes: BCH codes, Hamming codes, and Reed–Solomon codes.

5.4.3 BCH Codes

Around 1960, Bose and Ray-Chaudhuri [23] and Hocquenghem [24] independently discovered what have become known as BCH codes. These codes probably have been the most profusely studied block codes and are important in practice for the following reasons:

1. There is an ample selection of block lengths and code rates.
2. For short-to-moderate block lengths and small code alphabets, BCH codes are essentially the most powerful linear codes.
3. Elegant general-purpose algebraic decoding algorithms exist.

Our description of these codes will be terse, but one that emphasizes the central ideas. We begin with a basic definition of the generator polynomial for BCH codes.

Definition of BCH codes over GF(q)

Given a field GF(q), a block length $n \geq 3$, which is a divisor of $q^m - 1$ for some m, and $3 \leq \delta \leq n$, an (n, k) BCH code over GF(q) is a cyclic code generated by

$$g(D) = \text{LCM}[m_{\beta^j}(D), m_{\beta^{j+1}}(D), m_{\beta^{j+2}}(D), \ldots, m_{\beta^{\delta-2+j}}(D)]. \tag{5.4.19}$$

Here the $m_{\beta^i}(D)$ are minimal polynomials of $\delta - 1$ *successive* powers of a field element β whose order is n in an extension field GF(q^m). (Usually, β is a primitive element in this extension field, as discussed later.) LCM denotes the least common multiple polynomial, or smallest-degree monic polynomial for which all the indicated minimal polynomials are divisors. It is noted that in the set of $\delta - 1$ minimal polynomials there often will be found repeated polynomials. Since the minimal polynomials are all irreducible, finding the LCM polynomial amounts to forming the product of the *distinct* minimal polynomials in (5.4.19).

Since each minimal polynomial is a divisor of $D^n - 1$, $g(D)$ defined by (5.4.19) will also be a divisor of this same polynomial, which is sufficient to generate a cyclic code of length n. The code's dimension, k, will depend on the degree of the polynomial in (5.4.19).

An equivalent definition that is fundamental to understanding decoding is that a BCH code is the largest set of codewords $\mathbf{x}$ whose corresponding polynomials $x(D)$ have as roots $\delta - 1$ successive powers of an element β of order n in an extension field of GF(q). That is, for every code polynomial

$$x(D)|_{D=\beta^i} = 0, \qquad i = j, j+1, j+2, \ldots, j+\delta-2. \tag{5.4.20}$$

where $\beta^n = 1$ in GF(q^m). This follows since these powers of a field element are roots of minimal polynomials making up the generator polynomial and are therefore roots of $g(D)$ and thus $x(D)$ as well.

A "frequency-domain" interpretation is also helpful. Recall from Section 5.1 that the DFT coefficients X_i are equivalent to evaluations of time-domain polynomials at various powers of the transform kernel. From (5.4.20), codewords in a BCH code have special transforms: $\delta - 1$ *consecutive* transform coefficients in positions $j, j+1, \ldots, j+\delta-2$

are identically zero, the remaining transform coefficients being unspecified, as depicted in Figure 5.4.9. Thus, we could also define a BGH code as a set of n-tuples over GF(q), each of whose DFTs are zero in a specified number of consecutive positions. In constructing such DFTs we use a transform kernel that is an nth-root of unity in an extension field GF(q^m).

Normally, the base field element, β, for the definition of the minimal polynomials is a *primitive* element, α, in some extension field GF(q^m), in which case the code is a ***primitive*** BCH code. The order of such an element is $n = q^m - 1$, which is also the block length of the code. Furthermore, we usually adopt for the definition $j = 1$, in which the root set of the code begins with the first power of α. In this case the code is said to be ***narrow sense***. Introductory treatments of BCH codes often default to the primitive, narrow-sense codes.

Notice that the degree of $g(D)$ is less than or equal to $m(\delta - 1)$, since there are at most $\delta - 1$ distinct minimal polynomials involved in the construction of $g(D)$, and each has degree at most m. Thus, for *BCH codes over any field* we have the following relations:

BCH Relations

$$n = q^m - 1 \quad \text{(or a divisor of } q^m - 1) \tag{5.4.21}$$

and

$$n - k = \deg g(D) \leq m(\delta - 1). \tag{5.4.22a}$$

Equation (5.4.22a) places a bound on the number of parity symbols of the codes. Note that the previous definition of the BCH code does not directly specify the dimension, k, of the code; this only becomes clear once the construction of $g(D)$ is completed.

For *binary narrow-sense* BCH codes, the number of parity bits, $n - k$, is more tightly upper bounded by $m\lceil(\delta - 1)/2\rceil$, since among the sequence of minimal polynomials employed in the previous definition of $g(D)$ the only unique polynomials are $m_{\alpha^1}(D), m_{\alpha^3}(D), \ldots, m_{\alpha^{\delta-1}}(D)$. (Recall the earlier discussion that conjugate elements of a field, for example, $\alpha, \alpha^2, \alpha^4, \ldots$, share identical minimal polynomials.) Thus, for binary codes, (5.4.22a) becomes

$$n - k \leq m\left\lceil \frac{\delta - 1}{2} \right\rceil \quad \text{(binary, narrow-sense case)} \tag{5.4.22b}$$

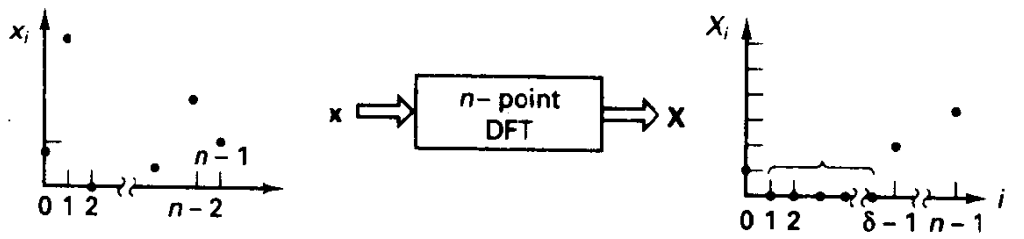


Figure 5.4.9 DFTs of BCH codewords have $\delta - 1$ successive zeros at $j, j + 1, \ldots, j + \delta - 2$.

We will soon demonstrate that the minimum distance can be lower bounded by

$$d_{\min} \geq \delta, \tag{5.4.23}$$

so δ is often called the ***design distance*** of the code; consequently, $t = \lfloor (\delta - 1)/2 \rfloor$ could be called the designed error correction capability. It may turn out that the true minimum distance, $d_{\min}$, is larger than the design distance and thus that an optimum decoder is capable of correcting more than t errors. However, the standard BCH decoding algorithms are incapable of exploiting this possibility. We remark that all primitive single- and double-error-correcting BCH codes are known to have true minimum distance equaling the design distance [25].

Example 5.19 (15, 7) BCH Code over GF(2)

This code represents perhaps the first interesting step beyond the Hamming codes, in that the code is double error correcting in a nontrivial way. We begin with $\delta = 5$, so that $t = 2$, and select $n = 15$, which implies that $m = 4$ by (5.4.21) for a primitive code. The dimension k (for a binary narrow-sense code) will be at least $15 - m\lceil (\delta - 1)/2 \rceil = 7$ by (5.4.22b). The generator polynomial can be taken as the least common multiple of the minimal polynomials for $\alpha, \alpha^2, \alpha^3$, and α^4, where α is a primitive element in GF(16). We recall, however, that the minimal polynomials for the elements α, α^2, and α^4 are identical, and thus there are but two distinct polynomials in the set used to define $g(D)$. From Figure 5.1.3, these minimal polynomials are seen to be

$$\begin{aligned} m_{\alpha^1}(D) &= D^4 + D + 1 \\ m_{\alpha^3}(D) &= D^4 + D^3 + D^2 + D^1 + 1. \end{aligned} \tag{5.4.24}$$

The generator polynomial is then the eighth-degree polynomial

$$g(D) = m_{\alpha^1}(D) m_{\alpha^3}(D) = D^8 + D^7 + D^6 + D^4 + 1. \tag{5.4.25}$$

Because $g(D)$ has degree 8, we conclude that k is indeed seven. Also, (5.4.23) holds that $d_{\min}$ will be 5 or larger, but since the number of nonzero coefficients in the generator polynomial is 5, this becomes the true $d_{\min}$. (This code should be located in Figure 5.4.2b of length 15 cyclic codes.)

Example 5.20 BCH Code with $n = 15, t = 3$

The binary BCH code with designed distance $\delta = 7$ (or $t = 3$) and $n = 15$ has a generator polynomial that is the LCM of minimal polynomials of six consecutive powers of a primitive element in GF(16), say $\alpha, \alpha^2, \ldots, \alpha^6$. Again, some of these minimal polynomials are identical. However, one of the polynomials has degree 2 (see Figure 5.1.3), and the generator polynomial has degree 10:

$$\begin{aligned} g(D) &= (D^4 + D + 1)(D^4 + D^3 + D^2 + D^1)(D^2 + D + 1) \\ &= D^{10} + D^8 + D^5 + D^4 + D^2 + D + 1. \end{aligned} \tag{5.4.26}$$

Thus, the number of parity symbols is $n - k = 10$, which is smaller than the BCH bound of (5.4.22b), or 12. This occurs precisely because the minimal polynomials are not all degree 4. Here, again, the actual $d_{\min}$ is equivalent to δ since the weight of $g(D)$ is 7. This code contains 32 codewords, which form a subcode of the code having 128 codewords in Example 5.19. In general, low-rate BCH codes are subcodes (or subspaces) of high-rate BCH codes of the same length, because the low-rate codes have root sets that include the root sets of the higher-rate codes.

In keeping with our frequency-domain definition of BCH codes, we could say that all 32 codewords of this code have DFT coefficients [in GF(16)] that are identically zero in the six consecutive positions $0, 1, \ldots, \delta - 2 = 5$. The remaining DFT coefficients depend on the actual message.

Figure 5.4.10 provides a compendium of binary, primitive, narrow-sense BCH codes for $n = 7, 15, 31, 63$, and 127, including the design distances, the actual dimension of the codes, the true distances, and the generator polynomials, expressed in octal form. The latter polynomials are extracted from Lin and Costello [3], where a more extensive table for longer block length codes is found. As indicated previously, these parameters emerge only after applying the definition of the code and determining the degree of $g(D)$, as well as determining the true $d_{\min}$.

We observe several interesting aspects in this tabulation. First, note that the dimension k usually decreases in steps of m units, where $n = 2^m - 1$, for each entry in the table, but not always. The unusual case occurs when a minimal polynomial with degree smaller than m is encountered in the sequence of (5.4.19). If n is prime, however, this never occurs (all minimal polynomials in this case have degree m), and the sequence of dimensions k is regular. Also, as δ is incremented we will occasionally find that a range of δ can be achieved with the same dimension k, and normally we would elect to know the largest possible design distance. We finally note that the first example (among primitive codes) for which the true minimum distance exceeds δ is $(127, 43)$. Usually, for primitive, narrow-sense BCH codes the minimum distance is either equivalent to the design distance or within a very few Hamming units of δ. This is often not the case with nonprimitive or nonnarrow-sense designs, however.

We have not claimed optimality for the BCH designs, and it is possible that other constructions will outperform the BCH construction. Chen [21] reports a $(63, 46)$ cyclic code with $d_{\min} = 7$ that is slightly superior to the $(63, 45)$ primitive, narrow-sense BCH code with design distance (and true $d_{\min}$) equaling 7. This seems rare for moderate block lengths and for $n = 2^m - 1$. Tables of cyclic codes in [21] and [22], however, show that for other block lengths cyclic codes often have true distance in excess of the BCH bound, that is, one greater than the maximum number of consecutive powers of β in the root set of the code, where β is a primitive nth root of unity. Techniques for narrowing in on the true minimum distance of cyclic codes are presented by van Lint and Wilson [26].

Example 5.21 The (23, 12) Golay Code as a Nonprimitive BCH Code

With helpful hindsight, we can realize that the Golay code is a special case of a BCH code. In $GF(2^{11})$, primitive elements would have order $2^{11} - 1 = 2047$, but there are also elements in this field with order 23, since $23(89) = 2047$. Let us adopt $\beta = \alpha^{89}$ (where α is primitive) as our base element for defining a BCH code, and adopt $j = 1$ and $\delta = 3$. Here, $g(D)$ is the LCM of the minimal polynomials for $\beta = \alpha^{89}$ and for $\beta^2 = \alpha^{178}$. These polynomials are identical since the extension field has characteristic 2, and a suitable tabulation of GF(2048) would reveal that this polynomial has degree 11; either found in (5.4.17) will suffice. Thus, the number of information bits in this nonprimitive BCH code is 12.

Surprisingly, the true d_{min} of the code is 7, not 3. This situation is more common with nonprimitive codes in general. However, the standard BCH decoder described in the next section is inefficient at exploiting the full error-correcting capability of the code, and special "error-trapping" procedures devoted to this code have been devised [27]. With

$n = 3$

δ	k	d_{min}	$g(D)$
3	1	3	7_8

$n = 7$

δ	k	d_{min}	$g(D)$
3	4	3	13
5 - 7	1	7	77

$n = 15$

δ	k	d_{min}	$g(D)$
3	11	3	23
5	7	5	721
7	5	7	2467
8 - 15	1	15	77777

$n = 31$

δ	k	d_{min}	$g(D)$
3	26	3	45
5	21	5	3551
7	16	7	107657
9 - 11	11	11	5423325
13 - 15	6	15	313365047
17 - 31	1	31	17777777777

$n = 63$

δ	k	d_{min}	$g(D)$
3	57	3	103
5	51	5	12471
7	45	7	1701317
9	39	9	166623567
11	36	11	1033500423
13	30	13	157464165347
15	24	15	17323260404441
17 - 21	18	21	1363026512351725
23	16	23	6331141367235453
25 - 27	10	27	472622305527250155
29 - 31	7	31	5231045543503271737
33 - 63	1	63	— all 1's —

$n = 127$

δ	k	d_{min}	$g(D)$
3	120	3	211
5	113	5	41567
7	106	7	11554743
9	99	9	3447023271
11	92	11	624730022327
13	85	13	130704476322273
15	78	15	26230002166130115
17 - 19	71	19	6255010713253127753
21	64	21	1206534025570773100045
23	57	23	335265252505705053517721
25 - 27	50	27	54446512523314012421501421
29	43	31	17721772213651227521220574343
31	36	≥31	314607466652207504476457472l735
33 - 43	29	≥43	403114461367670603667530141176155
45 - 47	22	≥47	12337607040472252243544562663764704 3
49 - 55	15	≥55	2205704244560455477052301376221760435 3
57 - 63	8	≥63	70472640527510306514762242715677331302 17
65 - 127	1	127	— all 1's —

Figure 5.4.10 Binary primitive BCH code parameters. Generator polynomials listed in octal; for example $\mathbf{g} = 13_8$ connotes $g(D) = D^3 + D + 1$. Generator polynomials taken from Lin and Costello [3].

modern technology, table lookup after syndrome computation represents the fastest decoding technique.

It is interesting to examine the behavior of binary BCH codes of a fixed rate as block length increases, with special attention paid to normalized $d_{\min}$ and how it compares with respect to the Varshamov–Gilbert bounds. Consider codes of rate $R \approx \frac{1}{2}$, listed in Figure 5.4.11. We list the design distance δ and the true minimum distance $d_{\min}$ and also the normalized minimum distance. We observe that with increasing block length the BCH codes are progressively poorer, and $d_{\min}/n$ eventually falls below the existence bound of Varshamov–Gilbert. (The Varshamov–Gilbert bound in its asymptotic form would say that codes exist whose $d_{\min}/n$ exceeds $h^{-1}(1 - 0.5) = 0.11$. This has prompted the declaration that long BCH codes are "bad," and in fact $d_{\min}/n$ approaches zero as n increases [28, 29].) On the positive side, we note that the normalized distance still exceeds the V–G lower bound for rather long block lengths, perhaps the range of most practical interest anyway. Furthermore, the code competition is not that strong, especially when we ask for reasonable decoding algorithms and for codes with rather general (n, k) parameters.

Our BCH construction incorporates nonbinary BCH codes at the outset. Thus, we could pursue the same constructions to produce a code over GF(4), say, whose block length is 63, since $4^3 - 1 = 63$. Roots of the generator polynomial would be taken as consecutive powers of a primitive element in GF(64). Nonetheless, these nonbinary BCH codes are of lesser practical importance, except for one special case taken up shortly, that of Reed–Solomon codes.

The BCH bound on $d_{\min}$

We now return to the issue of minimum distance and prove the ***BCH bound*** (5.4.23). For notation we will adopt a primitive base element α, although this is not restricting. From the definition, we see that valid codewords in a BCH code must have a sequence of $\delta - 1$ powers of α as roots of the corresponding code polynomial $x(D)$; that is,

$$x(\alpha^i) = \sum_{m=0}^{n-1} x_m(\alpha^i)^m = 0, \qquad i = j, j+1, \ldots, j+\delta-2. \tag{5.4.27}$$

(n, k)	δ	$d_{\min}$	$d_{\min}/n$
(7, 4)	3	3	0.42
(15, 7)	5	5	0.33
(31, 16)	7	7	0.22
(127, 64)	21	21	0.16
(511, 255)	57	57(?)	0.12

Figure 5.4.11 Listing of BCH codes with $R \approx \frac{1}{2}$.

These $\delta - 1$ equations may be represented in matrix form by $\mathbf{x}\mathbf{H}^T = \mathbf{0}$, where

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha^j & \cdot & \alpha^{(n-1)j} \\ 1 & \alpha^{j+1} & \cdot & \alpha^{(n-1)(j+1)} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 1 & \alpha^{j+\delta-2} & \cdot & \alpha^{(n-1)(j+\delta-2)} \end{bmatrix} \tag{5.4.28}$$

$\mathbf{H}$ is a matrix of $\delta - 1$ rows and n columns in the general case; however, some of the rows of $\mathbf{H}$ may be linearly dependent, which is to say that they do not provide independent parity check information. In the case of binary codes with $j = 1$, for example, only $\alpha, \alpha^3, \ldots, \alpha^{\delta-2}$ (assuming δ is odd) must be accounted for in the parity check matrix.

Now we appeal to a result of Section 5.2: if the parity check matrix is such that all sets of $d - 1$ columns are linearly independent, then $d_{\min} \geq d$. We wish to show then that any linear combination of $\delta - 1$ columns in (5.4.28) cannot be zero, unless the multiplier coefficients are all zero. If we let n_p, $p = 1, 2, \ldots, \delta - 1$, designate column indexes selected for linear combination, such a linear combination would be equivalent to the system of equations

$$\sum_{p=1}^{\delta-1} x_{n_p} \alpha^{in_p} = 0, \qquad i = j, j+1, \ldots, j+\delta-2. \tag{5.4.29}$$

The question is whether nonzero solutions exist for the $\delta - 1$ unknowns x_{n_p}. Such nontrivial solutions exist only if the determinant of the system matrix is zero. Therefore, we consider the $\delta - 1$ by $\delta - 1$ determinant

$$\Delta = \begin{vmatrix} \alpha^{n_1} & (\alpha^2)^{n_1} & \cdot & (\alpha^{\delta-1})^{n_1} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \alpha^{n_{\delta-1}} & (\alpha^2)^{n_{\delta-1}} & \cdot & (\alpha^{\delta-1})^{n_{\delta-1}} \end{vmatrix} \tag{5.4.30}$$

We can proceed to factor from each row of the determinant above a common term, thereby obtaining the reduced form

$$\Delta = \alpha^{n_1}\alpha^{n_2}\cdots\alpha^{n_{\delta-1}} \begin{vmatrix} 1 & \alpha^{n_1} & (\alpha^{\delta-2})^{n_1} \\ 1 & \alpha^{n_2} & (\alpha^{\delta-2})^{n_2} \\ \cdot & \cdot & \cdot \\ 1 & \alpha^{n_{\delta-1}} & (\alpha^{\delta-2})^{n_{\delta-2}} \end{vmatrix} = \alpha^{n_1}\alpha^{n_2}\cdots\alpha^{n_{\delta-1}}|\mathbf{V}|. \tag{5.4.31}$$

The remaining determinant $|\mathbf{V}|$ is a ***Vandermonde determinant***, known to be expressible as [30]

$$|\mathbf{V}| = \prod_{i=2}^{\delta-2} \prod_{j=1}^{i-1} (\alpha^{n_i} - \alpha^{n_j}). \tag{5.4.32}$$

This determinant is nonzero by specification that the base element is of order n so that $\alpha^{n_i} \neq \alpha^{n_j}$, $j \neq i$, and thus the complete determinant is nonzero. Thus, we have shown that no linear combination of $\delta - 1$ columns may sum to zero, in turn guaranteeing that $d_{\min} \geq \delta$. It may be that linear independence can be claimed for more than δ columns,

implying that the true distance exceeds the design distance, but this cannot be guaranteed at the start.

By inspecting the complete argument, we can claim a somewhat more general result: for any (n, k) cyclic code (BCH or not), the true minimum distance is at least as large as the number of consecutive powers of α found in the root set of the code, where α is of order n in GF(q^m).

5.4.4 Cyclic Hamming Codes

We have already encountered the Hamming codes in Section 5.2 as a class of perfect, single-error-correcting codes over any field, with $d_{\min} = 3$. Normally, they are described as binary codes. Recalling the development of Example 5.9, we have that

$$n = \frac{q^{n-k} - 1}{q - 1} \tag{5.4.33}$$

is the expression constraining n, k, and q. For a cyclic code to exist with these parameters, we need a generator polynomial of degree $n - k$, which factors $D^n - 1$. The existence of such codes is easily seen with appeal to the BCH construction, with $\delta = 3$. The BCH codes have length $n = q^{n-k} - 1$ or any divisor of this number. Clearly, $q - 1$ is such a divisor, so we consider BCH codes with length $n = (q^{n-k} - 1)/(q - 1)$, as required for the Hamming codes. Let β be an element of order n. Then the generator polynomial is, from (5.4.19) with $j = 1$,

$$g(D) = \text{LCM}[m_\beta(D), m_{\beta^2}(D)]. \tag{5.4.34}$$

However, we have the usual result that β and β^2 have the same minimal polynomial for fields of characteristic 2. Furthermore, this polynomial will be of degree $n - k$ over GF(q). For the case of binary Hamming codes, all this simplifies to the fact that the generator polynomial is a *primitive* polynomial over GF(2) of degree $n - k$. Thus, the generator polynomials for binary Hamming codes may be taken from the table of primitive polynomials given earlier. A (31, 26) binary cyclic Hamming code is then produced by $g(D) = D^5 + D^2 + 1$. Nonbinary cyclic Hamming codes are similarly produced: the shortest Hamming code over GF(16) is obtained with $n - k = 2$, so from (5.4.33) the block length is $n = 17$ and $k = 15$. The generator polynomial is the minimal polynomial of an element of order 17 in GF(256), which will be a second-degree polynomial over GF(16) of the form $g(D) = g_0 + g_1 D^1 + D^2$.

The BCH bound guarantees that the minimum distance is at least the design distance, 3. However, the Hamming upper bound prevents the minimum distance from exceeding 3. Also, inspection of the binary generator polynomials, which are primitive polynomials taken from Figure 5.1.2, shows that (except for the degree 8 case) the number of nonzero terms is generally 3, demonstrating the existence of weight 3 codewords. We will not proceed to verify that in fact these are all perfect codes; such could be done by appealing to the parity check matrix, in turn provided by the parity check polynomial, and showing that all q^{n-k} columns of $\mathbf{H}$ are nonzero and distinct. Thus, every single-error pattern produces a unique syndrome, while all higher-weight error patterns produce syndromes corresponding to single-error patterns.

5.4.5 Reed-Solomon Codes

Originally conceived in 1960 [31] as a class of maximum distance separable codes, these nonbinary codes were subsequently seen to be a special case of nonbinary BCH codes. [A possible alternative view is that BCH codes are a subfield subcode[15] of Reed–Solomon (RS) codes, in which case Reed–Solomon codes become the general class.]

Although conceived early in the history of coding theory, it is fair to say that RS codes have only come into practical prominence in the last several years, due largely to the developments in VLSI processing technology that make the decoding computations feasible. At the same time there has been increased interest in coding for nonbinary modulation. The best evidence of the penetration of RS codes into modern practice is their use in compact disc players [32]. RS codes have also recently become prominent in recording on magnetic and optical media, and high-speed single-chip decoders have been produced for certain standards [33].

We shall view the RS codes as special cases of BCH codes. Recall that primitive q-ary BCH codes have block lengths $n = q^m - 1$, and the extension field GF(q^m) is invoked for locating roots of $g(D)$. Taking $m = 1$, we have $n = q - 1$, which governs the standard Reed–Solomon (RS) codes. Notice that the code symbol field and the field of the roots of $g(D)$ are the same.

For the primitive BCH codes, the generator polynomial is given by

$$g(D) = \text{LCM}[m_{\alpha^l}(D), m_{\alpha^{l+1}}(D), \ldots, m_{\alpha^{l+\delta-2}}(D)], \tag{5.4.35}$$

where $m_{\alpha^i}(D)$ are minimal polynomials of elements α^i, powers of a primitive element in GF(q^m), and δ is the designed distance. With $m = 1$, however, the minimal polynomials over GF(q) of elements in GF(q) are first degree, of the form $(D - \alpha^j)$, and $g(D)$ reduces to

$$g(D) = (D - \alpha^l)(D - \alpha^{l+1}) \cdots (D - \alpha^{l+\delta-2}). \tag{5.4.36a}$$

This is a polynomial over GF(q) of degree exactly $\delta - 1$ (in contrast to the general BCH case, which may produce a generator of degree less than the number of minimal polynomials would suggest.) Thus, $n - k = \delta - 1$ for Reed–Solomon codes.

Because these codes are BCH codes, the true minimum distance must be at least $d_{\min} \geq \delta = n - k + 1$. However, any linear code has distance less than or equal to $n - k + 1$ by the Singleton bound (Section 5.3.2), so

$$d_{\min} = n - k + 1. \tag{5.4.36b}$$

and RS codes are ***maximum distance separable (MDS)***. Note also that, whereas the block length of the code is tied to the field size, the dimension k may be selected as any integer up to $n - 1$. For *all* such choices, we have an MDS code. The restriction on block lengths may seem quite constraining, but as with binary codes, RS codes can be shortened without decreasing $d_{\min}$ or changing $n - k$, and thus shortened RS codes are also MDS. Furthermore, Wolf [34] has shown that up to two stages of lengthening, that is, adding more information symbols, while keeping $n - k$ fixed, can be performed

[15] A subfield subcode of a code C over GF(q) is a code contained in C whose code symbols are in a subfield of GF(q).

without changing $d_{\min}$. These shortened or lengthened codes are not cyclic, but cyclic encoding and decoding procedures may be employed with minor modifications.

In defining the RS codes, the choice of j, that is, the starting exponent in the sequence of roots, is completely arbitrary ($d_{\min}$ and the dimension k are unaffected) and is commonly taken as $j = 1$. In contrast, for BCH codes, choice of j may affect the dimension and minimum distance of the code. Finally, the dual code for an (n, k) RS code is an $(n, n-k)$ RS code. In contrast, duals of BCH codes are not, in general, BCH codes.

A summary of the relevant q-ary RS code relations is given by

RS Relations

$$n = q - 1$$
$$n - k = \delta - 1$$
$$d_{\min} = \delta = n - k + 1 \tag{5.4.37}$$

The weight spectrum of MDS codes, including RS codes, is known in exact form [1] and is given by

$$A_w = C_w^n \sum_{m=0}^{w-(n-k+1)} (-1)^m C_m^w (q^{w-m-(n-k)} - 1), \qquad w = n - k + 1, \ldots, n. \tag{5.4.38}$$

It follows that the number of codewords having minimum nonzero weight is

$$A_{d_{\min}} = C_{d_{\min}}^n (q - 1). \tag{5.4.39}$$

Example 5.22 RS Codes over GF(16)

The natural block length is $n = q - 1 = 15$ code symbols. Suppose that we seek a $d_{\min} = 9$ code, producing $t = 4$. This implies by (5.4.37) that $n - k = 8$, or $k = 7$. [As a point of comparison, the (15, 7) BCH code over GF(2) has $d_{\min} = 5$.] This RS code has $2.7 \cdot 10^8$ codewords; each word has 75,075 nearest neighbors at distance 9.

If we wished to find a code with the same distance, but with only five information symbols, the original (15, 7) code could be shortened to (13, 5). On the other hand, the code could be lengthened to (16, 8) and (17, 9) [34], both retaining $d_{\min} = 9$. (Such modifications are described in Section 5.6.) The (16, 8) code has $R = \frac{1}{2}$, which is attractive in certain hardware implementations.

Example 5.23 RS Codes over GF(2^8) = GF(256)

Here the field size is 256, and the information and code symbols can be regarded as 8-bit bytes. The natural block length is $n = 255$. Selection of $d_{\min} = 33$ guarantees a $t = 16$ symbol-error-correcting capability. The number of information symbols is 223, and the fractional redundancy is only $\frac{32}{255} \approx \frac{1}{8}$. (It may seem remarkable that a high-rate code as this is can have such powerful error-correction features, but this is typical of codes on larger field sizes).

The generator polynomial for this code is a 32nd-degree polynomial over GF(256), certainly rather complicated to implement, with all arithmetic over GF(256). VLSI implementation of an encoder has been described in [35]; Blahut [36] also reports architectures for universal RS encoders and decoders, not restricted to this configuration. The particular

code cited here has become part of a NASA/ESA[16] deep-space coding standard (CCSDS), which employs the RS code in a concatenated arrangement with a trellis code. We will study this interesting (and very powerful) design later.

Compact disc players [32] also employ a RS code defined on 8-bit bytes for purposes of correcting electronic errors or surface defects in the disc; however, the codes used employ block length $n = 16$ and are referred to as shortened RS codes.

Reed–Solomon codes have three important applications. First, by choosing $q = M$, they are a natural means of coding for M-ary modulation. There is special advantage to nonbinary modulation for noncoherent detection on AWGN channels, fading channels, and certain spread-spectrum channels, as described in Chapter 4. Since RS codes are MDS, there is no need to look further among block codes for better q-ary codes, at least if Hamming distance is the measure of interest for codes, as it would be on uniform M-ary channels. Second, these codes may be used in conjunction with binary transmission, letting each q-ary symbol be represented as $m = \log_2 q$ bits. With this understanding, a RS code over GF(q) becomes an $(n \log_2 q, k \log_2 q)$ binary code. While perhaps not the design with largest Hamming distance (in binary units), the code does provide intrinsic robustness to bursty error phenomena. Specifically, if binary channel errors are confined to m bits and properly phased relative to symbol boundaries, the RS decoder sees these bursts as symbol errors in GF(q), so a burst is no worse than an isolated bit error. If the RS code has capability for correcting t symbol errors, it may correct single bursts up to length $m(t-1) + m - 1$ (now allowing for arbitrary placement of the burst with respect to symbol boundaries) or multiple shorter bursts. Consequently, RS codes have found application in computer memory systems organized with a byte-oriented memory architecture, where the binary storage mechanism exhibits error clustering [37]. Finally, RS codes are a frequent contributor to concatenated coding approaches, introduced by Forney [38], where RS codes form the outer code for a binary inner code. The outer code is normally seen as a "mop-up" code for residual inner-code errors, which tend to be bursty; hence the efficacy of a code over a larger field size. Concatenation schemes are discussed in more detail in Section 5.8.

5.5 DECODING OF CYCLIC CODES

In Section 5.2, we presented a general ML decoder for linear block codes when employed on a q-ary input, q-ary output channel. This procedure involved syndrome computation followed by table lookup of the most likely error pattern. This general procedure has two practical deficiencies. Most importantly, the table is kq^{n-k} symbols in size (normally only the k information symbols must be repaired), and this is unmanageably large for many codes of practical interest. Of lesser importance is the complexity of syndrome computation: matrix multiplication would require on the order of $n(n-k)$ GF(q) multiplications and additions. For general cyclic codes, a simpler procedure is available and we present this next. Following that discussion, we describe elegant decoding algorithms for decoding the most important of cyclic codes, the BCH/RS codes. In fact, it is the

[16]NASA denotes the National Aeronautics and Space Administration in the United States; ESA is the European Space Agency.

existence of efficient decoding algorithms that has made these codes popular. Such decoders are generally referred to as ***algebraic decoders***, for they utilize the algebra of finite fields to solve for most likely error patterns.

A shortcoming of all algebraic decoding procedures is that channel quality information in the form of symbol likelihoods is not easily incorporated, when available. One improvement is afforded by ***errors-and-erasures decoding***, wherein the demodulator produces an erasure symbol in code positions for which the decision has low confidence. This decoding algorithm can be treated as a modification of the standard errors-only algorithm. More ambitiously, we might be interested in ***maximum likelihood decoding*** for the channel at hand. Although ML decoding for general codes remains practically infeasible in many cases, it is readily implemented for certain small codes. Furthermore, procedures exist that approach maximum likelihood decoder performance by employing multiple decodings with an algebraic decoder. These will be discussed at the end of the section.

5.5.1 General-purpose Decoding of Cyclic Codes over GF(q)

Let us return to the general situation of transmitting a codeword $\mathbf{x}$ from a cyclic code by a uniform q-ary input, q-ary output channel. We seek the codeword that is nearest in Hamming distance to the received sequence, $\mathbf{r}$. The additional structure resident in a cyclic code provides a twofold economy of implementation relative to the general syndrome decoder of Section 5.2. First, computing the syndrome may be done with a simple feedback shift register similar to the encoder circuit. This is an improvement over building (or programming) a general-purpose multiplication of the received sequence $\mathbf{r}$ with a parity check matrix $\mathbf{H}$. Second, once the syndrome is computed, it is possible to iteratively determine which code positions should apparently be corrected by cycling the syndrome register.

We again represent the actions of the channel by

$$\mathbf{r} = \mathbf{x} + \mathbf{e}, \tag{5.5.1}$$

where $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ represents the transmitted codeword and $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$ is the error sequence, with both sequences from GF(q). In polynomial form, we express the channel action by

$$r(D) = x(D) + e(D). \tag{5.5.2}$$

Now consider division of $r(D)$ by $g(D)$, the generator polynomial for the code. This is motivated by recalling that valid codewords are exactly divisible by $g(D)$; that is, they produce zero remainder upon such division. We shall denote the remainder of this division as another polynomial, $s(D)$, the ***syndrome polynomial:***

$$\begin{aligned} s(D) &= s_0 + s_1 D + \cdots + s_{n-k} D^{n-k} \\ &= r(D) \bmod g(D) = [x(D) + e(D)] \bmod g(D) \\ &= x(D) \bmod g(D) + e(D) \bmod g(D) \\ &= e(D) \bmod g(D) \end{aligned} \tag{5.5.3}$$

We have invoked the distributive property of the modulus operation (see Exercise 5.5.1), and the last step follows from the fact that $g(D)$ divides $x(D)$.

Euclid's division theorem implies that the syndrome polynomial is exactly determined by the error sequence, and $s(D)$ will have degree $n-k-1$ or less. Of course, for every $s(D)$ there are q^k solutions for $e(D)$ corresponding to the q^k choices for the transmitted codeword, and our task is to find the most likely error pattern. Again, this corresponds to locating a minimum-weight error pattern. As argued in Section 5.2, initial reduction of the received vector to the syndrome vector entails no loss in ability to infer the most likely error pattern.

To compute the syndrome polynomial, we require a circuit for computing the remainder upon polynomial division. We have provided such a circuit in Figure 5.4.5, as a step toward formulating the systematic encoder, and repeat the generic structure of the ***syndrome computer*** in Figure 5.5.1, a device having $n-k$ q-ary register cells.[17] The circuit is clocked n times, at which time the syndrome vector (or polynomial) resides in the $(n-k)$-stage register, s_0 in the leftmost memory cell. Notice also that the encoder and corresponding syndrome computer differ only in the location of the input to the register and in the number of cycles to perform. This allows essentially the same circuitry (or chip) to perform both encoding and syndrome computation.

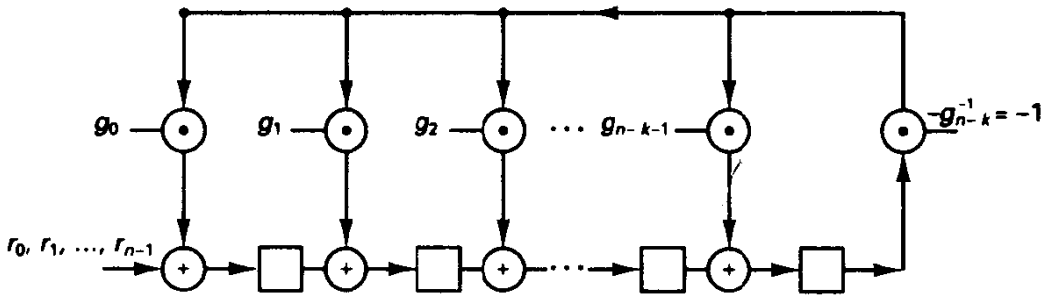


Figure 5.5.1 Syndrome forming circuit to compute $r(D)/g(D)$.

Once the syndrome vector is determined, we have several courses of action. If all we seek is error detection (mode 1 of Section 5.2), we merely check for a zero syndrome, since $s(D) = 0$ if and only if $r(D)$ corresponds to a valid codeword. Thus, error detection with cyclic codes is especially simple. More attention to this important topic is given in Section 5.7.

Usually, however, we will want to attempt error correction. Given the syndrome we could proceed with table lookup of the coset leader, the minimum-weight error pattern consistent with the observed syndrome. When q^{n-k} is reasonable in size, this is the preferred choice nowadays due to the availability of fast, inexpensive read-only memory. When such an approach is not feasible, we may exploit the following property

[17] An alternative implementation has k cells and is more efficient for low-rate codes; however, most applications of modern interest are such that $n-k$ is smaller than k.

of cyclic codes:

If $s(D)$ is the syndrome corresponding to a received sequence $r(D)$, then the syndrome of a right-cyclic shift of $r(D)$, denoted $r^{(1)}(D)$, is

$$s^{(1)}(D) = r^{(1)}(D) \bmod g(D) = Ds(D) \bmod g(D). \tag{5.5.4}$$

Thus, to obtain the syndrome polynomial for $r^{(1)}(D)$, we merely right-shift the original syndrome and then reduce modulo $g(D)$. This operation can be implemented with the circuit of Figure 5.5.1 by disabling further inputs from the left.

We can justify (5.5.4) as follows. From the fundamental properties of cyclic shifting [see (5.4.4)]

$$Dr(D) = r_{n-1}(D^n - 1) + r^{(1)}(D) \tag{5.5.5}$$

so

$$r^{(1)}(D) = Dr(D) - r_{n-1}(D^n - 1). \tag{5.5.6}$$

We may use Euclid's theorem to uniquely express the left-hand side of (5.5.6) as $r^{(1)}(D) = a(D)g(D) + z(D)$. This defines $z(D)$ as the remainder upon division of $r^{(1)}(D)$ by $g(D)$ or, equivalently, $z(D) = s^{(1)}(D)$ is the syndrome of $r^{(1)}(D)$. Furthermore, we have by definition that $r(D) = b(D)g(D) + s(D)$, and we can also recall that $g(D)h(D) = D^n - 1$. Substituting these relations into (5.5.6) yields

$$r^{(1)}(D) = D[b(D)g(D) + s(D)] - r_{n-1}g(D)h(D). \tag{5.5.7}$$

By rewriting (5.5.7) and substituting for $r^{(1)}(D)$, we find that

$$Ds(D) = [a(D) + r_{n-1}h(D) - Db(D)]g(D) + z(D). \tag{5.5.8}$$

Therefore, $z(D)$ is also the remainder upon dividing $Ds(D)$ by $g(D)$, which thus shows that $z(D) = Ds(D) \bmod g(D) = s^{(1)}(D)$.

We may exploit this property of the syndrome to sequentially correct suspected errors in the n positions of a codeword, assuming that the error pattern is indeed correctable. To begin, we examine $s(D)$ after the syndrome is computed and decide whether the given syndrome corresponds to a correctable error pattern with a nonzero symbol in the *highest-order* position of the received sequence. That is, we infer whether e_{n-1} is nonzero and, if so, its most likely value. (In the binary case, we simply ask whether an error occurred in position $n - 1$ or not.) In essence, we are asking for the *leading* symbol of the coset leader corresponding to the observed syndrome, and this symbol can be produced by a device having $n - k$ q-ary input lines and a single q-ary output line, implemented either with combinational logic or read-only memory. (Note that the complexity of this function is considerably less than that which estimates the entire error pattern at once.)

After e_{n-1} is resolved (and r_{n-1} repaired by subtracting e_{n-1}), we perform a cyclic shift of the syndrome register with further input inhibited. If e_{n-1} was determined to be zero, then this cyclic shift produces the syndrome $s^{(1)}(D) = Ds(D) \bmod g(D)$ and reveals whether a correctable error exists in position $n-2$. This cycling of the syndrome register would continue to correctly indicate the presence or absence of error as long as

errors were not previously found. Once an error is located, however, we must remove its influence from the syndrome so that future error positions can be properly estimated.

To accomplish this correction, suppose that there is a correctable error in position $n-1$ with value e_{n-1}. We wish to compute the syndrome that would have been produced had this error symbol been removed at the outset; that is, we wish to postcompensate the current syndrome for the influence of e_{n-1}. The received polynomial, after error correction, is

$$\tilde{r}(D) = r(D) - e_{n-1}D^{n-1}. \tag{5.5.9}$$

A right-cyclic shift of this received vector would correspond to the polynomial

$$\tilde{r}^{(1)}(D) = (r_{n-1} - e_{n-1}) + r_0 D + \cdots + r_{n-2}D^{n-1}. \tag{5.5.10}$$

The syndrome of this vector would be

$$\tilde{s}^{(1)}(D) = \tilde{r}^{(1)}(D) \bmod g(D), \tag{5.5.11}$$

which, again by the distributive property is,

$$\begin{aligned} \tilde{s}^{(1)}(D) &= r^{(1)}(D) \bmod g(D) - e_{n-1} \bmod g(D) \\ &= Ds(D) \bmod g(D) - e_{n-1} \bmod g(D) \\ &= Ds(D) \bmod g(D) - e_{n-1}. \end{aligned} \tag{5.5.12}$$

Therefore, the proper means to remove the influence of a symbol thought to be in error is to subtract the estimate of such symbols from the left-hand input to the syndrome computation circuit. After n iterations of this estimate/cycle-and-correct operation, we will have removed all errors in a correctable error pattern. Of course, if the original error pattern is not correctable by a table-lookup decoder as discussed in Section 5.2, the cyclic testing procedure here will be unable to correct the error pattern as well.

The decoder just described is sometimes referred to as a Meggit decoder [39] and is illustrated in general form in Figure 5.5.2. While it is applicable to any q-ary cyclic code, its shortcomings are that the complexity of the error decision circuit is still rather

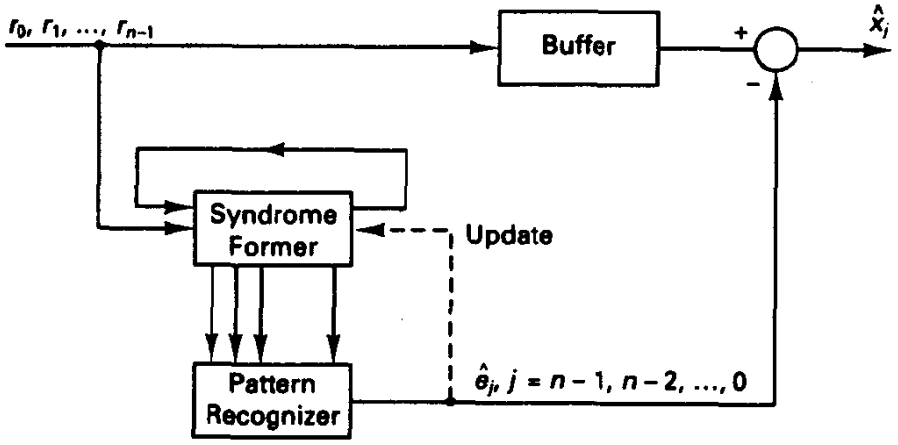


Figure 5.5.2 Meggit decoder for general cyclic codes. Following computation of $s(D)$, syndrome register is cycled another $n-1$ times.

forbidding if $n - k$ is large. Furthermore, $n - 1$ additional cycles are needed after the initial syndrome computation, whereas the table-lookup decoder operating on the original syndrome could immediately find the *entire* error pattern. Clearly, the two approaches trade memory requirements (space) against decoding speed (time).

Example 5.24 Meggit Decoding for (127, 113) Binary BCH Code

To illustrate the preceding discussion, consider the double-error-correcting binary (127, 113) BCH code. The syndrome computation can be performed with a 14-bit shift register with feedback taps set according to the generator polynomial for the code. After 127 received symbols have entered the syndrome computer, $s(D)$ resides in the register, whereupon table lookup of the entire error pattern in a table with $2^{14} = 16,384$ words could produce the error pattern.[18] (Instead of storing the complete binary error pattern of 127 bits, we may instead store the error locations of up to two error positions, which consumes at most 14 bits per table entry.)

To illustrate the Meggit decoder principle, however, we note that there is one syndrome corresponding to a single error in position 127 and 126 syndromes corresponding to a 2-error pattern having one of the errors in position 127. (These syndromes are distinct by virtue of the fact that the code is guaranteed to be double error correcting.) Thus, a total of 127 distinct 14-bit syndromes need to be recognized by the error pattern detector. To implement this function efficiently in combinational logic, we would minimize the Boolean function whose value is 1, or "true," only for the 127 special 14-bit syndrome input vectors.

Although we could readily show that the gate complexity of the error pattern recognizer is much less than that of the complete table-lookup approach, we suffer the rather slow operation of the Meggit decoder: 127 cycles of the error pattern testing process are required to complete a decoding cycle. Of course, these cycles might be short compared to a bit duration, whence the decoding delay is short relative to a codeword duration. In any case, some additional buffering would be required to handle incoming data of the next codeword while decoding of a given word is completed.

For some codes it is possible to simplify the error recognition process by performing a majority vote on certain syndrome positions in the register at each cycle. These codes are known as ***majority logic decodable***. Interested readers are referred to Lin and Costello [3] for an ample treatment of this class of codes. Generally, however, applications have evolved to the use of longer, more powerful codes, for which the complexity of the Meggit or table-lookup approach is infeasible, even with the rapid advance of memory technology. Based on additional algebraic structure, BCH and RS codes have decoding algorithms that involve direct solution for the error pattern from the syndrome. Although far less simple to describe than the decoding procedures presented thus far, the codes have been widely adopted in modern communications practice.

5.5.2 Algebraic (Errors Only) Decoding of BCH Codes and RS Codes

Because the BCH family is such a large class of codes and because in typical coding applications these codes are essentially as powerful as more general block codes, a large amount of attention has been given to efficient decoding algorithms. Clever utilization of

[18] Such a memory is referred to as a 16K ROM, a rather elementary requirement these days.