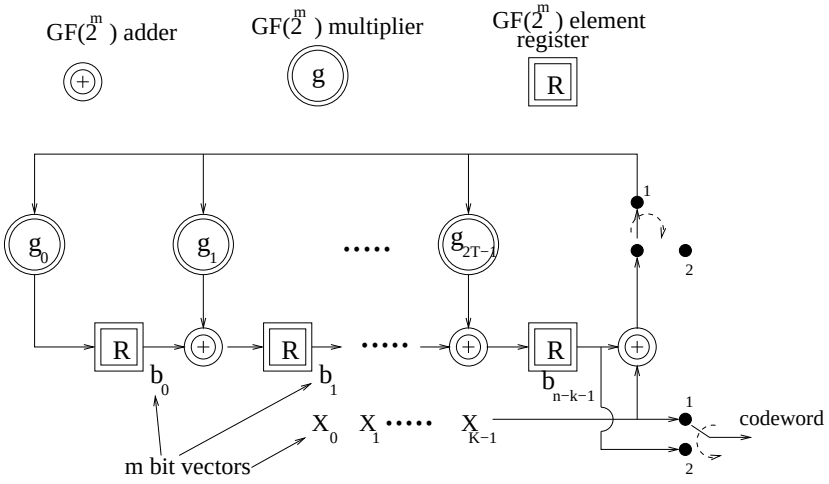
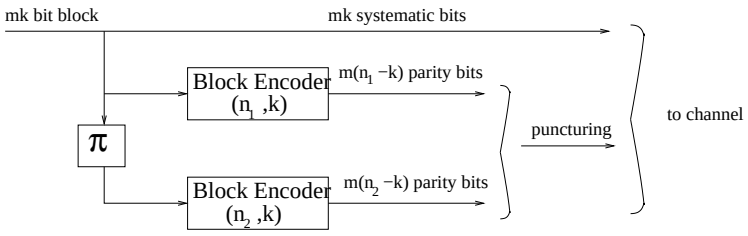
ECE8771
Information Theory & Coding for Digital Communications

Villanova University
ECE Department

Prof. Kevin M. Buckley

Lecture Set 2
Block Codes





Contents

List of Figures

7 Block Codes

In the last section of this Course we established that the channel capacity C is the lower bound for the rate R at which information can be reliably transmitted over a given channel. First we saw that, as the number of orthogonal waveforms increases to infinity, an orthogonal modulation scheme can provide rates approaching this capacity limit. We then noted that channel capacity can be approached with randomly selected codewords, by using codewords whose lengths goes to infinity. We then stated the noisy channel coding theorem, which establishes that reliable communications is possible over any channel as long as the transmission information rate R is not greater than the channel capacity C. This is a general result applying to any channel. Figure 51 illustrates channel coding for a general channel. At the heart of the coding scheme is the forward error correction (FEC) code, which facilitates the detection and/or correction of information bit transmission errors. Optionally, the channel may include an automatic repeat request (ARQ) capability, whereby received information bit errors are detected, initiating a request by the receiver for the transmitter to resend the faulty information bits. In this Course we focus on FEC coding. ARQ coding schemes employ FEC codes to detect errors for the ARQ scheme.

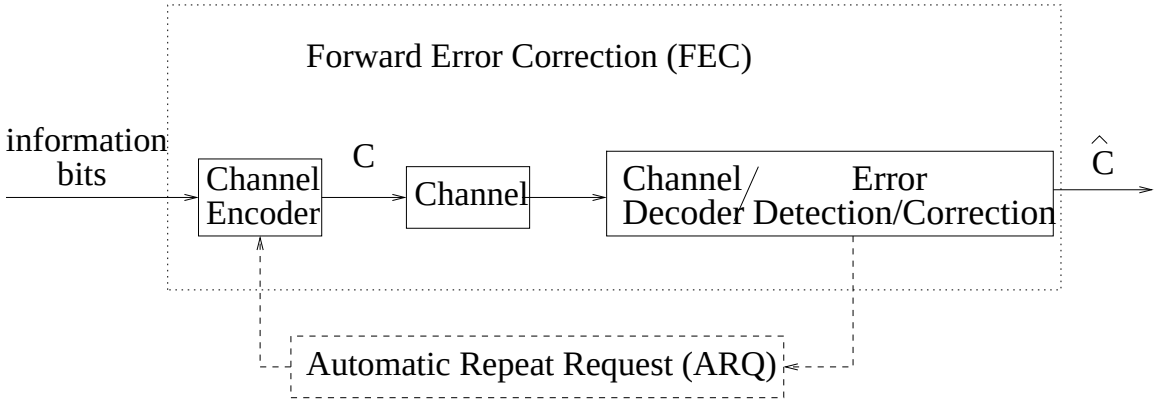


Figure 51: General channel coding block diagram.

We now turn our attention to practical channel coding. First, in this Section, we consider basic block codes. Next, in Section 8 of the Course, we cover standard convolutional codes. In these Sections we will describe channel coding methods which are commonly used in application. We will be concerned mainly with code rate, codeword generation, AWGN channels, hard and soft decision decoding, and bit error rate performance analysis. In Section 9 we will discuss some important recent developments in channel coding. Consideration of bandwidth requirements is postponed to Section 11 of the Course.

7.1 Introduction to Block Codes

In a block code, k information bits are represented by a block of N symbols to be transmitted. That is, as illustrated in Figure 52, a vector of k information bits is represented by a vector of N symbols. The N dimensional representation is called a *codeword*. Generally, the elements of a codeword are selected from an alphabet of size q. Since there are $M = 2^k$ unique input information vectors, and q^N unique codewords, $q^N \geq 2^k$ is necessary for unique representation of the information. (Note that $q^N = 2^k$ is not of interest, since without redundancy errors cannot be detected and/or corrected.) If the elements of the codewords are binary, the code is called a *binary block code*. For a binary block code, with codeword length n, $n > k$ is required. For nonbinary block codes with alphabet size $q = 2^b$, the codeword length, converted to bits, is $n = bN$.

A block code is a *linear block code* if it adheres to a linearity property which will be identified below. Because of implementation and performance considerations, essentially all practical block codes are linear, and we will therefore restrict our discussion to them. We will begin by focusing on binary linear block codes. Non-binary linear linear codes can be employed to generate long codewords. We will close this Section with a description of a popular class of nonbinary linear block codes – Reed-Solomon codes.

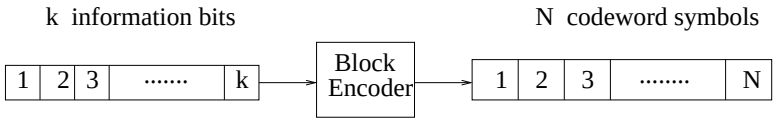


Figure 52: Block code encoding.

A binary block code of k length input vectors and n length codewords is referred to a an (n, k) code. The *code rate* of an (n, k) code is $R_c = \frac{k}{n}$. Note that $R_c < 1$. The greater R_c is, the more efficient the code is. On the other hand, the purpose of channel coding is to provide protection against transmission errors. For well designed block codes, error protection will improve in some sense as R_c decreases. To begin consideration of code error protection characteristics, we define the codeword *weight* as the number of non-zero elements in the codeword. Considering the M codewords used for a particular binary block code to represent the $M = 2^K$ input vectors, the *weight distribution* is the set of all M codeword weights.

We will see below that the weight distribution of a linear block code plays a fundamental role in code performance (i.e. error protection capability). We begin by building a mathematical foundation – an algebra for finite alphabet numbers. Based on this we will then develop a general description of a binary linear block code, and we will describe some specific codes which are commonly used. We will then consider decoding, describing two basic approaches to decoding binary linear block codes (hard and soft decision decoding), and we will evaluate code/decoder performance. Finally we will overview several advanced topics, including: non-binary linear block coding (e.g. Reed-Solomon codes), interleaving and concatenated codes (e.g. turbo codes).

7.2 A Galois Field Primer

In this Subsection we introduce just enough Galois field theory to get started considering binary linear block codes. Later, for our coverage of Reed-Solomon codes, we will expand on this.

Elements and Groups

Consider a set of *elements* $\mathcal{G}$, along with an operator "*" that uniquely defines an element $c = a*b$ from elements a and b (where $a, b, c \in \mathcal{G}$). If $\mathcal{G}$ and the operator satisfy the following properties:

- associativity: $a * (b * c) = (a * b) * c$ all $a, b, c \in \mathcal{G}$
- identity element e: such that $a * e = e * a = a$
- inverse element $a^{'}$ for each a: such that $a * a^{'} = a^{'} * a = e$ all $\in \mathcal{G}$,

then $\mathcal{G}$, along with operator $*$, is called a *group*. Additionally, if

- $a * b = b * a$ all $a, b \in \mathcal{G}$,

the group is *commutative*. The number of elements in a group is called the *order* of the group. Here we are interested in *finite order groups*. A group is an *algebraic* system.

Algebraic Fields

Another system of interest is an *algebraic field*. An algebraic field consists of a set of elements (numbers) along with defined addition and multiplication operators on those numbers that adhere to certain properties. Consider a field $\mathcal{F}$ and elements a, b, c of that field (i.e. $a, b, c \in \mathcal{F}$). Let $a + b$ denote the addition of a and b, and ab denote multiplication.

The addition properties are:

- *Closure*: $a + b \in \mathcal{F}$; $\forall\ a, b \in \mathcal{F}$.
- *Associativity*: $(a + b) + c = a + (b + c)$; $\forall\ a, b, c \in \mathcal{F}$.
- *Commutativity*: $a + b = b + a$; $\forall\ a, b \in \mathcal{F}$.
- *Zero element*: There exist an element in $\mathcal{F}$, called the zero element and denoted 0, such that $a + 0 = a$; $\forall\ a \in \mathcal{F}$.
- *Negative elements*: For each $a \in \mathcal{F}$, there is an element in $\mathcal{F}$, denoted $-a$, such that $a + (-a) = 0$. *Subtraction*, denoted $-$, is defined as $a - b = a + (-b)$.

The multiplication properties are:

- *Closure*: $ab \in \mathcal{F};\ \forall\ a, b \in \mathcal{F}$.
- *Associativity*: $(ab)c = a(bc);\ \forall\ a, b, c \in \mathcal{F}$.
- *Commutativity*: $ab = ba;\ \forall\ a, b \in \mathcal{F}$.
- *Distributivity of multiplication over addition*: $a(b + c) = ab + ac;\ \forall\ a, b, c \in \mathcal{F}$.
- *Identity element*: There exist an element in $\mathcal{F}$, called the identity element and denoted 1, such that $a(1) = a;\ \forall\ a \in \mathcal{F}$.
- *Inverse elements*: For each $a \in \mathcal{F}$ except 0, there is an element in $\mathcal{F}$, denoted a^{-1}, such that $a(a^{-1}) = 1$. *Division*, denoted $\div$, is defined as $a \div b = ab^{-1}$.

The Binary Galois Field GF(2):

As mentioned earlier, most of our discussion of block codes will focus on binary codes. Thus, we will mainly deal with the binary field, GF(2). This field consists of two elements, $\{0, 1\}$. That is, its consists of only the zero and identity elements. For GF(2), the addition operator is:

$$a + b \ = \ \begin{cases} 0 & a = 0,\ b = 0 \\ 1 & a = 1,\ b = 0 \\ 1 & a = 0,\ b = 1 \\ 0 & a = 1,\ b = 1 \end{cases} \quad . \tag{1}$$

The multiplication operator is:

$$ab \ = \ \begin{cases} 0 & a = 0,\ b = 0 \\ 0 & a = 1,\ b = 0 \\ 0 & a = 0,\ b = 1 \\ 1 & a = 1,\ b = 1 \end{cases} \quad . \tag{2}$$

Prime and Extension Fields

Let q be a prime number. A q-order *Galois field* GF(q) is a prime-order finite-element field, with the addition and multiplication operators are defined as modulo operations (i.e. mod q). This is the class of fields we are interested in since we are talking about coding (i.e. bits and symbols). We label the elements of GF(q) as $\{0, 1, \cdots, q - 1\}$. Note that if q is not prime, then $\mathcal{G} = 1, 2, \cdots, q - 1$ is not a group under modulo q multiplication. For prime numbers q, and integers m, a GF(q) field is called a prime field, and a GF(q^m) field is an *extension field* of GF(q).

Polynomials

Consider polynomials $f(p)$ and $g(p)$ with variable p, degree (highest power of p) m, and coefficients in field $GF(q)$:

$$\begin{aligned} f(p) &= f_m p^m + f_{m-1} p^{m-1} + \cdots + f_1 p + f_0 \\ g(p) &= g_m p^m + g_{m-1} p^{m-1} + \cdots + g_1 p + g_0 \ . \end{aligned} \tag{3}$$

The addition of these two polynomials is

$$f(p) + g(p) \ = \ (f_m + g_m) p^m + (f_{m-1} + g_{m-1}) p^{m-1} + \cdots + (f_1 + g_1) p + (f_0 + g_0) \ , \tag{4}$$

and their multiplications is

$$f(p) \ g(p) \ = \ c_{2m} p^{2m} + c_{2m-1} p^{2m-1} + \cdots + c_1 p + c_0 \tag{5}$$

where

$$\begin{aligned} c_{2m} &= f_m g_m; \ \ c_{2m-1} = f_m g_{m-1} + f_{m-1} g_m; \ \ c_{2m-2} = f_m g_{m-2} + f_{m-1} g_{m-1} + f_{m-2} g_m; \\ &\cdots \ c_1 = f_1 g_0 + f_0 g_1 \ ; \ \ c_0 = f_0 g_0 \end{aligned} \tag{6}$$

(i.e. the convolution of the $f(p)$ coefficient sequence with the $g(p)$ coefficient sequence). Above, all arithmetic is GF(q) (i.e. modulo q) arithmetic.

Example 7.1: Consider two 3-rd order polynomials over $GF(2)$, $f(p) = p^3 + p^2 + 1$ and $g(p) = p^2 + 1$. Then,

$$c(p) \ = \ f(p) \cdot g(p) \ = \ (p^3 + p^2 + 1) \cdot (p^2 + 1) \ = \ p^5 + p^4 + p^3 + 1 \quad . \tag{7}$$

Equivalently, convolving the vector $F = [1 \ 1 \ 0 \ 1]$ with $G = [0 \ 1 \ 0 \ 1]$ we get $C = [1 \ 1 \ 1 \ 0 \ 0 \ 1]$.

Let $f(p)$ be of higher degree than $g(p)$. Then $f(p)$ can be divided by $g(p)$ resulting in

$$f(p) \ = \ q(p) \ g(p) \ + \ r(p) \tag{8}$$

where $q(p)$ and $r(p)$ are the quotient and remainder, respectively.

Example 7.2: Divide the GF(2) polynomial $f(p) = p^6 + p^5 + p^4 + p + 1$ by GF(2) polynomial $g(p) = p^3 + p + 1$.

$$\begin{array}{r} p^3 + p^2 \\ p^3 + p + 1 \ \overline{)\ p^6 + p^5 + p^4 \qquad + p + 1} \\ \qquad\quad p^6 \qquad + p^4 + p^3 \\ \hline \qquad\quad p^5 \qquad + p^3 \qquad + p + 1 \\ \qquad\quad p^5 \qquad + p^3 + p^2 \\ \hline \qquad\qquad\qquad\qquad p^2 + p + 1 \end{array} \tag{9}$$

Thus, $q(p) = p^3 + p^2$ and $r(p) = p^2 + p + 1$.

The quotient $q(p)$ and remainder $r(p)$ of a binary polynomial division $\frac{x(p)}{g(p)}$ can be efficiently implemented using the feedback shift register structure shown in Figure 53. This structure will be used later for both encoding and decoding.

Let k and m be the degrees of $x(p)$ and $g(p)$ respectively. Assume $x_k = 1$. Initially, the shift register is loaded with zeros. The coefficients of $x(p)$ are loaded into the shift register in reverse order, i.e. starting with x_k at time $i = 0$. At time $i = m$ the first quotient coefficient is generated, q_{k-m}, and after the additions the shift registers contain the remainder coefficients for the 1^{st} stage of the long division. For each i thereafter, the next lower quotient coefficient, q_{k-i}, and corresponding remainder coefficients are generated. The process stops at time $i = k$, at which time the shift register contains the final remainder coefficients in reverse order.

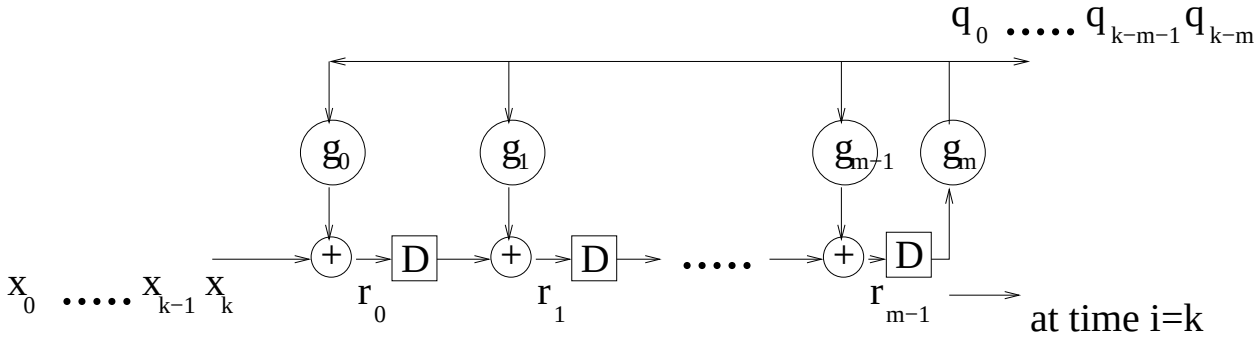


Figure 53: Feedback shift register for binary polynomial division.

A *root* of a polynomial $f(p)$ is a value a such that $f(a) = 0$. Given root a of a polynomial $f(p)$, then $p - a$ is a *factor* of $f(p)$. That is, $f(p)$ divided by $p - a$ has zero remainder. An *irreducible polynomial* is a polynomial with no factors. Examples of irreducible polynomials in GF(2) are:

$$\begin{array}{l} p^2 + p + 1 \\ p^3 + p + 1 \\ p^4 + p + 1 \\ p^5 + p^2 + 1 \end{array} \tag{10}$$

It can be shown that any irreducible polynomial in GF(2) of degree m is a factor of $p^{2^m-1}+1$. A *primitive polynomial* $g(p)$ is an irreducible polynomial of degree m such that $n = 2^m - 1$ is the smallest integer such that $g(p)$ is a factor of $p^n + 1$. The polynomials is Eq(10) are all primitive.

Vector Spaces in GF(q)

Let $\mathcal{V}$ be the set of all vectors with N elements, with each element from a GF(q) field (i.e. a set of q^N vectors in GF(q)). $\mathcal{V}$ is called the vector space over GF(q) because:

- $\mathcal{V}$ is commutative under modulo-q addition – for any $U, V \in \mathcal{V}$,
- $\mathcal{V}$ is closed under vector addition and scalar multiplication – for any $a \in$ GF(q) and $U, V \in \mathcal{V}$, $aV \in \mathcal{V}$ and $U + V \in \mathcal{V}$

- distributivity of scalar multiplication over vector addition holds – for any $a \in \mathrm{GF}(q)$ and $U, V \in \mathcal{V}$, $a(U+V) = aU + aV$
- distributivity of vector multiplication over scalar addition holds – for any $a, b \in \mathrm{GF}(q)$ and $V \in \mathcal{V}$, $(a+b)V = aV + bV$
- associativity holds for vector addition and scalar multiplication – for any $a, b \in \mathrm{GF}(q)$ and $U, V, W \in \mathcal{V}$, $(ab)V = a(bV)$ and $(U+V)+W = U+(V+W)$
- there exists a vector additive identity – for any $V \in \mathcal{V}$, $V + 0_N = V$, where O_N is the vector of N zeros (note $0_N \in calV$)
- there exists an additive inverse – for each $V \in \mathcal{V}$ there exists a $U \in \mathcal{V}$ such that $V + U = 0_N$
- there exists a scalar multiplicative identity – for each $V \in \mathcal{V}$, $1V = V$.

These are the same requirements we are familiar with for a Euclidean vector space.

For coding, it is standard to consider vectors as column vectors. The *inner product* of a vector V with U is $V\ U^T$, where the superscript "T" denotes transpose.

Example 7.3: Consider the $N = 4$ dimensional vector space over GF(2). Given two vectors, say $\mathbf{C}_1 = [1\ 1\ 0\ 1]$ and $\mathbf{C}_2 = [0\ 1\ 0\ 1]$,

$$\mathbf{C}_1 \ + \ \mathbf{C}_2 \ = \ [1\ 1\ 0\ 1] \ + \ [0\ 1\ 0\ 1] \ = \ [1\ 0\ 0\ 0] \quad . \tag{11}$$

Any vector is its own additive inverse, e.g.

$$\mathbf{C}_1 \ + \ \mathbf{C}_1 \ = \ [1\ 1\ 0\ 1] \ + \ [1\ 1\ 0\ 1] \ = \ [0\ 0\ 0\ 0] \quad . \tag{12}$$

Also,

$$\mathbf{C}_1\ \mathbf{C}_2^T \ = \ [1\ 1\ 0\ 1]\ [0\ 1\ 0\ 1]^T \ = \ 0 \quad . \tag{13}$$

Codewords & Hamming Distance

Consider an (n, k) block code with codewords $\mathbf{C}_i;\ i = 1, 2, \cdots, M$. The *Hamming distance* d_{ij} between codewords $\mathbf{C}_i$ and $\mathbf{C}_j$ is defined as the number of elements that are different between the codewords. Note that $0 \leq d_{ij} \leq n$, with $d_{ij} = 0$ indicating that $\mathbf{C}_i = \mathbf{C}_j$. We refer to

$$\min\{d_{ij}\};\ i, j = 1, 2, \cdots, M;\ i \neq j \tag{14}$$

as the *minimum distance*, and denote it as d_{min}. We will see that d_{min} is the principal performance characteristic of practical block codes.

7.3 Linear Block Codes

In this Subsection we describe a general class of block codes, *linear block codes*, which constitutes essentially all practical block codes. We begin with the general field GF(q), and switch specifically to a discussion in terms of GF(2) starting with the topic **Generator Matrix** below.[1] In Subsections 7.5 and 7.7 respectively, we then describe specific codes for the most common fields used for linear block codes – prime field GF(2) and the extension field GF(2^m).

Linear Block Codes

Consider a k-dimensional information bit vector, $\mathbf{X}_m = [x_{m1}, x_{m2}, \cdots, x_{mk}]$. There are $M = 2^k$ of these vectors, $\mathbf{X}_m;\ m = 1, 2, \cdots, M$. The block coding objective is to assign to each of these $\mathbf{X}_m$ a unique N-dimensional codeword $\mathbf{C}_i;\ i = 1, 2, \cdots, M$, where

$$\mathbf{C}_m = [c_{m1}, c_{m2}, \cdots, c_{mN}] \tag{15}$$

and the codeword symbols (or elements) are $c_{mj} \in \text{GF}(q)$.

The block code is termed a *linear* code if, given any two codewords $\mathbf{C}_i$ and $\mathbf{C}_j$ and any two scalars $a_1, a_2 \in \text{GF}(q)$, we have that $a_1\mathbf{C}_i + a_2\mathbf{C}_j$ is also a codeword. As we shall see, linear block codes offer several advantages, including that, relative to nonlinear codes, they are: easily encoded; easily decoded; and easily analyzed (i.e. their performance can be simply characterized).

Note that the N dimensional vector of all zeros, $\underline{0}_N$, must be a codeword for a linear code since $a_1\mathbf{C}_i - a_1\mathbf{C}_i = \underline{0}_N$. Also, let $\mathcal{S}$ denote the N-dimensional vector space of all N-dimensional vectors for GF(q). As noted earlier, we consider these to be column vectors. There are q^N vectors in this space. Consider $k < N$ linear independent vectors in $\mathcal{S}$:

$$\{\mathbf{g}_i;\ i = 1, 2, \cdots, k\} \quad . \tag{16}$$

(These vectors are linearly independent if no one vector can be written as a linear combination of the others. For GF(q), by linear combination we mean weighted sum where the weights are elements of GF(q).) The set of all linear combinations of these k vectors form a k-dimensional subspace $\mathcal{S}_c$ of $\mathcal{S}$. These k vectors form a basis for $\mathcal{S}_c$. If these k vectors are orthonormal , i.e. if

$$\mathbf{g}_i\ \mathbf{g}_j^T = \delta(i - j) \quad , \tag{17}$$

where GF(q) algebra is assumed, then they form an orthonormal basis for $\mathcal{C} = \mathcal{S}_c$. We denote as $\overline{\mathcal{C}}$ the null space of $\mathcal{C}$. It is the $(N - k)$-dimensional subspace containing all of the vectors orthogonal to $\mathcal{C}$.

> An N-dimensional block code is linear if and only if its codewords, $\mathbf{C}_i;\ i = 1, 2, \cdots, M$, fill a k-dimensional subspace of the N-dimensional vector space in GF(q). Then, each codeword can be generated as a linear combination of k N-dimensional basis vectors, $\mathbf{g}_i;\ i = 1, 2, \cdots, k$, for the code space. We call this set of codewords, or equivalently the subspace they fill, the code. We denote this linear block code $\mathcal{C}$.

[1] The GF(2) discussion subsequent to that point would generalize to GF(q) fields in a straightforward manner, by grouping information bits into a GF(q) representation before coding.

Codeword Weights and Code Weight Distribution

The *weight* of a codeword is defined as the number of its non-zero elements. For a linear block code, we use the convention $\mathbf{C}_1 = \underline{0}_N$. We let w_m denote the weight of the m-th codeword. So, the weight of a codeword is its Hamming distance from $\mathbf{C}_1$ – i.e. $w_1 = 0$ and $d_{1m} = w_m;\ m = 1, 2, \cdots, M$. The function, number of codewords wn_j vs. weight number $j = 0, 1, \cdots, N$, is called the *weight distribution* of a linear code.

For a binary linear block code (i.e. a code in GF(2), the Hamming distance d_{ij} between codewords $\mathbf{C}_i$ and $\mathbf{C}_j$ is the weight of $\mathbf{C}_i + \mathbf{C}_j$, which is also a codeword. Thus, the minimum distance between ant codewords is:

$$d_{min} \ = \ \min_{m, m \neq 1} \{w_m\} \quad . \tag{18}$$

For GF(2), the n-dimensional vector space $\mathcal{S}$ consists of 2^n vectors. A code (n, k) consists of 2^k codewords which are all of the vectors in the k-dimensional subspace $\mathcal{C}$ spanned by these codewords. The $(n-k)$-dimensional null space of $\mathcal{C}$, $\overline{\mathcal{C}}$, contains 2^{n-k} vectors.

> The weight distribution wn_j and the minimum distance d_{min} of a linear block code $\mathcal{C}$ determine the performance of the code. That is, they characterize the ability to differentiate between codewords received in additive noise.

The Generator Matrix

One of the advantages of linear block codes is the relative ease of codeword generation. Here we describe one way to generate codewords. In this Subsection, from this point on, we restrict the description to binary codes. The extension to general GF(q) codes is straightforward.

Consider the set of k-dimensional vector of k information bits,

$$\mathbf{X}_m \ = \ [x_{m1}, x_{m2}, \cdots, x_{mk}]; \qquad m = 1, 2, \cdots, M = 2^k \quad . \tag{19}$$

To each $\mathbf{X}_m$, a codeword

$$\mathbf{C}_m \ = \ [c_{m1}, c_{m2}, \cdots, c_{mn}] \tag{20}$$

is assigned, where the c_{mj} are codeword bits. For a linear binary block code, a codeword bit c_{mj} is generated as a linear combination of the information bits in $\mathbf{X}_m$:

$$c_{mj} \ = \ \sum_{i=1}^{k} x_{mi} \ g_{ij} \quad . \tag{21}$$

Let

$$\mathbf{g}_i \ = \ [g_{i1}, g_{i2}, \cdots, g_{in}] \quad . \tag{22}$$

The vectors $\mathbf{g}_i; i = 1, 2, \cdots, k$ characterize the particular (n, k) code $\mathcal{C}$ Specifically, they form a basis for the code subspace $\mathcal{C}$. The $\mathbf{g}_i; i = 1, 2, \cdots, k$ must be linearly independent in order to represent the $\mathbf{X}_m$ without ambiguity.

In vector/matrix form, the codewords are computed from the information vectors as

$$\mathbf{C}_m \ = \ \mathbf{X}_m \ \mathbf{G} \tag{23}$$

where

$$\mathbf{G} \;=\; \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_k \end{bmatrix} \;=\; \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \cdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix} \tag{24}$$

is the $(k \times n)$-dimensional code *generation matrix*. Note that codewords formed in this manner constitute a linear code. The k n-dimensional vectors $\mathbf{g}_i$ form a basis for the codewords, since

$$\mathbf{C}_m \;=\; \sum_{i=1}^{k} x_{mi}\, \mathbf{g}_i \quad . \tag{25}$$

Systematic Block Codes

As already noted, the generator matrix $\mathbf{G}$ characterizes the code. It also generates the codewords from the information vectors. To design a linear block code is to design its generator matrix. To implement one is to implement $\mathbf{C}_m \;=\; \mathbf{X}_m\, \mathbf{G}$, either directly or indirectly. Before we move on to discuss commonly used block codes, let's look at an important class of linear block codes, and take an initial look at decoding.

The generator matrix for a *systematic block code* has the following form:

$$\mathbf{G} \;=\; [\mathbf{I}_k\; \mathbf{P}] \;=\; \left[\begin{array}{ccccc|cccc} 1 & 0 & 0 & \cdots & 0 & p_{11} & p_{12} & \cdots & p_{1(n-k)} \\ 0 & 1 & 0 & \cdots & 0 & p_{21} & p_{22} & \cdots & p_{2(n-k)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \cdots & \\ 0 & 0 & 0 & \cdots & 1 & p_{k1} & p_{k2} & \cdots & p_{k(n-k)} \end{array}\right] \quad , \tag{26}$$

where $\mathbf{I}_k$ is the k-dimensional identity matrix and $\mathbf{P}$ is a $k \times (n-k)$ dimensional *parity bit generator matrix*. Note that with a systematic code, the first k elements of a codeword $\mathbf{C}_m$ are the elements of the corresponding information vector $\mathbf{X}_m$. This is what is meant be systematic. The remaining $(n-k)$ bits of a codeword $\mathbf{C}_m$ are generated as $\mathbf{X}_m\, \mathbf{P}$. These bits are used by the decoder, effectively, to check for and correct bit errors. That is, they are parity bits, and thus the term parity bit generator matrix.

If a code's generator matrix does not have the structure described above, the code is nonsystematic. Any nonsystematic generator matrix can be transformed into a systematic generator matrix via elementary row operations and column permutations. Column permutations correspond to swapping the same elements of all of the codeword (which does not change weight distribution of the code). Elementary row operations correspond to linear operations on the elements of $\mathbf{X}_m$, which generate other input vectors. This does not change the set of codewords, so it does not change the space spanned by the codewords, and thus again the weight distribution is unchanged. Therefore, any nonsystematic code is equivalent to a systematic code in the sense that the two have the same weight distribution. This is why a linear block code is often designated by $\mathcal{C}$, the subspace spanned by the codewords, rather then by a specific generator matrix.

The Parity Check Matrix for Systematic Codes

For a given $k \times n$-dimensional generator matrix $\mathbf{G}$ with rows $\mathbf{g}_i; i = 1, 2, \cdots, k$ that span the k-dimensional code subspace $\mathcal{C}$, consider a $(n-k) \times n$-dimensional matrix $\mathbf{H}$ with rows

$\mathbf{h}_l; l = 1, 2, \cdots, n-k$ that span the $(n-k)$-dimensional code subspace $\overline{\mathcal{C}}$. The $\mathbf{g}_i$ are orthogonal to the $\mathbf{h}_l$, so

$$\mathbf{G}\ \mathbf{H}^T\ =\ \mathbf{0}_{k\times(n-k)} \tag{27}$$

where $\mathbf{0}_{k\times(n-k)}$ is the $k \times (n-k)$-dimensional matrix of zeros. Thus

$$\mathbf{C}_m\ \mathbf{H}^T\ =\ \mathbf{0}_{1\times(n-k)}\ ;\qquad m = 1, 2, \cdots, M\ \ . \tag{28}$$

If we assume $\mathbf{G}$ is systematic, so that $\mathbf{G} = [\mathbf{I}_k\ \mathbf{P}]$, then Eq (27), i.e.
$[\mathbf{I}_k\ \mathbf{P}]\ \mathbf{H}^T = \mathbf{0}_{k\times(n-k)}$, is satisfied with

$$\mathbf{H}\ =\ [-\mathbf{P}^T\ \mathbf{I}_{n-k}]\ =\ [\mathbf{P}^T\ \mathbf{I}_{n-k}]\quad , \tag{29}$$

where $-\mathbf{P} = \mathbf{P}$ for a binary code (i.e. in the GF(2) field).

Since $\mathbf{H}$ can be thought of as a generator matrix, we can think of it as representing a *dual code*. More to the point, as we now show, $\mathbf{H}$ points to a decoding scheme that illustrates the error protection capabilities of the original code described by $\mathbf{G}$.

Let the $(l,j)^{th}$ element of $\mathbf{H}$ be denoted $\mathbf{H}_{l,j} = h_{l,j}$. Then for any codeword we have that

$$\mathbf{C}_m\ \mathbf{H}^T\ =\ \begin{bmatrix} \sum_{j=1}^{n} c_{m,j}h_{1,j} \\ \sum_{j=1}^{n} c_{m,j}h_{2,j} \\ \vdots \\ \sum_{j=1}^{n} c_{m,j}h_{(n-k),j} \end{bmatrix}^T\ =\ \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}^T\quad , \tag{30}$$

or

$$\sum_{j=1}^{n} c_{m,j}h_{l,j}\ =\ 0\ ;\qquad l = 1, 2, \cdots, n-k\ . \tag{31}$$

For any valid codeword, $\mathbf{C}_m\ \mathbf{H}^T = 0$. Also, for any n-dimensional vector $\mathbf{Y}$ which is not a valid codeword, $\mathbf{Y}\ \mathbf{H}^T \neq 0$. Thus, $\mathbf{Y}\ \mathbf{H}^T$ is a simple test to determine if $\mathbf{Y}$ is a valid codeword.

Considering further this idea that $\mathbf{Y}\ \mathbf{H}^T$ is a test for the validity of $\mathbf{Y}$ as a codeword, let $\mathbf{Y} = [y_1, y_2, \cdots, y_n]$ be a candidate codeword. Recalling that $\mathbf{H} = [\mathbf{P}^T\ \mathbf{I}_{n-k}]$, for $\mathbf{Y}$ to be valid, the following must hold:

$$\sum_{j=1}^{k} y_j p_{l,j}\ =\ y_{k+l}\ ;\qquad l = 1, 2, \cdots, n-k\ . \tag{32}$$

Thus the validity test is composed of $n-k$ parity tests. The l^{th} parity test consists of summing elements from $[y_1, y_2, \cdots, y_k]$ (i.e. information bits), selected by the parity matrix elements $p_{l,j}; j = 1, 2, \cdots, k$, and comparing this sum to the parity bit y_{l+k}. $\mathbf{Y}$ is a valid codeword if and only if it passes all $n-k$ parity tests. An equivalent form of the l^{th} parity test is to sum the selected information elements with y_{k+l}. If the sum is zero (i.e. even parity), the parity test is passed. If the sum is one (i.e. odd parity), the parity test is failed.

Because $\mathbf{H}$ defines a set of parity checks to determine if a vector $\mathbf{Y}$ is a codeword, we refer to $\mathbf{H}$ as the *parity check matrix* associated with the generator matrix $\mathbf{G}$. Note from Eq (29) that the parity check matrix is easily designed from the generator matrix.

Example 7.4: Consider a binary linear block $(6, 3)$ code with generator matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \quad . \tag{33}$$

1. Determine the codeword table.

	$\mathbf{C}_m = \mathbf{X}_m\mathbf{G}$		
$\mathbf{X}_m$	$\mathbf{X}_m$	$\mathbf{X}_m\mathbf{P}$	w_m
000	000	000	0
001	001	101	3
010	010	011	3
011	011	110	4
100	100	110	3
101	101	011	4
110	110	101	4
111	111	000	3

2. What is the weight distribution and the minimum distance?

weight # j	# codewords wn_j
0	1
3	4
4	3

$d_{min} = 3$

3. What is the parity bit generator matrix $\mathbf{P}$? Give parity bit equations.

$$\mathbf{P} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} , \qquad \begin{cases} c_{m4} = x_{m1} + x_{m3} \\ c_{m5} = x_{m1} + x_{m2} \\ c_{m6} = x_{m2} + x_{m3} \end{cases} \tag{34}$$

4. Determine the parity check matrix $\mathbf{H}$, and show that it is orthogonal to $\mathbf{G}$.

$$\mathbf{H} = [\mathbf{P}^T|\mathbf{I}_3] = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} ; \quad \mathbf{G}\,\mathbf{H}^T = [\mathbf{I}_3|\mathbf{P}]\,[\mathbf{P}|\mathbf{I}_3]^T = \mathbf{P}+\mathbf{P} = \mathbf{0}_3 \tag{35}$$

5. Is each of the following vectors a valid codeword? If not, which codeword(s) is it closest to?
   - $\mathbf{C}_a = [111000]$: Yes, corresponding to $\mathbf{X}_m = [1\ 1\ 1]$
   - $\mathbf{C}_b = [111001]$: No, it is 1 bit from $\mathbf{C}_8$, at least 2 bits from all others.
   - $\mathbf{C}_c = [011001]$: No, it is 2 bits from both $\mathbf{C}_3$ and $\mathbf{C}_8$

7.4 Initial Comments on Performance and Implementation

To be useful, a linear block code must have good performance and implementation characteristics. In this respect, identifying good codes is quite challenging. Fortunately, with over 55 years of intense investigation, numerous practical linear block codes have been identified. In the next two Subsections we identify the most important of these. Here, to assist in the description of these codes, i.e. to identify why they are good, we discuss some basic performance and implementation issues.

7.4.1 Performance Issues

Generally speaking, channel codes are used to manage errors caused by channel corruption of transmitted digital communications signals. To gain a sense of the capability of linear block codes in this regard, it is useful to consider a BSC, which implies: a two symbol modulation scheme; GF(2) block codes; and hard decision decoding[2]. Figure 54 illustrates this channel. For the transmission of a linear binary block codeword, the BSC is used n times, once for each codeword bit. Codeword bit errors are assumed statistically independent across the codeword. With each codeword bit, an error is made with probability ρ. With the linear binary block code, these codeword bit errors can be managed. Specifically, it is insightful to look at the ability to detect and correct errors made in detecting individual codeword bits.

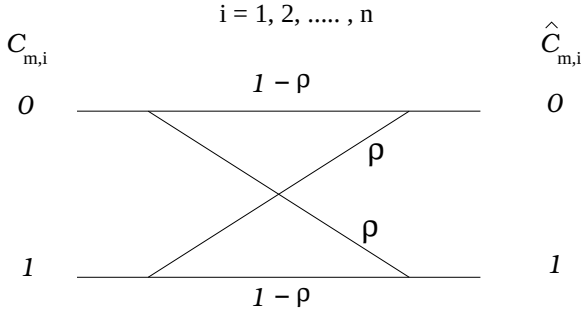


Figure 54: The Binary Symmetric Channel (BSC) used to transmit a block codeword.

Error Correction Capability

Consider a linear binary block (n, k) code. Let $\mathbf{Y}$ represent the received binary codeword, which potentially has codeword bit errors. Say that the error management strategy is simply to pick the codeword $\mathbf{C}_m$ which is closest to $\mathbf{Y}$. That is, say that $\mathbf{Y}$ is used *strictly to correct errors*. Clearly, whether or not the correct codeword is selected depends on how many codeword bit errors are made (i.e. probabilistically, it depends on ρ) and on how close, in Hamming distance, the actual codeword is to other codewords. So, the probability of codeword error, denoted here as $P(e)$, will depend on the minimum distance d_{min} of the code.

For example, consider the binary linear block code of Example 7.4 above. Since $d_{min} = 3$, if one codeword bit error is made, then the received vector $\mathbf{Y}$ will still be closest to the true codeword, and a correct codeword decision will be made. On the other hand, if two or more

[2]We will discuss other decoding approaches (i.e. hard decision, soft decision, and quantized) a little later.

codeword bit errors are made, then it is possible that the wrong codeword decision will be made. In general, the upper bound on the number of codeword bit errors that can be made without causing a codeword decision error is $t = \lfloor (d_{min} - 1)/2 \rfloor$, where the "floor" operator $\lfloor x \rfloor$ denotes the next integer less than x.

Figure 55 depicts the codewords $\mathbf{C}_m$ and possible received vectors $\mathbf{Y}$ in the codeword vector space. The spheres around the valid codewords represent the received vectors $\mathbf{Y}$ corresponding to t or fewer codeword bit errors. As illustrated, in general, for a given code, there may be some $\mathbf{Y}$ that are not within Hamming distance t to any valid codeword. A codeword error is made if the received vector $\mathbf{Y}$ is outside the true codeword sphere *and* closer in Hamming distance to the sphere of another codeword. Thus, the probability of a codeword error is upper bounded as:

$$P(e) \;\leq\; \sum_{j=t+1}^{n} \left(\begin{array}{c} n \\ j \end{array} \right) \rho^j (1-\rho)^{n-j} \tag{36}$$

where $\rho^j(1-\rho)^{n-j}$ is the probability of making exactly j codeword bit errors, and the number of ways that exactly j codeword bit errors can be made is $\left(\begin{array}{c} n \\ j \end{array} \right)$ (i.e. "n take j").

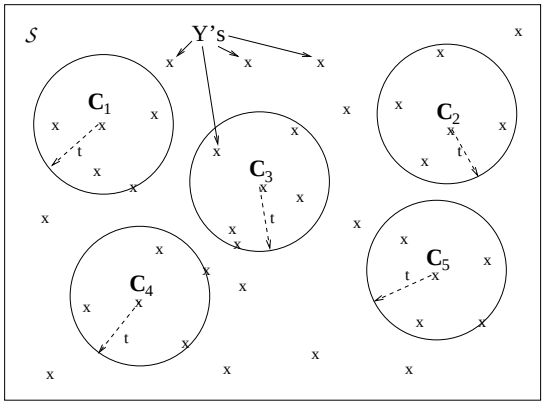


Figure 55: Codewords and received vectors in the code vector space.

Error Detection Capability

Since d_{min} or more errors are necessary to mistake one codeword for another, any $d_{min} - 1$ errors can be detected. The *error detection probability* of a binary linear block code is governed by its codeword weight distribution wn_j; $j = 0, 1, 2, \cdots, n$. If used *strictly to detect errors*, i.e. if a received binary n-dimensional vector Y is tagged as in error if it is not a valid codeword X_m, the probability of *undetected* codeword error, denoted as $P_u(e)$, is given by

$$P_u(e) \;=\; \sum_{j=1}^{n} \; wn_j \; \rho^j (1-\rho)^{n-j}$$

That is, since $\mathbf{Y}$ is an incorrect codeword if and only if the error pattern across the codeword looks like a codeword, this is the sum of the probabilities of all the possible error combinations that result in $\mathbf{Y}$ being exactly an incorrect codeword.

Combined Error Correction & Detection

When correcting $t = \lfloor (d_{min} - 1)/2 \rfloor$ errors, t errors are effectively detected, but no more. On the other hand, when detecting $d_{min} - 1$ errors, no errors will be corrected. Let e_c and e_d denote, respectively, the number of errors we can correct and detect. If we reduce the number of errors we intend correct to $e_c < t$, it is possible to then also detect $e_d > e_c$ errors. Referring to Figure 55, we do this by reducing the sphere radii to $e_c < t$. Then,

$$e_c \ + \ e_d \ \leq \ d_{min} - 1 \quad . \tag{37}$$

Perfect Codes

A linear binary block code is a *perfect code* if every binary n-dimensional vector is within Hamming distance $t = \lfloor (d_{min} - 1)/2 \rfloor$ of a codeword. This is depicted below in Figure 56. Given a perfect code, the probability of codeword error for a strictly error correction mode of reception, is exactly

$$P(e) \ = \ \sum_{m=t+1}^{n} \left(\begin{array}{c} n \\ m \end{array} \right) \rho^m (1-\rho)^{n-m} \quad . \tag{38}$$

There are only a few known perfect codes – Hamming codes, the Golay (23,12) code, and a few trivial (2 codeword, odd length) codes. Hamming and Golay codes are described below.

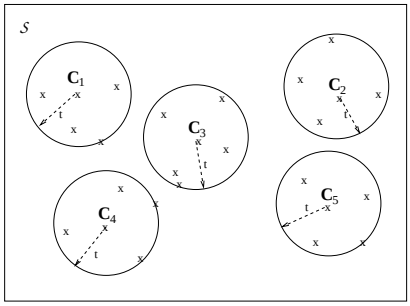


Figure 56: For a perfect code, codewords and received vectors in the code vector space.

7.4.2 From Performance to Implementation Considerations

Figure 57 is a block diagram representing possible decoding schemes for a linear block code. It depicts the transmission/reception of one information bit vector $\mathbf{X}_m$. The received, noisy codeword is $\underline{\mathbf{r}}$, which is the sampled output of the receiver filter which is matched to the codeword bit waveform. $\underline{\mathbf{r}}$ is n-dimensional and continuous in amplitude. The figure shows three basic options for codeword detection:

- Soft decision decoding is shown on top, in which $\underline{\mathbf{r}}$ is compared directly to the M possible codewords. The comparison is typically a ML detector (for $\mathbf{C}_m$ given $\underline{\mathbf{r}}$).
- Hard decision decoding is shown on the bottom, in which each codeword bit is first detected, typically using a ML bit detector, to form a received binary vector $\mathbf{Y}$. Then $\mathbf{Y}$ is effectively compared to the M possible codewords. Again, the comparison is typically a ML detector (this time for $\mathbf{C}_m$ given $\mathbf{Y}$). $\mathbf{Y}$ can be considered a severely quantized version of $\underline{\mathbf{r}}$.

- Something in between soft and hard decision decoding is shown in between, in which $\underline{\mathbf{r}}$ is quantized, not as severely as with hard decision decoding, to form $\mathbf{R}$. Then $\mathbf{R}$ is compared to the M possible codewords. Again, the comparison is typically a ML detector (this time for $\mathbf{C}_m$ given $\mathbf{R}$).

Concerning performance, soft decision decoding will be best, followed by quantized decoding and then hard decision decoding. This is because information is lost through the quantization process. On the other hand, hard decision decoders can be implemented most efficiently. As we will see, linear block codes are often designed for efficient hard decision decoding.

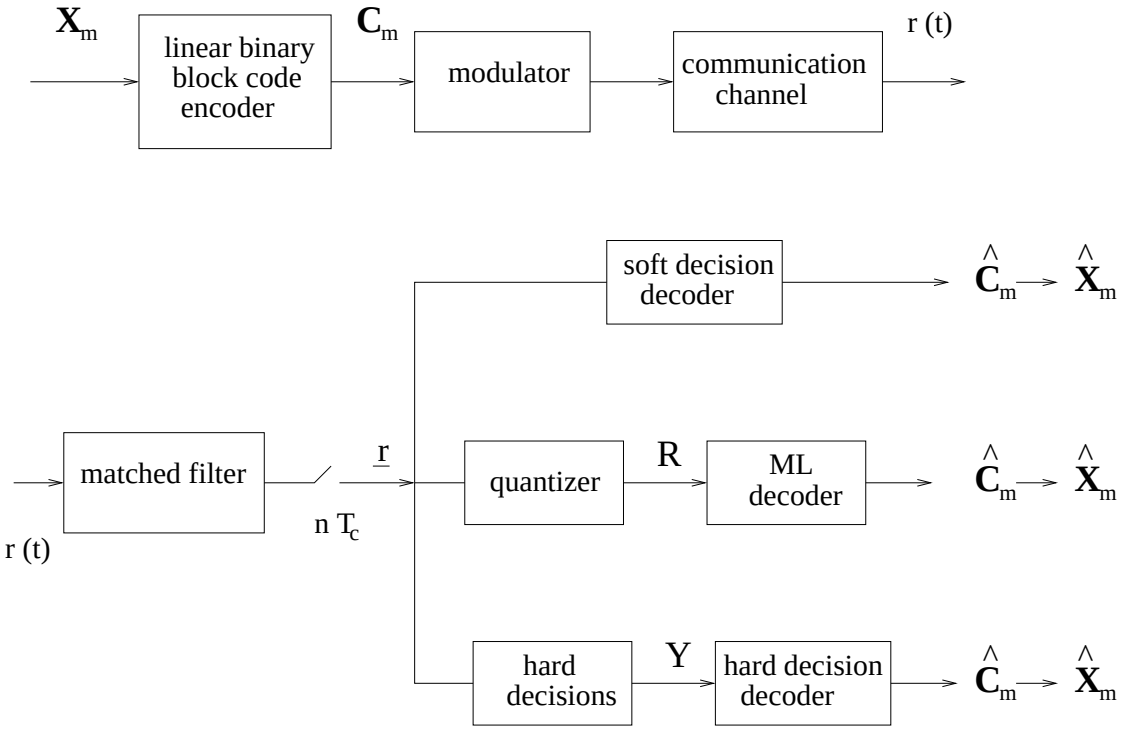


Figure 57: Decoding schemes for block codes.

These decoding schemes can all be used for strictly error correction, i.e. for FEC (forward error correction) coding. Alternatively, for ARQ (automatic repeat request) channel coding, the hard decision vector $\mathbf{Y}$ can be used to detect errors and initialize an ARQ. As noted earlier, in this Course we will focus on FEC coding.

Earlier in this Section we established that d_{min} and the code weight function wn_j dictate performance. Also, in Section 6 of this Course we established that large codes (e.g., long block length n and number of codewords M) are important for realizing reliable communications at information rates approaching capacity. However, large block codes, if implemented directly by comparison of the received data vector ($\underline{\mathbf{r}}$, $\mathbf{R}$ or $\mathbf{Y}$) will be computationally expensive both because comparison to each $\mathbf{C}_m$ will be costly and because there will be a lot of $\mathbf{C}_m$'s.

So, it looks like we have a classic performance vs. cost trade off problem. We push the performance, towards the channel capacity bound, by using as large as possible block codes, with good d_{min} and weight distribution wn_j characteristics, which have structures that allow for efficient optimum (e.g. ML) or suboptimum decoding.

7.4.3 Implementation Issues

With the large block code objective in mind, and d_{min} and weight distribution wn_j characteristics, effective linear block codes have been developed. In describing the major linear block codes below, we will consider implementation issues and techniques. We will consider, for example

- systematic codes
- syndrome decoding (with associated concepts like standard arrays, coset headers and error patterns)
- shift register based encoding
- check sums & majority decision rule decoding
- puncturing
- concatenated codes, and
- interleaving.

7.4.4 Code Rate

The code rate for a binary block code is defined as $R_c = \frac{k}{n}$ (or, for a nonbinary block code, $R_c = \frac{k}{N}$). That is, it is the ratio of the number of bits represented to the number of symbols in a codeword. For any block code, $R_c > 0$. For binary block codes, $R_c < 1$. The higher the rate, the more efficiently the channel is used. Ideally, a high rate, large d_{min} code is desired, although these tend to be conflicting goals, since $k \approx n$ means the codewords fill up most of the vector space. However, decreasing n can improve R_c for a fixed d_{min}.

7.5 Important Binary Linear Block Codes

The Lin & Costello book [1], Chapters 3 through 10, is excellent reference on block codes. The authors suggest using these Chapters as the basis for a one semester graduate level course. In this Subsection we provide an overview of binary linear block codes. We briefly describe the most important ones, and discuss their performance and implementation characteristic. To illustrate a more involved treatment of linear block codes, will consider Reed-Solomon codes, the most popular non-binary block code, in Subsection 6.7.

7.5.1 Single Parity Check Codes

Most communication system engineers have heard of single parity check block coding. This is often used, for example, with 7 bit ASCII characters to provide some error detection capability. In the even parity bit version, a bit is added to each 7 bit ASCII character so that the total number of "1" bits is even. Thus the codewords are 8 bits long, and the code is $(n, k) = (8, 7)$.

Consider a $k = 7$ bit information vector $\mathbf{X}_m$. The 7×8 dimensional generator matrix $\mathbf{G}$ for the even parity check block code is

$$\mathbf{G} \;=\; [\; \mathbf{I}_7 \mid \mathbf{1}_7^T \;] \tag{39}$$

where $\mathbf{I}_N$ is the $N \times N$ identity matrix and $\mathbf{1}_N$ is the row vector of N 1's. The parity bit generator matrix and parity check matrices are

$$\mathbf{P} \;=\; \mathbf{1}_7^T \;; \qquad \mathbf{H} \;=\; \mathbf{1}_8^T \quad . \tag{40}$$

This code is systematic. Its minimum distance is $d_{min} = 1$, so $t = 0$ codeword bit errors can be corrected. A codeword has the form

$$\mathbf{C}_m \;=\; [\; \mathbf{X}_m \mid p_{m1}] \;; \qquad p_{m1} \;=\; c_{m8} \;=\; \mathbf{X}_m \; \mathbf{1}_7^T \quad . \tag{41}$$

The parity check bit p_{m1} detects any odd number of codeword bit errors.

7.5.2 Repetition Codes

A repetition code is a $(n, 1)$ linear block code with generator matrix $\mathbf{G} = \mathbf{1}_n$. It generates a codeword by repeating each information bit n times. Strictly speaking, it is a systematic code with a parity bit generator matrix $\mathbf{P} = \mathbf{1}_{n-1}$. $d_{min} = n$, so it can correct any $t = \lfloor (n-1)/2 \rfloor$ codeword bit errors.

7.5.3 Hamming Codes

For any positive integer m, a Hamming code can be easily described in terms of a basic characteristic of its $m \times n$-dimensional parity check matrix $\mathbf{H}$, where $m = n - k$. That is, the columns of $\mathbf{H}$ are the $2^m - 1$ non-zero binary vectors of length m. Then, in terms of m, we have $n = 2^m - 1$ and $k = n - m = 2^m - 1 - m$, or

$$(n, k) \;=\; (2^m - 1, 2^m - 1 - m) \quad . \tag{42}$$

Table 7.1 shows (n, k) for several Hamming codes. Only $m \geq 1$ are of interest. All Hamming codes have $d_{min} = 3$, so that $t = \lfloor (d_{min} - 1)/2 \rfloor = 1$ error can be corrected. They are also perfect code, so that every binary n vector is within Hamming distance $t = 1$ of a codeword.

m	$(n, k) = (2^m - 1, 2^m - 1 - m)$	R_c
1	$(1, 0)$	xx
2	$(3, 1)$	$1/3$
3	$(7, 4)$	$4/7$
4	$(15, 11)$	$11/15$
5	$(31, 26)$	$26/31$
$\downarrow$	$\downarrow$	$\downarrow$
∞	(∞, ∞)	1

Table 7.1: Hamming code lengths.

Example 7.5 - The (3,1) Hamming code: Its parity check matrix is

$$\mathbf{H} = \left[\begin{array}{c|cc} 1 & 1 & 0 \\ 1 & 0 & 1 \end{array}\right] \tag{43}$$

so that

$$\mathbf{G} = [1\ 1\ 1]\ ; \qquad \mathbf{P} = [1\ 1] \tag{44}$$

The code generation table is

$\mathbf{X}_m$	$\mathbf{C}_m$	w_m
0	000	0
1	111	3

from which we can see that $d_{min} = 3$. This is a systematic code. In fact, it is the (3,1) repetition code.

Example 7.6 - The (7,4) Hamming code: Its parity check matrix is

$$\mathbf{H} = \left[\begin{array}{cccc|ccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{45}$$

so that

$$\mathbf{G} = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right] \ ; \quad \mathbf{P} = \left[\begin{array}{ccc} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{array}\right] \tag{46}$$

The code generation table, shown below, indicates that $d_{min} = 3$.

	$\mathbf{C}_m = \mathbf{X}_m\mathbf{G}$		
$\mathbf{X}_m$	$\mathbf{X}_m$	$\mathbf{X}_m\mathbf{P}$	w_m
0000	0000	000	0
0001	0001	011	3
0010	0010	110	3
0011	0011	101	4
0100	0100	111	4
0101	0101	100	3
0110	0110	001	3
0111	0111	010	4
1000	1000	101	3
1001	1001	110	4
1010	1010	011	4
1011	1011	000	3
1100	1100	010	3
1101	1101	001	4
1110	1110	100	4
1111	1111	111	7

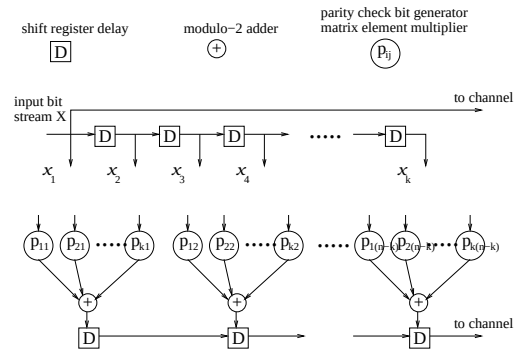
The weight distribution for this example is given in the following table.

weight # j	# codewords wn_j
0	1
3	7
4	7
7	1

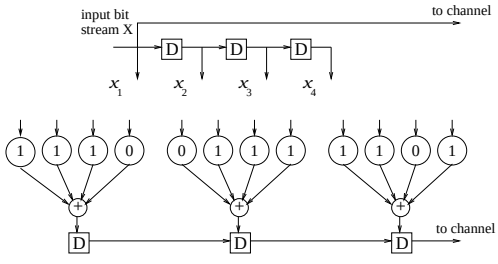
Note that any circular shift of a codeword is also a codeword. That is

- The set of weight 3 code words contains all circular shifts of $\mathbf{C}_2$: i.e. $\{bfC_3$ is $\mathbf{C}_2$ circularly shifted 1 (to the left), $\mathbf{C}_6$ is shifted 2, $\mathbf{C}_{12}$ is shifted 3, $\mathbf{C}_7$ is shifted 4, $\mathbf{C}_{13}$ is shifted 5, and $\mathbf{C}_9$ is shifted 6.
- The set of weight 4 code words contains all circular shifts of $\mathbf{C}_4$: i.e. $\mathbf{C}_8$ is shifted 1, $\mathbf{C}_{15}$ is shifted 2, $\mathbf{C}_{14}$ is shifted 3, $\mathbf{C}_{11}$ is shifted 4, $\mathbf{C}_5$ is shifted 5, and $\mathbf{C}_{10}$ is shifted 6.
- $\mathbf{C}_1$ is all circular shifted versions of itself.
- $\mathbf{C}_{16}$ is all circular shifted versions of itself.

Binary linear block code encoders (i.e. the $\mathbf{C}_m = \mathbf{X}_m\mathbf{G}$ operation) are often implemented with shift register circuit. Figure 58(a) illustrates the general implementation, while Figure 58(b) is a Hamming (7,4) encoder.



(a) general shift register encoder



(b) Hamming (7,4) code encoder

Figure 58: Shift register encoders.

7.5.4 Shortened Hamming and SEC-DED Codes

Shortened Hamming codes are designed by eliminating columns of a Hamming code generator matrix $\mathbf{G}$. In this way, codes have been identified with $d_{min} = 4$, that can be used to correct any 1 error (i.e. $t = 1$) while detecting any 2 errors.

SEC-DED (single error correcting, double error detecting) codes were first described by Hsiao. As noted directly above, some SEC-DED codes have been designed as shortened Hamming codes.

7.5.5 Reed-Muller codes

Reed-Muller codes are particular binary linear block (n, k) codes for which, for integer r and m such that $0 \leq r \leq m$, $n = 2^m$ and

$$k \;=\; 1 \;+\; \left(\begin{array}{c} m \\ 1 \end{array}\right) \;+\; \left(\begin{array}{c} m \\ 2 \end{array}\right) \;+\; \cdots \;+\; \left(\begin{array}{c} m \\ r \end{array}\right) \quad . \tag{47}$$

These codes have several attractive attributes. First, $d_{min} = 2^{m-r}$, so they facilitate multiple error correction. Also, the codewords for these codes have a particular structure which can be efficiently decoded. There are not systematic codes. Instead, codewords have an alternating 0's/1's, self-similar structure (somewhat reminiscent of wavelets).

> *Example 7.7:* For $m = 4$ (i.e. $n = 16$), and $r = 2$ so that $k = 11$, codewords are generated using a generator matrix whose rows are the vectors
>
> $$\begin{aligned}
> \mathbf{g}_0 &= \mathbf{1}_{16} \\
> \mathbf{g}_1 &= [\mathbf{0}_8 \; \mathbf{1}_8] \\
> \mathbf{g}_2 &= [\mathbf{0}_4 \; \mathbf{1}_4 \; \mathbf{0}_4 \; \mathbf{1}_4] \\
> \mathbf{g}_3 &= [0\;0\;1\;1\;0\;0\;1\;1\;0\;0\;1\;1\;0\;0\;1\;1] \\
> \mathbf{g}_4 &= [0\;1\;0\;1\;0\;1\;0\;1\;0\;1\;0\;1\;0\;1\;0\;1] \\
> \mathbf{g}_5 &= \mathbf{g}_1 \cdot \mathbf{g}_2 \\
> \mathbf{g}_6 &= \mathbf{g}_1 \cdot \mathbf{g}_3 \\
> \mathbf{g}_7 &= \mathbf{g}_1 \cdot \mathbf{g}_4 \\
> \mathbf{g}_8 &= \mathbf{g}_2 \cdot \mathbf{g}_3 \\
> \mathbf{g}_9 &= \mathbf{g}_2 \cdot \mathbf{g}_4 \\
> \mathbf{g}_{10} &= \mathbf{g}_3 \cdot \mathbf{g}_4
> \end{aligned} \tag{48}$$
>
> where here $\mathbf{g}_i \cdot \mathbf{g}_j$ is the element-by-element modulo-2 multiplication.

With this codeword structure, an efficient multistage decoder has been developed, with each stage consisting of majority-logic decisions[3] using check sums of blocks of received bits.

[3]Majority-logic decision is described in Subsection 9.5 of this Course on LDPC codes.

7.5.6 The Two Golay Codes

The original Golay code is a $(23, 12)$ binary linear block code. It is a perfect code with $d_{min} = 7$, so that $t = 3$ codeword bit errors can be corrected. This code can be described in terms of a *generator polynomial* $g(p)$ from which a generator matrix $\mathbf{G}$ can be constructed. The generator polynomial for this Golay code is

$$g(p) \;=\; p^{11} \;+\; p^{9} \;+\; p^{7} \;+\; p^{6} \;+\; p^{5} \;+\; p \;+\; 1 \quad . \tag{49}$$

Let

$$\mathbf{g} \;=\; [1\;0\;1\;0\;1\;1\;1\;0\;0\;0\;1\;1] \tag{50}$$

be the 12-dimensional vector of generator polynomial coefficients. The 12×23 dimensional Golay code generator matrix is

$$\mathbf{G} \;=\; \begin{bmatrix} \mathbf{g} & \mathbf{0}_{11} & \\ \mathbf{0}_1 & \mathbf{g} & \mathbf{0}_{10} \\ \mathbf{0}_2 & \mathbf{g} & \mathbf{0}_9 \\ \mathbf{0}_3 & \mathbf{g} & \mathbf{0}_8 \\ \vdots & \vdots & \vdots \\ \mathbf{0}_{10} & \mathbf{g} & \mathbf{0}_1 \\ & \mathbf{0}_{11} & \mathbf{g} \end{bmatrix} \quad . \tag{51}$$

As with other linear block codes, codewords can be generated by premultiplying the generator matrix $\mathbf{G}$ with the information bit vector $\mathbf{X}_m$. Equivalently, the generator polynomial vector $\mathbf{g}$ can be convolved with $\mathbf{X}_m$.

Example 7.8: Determine the Golay codeword for the information vector $\mathbf{X}_m = [111111000000]$.

$$[1\;0\;1\;0\;1\;1,\;\;1\;0\;0\;0\;1\;1]$$

$$[0\;0\;0\;0\;0\;0\;1\;1\;1\;1\;1\;1]$$

$$\mathbf{C}_m \;=\; [1\;1\;0\;0\;1\;0,\;\;0\;0\;1\;1\;1\;1,\;\;0\;0\;0\;0\;0\;1\;0,\;\;0\;0\;0\;0\;0\;0]$$

Efficient hard decision decoders exist for the (23,12) Golay code.

The Golay (24,12) code is generated from the (23,12) Golay code by appending an even parity bit to each codeword. $d_{min} = 8$, so any $t = 3$ errors can be corrected.

7.5.7 Cyclic Codes

Cyclic codes constitute a very large class of binary and nonbinary linear block codes. Their popularity stems from efficient encoding and decoding algorithms, which allow for large codes. They also have good performance characteristics. Here we describe the binary cyclic code class, which include Hamming and the Golay (23,12) codes already discussed, as well as binary BCH codes introduced below. The nonbinary cyclic code class includes nonbinary BCH code class, which in turn includes Reed-Solomon codes.

In this Subsection we represent a message vector as $\mathbf{X}_m = [x_{m(k-1)},\ x_{m(k-2)},\ \cdots,\ x_{m1},\ x_{m0}]$ or equivalently as a message polynomial $x_m(p) = x_{m(k-1)}p^{k-1} + x_{m(k-2)}p^{k-2} + \cdots + x_{m1}p + x_{m0}$. That is, in the vector, the highest powers are listed first. Similarly, a codeword vector is denoted $\mathbf{C}_m = [c_{m(n-1)},\ c_{m(n-2)},\ \cdots,\ c_{m1},\ c_{m0}]$ and the corresponding codeword polynomial is $c_m(p) = c_{m(n-1)}p^{n-1} + c_{m(n-2)}p^{n-2} + \cdots + c_{m1}p + c_{m0}$.

Cyclic Code Structure

Consider the set of codewords $\{\mathbf{C}_m; m = 1, 2, \cdots, M\}$ of a binary linear block code. The code is cyclic if, for each codeword $\mathbf{C}_m$, all cyclic shifts of $\mathbf{C}_m$ are codewords. That is, if $\mathbf{C}_m = [c_{m,(n-1)}, c_{m,(n-2)}, \cdots, c_{m,0}]$ is a codeword, then so are

$$[c_{m,mod2(n-1-i)}, c_{m,mod2(n-2-i)}, \cdots, c_{m,mod2(-i)}] \qquad i = 1, 2, \cdots, n-1 \quad . \tag{52}$$

Note that the number or unique vectors in the set listed in Eq (52) may be less than n. In general for a cyclic code there will be more than one set of cyclic shift related codewords.

> *Example 7.9:* List all of the unique cyclic shifts of the vectors [0000000], [1111111], [0001011] and [0011101]. (See the Hamming (7,4) code, Example 7.5.)

In terms of a codeword polynomial,

$$C(p) \ = \ c_{n-1}p^{n-1} \ + \ c_{n-2}p^{n-2} \ + \ \cdots \ + c_0 \quad , \tag{53}$$

the cyclic shift property of cyclic codes means that the

$$C_i(p) = c_{n-1-i}p^{n-1} \ + \ c_{mod2(n-2-i)}p^{n-2} \ + \ \cdots \ + c_{mod2(-i)} \ ; \quad i = 1, 2, \cdots, n-1 \tag{54}$$

correspond to codewords. Consider

$$\frac{pC(p)}{p^n+1} \ = \ \frac{c_{n-1}p^n \ + \ c_{n-2}p^{n-1} \ + \ \cdots \ + c_0 p}{p^n+1} \ = \ c_{n-1} \ + \ \frac{C_1(p)}{p^n+1} \ . \tag{55}$$

We see that $C_1(p)$ is the remainder of $pC(p)$ divided by p^n+1. Applying this next to $p^2C(p)$, and so on, we see that

$$C_i(p) \ = \ p^i \ C(p) \ \ mod(p^n+1) \quad . \tag{56}$$

This result will be used directly below.

Cyclic Codeword Generation

As with other codes, cyclic codes can be generated from a generator matrix. However, it is easier to describe codeword generation (as well as generator matrix derivation) in terms of a *generator polynomial*. For a (n, k) code, the generator polynomial $g(p)$ will be degree $n - k$, i.e.

$$g(p) \;=\; p^{n-k} \;+\; g_{n-k-1}p^{n-k-1} \;+\; \cdots \;+\; g_0 \quad . \tag{57}$$

Consider a $k-1$ degree *message polynomial*

$$X_m(p) \;=\; x_{k-1}p^{k-1} \;+\; x_{k-2}p^{k-2} \;+\; \cdots + x_0 \quad , \tag{58}$$

for some information bit vector $\mathbf{X}_m$. Then, a codeword polynomial is computed as

$$C_m(p) \;=\; X_m(p)\; g(p) \quad . \tag{59}$$

The coefficient vector of $C_m(p)$ is the codeword $\mathbf{C}_m$. We next show that if $g(p)$ is a factor of the polynomial $p^n + 1$, it is the generator polynomial of a cyclic code.

Consider a $n - k$ degree polynomial $g(p)$ which is a factor of $p^n + 1$, and a $k - 1$ degree message polynomial of the Eq (58) form. Note that $g(p)$ is a factor of $C_m(p)$ since $C_m(p) = X_m(p)g(p)$. Now, consider a codeword polynomial $C(p)$. Eq (55) indicates that

$$C_1(p) \;=\; pC(p) + c_{n-1}\;(p^n + 1) \quad . \tag{60}$$

Now, since $g(p)$ is a factor of both $C(p)$ and $p^n + 1$, it is also a factor of $C_1(p)$. That is, $C_1(p) = X_1(p)\; g(p)$ for some degree $(k - 1)$ $X_1(p)$. So $C_1(p)$, the cyclic shift of codeword polynomial $C(p)$, is also a codeword polynomial. Any cyclic shifted codeword is a codeword.

In summary, with a degree $n - k$ generator polynomial $g(p)$ which is a factor of $p^n + 1$, the corresponding code is cyclic.

Example 7.10: $p^7 + 1$ factors as

$$\begin{aligned} p^7 + 1 \;&=\; (p^3 + p^2 + 1)(p^4 + p^3 + p^2 + 1) \\ &=\; g(p)\; h(p) \quad . \end{aligned} \tag{61}$$

The polynomial $g(p) = (p^3 + p^2 + 1)$ is a generator polynomial for a $(7, 4)$ binary cyclic code, while $h(p) = (p^4 + p^3 + p^2 + 1)$ is a generator polynomial for a $(7, 3)$ code.

Since p^n+1 has a number of factors, there are a number of binary cyclic codes with n-dimensional codewords. The polynomial $h(p)$ is termed the *parity polynomial* of the generator polynomial $g(p)$. Conversely, $g(p)$ is the parity polynomial of the generator polynomial $h(p)$.

Example 7.11: Determine the codeword table for the $(7,4)$ binary cyclic code identified by generator polynomial $g(p) = (p^3 + p^2 + 1)$.

$$g(p) \;=\; p^3 \;+\; p^2 \;+\; 0p \;+\; 1 \quad\longrightarrow\quad \mathbf{g} \;=\; [1\ 1\ 0\ 1]$$

Convolve $\mathbf{g}$ with the $\mathbf{X}_m$. Note that this code is not systematic.

$\mathbf{X}_m$	$\mathbf{C}_m$	w_m
0000	0000000	0
0001	0001101	3
0010	0011010	3
0011	0010111	4
0100	0110100	3
0101	0111001	4
0110	0101110	4
0111	0100011	3
1000	1101000	3
1001	1100101	4
1010	1110010	4
1011	1111111	7
1100	1011100	4
1101	1010001	3
1110	1000110	4
1111	1001011	4

The Generator Matrix of a Cyclic Code

Recall that we first described the Golay (23,12) code in terms of its generator polynomial

$$g(p) \;=\; p^{11} \;+\; p^9 \;+\; p^7 \;+\; p^6 \;+\; p^5 \;+\; p \;+\; 1 \quad . \tag{62}$$

$g(p)$ is a factor of $p^{23} + 1$, so the code is cyclic. We saw that its generator matrix is

$$\mathbf{G} \;=\; \begin{bmatrix} \mathbf{g} & \mathbf{0}_{11} & \\ \mathbf{0}_1 & \mathbf{g} & \mathbf{0}_{10} \\ \mathbf{0}_2 & \mathbf{g} & \mathbf{0}_9 \\ \mathbf{0}_3 & \mathbf{g} & \mathbf{0}_8 \\ \vdots & \vdots & \vdots \\ \mathbf{0}_{10} & \mathbf{g} & \mathbf{0}_1 \\ & \mathbf{0}_{11} & \mathbf{g} \end{bmatrix} \quad , \tag{63}$$

where $\mathbf{g} \;=\; [1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1]$ is the generator polynomial coefficient vector.

Generally, the generator matrix $\mathbf{G}$ for a cyclic code can be described in terms of the generator polynomial $g(p)$. Since the codewords can be generated by convolving each information bit vector with a generator polynomial coefficient vector, the generator matrix will effectively perform this convolution. Since $\mathbf{C}_m = \mathbf{X}_m\mathbf{G}$, each element of $\mathbf{C}_m$ is generated as an inner

product of $\mathbf{X}_m$ with a column of $\mathbf{G}$. From Example 7.11, we see that each element of $\mathbf{C}_m$ is also an inner product with a portion of the coefficient vector of the generator matrix. Thus, the columns of the generator matrix are constructed from the generator polynomial coefficient vector, with each column being a next shifted k-dimensional perhaps zero-padded window of the generator polynomial coefficient vector. Thus, in general the generator matrix for a cyclic code can be formed as it was for the Golay code (i.e. as in Eq. (63)).

Example 7.12: For the cyclic code in Example 7.11, describe the generator matrix.

$$g(p) \;=\; p^3 \;+\; p^2 \;+\; 0p \;+\; 1 \quad\longrightarrow\quad \mathbf{g} \;=\; [1\;1\;0\;1] \tag{64}$$

$$\mathbf{G} \;=\; \left[\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right] \;=\; \left[\begin{array}{ccc} \mathbf{g} & \mathbf{0}_3 & \\ 0 & \mathbf{g} & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{g} & 0 \\ & \mathbf{0}_3 & \mathbf{g} \end{array}\right] \;. \tag{65}$$

The codewords in Example 7.11 can be generated as $\mathbf{C}_m = \mathbf{X}_m\mathbf{G}$. Again note, this time by observation of the generator matrix, that this code is not systematic.

Systematic Cyclic Codes

Recall that for linear block codes, the codewords are all of the linear combinations of the rows of the generator matrix. That is, the codewords fill a k dimensional subspace we denote as $\mathcal{C}$. We noted earlier that we can consider $\mathcal{C}$ to be the code. Any $k \times n$ dimensional matrix whose rows span $\mathcal{C}$ is a generator matrix for code $\mathcal{C}$. Since elementary row operations on a matrix do not change the span of the rows, a systematic cyclic generator matrix can be derived from a non-systematic one through elementary row operations.

Example 7.13: Derive a systematic generator equivalent to the one in Example 7.12. First, adding row 4 to rows 1 and 3, we get

$$\left[\begin{array}{cccc|ccc} 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right] \;. \tag{66}$$

Next, adding row 3 to row 2, we get

$$\left[\begin{array}{cccc|ccc} 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right] \;. \tag{67}$$

Finally, adding row 2 to row 1, we get the desired systematic generator matrix

$$\mathbf{G} \;=\; \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right] \;. \tag{68}$$

This generator matrix will generate the all the codewords listed in Example 7.11. So, as expected, the code is still cyclic. However, the codewords will be associated with different information bit vectors.

An Efficient Systematic Cyclic Code Encoder

We have just seen that any cyclic code can be put in systematic form. In terms of polynomial coefficient vectors, this means that codewords can be constructed as follows:

$$\mathbf{C}_m \;=\; [x_{m(k-1)},\; x_{m(k-2)},\; \cdots,\; x_{m1},\; x_{m0},\; p_{m(n-k-1)},\; p_{m(n-k-2)},\; \cdots,\; p_{m0}] \tag{69}$$

where the $p_{mi}; i = 1, 2, \cdots, n-k-1$ are parity check bits. The codeword polynomial can therefore be expressed as

$$c_m(p) \;=\; p^{n-k}\; x_m(p) \;+\; p_m(p) \tag{70}$$

where

$$p_m(p) = p_{m(n-k-1)}p^{n-k-1} \;+\; p_{m(n-k-2)}p^{n-k-2} \;+\; \cdots \;+\; p_{m1} \;+\; p_{m0} \tag{71}$$

is the parity check polynomial. Consider dividing the degree $n-1$ polynomial $p^{n-k}\; x_m(p)$ by the generator polynomial $g(p)$:

$$p^{n-k}\; x_m(p) \;=\; q(p)\; g(p) \;+\; b(p) \tag{72}$$

where $q(p)$ and $b(p)$ are the quotient and remainder polynomials respectively. Rearranging Eq (72), we have

$$q(p)\; g(p) \;=\; p^{n-k}\; x_m(p) \;+\; b(p) \quad . \tag{73}$$

The polynomial $q(p)\; g(p)$ represents a valid codeword since it is the product of the generator polynomial $g(p)$ and another polynomial, $q(p)$ of degree $\leq k$. Thus, $p^{n-k}\; x_m(p) \;+\; b(p)$ represents a valid codeword, and it is in systematic form. Thus, the remainder of $p^{n-k}\; x_m(p)$ divided by generator polynomial $g(p)$ is the parity check polynomial, i.e. $p_m(p) \;=\; b(p)$.

Recall that in Subsection 7.2 we presented an efficient feedback shift register implementation of binary polynomial division, i.e. see Figure 53. This suggests an efficient encoder structure for a systematic binary cyclic code based on dividing the message polynomial $p^{n-k}\; x_m(p)$ by the generator polynomial $g(p)$ to compute the parity bits $p_{mi}; i = 1, 2, \cdots, n-k-1$. Figure 59 depicts feedback shift register modified to generate systematic cyclic codewords. Note that for a cyclic code generator polynomial, $g_0 = g_{n-k} = 1$. Next note that, to implement the division of $p^{n-k}\; x_m(p)$, as opposed to $x(p)$, the information bits are loaded into the end of the shift register instead of the start. To generate the systematic codeword described by Eq (69), the information bits are simultaneously loaded into the codeword register. When the information bits are finished loading, the parity bit computations are complete and are stored in the shift register. The two switches shown in the figure are moved from the 1 to the 2 positions, and the parity bits are loaded into the codeword register.

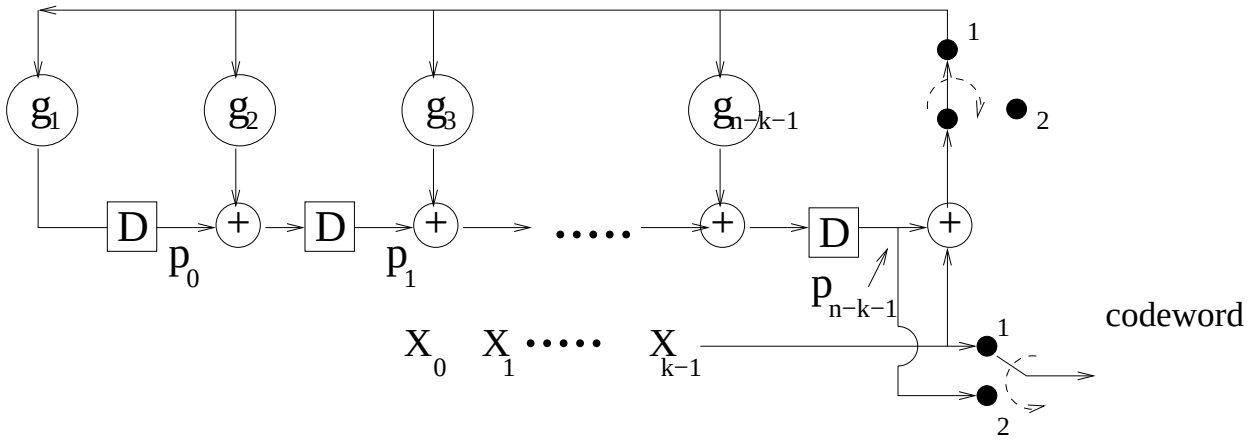


Figure 59: Linear feedback shift register implementation of a systematic binary cyclic encoder.

7.5.8 BCH Codes

Bose-Chaudhuri-Hocquenghem (BCH) codes are a general class of both binary and non-binary codes with attractive implementation and performance characteristics. They are a subclass for cyclic codes, and include Reed-Solomon codes as a subclass. Binary BCH codes are overviewed here.

Let $m \geq 3$ be an integer and let t be a positive integer (i.e. the error correction capability of the code). Then, binary BCH codes exist for (n, k) where

$$n \;=\; 2^m \;-1\;; \qquad n - mt \;\leq k \;< n \quad . \tag{74}$$

Generator polynomials for a large set of binary BCH codes can be found in a number of digital communication texts (e.g. Proakis [2]).

Example 7.14: In Table 7.10-1 of Proakis [2], find the generator polynomial for the $(15, 7)$ binary BCH code. What is the codeword for information vector $\mathbf{X}_m = [0001111]$.

From Proakis [2], Table 7.10-1, the generator polynomial coefficients are given by the octal number

$$7\;2\;1 \;=\; [1\;1\;1\;0\;1\;0\;0\;0\;1] \quad .$$

The polynomial has degree $n - k = 8$. It is

$$g(p) \;=\; p^8 \;+\; p^7 \;+\; p^6 \;+\; p^4 \;+\; 1 \quad .$$

$$[1\;1\;1\;0\;1\;0\;0\;0\;1] \tag{75}$$

$$[1\;1\;1\;1\;0\;0\;0]$$

$$\mathbf{C}_m \;=\; [0\;0\;0,\;\; 1\;0\;1\;1,\;\; 1\;0\;1\;1\;1,\;\; 1\;1\;1]$$

7.5.9 Other Linear Block Codes

In this Subsection, we overviewed a number of important binary linear block codes. These are important in their own right, and they are also often used as building blocks for other codes, such as concatenated and product codes, which we will investigate later. Descriptions of other useful codes, such as Euclidean geometry, projective geometry, quadratic residue and fire codes, can be found in Lin & Costello's text [1].

We have considered binary linear block codes, there performance parameters d_{min} and the weighting function wn_j, and some encoder structures. We now turn to decoding algorithms for and performance analysis of these codes.

7.6 Binary Linear Block Code Decoding & Performance Analysis

In a digital communication system, we receive noisy symbols. For a system employing block coding and a given modulation scheme, we will receive a sequence of waveforms, that represent a block codeword $\mathbf{C}_m$, which is corrupted by noise. Under the assumptions that:

- the sequence of information bits is statistically independent,
- a memoryless modulation scheme is employed,
- channel is memoryless; and
- the noise is white,

the optimum receiver front end will consist of a filter matched to the symbol waveform followed by a symbol-rate sampler. This receiver structure is illustrated in Figure 60. The matched filter output vector $\underline{r}$ contains the samples corresponding to a transmitted codeword $\mathbf{C}_m$. That is, it contains the n matched filter output samples corresponding to the n symbols representing $\mathbf{C}_m$. Under the assumptions stated above, this vector can be processed (without any information of the received data for other codewords) to optimally estimate $\mathbf{C}_m$. The exact statistical characteristics $\underline{r}$, and this the communications system performance, will depend on the modulation scheme employed.

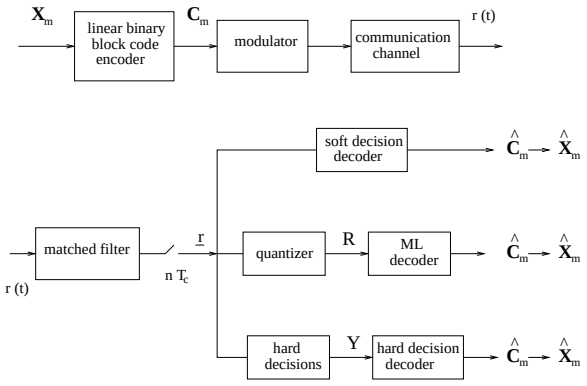


Figure 60: Digital communication system with block channel encoding.

From the matched filter output samples, there are several basic approaches to information vector detection. On the one hand, the matched filter output samples can be detected one symbol at a time to generate a binary (hard) received vector $\mathbf{Y}$, which is a codeword possibly corrupted by noise (i.e. $\mathbf{Y} = \mathbf{C}_m + \mathbf{e}$). From this binary vector a codeword estimate $\hat{\mathbf{C}}_m$ and corresponding information bit vector estimate $\hat{\mathbf{X}}_m$ are generated. This is called *hard-decision decoding* since hard decisions are made on each symbol prior to decoding. Alternatively, the undetected (soft) matched filter output can be processed directly as a block to estimate the transmitted codeword and corresponding information bit vector. This is called *soft-decision decoding*. In between these schemes, there is the possibility of quantizing the sampled matched filter output vector $\mathbf{r}$ to form discrete valued $\mathbf{R}$. *Note that a soft-decision decoder will outperform a hard-decision or quantized* $\mathbf{r}$ *decode because the these latter decoders processes quantized data.*

In this Subsection we consider decoding of linear binary block codes for AWGN channels. We will first describe soft-decision decoding and analyze it for several of the modulation schemes considered in Section 3 of this Course. Results will be in terms of the identified block code properties. Analysis will take advantage of previous results from Section 3. We will then describe and analyze hard-decision decoding, which is more practical to implement. We conclude with a comparison of soft and hard-decision decoding performance.

7.6.1 Soft-Decision Decoding

In this Subsection we investigate binary linear block code performance for AWGN channels and either $M = 2, 4$ symbol coherent PSK or coherent/noncoherent $M = 2$ (i.e. binary) orthogonal FSK. We present expressions for codeword error as a function of SNR per bit $\gamma_b = \mathcal{E}_b/N_0$.

We define $\mathcal{E} = n\mathcal{E}_c$ as the codeword energy, where $\mathcal{E}_c$ is the energy per codeword bit. Then the information bit energy is

$$\mathcal{E}_b \;=\; \frac{\mathcal{E}}{k} \;=\; \frac{n}{k}\mathcal{E}_c \quad . \tag{76}$$

We will consider maximum likelihood (ML) codeword detection, given $\mathbf{r}$, and assume that all M codewords have equal probabilities. The ML approach then minimizes probability of codeword error.

Let $T = nT_c$ be the codeword duration, where T_c is the duration of each bit in the codeword. Denote as $s_i(t);\ 0 \le t \le T$ the modulated waveform corresponding to the i^{th} codeword, $i = 1, 2, \cdots, M = 2^k$. The exact form of an $s_i(t)$ depends on the modulation scheme and the codeword $\mathbf{C}_i$ it represents.

Binary PSK with Coherent Reception

For binary PSK, the codeword waveforms $s_i(t)$ are of the form

$$s_i(t) \;=\; \sqrt{\mathcal{E}_c}\ \sum_{j=1}^{n}\ (2c_{ij} - 1)\ f(t - (j-1)T_c) \quad i = 1, 2, \cdots, M = 2^k, \tag{77}$$

where

$$f(t) \;=\; \sqrt{\frac{2}{\mathcal{E}_g}}\ g(t)\ \cos(2\pi f_c t) \quad 0 \le t \le T_c \quad , \tag{78}$$

and $\mathbf{C}_i = [c_{i1}, c_{i2}, \cdots, c_{in}]$. The received waveform is

$$r(t) \;=\; s_i(t) \;+\; n(t)\ ; \quad 0 \le t \le T \quad . \tag{79}$$

In Figure 61 we illustrate a receiver in which each bit of a codeword is match-filtered and sampled. Let's take a step back and discuss why this is the first step in an ML receiver. Consider Figure 61(a). Paralleling the discussion on optimum detection of M symbols in an M-ary memoryless modulation scheme, under assumptions stated at the beginning of Subsection 7.6, the optimum receiver front end consists of a bank of M filters, each matched to a codeword waveform. Each matched filter effectively correlates the received waveform $r(t)$ with a codeword waveform $s_i(t)$. [4] Each filter is followed by a codeword rate sampler.

[4] This use of matched filters presumes coherent detection.

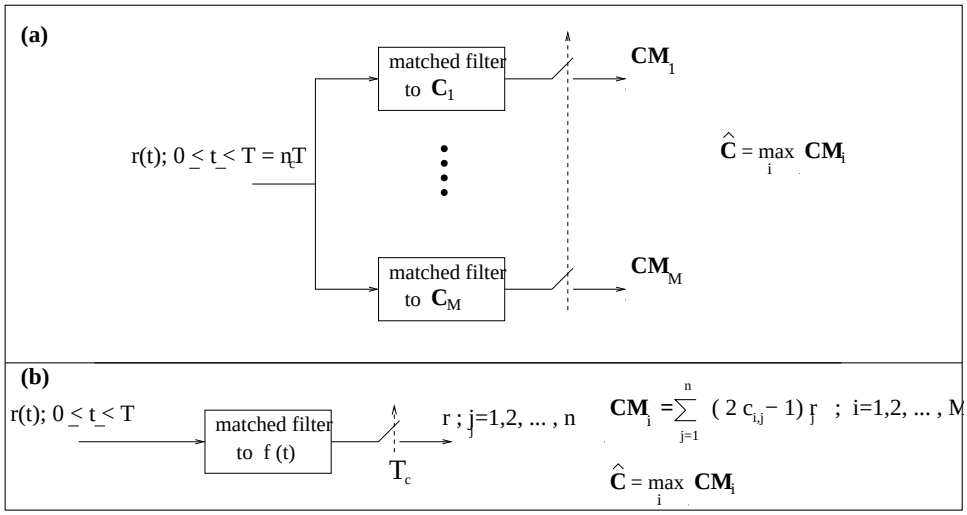


Figure 61: ML receiver for an M codeword block code: (a) using filters matched to each codeword waveform; (b) practical implementation using a filter matched to the symbol shape.

The estimated codeword corresponds to the largest matched filter output sample. That is, the ML codeword detector is the nearest neighbor detector.

The i^{th} matched filter output is

$$\begin{aligned} CM_i &= \int_0^T r(t)\ s_i(t)\ dt \\ &= \int_0^T r(t)\ \left(\sum_{j=1}^{n}(2c_{ij}-1)\ f(t-(j-1)T_c)\right)\ dt \\ &= \sum_{j=1}^{n}(2c_{ij}-1)\ \int_0^T r(t)\ f(t-(j-1)T_c)\ dt \\ &= \sum_{j=1}^{n}(2c_{ij}-1)\ \int_{(j-1)T_c}^{jT_c} r(t)\ f(t-(j-1)T_c)\ dt \\ &= \sum_{j=1}^{n}(2c_{ij}-1)\ r_j \quad , \end{aligned} \tag{80}$$

where

$$r_j = \int_{(j-1)T_c}^{jT_c} r(t)\ f(t-(j-1)T_c)\ dt \quad . \tag{81}$$

Eqs (80,81) suggest the alternative front end, shown in Figure 61(b), based on a filter matched to a single codeword element waveform and sampled at the codeword element rate.

Eqs (80,81) describe the *correlation matrix* for the ML codeword detector with binary PSK modulation. The ML estimator is

$$\hat{\mathbf{C}}_m = \arg\max_{\mathbf{C}_i} \quad CM_i \quad . \tag{82}$$

Performance Analysis:

The general approach for determining the codeword error probability is straightforward. Let P_e denote the codeword error probability, $P(e|\mathbf{C}_m)$ denote the probability of a codeword error given that $\mathbf{C}_m$ is sent, and $P(\mathbf{C}_i|\mathbf{C}_m)$ be the probability of detecting $\mathbf{C}_i$ given that $\mathbf{C}_m$ is sent. Then,

$$P_e \;=\; \sum_{m=1}^{M} P(\mathbf{C}_m)\; P(e|\mathbf{C}_m) \;=\; \frac{1}{M}\sum_{m=1}^{M} P(e|\mathbf{C}_m) \quad . \tag{83}$$

Note that for the AWGN channel considered here, the correlation metrics (i.e. the CM_i's) are Gaussian. Also note that for binary linear block codes, an AWGN channel and ML detection, the codeword detection error probability is equal for all codewords. Thus, we will consider the probability of error in detecting the all-zero codeword, $\mathbf{C}_1$, which is the easiest to analyze because the codeword weights describe the Hamming distances of all other codewords to $\mathbf{C}_1$. So we consider

$$P_e \;=\; P(e|\mathbf{C}_1) \quad . \tag{84}$$

The difficulty in determining P_e is that

$$P(e|\mathbf{C}_1) \;=\; P\left(\bigcup_{i=2}^{M} [(CM_i > CM_1)|\mathbf{C}_1]\right) \tag{85}$$

is difficult to evaluate because the events $[(CM_i > CM_1)|\mathbf{C}_1];\; i = 2, 3, \cdots, M$ are not disjoint. The exact probability will depend on many joint codeword probabilities conditioned on $\mathbf{C}_1$. So, rather than attempt to derive an explicit expression for P_e, we must resort to deriving bounds, and we strive to make these bounds simple but tight. Here we resort to the union bound

$$P\left(\bigcup_{i=2}^{M} [(CM_i > CM_1)|\mathbf{C}_1]\right) \;\leq\; \sum_{i=2}^{M} P(\mathbf{C}_i|\mathbf{C}_1) \quad , \tag{86}$$

where $P(\mathbf{C}_i|\mathbf{C}_1) = P[(CM_i > CM_1)|\mathbf{C}_1]$. So we need the $P(\mathbf{C}_i|\mathbf{C}_1)$. For these, we need the means and variances of the Gaussian correlation metrics $CM_i;\; i = 1, 2, \cdots, M$, conditioned on $\mathbf{C}_1$ having been transmitted, we start with

$$r_j = \int_{(j-1)T_c}^{jT_c} \left(\sqrt{\mathcal{E}_c}(2c_{1j} - 1)f(t - (j-1)T_c) + n(t)\right) f(t - (j-1)T_c)\; dt \; . \tag{87}$$

So,

$$E\{r_j\} \;=\; \sqrt{\mathcal{E}_c}\;(2c_{1j} - 1) \;=\; -\sqrt{\mathcal{E}_c} \quad . \tag{88}$$

For $E\{CM_1\}$, we have

$$\begin{aligned} E\{CM_1\} \;&=\; \sum_{j=1}^{n}(2c_{1j} - 1)\; E\{r_j\} \\ &=\; -\sum_{j=1}^{n} E\{r_j\} \\ &=\; n\sqrt{\mathcal{E}_c} \quad . \end{aligned} \tag{89}$$

Similarly,

$$E\{CM_m\} \;=\; (n-2w_m)\sqrt{\mathcal{E}_c}\;; \quad m=2,3,\cdots M\;, \tag{90}$$

where w_m is the weight of the m^{th} codeword. Also,

$$Var\{r_j\} \;=\; Var\{\int_{(j-1)T_c}^{jT_c} n(t)f(t-(j-1)T_c)\;dt\;\} \;=\; \frac{N_0}{2} \quad . \tag{91}$$

Now, getting back to the $P[(CM_i > CM_1)|\mathbf{C}_1]$, this is the probability that, given $\mathbf{C}_1$, $CM_i - CM_1 > 0$. Since, given $\mathbf{C}_1$, CM_i and CM_1 are independent Gaussians, $CM_i - CM_1$ is Gaussian with mean

$$E\{CM_i - CM_1\} \;=\; E\{CM_i\} \;-\; E\{CM_1\} \;=\; -2w_i\sqrt{\mathcal{E}_c} \tag{92}$$

and variance

$$\begin{aligned} Var\{CM_i - CM_1\} &= Var\left\{\sum_{j=1}^{n}(2c_{ij}-1)r_j \;-\; \sum_{j=1}^{n}(2c_{1j}-1)r_j\right\} \\ &= \sum_{j=1}^{n}\left[2(c_{ij}-c_{1j})\right]^2\;Var\{r_j\} \\ &= 2w_iN_0 \quad . \end{aligned} \tag{93}$$

So, $P[(CM_i > CM_1)|\mathbf{C}_1]$ is the area, from 0 to ∞, of a Gaussian PDF with mean $2w_i\sqrt{\mathcal{E}_c}$ and variance $2w_iN_0$. That is, simplifying the notation $P[(CM_i > CM_1)|\mathbf{C}_1]$ to $P(\mathbf{C}_i|\mathbf{C}_m)$,

$$\begin{aligned} P(\mathbf{C}_i|\mathbf{C}_m) &= Q\left(\frac{0+2w_i\sqrt{\mathcal{E}_c}}{\sqrt{2w_iN_0}}\right) \;=\; Q\left(\sqrt{\frac{2w_i\mathcal{E}_c}{N_0}}\right) \\ &= Q\left(\sqrt{\frac{\mathcal{E}}{N_0}(1-\rho_i)}\right) \end{aligned} \tag{94}$$

where $\rho_i = 1 - \frac{2w_i}{n}$ is the correlation between $\mathbf{C}_1$ and $\mathbf{C}_i$ and $\mathcal{E} = n\mathcal{E}_c$.

Now, recalling that $R_c = \frac{k}{n}$ is the code rate, $\gamma_b = \frac{\mathcal{E}_b}{N_0}$ is the SNR per bit, and $\mathcal{E}_c = R_c\mathcal{E}_b$, we have the following upper bound on the codeword error probability:

$$P_e \;\leq\; \sum_{i=2}^{M} Q\left(\sqrt{2\gamma_bR_cw_i}\right) \quad . \tag{95}$$

This upper bound requires the code weight distribution. Alternatively, since the codeword weight w_i is lower bounded by the code minimum distance d_{min} and the Chernov bound on $Q(x)$ is $e^{-x^2/2}$, we have the looser but simpler upper bound

$$P_e \;\leq (M-1)\;e^{-\gamma_bR_cd_{min}} \;<\; e^{(-\gamma_bR_cd_{min}+k\ln(2))} \tag{96}$$

which does not depend on the code weight distribution. (This is Eq (7.4-7) of the Text.)

Coding gain is the performance improvement achieved through coding compared to uncoded transmission. It is usually defined as the SNR gain for a particular level of performance, P_e. We will explore this later, when we compare performance of uncoded and block coded transmission using both Monte Carlo simulations and the performance equations derived above. For now, consider the uncoded BPSK symbol error probability (SEP)

$$P_b \;=\; P_e \;=\; Q\left(\sqrt{2\gamma_b}\right) \tag{97}$$

vs. γ_b. A plot of this can be found in most digital communications texts, for example on p. 177 of Proakis & Salehi [2]. Also, the block coding soft decoding performance bounds, Eqs (95,96), are plotted for the Golay (23,12) code on p. 438 of Proakis & Salehi [2]. For $P_e = 10^{-6}$, we observe from these plots a coding gain of at least 4dB (i.e. about 10.5dB for BPSK, and an upper bound of 6.5dB for the Golay (23,12) code). Of course, with $R_c = \frac{12}{23} \approx \frac{1}{2}$, the price paid is a doubling of code rate (i.e. doubling the bandwidth).

Information bit error probability is difficult to derive from the codeword error probability P_e because of the codeword dependence between a codeword error and the corresponding number of bit errors (i.e. the number of bit errors resulting from incorrectly deciding $\mathbf{C}_i$ over $\mathbf{C}_m$ depends on both $\mathbf{C}_i$ and $\mathbf{C}_m$. The useful relationship $P_b \leq \frac{P_e}{2}$ is often used.

Quaternary (i.e. $M = 4$) PSK with Coherent Reception

Since uncoded (binary) PSK and QPSK have identical performance, so does block coded PSK and QPSK. Thus, Eqs (95,96) bound performance for QPSK.

Binary Orthogonal FSK with Coherent Reception

Compared to PSK or QPSK, for which $P(\mathbf{C}_i|\mathbf{C}_1) = Q\left(\sqrt{2\gamma_b R_c w_i}\right)$, for coherent binary orthogonal FSK,

$$P(\mathbf{C}_i|\mathbf{C}_1) \;=\; Q\left(\sqrt{\gamma_b R_c w_i}\right) \quad . \tag{98}$$

That is, its performance is 3dB worse, and therefore so are its performance bounds. For the same channel code, to achieve the same codeword error probability, twice the SNR per bit is required for coherent FSK as for coherent PSK or QPSK.

Binary Orthogonal FSK with Noncoherent Reception

In general, compared to coherent detection, there is a degradation in performance associated with noncoherent detection. The rule-of-thumb is that the degradation is about 3dB.

7.6.2 Hard-Decision Decoding

In the last Subsection we considered an optimum soft-decision approach to decoding block codes. We specifically considered binary linear block codes, although the formulation is general enough to be applicable to nonlinear and nonbinary block codes. The approach is optimum in that, for codeword transmission duration $0 \leq t \leq T$, the codeword estimate $\hat{\mathbf{C}}_m$ is the ML estimate based on observation $r(t);\ 0 \leq t \leq T$. Though the approach requires the computation and comparison of M correlation matrices $CM_i;\ i = 1, 2, \cdots, M$, these metrics are efficiently computed using a single filter, matched to the codeword elements, followed by a codeword element rate sampler. Even so, one shortcoming of this approach is its computational requirements, which can be impractical for large M.

We now consider hard-decision decoding, a computationally attractive approach which, from the perspective of starting with $r(t);\ 0 \leq t \leq T$ and generating an estimate of $\mathbf{C}_m$, is suboptimum. It differs from the soft-decision approach in that decisions are made on the matched filter sampled outputs for each codeword element. This can be considered a severe form of quantization. This quantization results in loss of information, i.e. the quantized outputs are not sufficient statistics of $r(t);\ 0 \leq t \leq T$ for the purpose of estimating $\mathbf{C}_m$.

Below we describe: the hard-decision processing structure; minimum Hamming distance (ML) decoding; syndrome decoding; an efficient decoder for cyclic codes, and error detection & correction performance.

The Hard-Decision Decoder Structure

Figure 62 shows the basic hard-decision decoder. The receiver front end consists of the same front end as for the soft-decision decoder, followed by an element-by-element ML detector which processes the n-dimensional vector $\mathbf{r}$ of matched filter output samples to give binary data vector $\mathbf{Y}$. The problem is then to derive a codeword estimate $\hat{\mathbf{C}}_m$ from $\mathbf{Y}$.

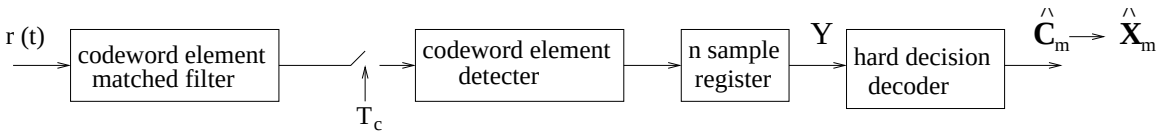


Figure 62: The hard-decision block code decoder.

We will consider hard-decision decoding for coherent PSK and coherent/noncoherent FSK. Note that for SNR per information bit $\gamma_b = \frac{\mathcal{E}_b}{N_0}$, the SNR per codeword element is $\gamma_b R_c$, where $R_c = \frac{k}{n}$ is the block code rate. Recall from earlier in the Course that the probabilities of bit errors for ML codeword element detection in AWGN channels are

$$\rho \ = \ Q(\sqrt{2\gamma_b R_c}) \ , \tag{99}$$

$$\rho \ = \ Q(\sqrt{\gamma_b R_c}) \ , \tag{100}$$

$$\rho \ = \ \frac{1}{2}\, e^{-\gamma_b R_c/2} \tag{101}$$

for coherent PSK, coherent FSK and noncoherent FSK respectively.

Maximum Likelihood Decoding of Y

Given the binary data vector $\mathbf{Y} = [y_1, y_2, \cdots, y_n]$, with possible element errors, the maximum likelihood codeword detection problem is

$$\max_{\mathbf{C}_m} \qquad P(\mathbf{Y}/\mathbf{C}_m) \;=\; \prod_{i=1}^{n} \; [(1-\rho)\; \delta(y_i - c_{mi}) \;+\; \rho\; \delta(y_i - \overline{c}_{mi})] \quad . \tag{102}$$

Assume $\rho < 0.5$. A direct solution approach is to directly compare each $\mathbf{C}_m$ to $\mathbf{Y}$ pick as $\hat{\mathbf{C}}_m$ the $\mathbf{C}_m$ with minimum Hamming distance from $\mathbf{Y}$. Therefore, an equivalent ML problem statement is:

$$\min_{\mathbf{C}_m} \qquad w_h(\mathbf{Y} - \mathbf{C}_m) \quad , \tag{103}$$

where $w_h(\mathbf{Y} - \mathbf{C}_m)$ is the Hamming weight of the binary hard-decision *error pattern* vector $\mathbf{e}_m \;=\; \mathbf{Y} - \mathbf{C}_m$.

The Syndrome, and Error Detection

More efficient hard-decision ML decoding algorithms (i.e. that solves Eq (102)) can be derived using the code parity check matrix $\mathbf{H}$, which has rows that span the codes $(n-k)$-dimensional orthogonal complement subspace $\overline{\mathcal{C}}$. The binary data vector $\mathbf{Y}$ can be written as

$$\mathbf{Y} \;=\; \mathbf{C_m} \;+\; \mathbf{e} \;=\; \{\mathbf{C_m} \;+\; \mathbf{e}_u\} \;+\; \mathbf{e}_d \quad , \tag{104}$$

where $\mathbf{C}_m$ is the transmitted codeword, $\mathbf{e}$ is the error vector, $\mathbf{e}_u$ is the part of the error vector in $\mathcal{C}$ and $\mathbf{e}_d$ is the part of the error vector in $\overline{\mathcal{C}}$. Since, for a linear block code, $\{\mathbf{C_m} \;+\; \mathbf{e}_u\}$ is a codeword, $\mathbf{e}_u$ is the undetectable part of the error vector and $\mathbf{e}_d$ is the detectable part. Let

$$\mathbf{Y}\;\mathbf{H}^T \;=\; (\mathbf{C_m} \;+\; \mathbf{e})\mathbf{H}^T \;=\; \mathbf{e}\;\mathbf{H}^T \;=\mathbf{S} \quad , \tag{105}$$

where $\mathbf{S} = \mathbf{Y}\;\mathbf{H}^T = [s_1,\; s_2, \cdots, s_{(n-k)}]$. $\mathbf{S}$, called the *syndrome* of the error vector, is the $(n-k)$-dimensional representation of $\mathbf{e}_d$. It is the error vector $\mathbf{e}$ collapsed onto $\overline{\mathcal{C}}$.

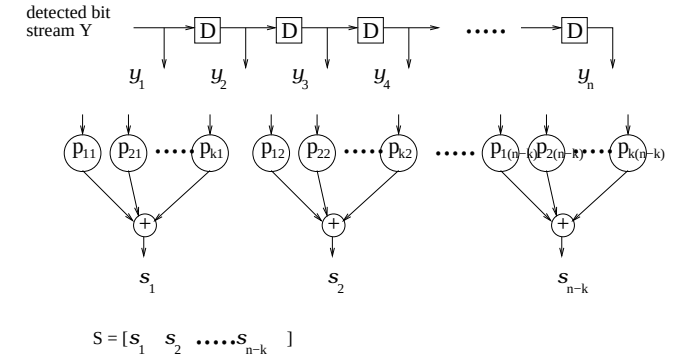
By processing the syndrome $\mathbf{S}$, we will be looking at detectable error patterns since a nonzero $\mathbf{S}$ indicates that there detectable errors. Processing based on the 2^{n-k} possible $(n-k)$-dimensional $\mathbf{S}$ has computational advantages compared to processing based on the 2^k possible n-dimensional $\mathbf{C}_m$.

For a systematic code, the $(n-k)$ elements of $\mathbf{S}$ are the results of the $(n-k)$ parity checks on $\mathbf{Y}$. Figure 63(a) shows a general syndrome calculation circuit. Figure 63(b) shows a syndrome circuit for the Hamming (7,4) code.

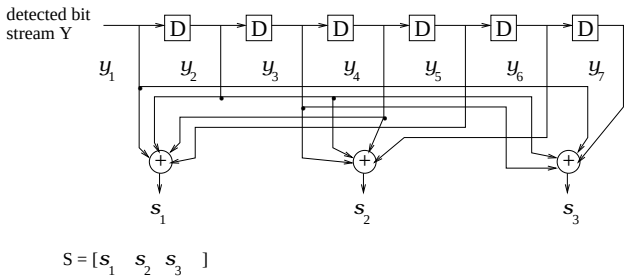
Syndrome Based Error Correction Decoding

In Subsection 6.4.2 we discussed the relationship between d_{min} and coding error detection e_d and error correction e_c capabilities for hard-decision decoding. Strategies that take advantage of error detection (with or without correction) capabilities employ automatic repeat-request (ARQ) and retransmit capabilities. Here we present an efficient error correction decoding algorithm for a general linear block code.

Syndrome based error correction decoding consists of three stages: 1) syndrome calculation; 2) association of the syndrome with an error pattern; and 3) error correction. This is



(a) general shift register syndrome calculation



(b) Hamming (7,4) code syndrome calculation

Figure 63: Syndrome calculation for a systematic block code.

illustrated in Figure 64. Syndrome calculation has already been addressed. Error correction involves simply adding the detected error pattern $\hat{\mathbf{e}}$ to the the received binary vector $\mathbf{Y}$ to form the estimated codeword

$$\hat{\mathbf{C}}_m \;=\; \mathbf{Y} \;+\; \hat{\mathbf{e}} \quad . \tag{106}$$

What remains to be established is the association of the syndrome with an error pattern.

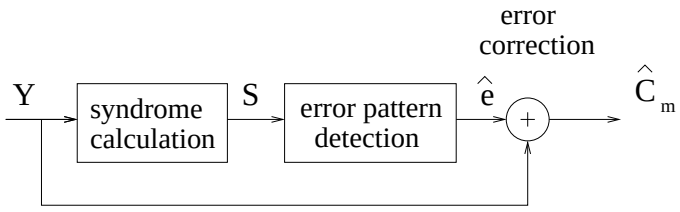


Figure 64: Syndrome based error correction decoding.

There are many error patterns that result in the same syndrome $\mathbf{S}$. The ML minimum $w_h(\mathbf{Y} - \mathbf{C}_m)$ objective is equivalent to selecting the minimum Hamming weight $\mathbf{e}$ consistent with $\mathbf{S} = \mathbf{Y}\ \mathbf{H}^T$, and then choosing $\hat{\mathbf{C}}_m$ as the codeword associated with that error. That is, given a $\mathbf{Y}$, resulting syndrome $\mathbf{S} = \mathbf{Y}\ \mathbf{H}^T$, and corresponding minimum Hamming weight error $\hat{\mathbf{e}}$, the ML codeword estimate is

$$\hat{\mathbf{C}}_m \;=\; \mathbf{Y} \;+\; \hat{\mathbf{e}} \quad . \tag{107}$$

The error pattern $\mathbf{e}$ of minimum Hamming weight associated with a particular syndrome $\mathbf{S}$ is termed the syndrome's *coset leader*[5]. For a given (n, k) block code, the *syndrome table* is a list of the coset leaders for each $(n-k)$-dimensional syndrome. To determine the syndrome table, first list the zero syndrome (associated with zero errors). Next determine the syndromes associated with all single errors (i.e. for the error vectors with Hamming weight 1). These syndromes are the rows of $\mathbf{H}^T$. For any syndromes not yet covered, determine syndromes associated with double errors. These syndromes are the linear combinations of two rows of $\mathbf{H}^T$. Continue this process until all syndromes are covered.

Example 7.15: Determine the syndrome table for the $(7, 4)$ Hamming code.

Since a Hamming code is perfect and capable of single error correction, the syndrome table is completely constructed using all of the single-error vectors. Equivalently, since for a Hamming code the columns of $\mathbf{H}$ are all the nonzero $(n-k)$-dimensional vectors, the rows of $\mathbf{H}^T$ are all the nonzero syndromes. For the (7,4) Hamming code,

$$\mathbf{H}^T = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{108}$$

The syndrome table is

S	**e**
000	0000000
001	0000001
010	0000010
011	0001000
100	0000100
101	1000000
110	0010000
111	0100000

Note that the syndrome indicates which element of $\mathbf{Y}$ is in error.

The mapping from the syndrome $\mathbf{S}$ to the minimum Hamming weight error pattern $\hat{\mathbf{e}}$, as indicated by the syndrome table, can be efficiently implemented using a combinational logic circuit (see Lin & Costello [1], pp. 88-89).

[5]The coset leaders are the entries in the first column of the code's *standard array*. These entries are the 2^{n-k} minimum weight error patterns. Each row of the standard array is called a *coset*. For a particular coset leader, the coset contains the possible received binary vectors $\mathbf{Y}$ for that error pattern (i.e. the set of the coset leader added to each codeword.

An Efficient Systematic Cyclic Code Decoder

An efficient syndrome based decoder for a systematic binary cyclic code is shown in Figure 64, where the error pattern detector is implemented as a combinational logic circuit as noted above, and the syndrome calculation is implemented using a feedback shift register. To see this, consider the polynomial representation of the received binary data vector $\mathbf{Y}$,

$$y(p) \;=\; y_{n-1}p^{n-1} \;+\; y_{n-2}p^{n-2} \;+\; \cdots y_1 p \;+\; y_0 \;=\; c_m(p) \;+\; e(n) \quad , \tag{109}$$

where $c_m(p)$ is the codeword polynomial and $e(n)$ is the binary error polynomial. Since $c_m(p)$ is divisible by the generator polynomial $g(p)$, we can express $y(p)$ as

$$y(p) \;=\; q(p)\; g(p) \;+\; r(p) \quad , \tag{110}$$

where $r(p)$, the remainder, is due to the error. In fact, it is the remainder polynomial of $e(n)$ divided by $g(p)$. The coefficients of this remainder are the syndromes. Thus, the syndromes can be calculated using the feedback shift register circuit described in Subsection 7.2 for binary polynomial division. Figure 65 illustrates the circuit. Note that $g_0 = g_{n-k} = 1$ is explicitly implemented. The syndromes are the contents of the shift register after the final received data vector bit has been shifted into the circuit.

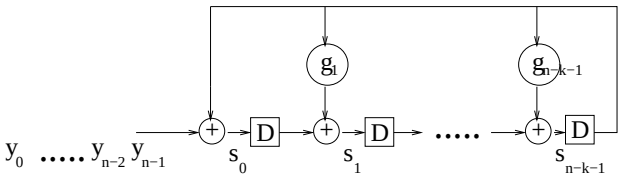


Figure 65: An efficient syndrome based decoder for systematic binary cyclic codes.

Probability of Codeword Error Analysis

Consider operating in strictly error correction mode, using ML detection based on the hard-decision data vector $\mathbf{Y}$. In this case, a codeword error can occur whenever the number of element errors is greater than $t = \lfloor \frac{1}{2}(d_{min} - 1) \rfloor$. Figures 55,56 illustrate this. The codeword error probability P_e will be less then or equal to the probability of making more than t errors.

A codeword element error can be thought of as a Bernoulli trial, with ρ being the probability of error and $(1 - \rho)$ the probability of correct detection. Then, since element errors are independent (under previously stated assumptions), the number of element errors in a codeword is binomial distributed. Letting l denote the number of element errors, its PDF is

$$P(l) \;=\; \sum_{m=0}^{n} P(m,n)\; \delta(l-m) \quad , \tag{111}$$

where

$$P(m,n) \;=\; \left(\begin{array}{c} n \\ m \end{array} \right) \; \rho^m \; (1-\rho)^{n-m} \quad . \tag{112}$$

Thus,

$$P_e \leq \sum_{m=t+1}^{n} P(m,n) \quad . \tag{113}$$

This result was established earlier, in Subsection 7.4.1. Values of ρ for the modulation schemes we consider here are given in Eqs(100, 100, 101).

For perfect codes (i.e. Hamming codes and the Golay code), every $\mathbf{Y}$ is within a sphere of radius t of a valid codeword, and these spheres are disjoint, so

$$P_e = \sum_{m=t+1}^{n} P(m,n) = 1 - \sum_{m=0}^{t} P(m,n) \quad . \tag{114}$$

So perfect codes are optimum codes for hard-decision ML decoding.

There are only a few perfect codes. However there are *quasiperfect codes* for most values of (n,k). These codes are optimum codes for hard-decision ML decoding. For these codes,

$$\sum_{m=t+2}^{n} P(m,n) \leq P_e \quad . \tag{115}$$

Eq (113) is an upper bound on the ML codeword probability of error for a block code. Eq (114) is the exact ML codeword probability of error for a perfect code. Since quasiperfect codes are optimum, Eq (115) is a lower bound on P_e for any block code.

7.6.3 A Comparison Between Hard-Decision and Soft-Decision Decoding

In this Subsection we resort to performance equations and Monte Carlo simulations to study the performance of the Hamming (15,11) block code. We assume BPSK modulation with coherent reception. We consider both hard and soft decision decoding. The parity bit generator matrix for this code, in systematic form, is

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}^T \quad . \tag{116}$$

This is a code rate $R_c = \frac{11}{15}$ block code, with $d_{min} = 3$ since it is a Hamming code. It's weight distribution is shown in the bottom left plot of Figure 66. The top left plot shows four BER vs. SNR/bit results:

- The theoretical BER curve for uncoded BPSK with coherent reception, $P_b = Q(\sqrt{2\gamma_b})$, is the solid red line.
- The tighter of the two soft decision bounds is shown as the solid magenta line. As noted earlier, $P_b = \frac{P_e}{2}$ where $P_e \leq \sum_{i=2}^{M} Q(\sqrt{2\gamma_b R_c w_i})$ where w_i is the weight of the $i-th$ codeword and $M = 2^k$ is the number of codewords for the code.
- The Monte Carlo hard decision results are given by the blue "+" curve. Note that these results indicate with hard decision decoding a significant advantage relative to no coding is realized only at higher SNR.

- The Monte Carlo soft decision results are given by the green "*" curve. These results follow closely the tighter bound curve for higher SNR. Soft decision decoding outperforms hard decision decoding & uncoded transmission, especially at higher SNR.

The top right plot shows five codeword error probability vs. SNR/bit results:

- The looser of the two soft decision bounds is shown as the solid red line. This bound, given by $P_e < (M-1)\, e^{-\gamma_b R_c d_{min} + l \ln(2)}$ as noted in Subsection 7.6.1, is not accurate in this case.
- The tighter of the two soft decision bounds, $P_e = \sum_{i=2}^{M} Q(\sqrt{2\gamma_b R_c w_i})$, is shown as the solid magenta line.
- As noted in Subsection 7.6.2, since the employed Hamming code is a perfect code, the performance curve given by Eq (114) is exact. This curve shown as the solid cyan line.
- The Monte Carlo hard decision results are given by the blue "+" curve. These results closely match the theoretical curve, as expected.
- The Monte Carlo soft decision results are given by the green "*" curve. These results follow closely the tighter bound curve for higher SNR, and the theoretical hard decision curve for lower SNR. Again, soft decision decoding outperforms hard decision decoding, especially at higher SNR.

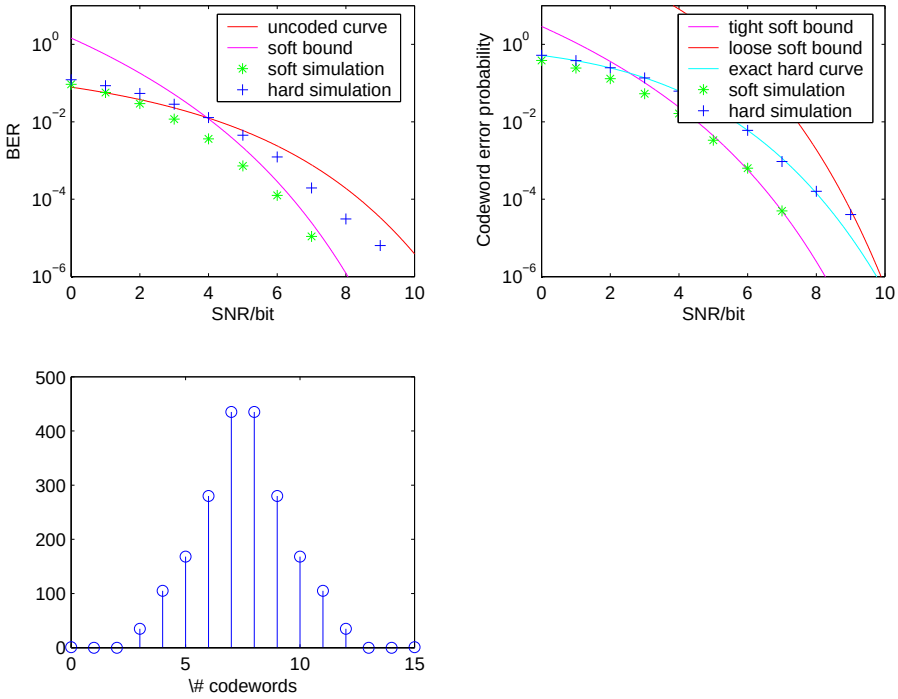


Figure 66: Performance evaluation of the Hamming (15,11) code with coherent reception and both hard & soft decision decoding: (a) BER vs. SNR/bit; (b) codeword error probability vs. SNR/bit; (c) codeword weight distribution.

7.6.4 Bandwidth Considerations

As defined earlier, the rate of an (n, k) binary block code is $R_c = \frac{k}{n}$, which is less than one. Given an information bit rate of R, after coding with an (n, k) binary block code, the codeword bit rate is increased to $\frac{R}{R_c}$. Compared to uncoded transmission, to maintain the same information transmission rate, the transmission bit rate must be increased. If the modulation scheme is fixed, then the symbol rate (and thus the transmission bandwidth) must be increased by a factor of R_c^{-1}. The bandwidth expansion factor,

$$B_e \;=\; \frac{1}{R_c} \;=\; \frac{n}{k} \tag{117}$$

(Eq (7.4-21) of the Text), quantifies this increase.

> *Example 7.17:* In this example we compare performance of three uncoded and coded binary PSK communication schemes. We compare SNR/information-bit γ_b required to achieve $P_b = 10^{-5}$. We also compare performance to capacity.
>
> - First consider uncoded binary PSK. Its bandwidth efficiency is $\frac{R}{W} = 1$. From your Computer Project I results, we see that $\gamma_b = 9.4\ dB$ is required for $P_e = 10^{-5}$ (also see Figure 4.2-5 of the Text). From Figure 4.6-1 of the Text we see that reliable communications is theoretically possible at $\frac{C}{W} = 1$ for $\gamma_b \geq 0\ dB$. So uncoded binary PSK is about 9.4 dB from capacity.
> - Second, consider using the (15,11) Hamming code with binary PSK as discussed directly above. From Figure 66, we see that $\gamma_b \approx 7.5\ dB$ is required to achieve $P_b = 10^{-5}$. This is about 1.9 dB better than uncoded binary PSK. However, this gain comes at the expense of an increase in bandwidth of $B_e = \frac{15}{11}$. It is more fair to compare performance with capacity. From Figure 4.6-1 of the Text, reliable communications is theoretically possible at $\frac{C}{W} = \frac{11}{15}$ for $\gamma_b \geq -0.5\ dB$. So the Hamming (15,11) code with binary BPSK is about 8.0 dB from capacity, which is 1.4 dB better than uncoded binary PSK.
> - Third, looking ahead to Subsection 8.9-2 of the Text, a rate $R_c = \frac{1}{2}$ turbo code is described and analyzed with binary PSK. As noted in that Subsection, $\gamma_b \approx 0.7\ dB$ is required to achieve $P_b = 10^{-5}$ (see Figure 8.9-4). This is about 8.7 dB better than uncoded binary PSK. However, this gain now comes with an increase in bandwidth of $B_e = 2$. From Figure 4.6-1 of the Text, reliable communications is theoretically possible at $\frac{C}{W} = \frac{1}{2}$ for $\gamma_b \geq -0.8\ dB$. So the rate $\frac{1}{2}$ turbo code with binary BPSK is about 1.5 dB from capacity, which is 7.9 dB better than uncoded binary PSK.

7.7 Nonbinary Block Codes - Reed-Solomon (RS) Codes

Nonbinary (N, K) block codes consist of codewords of length N with elements selected from the q letter alphabet of symbols $\{0, 1, 2, \cdots, q-1\}$. Each codeword represents an information vector of K symbols, each symbol being from the q letter alphabet. Thus q^K different information symbol vectors are represented by q^K codewords out of a possible q^N N-dimensional vectors. So $N > K$. Typically, $q = 2^m$ so that an information vector element represents m bits. For a systematic (N, K) code, the first K elements of a codeword are the information symbols and the remaining $(N-K)$ elements are parity check symbols. Nonbinary block codes are characterized by their minimum distance D_{min}, which is the minimum Hamming distance between any two codewords. Small $\frac{q^K}{q^N}$ can result in large D_{min}. A principal objective of nonbinary block codes is to employ an efficient algebraic formulation to develop large codes, with good performance characteristics and practical implementations.

Recall from our discussion on binary block codes that BCH codes are a subset of cyclic codes. This is true for nonbinary codes as well, and Reed Solomon (RS) codes form a subset of BCH codes that has particularly good distance properties, known weight enumerating polynomials, efficient hard-decision decoding algorithms, good burst error characteristics, and good self-synchronization properties. Nonbinary linear block codes, in particular BCH and specifically RS codes, are widely used. Examples of their successful application include: CD-ROMs, wireless communications, space communications, DSL, DVD's, and digital TV.

Because an understanding RS codes requires more advanced Galois field concepts, we begin this Subsection with an overview of GF(2^m) fields. We then describe RS codes, and discuss practical encoding and decoding algorithms.

7.7.1 A GF(2^m) Overview for Reed-Solomon Codes

GF(2^m) is an expansion Galios field consisting of an alphabet of $q = 2^m$ digits (where m is a positive integer), along with addition and multiplication operators. These operators are described below, after we develop several representations of the digits.

It is useful, but not necessary, to think of the alphabet of a GF(2^m) as the set of all m-bit bytes, denoted as $\underline{\mathbf{b}}_i; i = 0, 1, \cdots, 2^m - 1$. For each digit of the alphabet, we can think of an $(m-1)$ order binary polynomial, denoted $b(p)$, whose coefficients are the elements of $\underline{\mathbf{b}}$. A third useful alphabet representation is in terms of base 2^m digits, $\mathbf{b}_i;\ i = 1, 2, \cdots, 2^m - 1$.

Example 7.18: For GF(2^4), the alphabet can be thought of as

$$\{\underline{\mathbf{b}}_0,\ \underline{\mathbf{b}}_1, \cdots, \underline{\mathbf{b}}_{15}\} \ = \ \{0000,\ 0001,\ 0010,\ \cdots,\ 1111\} \tag{118}$$

with corresponding binary polynomials

$$\{b_0(p),\ b_1(p), b_2(p), \cdots,\ b_{15}(p)\} \ = \ \{0,\ 1,\ p,\ \cdots,\ p^3 + p^2 + p + 1\} \ . \tag{119}$$

In this example the base 16, or *hexagonal*, representation is

$$\{\mathbf{b}_0,\ \mathbf{b}_1, \cdots, \mathbf{b}_{15}\} \ = \ \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ a,\ b, c,\ d,\ e,\ f\} \ . \tag{120}$$

Using this notation, a GF(2^m) is defined by the elements, along with a particular arithmetic (addition and multiplication operators) based on binary polynomials.

Addition

The addition of two digits is bit-by-bit modulo-2 addition.

Example 7.19: For GF(2^4),

$$\underline{\mathbf{b}}_5 \; + \; \underline{\mathbf{b}}_{11} = [0101] \; + \; [1011] \; = \; [1110] \; = \; \underline{\mathbf{b}}_{14} \tag{121}$$

or

$$b_5(p) \; + \; b_{11}(p) \; = \; (p^2+1) \; + \; (p^3+p+1) \; = \; p^3+p^2+p \; = \; b_{14}(p) \; . \tag{122}$$

This definition of addition adheres to all required properties for a field (i.e. closure, associativity, commutativity, zero and negative elements).

Multiplication

For multiplication, we start with a rank m binary irreducible polynomial $p(p)$. (Recall from Subsection 7.2 that an irreducible polynomial can not be factored into a product of polynomials of lower rank; e.g. p, p+1, p^2+1, p^2+p+1 are the irreducible binary polynomials of order 1 & 2).

Example 7.20: For GF(2^4), an irreducible polynomial used for RS code definition is $p(p) \; = \; p^4+p+1$.

Multiplication for GF(2^m) is then defined as modulo-$p(p)$ binary polynomial multiplication, where $p(p)$ is binary irreducible.

Example 7.21: For GF(2^4) and $p(p) = p^4+p+1$,

$$\begin{aligned}
c(p) &= b_5(p) \cdot b_{11}(p) && && (123)\\
&= (p^2+1)(p^3+p+1) && mod\ p(p) && (124)\\
&= p^5+p^2+p+1 && mod\ (p^4+p+1) && (125)\\
&= 1 && (the\ remainder) && (126)
\end{aligned}$$

The *mod* (p^4+p+1) operation, i.e. the calculation of the remainder, is

$$\begin{array}{rl}
 & p \\
p^4+p+1 \; \sqrt{} & p^5 + \qquad\qquad +p^2+p+1 \\
 & \underline{p^5 \qquad\quad +p^2+p} \\
 & \qquad\qquad\qquad\quad 1
\end{array} \tag{127}$$

i.e. $\underline{b}_5 \cdot \underline{b}_{11} = \underline{b}_1 = [0001]$.

This multiplication adheres to all required properties for a field (i.e. closure, associativity, commutativity, distributivity, identity and inverse elements).

Primitive Element of GF(2^m)

A primitive element of GF(2^m) is any element whose powers generate all non-zero elements of GF(2^m). That is, using a base 2^m representation, if α is a primitive element of GF(2^m), then $\{1 = \alpha^{2^m-1}, \alpha^1, \alpha^2, \alpha^3, \cdots, \alpha^{2^m-2}\}$ is the set of all of the non-zero elements.

Example 7.22: For GF(2^4) and $p(p) = p^4 + p + 1$, $\underline{\alpha} = \underline{\mathbf{b}}_3 = [0011]$ or $\alpha(p) = b_3(p) = p + 1$ is a primitive element, since

$$
\begin{array}{lcccc}
(\alpha(p))^1 = p+1 & \rightarrow & [0011] & \rightarrow & 3 \\
(\alpha(p))^2 = p^2+1 & \rightarrow & [0101] & \rightarrow & 5 \\
(\alpha(p))^3 = p^3+p^2+p+1 & \rightarrow & [1111] & \rightarrow & f \\
(\alpha(p))^4 = p & \rightarrow & [0010] & \rightarrow & 2 \\
(\alpha(p))^5 = p^2+p & \rightarrow & [0110] & \rightarrow & 6 \\
(\alpha(p))^6 = p^3+p & \rightarrow & [1010] & \rightarrow & a \\
(\alpha(p))^7 = p^3+p^2+1 & \rightarrow & [1101] & \rightarrow & d \\
(\alpha(p))^8 = p^2 & \rightarrow & [0100] & \rightarrow & 8 \\
(\alpha(p))^9 = p^3+p^2 & \rightarrow & [1100] & \rightarrow & c \\
(\alpha(p))^{10} = p^2+p+1 & \rightarrow & [0111] & \rightarrow & 7 \\
(\alpha(p))^{11} = p^3+1 & \rightarrow & [1001] & \rightarrow & 9 \\
(\alpha(p))^{12} = p^3 & \rightarrow & [1000] & \rightarrow & 8 \\
(\alpha(p))^{13} = p^3+p+1 & \rightarrow & [1011] & \rightarrow & b \\
(\alpha(p))^{14} = p^3+p^2+p & \rightarrow & [1110] & \rightarrow & e \\
(\alpha(p))^{15} = 1 & \rightarrow & [0001] & \rightarrow & 1
\end{array}
\qquad (128)
$$

7.7.2 Reed-Solomon (RS) Codes

Generally, RS codes are defined over an extension field GF(p^m). In application, $p = 2$ is typical, so we will restrict our discussion to codes over GF(2^m). For a RS code, we have

- $N = q - 1$ (codeword length)
- $N - K = 2T$ (number of parity digits)
- $D_{min} = 2T + 1$ (minimum distance)
- $T \equiv$ number of digits that can be corrected.

For example, for $m = 4$, we have $q = 16$ and $N = 15$. A RS code of codeword length N exists if N divides into $2^m - 1$. Thus, a $(N, N - 2T)$ RS code can be constructed for any $T \leq \lfloor N/2 \rfloor$.

For a given m, irreducible polynomial $p(p)$, and primitive element α (i.e. $\underline{\alpha}$ or $\alpha(p)$), the RS code generator polynomial is

$$
\begin{aligned}
g(p) &= (p-\alpha)(p-\alpha^2)\cdots(p-\alpha^{2T}) = (p+\alpha)(p+\alpha^2)\cdots(p+\alpha^{2T}) && (129)\\
&= p^{2T} + g_{2T-1}p^{2T-1} + \cdots + g_1 p + g_0 && (130)
\end{aligned}
$$

where $\alpha^j;\ j = 1, 2, \cdots 2T$, i.e. powers of α, are the roots of $g(p)$. Note that $-\alpha^m = \alpha^m$ since these elements represent $GF(2)$ vectors. Since these are also roots of $p^N + 1$, $g(p)$ divides

$p^N + 1$. $g(p)$ is therefore the generating polynomial of a GF(2^m) cyclic code. The generator polynomial is of degree $2T = N - K$. In systematic form, this code will have $2T$ parity check symbols.

Example 7.23: Determine the generator polynomial for the $(7, 5)$ RS code in GF(2^3) with $p(p) = p^3 + p + 1$, $\underline{\alpha} = [010]$ (i.e. $\alpha = \mathbf{2}$ in octal) and $T = 1$.

Solution: First note that the given $p(p)$ is irreducible, $N = 7$ divides into $2^3 - 1 = 7$, the given $\alpha(p) = p$ is a primitive element, and $K = 5 = N - 2T$. So this is a valid RS code description. To determine the generator we need the following element table:

$$\begin{array}{lcll} \alpha = p & \rightarrow & [010] & \rightarrow \ 2 \\ \alpha^2 = p^2 & \rightarrow & [100] & \rightarrow \ 4 \\ \alpha^3 = p + 1 & \rightarrow & [011] & \rightarrow \ 3 \\ \alpha^4 = p^2 + p & \rightarrow & [110] & \rightarrow \ 6 \\ \alpha^5 = p^2 + p + 1 & \rightarrow & [111] & \rightarrow \ 7 \\ \alpha^6 = p^2 + 1 & \rightarrow & [101] & \rightarrow \ 5 \\ \alpha^7 = 1 & \rightarrow & [001] & \rightarrow \ 1 \end{array} \tag{131}$$

The generator polynomial is

$$g(p) = (p - \alpha)(p - \alpha^2) \tag{132}$$

$$= p^2 - (\alpha + \alpha^2)p + \alpha^3 = p^2 + \alpha^4 p + \alpha^3 \tag{133}$$

$$= p^2 + 6p + 3 \ . \tag{134}$$

In the last line of this equation, the coefficients are represented in octal.

With $K = 5$ octal information digits, there are $8^5 = 32,768$ information vectors and corresponding codewords, representing all combinations of 15 bits. Since there are 7 three bit symbols in a codeword, there are 2,097,152 binary vectors from which the codewords are selected.

Note that the RS code formulation allows for a compact way to represent long codewords. To generate codeword vectors, we can multiply the generator polynomial $g(p)$ with the message polynomial $x(p)$ to generate the codeword polynomial $c(p)$. However, as was the case for general binary cyclic codes, this approach to encoding does not generate systematic codewords. Systematic codewords can be generated by generalizing the polynomial remainder procedure used for binary cyclic codes. The following procedure generates systematic codewords:

$$r(p) = p^{2T} \cdot x(p) \quad mod \ g(p) \tag{135}$$

$$c(p) = p^{2T} \cdot x(p) + r(p) \ . \tag{136}$$

Then the codeword vector is

$$\mathbf{C} = [\mathbf{X}, \ \mathbf{R}] \tag{137}$$

where $\mathbf{X}$ is the information vector and $\mathbf{R}$ is the vector of remainder polynomial $r(p)$ coefficients.

Example 7.24: Continuing Example 7.23, determine the systematic codewords for the following information vectors:

1. $\mathbf{X} = [0\ 0\ 0\ 0\ 1]$
2. $\mathbf{X} = [0\ 0\ 0\ 0\ 3]$
3. $\mathbf{X} = [0\ 0\ 1\ 0\ 3]$
4. $\mathbf{X} = [0\ 0\ 0\ 7\ 3]$.

Solution: For this $T = 1$ error correcting RS code, the remainder polynomial is of the form $r(p) = r_1 p + r_0$, with the corresponding coefficient vector $\mathbf{R} = [r_1,\ r_0]$. The problem is to find the $\mathbf{R}$ for each $\mathbf{X}$, and then form the codeword $\mathbf{C} = [\mathbf{X},\ \mathbf{R}]$.

1. For $\mathbf{X} = [0\ 0\ 0\ 0\ 1]$, we have

$$\begin{aligned} r(p) &= p^2\ x(p)\ \ mod\ \{p^2 + \alpha^4 p + \alpha^3\} && (138)\\ &= p^2\ \ mod\ \{p^2 + \alpha^4 p + \alpha^3\}\ . && (139)\end{aligned}$$

   Performing the long division required for the *mod* operator, we get

$$r(p) = \alpha^4 p + \alpha^3\ , \qquad (140)$$

   so that

$$c(p) = p^2\ x(p)\ + r(p) = p^2 + \alpha^4 p + \alpha^3\ , \qquad (141)$$

   or equivalently

$$\mathbf{C} = [0\ 0\ 0\ 0\ 1\ 6\ 3]\ . \qquad (142)$$

2. For $\mathbf{X} = [0\ 0\ 0\ 0\ 3]$, we have

$$\begin{aligned} r(p) &= p^2\ x(p)\ \ mod\ \{p^2 + \alpha^4 p + \alpha^3\} && (143)\\ &= \alpha^3 p^2\ \ mod\ \{p^2 + \alpha^4 p + \alpha^3\}\ . && (144)\end{aligned}$$

   Performing the long division required for the *mod* operator, we get

$$r(p) = p + \alpha^6\ , \qquad (145)$$

   so that

$$\mathbf{C} = [0\ 0\ 0\ 0\ 3\ 1\ 5]\ . \qquad (146)$$

3. For $\mathbf{X} = [0\ 0\ 1\ 0\ 3]$, we have

$$\begin{aligned} r(p) &= p^2\ x(p)\ \ mod\ \{p^2 + \alpha^4 p + \alpha^3\} && (147)\\ &= p^4 + \alpha^3 p^2\ \ mod\ \{p^2 + \alpha^4 p + \alpha^3\}\ . && (148)\end{aligned}$$

   Performing the long division required for the *mod* operator, we get

$$r(p) = \alpha^4 p + \alpha^4\ , \qquad (149)$$

   so that

$$\mathbf{C} = [0\ 0\ 1\ 0\ 3\ 6\ 6]\ . \qquad (150)$$

4. For $\mathbf{X} = [0\ 0\ 0\ 7\ 3]$, we have

$$\begin{aligned} r(p) &= p^2\ x(p)\ \ mod\ \{p^2 + \alpha^4 p + \alpha^3\} && (151)\\ &= \alpha^5 p^3 + \alpha^3 p^2\ \ mod\ \{p^2 + \alpha^4 p + \alpha^3\}\ . && (152) \end{aligned}$$

Performing the long division required for the *mod* operator, we get

$$r(p) = \alpha^4 p + \alpha\ \ , \qquad (153)$$

so that

$$\mathbf{C} = [0\ 0\ 0\ 7\ 3\ 6\ 2]\ \ . \qquad (154)$$

Note that the Matlab communications toolbox has GF(2^m) and Reed Solomon code functions that can be used to verify and extend these examples.

7.7.3 Encoding Reed-Solomon (RS) Codes

An efficient RS code encoder can be developed as an extension of the feedback shift register encoder for systematic binary cyclic codes described in Subsection 7.5.7. The extension simply involves substituting the GF(2) adders, multipliers and shift registers with their GF(2^m) counterparts. Figure 67 illustrates this encoder.

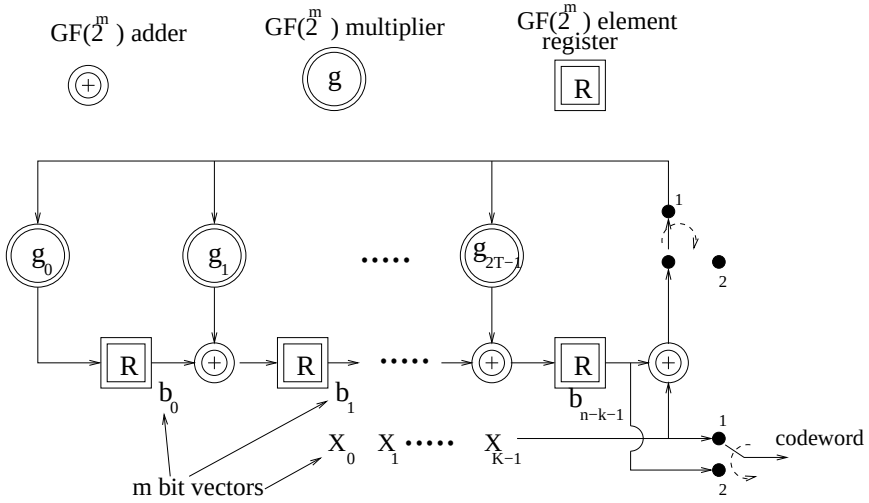


Figure 67: An encoder for systematic GF(2^m) Reed-Solomon codes.

Alternatively, RS code codewords can be calculated directly from the $2T$ order generator polynomial $g(p)$, by convolving its $2T+1$-dimensional coefficient vector $\mathbf{g}$ with the $K = (N - 2T)$-dimensional information vector $\mathbf{X}$. This is equivalent to post multiplying $\mathbf{X}$ with the $2T \times N$-dimensional RS code generator matrix $\mathbf{G}$. This approach does not in general provide a systematic form of the RS code.

7.7.4 Decoding Reed-Solomon (RS) Codes

A RS code codeword can be represented in binary form. That is, for a code in GF(2^m), each codeword symbol can be represented by m bits. The corresponding binary codeword can then be transmitted using any digital modulation scheme. Since large binary codewords with good d_{min} and code weight wn_j characteristics can be generated using RS codes, these codes can be used in digital communications systems to provide very good bit error probability (BEP) performance at reasonably high code rate R_c.

Moreover, since T codeword symbol errors can be corrected, RS codes provide effective protection against burst errors (i.e. a burst of receiver bit errors). For example, for a $T = 2$ symbol error correction RS code in GF(2^5), two adjacent symbols or equivalently up to 10 successive bit errors can be tolerated without a codeword error. Burst errors are common, for example, in wireless communications systems because of channel fading, and in CD data storage applications because of disc defects.

What remains to be established is that RS codes can be efficiently decoded. This is the objective of this Subsection.

A Reed Solomon Decoding Structure

We now describe a basic approach to RS code decoding, and explain one set of algorithms for implementing this approach. For a more comprehensive discussion of RS code decoding approaches and algorithms, refer to Lin and Costello [1].

Recall that we previously described the following three step decoding procedure for binary cyclic codes (systematic or not):

1. syndrome calculation;
2. error pattern identification based on the syndrome table; and
3. error correction.

A RS code can be decoded using essentially the same basic procedure, with the exception that since the code is nonbinary, an additional step is required to enumerate the values of the errors. Figure 68 illustrates this decoding procedure.

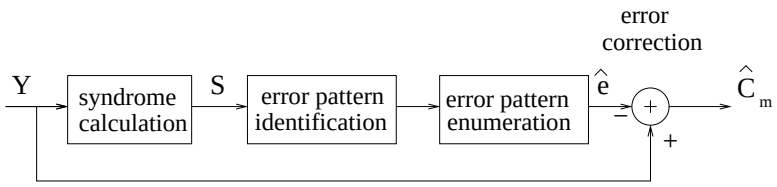


Figure 68: A decoder block diagram for a RS code.

Syndrome Calculation

The decoder described here assumes codewords have been generated as

$$\mathbf{C} \;=\; \mathbf{X}\,\mathbf{G} \tag{155}$$

where $\mathbf{G}$ corresponds to the RS generator polynomial

$$\begin{aligned} g(p) &= (p-\alpha)(p-\alpha^2)\;\cdots\;(p-\alpha^{2T}) &(156)\\ &= p^{2T} \;+\; g_{2T-1}p^{2T-1} \;+\; \cdots \;+\; g_1 p \;+\; g_0 &(157)\end{aligned}$$

where $\alpha^i;\ i = 1, 2, \cdots, 2T$ are the roots. Recall that for any linear block code the parity check matrix $\mathbf{H}$ has the property $\mathbf{G}\,\mathbf{H}^T \;=\; \mathbf{0}_{2T,N-2T}$. Thus, the rows of $\mathbf{H}$ are determined from the roots of $g(p)$. Specifically,

$$\mathbf{H} \;=\; \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{N-1} \\ 1 & \alpha^2 & \alpha^{2\cdot 2} & \cdots & \alpha^{2\cdot(N-1)} \\ 1 & \alpha^3 & \alpha^{3\cdot 2} & \cdots & \alpha^{3\cdot(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{2T} & \alpha^{2T\cdot 2} & \cdots & \alpha^{2T\cdot(N-1)} \end{bmatrix} \quad . \tag{158}$$

Given the N-dimensional GF(2^m) hard decision received data vector $\mathbf{Y}$, the syndrome vector $\mathbf{S}$ is computed as

$$\mathbf{S} \;=\; [s_1,\; s_2, \cdots,\; s_{2T}] \;=\; \mathbf{Y}\,\mathbf{H}^T \quad . \tag{159}$$

In polynomial form, we represent the data vector as

$$y(p) \;=\; y_{N-1}p^{N-1} \;+\; y_{N-2}p^{N-2} \;+\; \cdots \;+\; y_1 p \;+\; y_0 \quad . \tag{160}$$

Given the structure of $\mathbf{H}$ indicated by Eq (158), we have that the syndrome values (the elements of $\mathbf{S}$) can be computed as

$$s_i \;=\; y(\alpha^i) \;; \quad i = 1, 2, \cdots, 2T \quad . \tag{161}$$

That is, the syndrome can be computed by evaluating the received data polynomial $y(p)$ at the RS code generator polynomial root values $\alpha_i; = 1, 2, \cdots, 2T$.

Error Pattern Identification

For a transmitted codeword, the received data polynomial is

$$y(p) \;=\; c(p) \;+\; e(p) \tag{162}$$

where $c(p)$ is the actual codeword polynomial and $e(p)$ is the hard decision error polynomial. As with the error for any linear block code, this error will in general consist of an undetectable and a detectable component. That is

$$e(p) \;=\; e_u(p) + e_d(p) \;=\; q(p)\; g(p) \;+\; e_d(p) \tag{163}$$

where $e_u(p) = \; q(p)\; g(p)$ looks like some codeword polynomial and $e_d(p)$ will be characterized by the syndrome. It is this detectable part of the error, $e_d(p)$, which is subtracted from $y(p)$ to generate the ML codeword polynomial estimate $\hat{c}_m(p)$.

In terms of this detectable error $e_d(p)$, the syndrome values are

$$s_i \;=\; y(\alpha^i) \;=\; c(\alpha^i) \;+\; q(\alpha^i)\; g(\alpha^i) \;+\; e_d(\alpha^i) \;=\; \; e_d(\alpha^i) \;; \;\; i = 1, 2, \cdots, 2T \quad . \tag{164}$$

Let $e_d(p) \;=\; e_{N-1}p^{N-1} \;+\; e_{N-2}p^{N-2} \;+\; \cdots \;+\; e_1 p \;+\; e_0$. Assume that there are ν errors at locations $j_i; \;\; i = 1, 2, \cdots, \nu$ (i.e. $e_j \neq 0; \;\; j = j_i; \;\; i = 1, 2, \cdots, \nu$). Then we have the following $2T$ equations, one for each syndrome value:

$$\begin{aligned}
s_1 \;&=\; e_{j_1}\alpha^{j_1} \;+\; e_{j_2}\alpha^{j_2} \;+\; \cdots \;+ e_{j_\nu}\alpha^{j_\nu} && (165)\\
s_2 \;&=\; e_{j_1}\alpha^{2\cdot j_1} \;+\; e_{j_2}\alpha^{2\cdot j_2} \;+\; \cdots \;+ e_{j_\nu}\alpha^{2\cdot j_\nu} && (166)\\
&\;\;\vdots && (167)\\
s_{2T} \;&=\; e_{j_1}\alpha^{2T\cdot j_1} \;+\; e_{j_2}\alpha^{2T\cdot j_2} \;+\; \cdots \;+ e_{j_\nu}\alpha^{2T\cdot j_\nu} \quad . && (168)
\end{aligned}$$

The error pattern identification task is to determine the unknown error locations $j_i; \;\; i = 1, 2, \cdots, \nu$. Given these locations, the error enumeration task is to determine the $e_j; \;\; j = j_i; \;\; i = 1, 2, \cdots, \nu$.

Let $\beta_i = \alpha^{j_i}; \;\; i = 1, 2, \cdots, \nu$ an $\delta_i = e_{j_i}; \;\; i = 1, 2, \cdots, \nu$. The T syndrome equations can be rewritten as

$$\begin{aligned}
s_1 \;&=\; \delta_1\beta_1 \;+\; \delta_2\beta_2 \;+\; \cdots \;+ \delta_\nu\beta_\nu && (169)\\
s_2 \;&=\; \delta_1\beta_1^2 \;+\; \delta_2\beta_2^2 \;+\; \cdots \;+ \delta_\nu\beta_\nu^2 && (170)\\
&\;\;\vdots && (171)\\
s_{2T} \;&=\; \delta_1\beta_1^{2T} \;+\; \delta_2\beta_2^{2T} \;+\; \cdots \;+ \delta_\nu\beta_\nu^{2T} \quad . && (172)
\end{aligned}$$

Define the *error locator polynomial* as

$$\sigma(p) \;=\; (1-\beta_1 p)(1-\beta_2 p)\cdots(1-\beta_\nu p) \quad . \tag{173}$$

The error location indicators, $\beta_i;\ i = 1,2,\cdots,\nu$ are the inverses of the roots of $\sigma(p)$. Berlekamp's algorithm can be used to generate this polynomial. See, for example, Lin and Costello [1], pp. 242-243 for a description of this algorithm. Given $\sigma(p)$, the roots can be computed using the Chien's algorithm referred to in Lin and Costello [1], p. 244.

Error Enumeration

Given the error location indicators, $\beta_i;\ i = 1,2,\cdots,\nu$, the error enumerations $\delta_i = e_{j_i};\ i = 1,2,\cdots,\nu$. Forney (see Lin and Costello [1] pp. 245-247) derived an algorithm for error enumeration. Let

$$z_0(p) \;=\; s_1 + (s_2+\sigma_1 s_1)p + (s_3+\sigma_1 s_2+\sigma_2 s_1)p^2 + \cdots + (s_\nu+\sigma_1 s_{\nu-1}+\cdots+\sigma_{\nu-1}s_1)p^{\nu-1} \quad . \tag{174}$$

and

$$\sigma^{'}(\beta_i^{-1}) \;=\; -\beta_i \prod_{k=1,k\neq i}^{\nu} (1-\beta_k\beta_i^{-1}) \quad . \tag{175}$$

then the Forney error value evaluator is

$$\delta_i \;=\; \frac{-z_o(\beta_i^{-1})}{\sigma^{'}(\beta_i^{-1})} \quad . \tag{176}$$

Example 7.25: Consider the GF(2^5) field with multiplication defined by irreducible polynomial $p(p) = p^5 + p^2 + 1$ and primitive element $\alpha(p) = p$.

1. For this field, determine the generator polynomial $g(p)$ for the Reed-Solomon code that can correct $T = 1$ error.
2. Using the systematic encoder for this code, determine the codeword polynomial $c(p)$ for the information polynomial $x(p) = \alpha^8 p^2 + \alpha^5$. What is the codeword vector $\mathbf{C}$?
3. For either a systematic or nonsystematic encoder, which uses a generator matrix $\mathbf{G}$ derived from the generator polynomial $g(p)$, describe the parity check matrix $\mathbf{H}$. For received hard-decision data polynomial $y(p) = p^{16} + \alpha^2 p^{11}$, determine the syndrome, the detectable error polynomial $e_d(p)$, and the ML codeword estimate polynomial $\hat{c}(p)$.

The following GF(2^5) element table is required to solve this problem.

0	00000	α^{16}	11011
α	00010	α^{17}	10011
α^2	00100	α^{18}	00011
α^3	01000	α^{19}	00110
α^4	10000	α^{20}	01100
α^5	00101	α^{21}	11000
α^6	01010	α^{22}	10101
α^7	10100	α^{23}	01111
α^8	01101	α^{24}	11110
α^9	11010	α^{25}	11001
α^{10}	10001	α^{26}	10111
α^{11}	00111	α^{27}	01011
α^{12}	01110	α^{28}	10110
α^{13}	11100	α^{29}	01001
α^{14}	11101	α^{30}	10010
α^{15}	11111	α^{31}	00001

Solution: Note that $q = 2^m = 2^5 = 32$, so that $N = q-1$. Also, $K = N-2T = 29$, so this is a (31,29) RS code with rate $R_c = \frac{29}{31}$.

1. For a $T = 1$ error correcting RS code,

$$g(p) \;=\; (p+\alpha)(p+\alpha^2) \;=\; p^2+(\alpha+\alpha^2)p+\alpha^3 \;=\; p^2+\alpha^{19}p+\alpha^3 \;\;. \quad (177)$$

2. For the systematic codeword, we need

$$r(p) \;=\; p^2\; x(p) \;\; mod\; g(p) \;=\; \alpha^8 p^4+\alpha^5 p^2 \;\; mod\; g(p) \quad . \quad (178)$$

The long division yields $r(p) = \alpha^{28}p + \alpha^{18}$. Thus

$$c(p) \;=\; \alpha^8 p^4 \;+\; \alpha^5 p^2 \;+\; \alpha^{28} p \;+\; \alpha^{17} \quad (179)$$

or equivalently

$$\mathbf{C} \;=\; [0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,\alpha^8\;0\;\alpha^5\;\alpha^{28}\;\alpha^{17}] \;\;. \quad (180)$$

3. For this (31,29) code, there will be $2T = 2$ syndrome values which can be used to identify (locate and enumerate) a single error to correct. The detectable error has polynomial $e_d(p) = e_j p^j$, where j indicates the error location and $e_j = \alpha^k$ is the error value. The problem is to determine j and k.

For the nonsystematic encoder, and the hard decision received data polynomial $y(p) = p^{16} + \alpha^2 p^{11}$, the syndromes are

$$s_1 \;=\; y(\alpha) \;=\; \alpha^{16}+\alpha^2\alpha^{11} \;=\; \alpha^{16}+\alpha^{13} \;=\; \alpha^{11} \quad (181)$$

$$s_2 \;=\; y(\alpha^2) \;=\; \alpha^{32}+\alpha^2\alpha^{22} \;=\; \alpha^{32}+\alpha^{24} \;=\; \alpha^{13} \quad . \quad (182)$$

Since $y(p) = \hat{c}(p) + e_d(p)$, where $\hat{c}(p)$ is a codeword, and $\hat{c}(\alpha^m) = 0;\ m = 1, 2$, we have that

$$\begin{aligned} s_1 &= \alpha^{11} = \alpha^k \alpha^j & (183) \\ s_2 &= \alpha^{13} = \alpha^k \alpha^{2j} \quad . & (184) \end{aligned}$$

Solving for k and j, we have $k = 9$ and $j = 2$. So, $e_d(p) = \alpha^9 p^2$, and our decoded codeword polynomial is

$$\hat{c}(p) = y(p) + e_d(p) = p^{16} + \alpha^2 p^{11} + \alpha^9 p^2 \quad . \qquad (185)$$

7.8 Techniques for Constructing More Complex Block Codes

In this Subsection we introduce several techniques used, in conjunction with the basic block codes we've just covered, in the construction of more complex block codes with attractive performance characteristics. The common theme of these techniques is to use basic block codes as constituent codes of larger codes which have good minimum distance characteristics and/or disperse the effects of burst errors. In consideration of the decoding of these larger codes, we will touch on iterative decoding. It is the search for better codes, based on these encoding techniques and iterative decoding, that has lead to turbo codes and their iterative decoding, which is one of the two main topics of Section 9 of this Course.

7.8.1 Product Codes

Consider two systematic binary codes, (n_1, k_1) code $\mathcal{C}_1$ with minimum distance $d_{1,min}$ and (n_2, k_2) code $\mathcal{C}_2$ with minimum distance $d_{2,min}$. A $(n_1 n_2, k_1 k_2)$ code $\mathcal{C}$ with minimum distance $d_{1,min} d_{2,min}$ is constructed as follows. Let

$$\mathbf{X}_m = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,k_1} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k_1} \\ x_{3,1} & x_{3,2} & \cdots & x_{3,k_1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{k_2,1} & x_{k_2,2} & \cdots & x_{k_2,k_1} \end{bmatrix} \tag{186}$$

be a $k_2 \times k_1$ matrix of information bits. If we first apply code $\mathcal{C}_1$ to the rows, generating

$$\mathbf{C}'_m = \left[\begin{array}{cccc|cccc} x_{1,1} & x_{1,2} & \cdots & x_{1,k_1} & p^{(1)}_{1,1} & p^{(1)}_{1,2} & \cdots & p^{(1)}_{1,n_1-k_1} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k_1} & p^{(1)}_{2,1} & p^{(1)}_{2,2} & \cdots & p^{(1)}_{2,n_1-k_1} \\ x_{3,1} & x_{3,2} & \cdots & x_{3,k_1} & p^{(1)}_{3,1} & p^{(1)}_{3,2} & \cdots & p^{(1)}_{3,n_1-k_1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{k_2,1} & x_{k_2,2} & \cdots & x_{k_2,k_1} & p^{(1)}_{k_2,1} & p^{(1)}_{k_2,2} & \cdots & p^{(1)}_{k_2,n_1-k_1} \end{array}\right] \tag{187}$$

We then apply code $\mathcal{C}_2$ to the columns, generating the code-matrix

$$\mathbf{C}_m = \left[\begin{array}{cccc|cccc} x_{1,1} & x_{1,2} & \cdots & x_{1,k_1} & p^{(1)}_{1,1} & p^{(1)}_{1,2} & \cdots & p^{(1)}_{1,n_1-k_1} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k_1} & p^{(1)}_{2,1} & p^{(1)}_{2,2} & \cdots & p^{(1)}_{2,k_1} \\ x_{3,1} & x_{3,2} & \cdots & x_{3,k_1} & p^{(1)}_{3,1} & p^{(1)}_{3,2} & \cdots & p^{(1)}_{3,n_1-k_1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{k_2,1} & x_{k_2,2} & \cdots & x_{k_2,k_1} & p^{(1)}_{k_2,1} & p^{(1)}_{k_2,2} & \cdots & p^{(1)}_{k_2,n_1-k_1} \\ \hline p^{(2)}_{1,1} & p^{(2)}_{1,2} & \cdots & p^{(2)}_{1,k_1} & p_{1,1} & p_{1,2} & \cdots & p_{1,n_1-k_1} \\ p^{(2)}_{2,1} & p^{(2)}_{2,2} & \cdots & p_{2,k^{(2)}_1} & p_{2,1} & p_{2,2} & \cdots & p_{2,n_1-k_1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x^{(2)}_{n_2-k_2,1} & x^{(2)}_{n_2-k_2,2} & \cdots & x^{(2)}_{n_2-k_2,k_1} & p_{n_2-k_2,1} & p_{n_2-k_2,2} & \cdots & p_{n_2-k_2,n_1-k_1} \end{array}\right] = \begin{bmatrix} \mathbf{X}_m & \mathbf{P}^{(1)}_m \\ \mathbf{P}^{(2)}_m & \mathbf{P}_m \end{bmatrix}, \tag{188}$$

where $\mathbf{P}_{1,m}$ is the $k_2 \times (n_1 - k_1)$-dimensional code $\mathcal{C}_1$ parity check matrix, $\mathbf{P}_{2,m}$ is the $(n_2 - k_2) \times k_1$-dimensional code $\mathcal{C}_2$ parity check matrix, and $\mathbf{P}_m$ is the $(n - 2 - k_2) \times (n - 1 - k_1)$-dimensional parity-checks of parity checks matrix.

Given codes $\mathcal{C}_1$ and $\mathcal{C}_2$, there will be $2^{k_1 k_2}$ code-matrices out of a possible $2^{n_1 n_2}$ $n_2 \times n_1$ binary matrices. Upon matched filter reception of a noisy code-matrix, we have a $n_2 \times n_1$-dimensional data matrix

$$\mathbf{R} \;=\; \mathbf{C}_m \;+\; \mathbf{E}^{'} \quad . \tag{189}$$

A soft decision decoder would have to compare each possible $\mathbf{C}_m$ to $\mathbf{R}$, which could be impractical. Consider instead the hard decision data matrix

$$\mathbf{Y} \;=\; \mathbf{C}_m \;+\; \mathbf{E} \quad . \tag{190}$$

Although again an optimum decoder could be impractical, an effective suboptimum decoding procedure involves: 1) first decoding the columns of $\mathbf{Y}$ using a code $\mathcal{C}_2$ decoder to form $\mathbf{Y}^{'}$; and 2) then decoding the rows of $\mathbf{Y}^{'}$ using a code $\mathcal{C}_1$ decoder to form an information matrix estimate $\hat{\mathbf{X}}_m$. Several iterations of row/column decoding will further improve results. In general, *iterative decoding* between constituent decoders of a large code composed of several constituent codes can be a practical, effective suboptimal approach. Iterative decoding is a key component of both turbo and LDPC channel coding systems.

Although the description above is for binary product codes, the basic development is applicable for codes in GF(q).

7.8.2 Interleaving

An *interleaver* is a bit or symbol shuffler. The two basic purposes of an interleaver are: 1) to spread burst errors, as illustrated below; and 2) to extend codeword lengths in a concatenated coder (as explained later). Common causes of burst errors include: fading channels and inner coder decoding (again, see concatenated codes below).

There are many types of interleavers. In Figure 69(a) we show a simple block interleaver. As shown, bits (or symbols) are read into a $m \times n$ register row-by-row, and read out column-by-column. Denote this interleaving operation as Π. As illustrated in Figure 69(c), the deinterleaver, denoted Π^{-1}, inverts the interleaver operation.

As an example, consider a sequence of bits $b_i; i = 1, 2, \cdots, nm$ at the output of a block coder. As illustrated in Figure 69(b), these bits are interleaved to generate output $c_i; i = 1, 2, \cdots, nm$. These interleaved bits are transmitted over a channel and hard decision decoding is performed at the channel output. Say the channel causes a burst of errors. Without interleaving, the high concentration of errors over the short burst duration would cause codeword decoding errors, even if the coder has good error correction capabilities. However, using the interleaver, after deinterleavng the errors are distributed throughout the mn block, so that error correction coding can better handle them.

One example of the effective use of interleavers is turbo coding, which will be overviewed in Section 9 (after the discussion on convolutional codes). For turbo codes, $mn \geq 65,536$ is not uncommon.

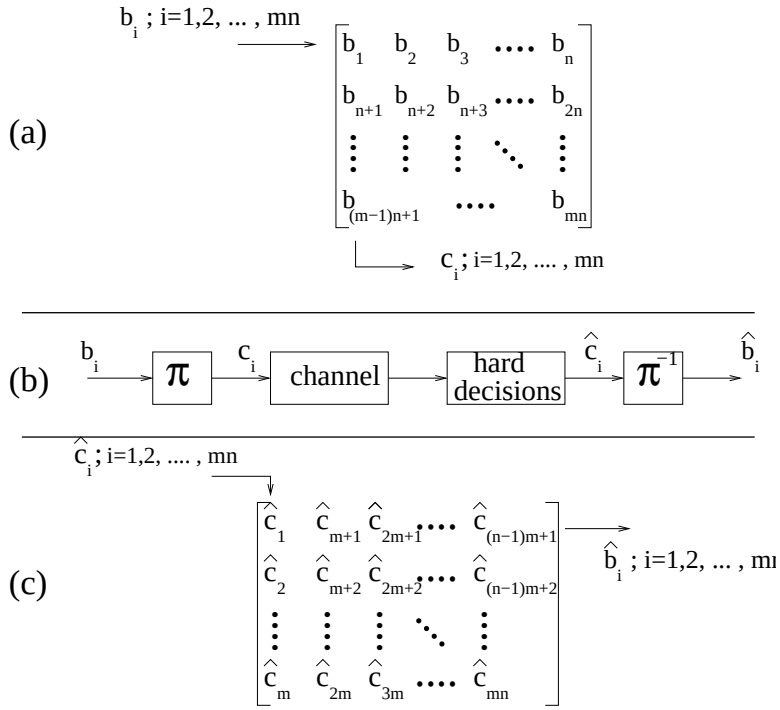


Figure 69: (a) an $m \times n$ block interleaver; (b) its use with a burst error channel; (c) an $m \times n$ block deinterleaver.

7.8.3 Concatenated Block Codes

Block codes can be concatenated either in series or in parallel, and an interleaver may be inserted between them. Similar to product coding, concatenation can have the effect of creating a composite coder with very long codeword lengths and very good minimum distance characteristics.

Serial Concatenation

A Serial Concatenated Block Code (SCBC) is exemplified in Figure 70. As illustrated, the outer code is a (N, K) nonbinary block code (with each symbol representing $\log_2 q$ bits) and the inner code is a (n, k) binary block code. With this figure, the idea is that k bits into the inner code will represent one symbol out of the outer code, so $k = \log_2 q$. The inner code serves to clean up the received outer code codeword elements prior to outer code decoding. The concatenated code takes kK information bits and generates a binary codeword of length nN. So the concatenated code is a (nN, kK) block code. For inner and outer code minimum distances D_{min} and d_{min}, the concatenated code has minimum distance $d_{min} \cdot D_{min}$.

Since lengths of the 2^{kK} different codewords are long, complexity of the optimum hard or soft-decision decoder is high. Notice in Figure 70 that instead of using one decoder to optimally decode the binary codewords of length nN, a two stage decoder is used. The inner and outer codes are decoded separately. This is a suboptimum, but practical, decoding scheme. As with product codes, several iterations between the two constituent decoders can improve performance.

Figure 70: A serial concatenated block code.

Serial Concatenation with Interleaving

Figure 71 illustrates a serial concatenated block code (SCBC) with interleaving between encoders. The outer coder generates m blocks of p bit codewords, each codeword representing k bits. The mp bits from the m codewords are interleaved. The resulting mp bits are formed into m blocks of p bits, and each block is encoded with an (n, p) block code. Altogether, mk bits are coded into mn bits, for a rate $R_c = \frac{k}{n}$ code. Effective codeword lengths are mn-dimensional, which can be very large.

Figure 71 also shows a decoder structure for this interleaved SCBC. The idea is a straightforward extension of the serial decoder in Figure 71, with the deinterleaver added for obvious reasons. Consider that it is possible to feed back information from the outer decoder, through an interleaver, for use as *prior* information for a second pass through the inner decoder. This suggests a very effective *iterative* decoding approach. This code concatenation with interleaving, along with this iterative decoding possibility, forms the basic idea behind *turbo coding & decoding*. We will visit this topic again, after we cover convolutional codes, and learn something about decoders which can be used to generate information that can be used by another decoder as prior information.

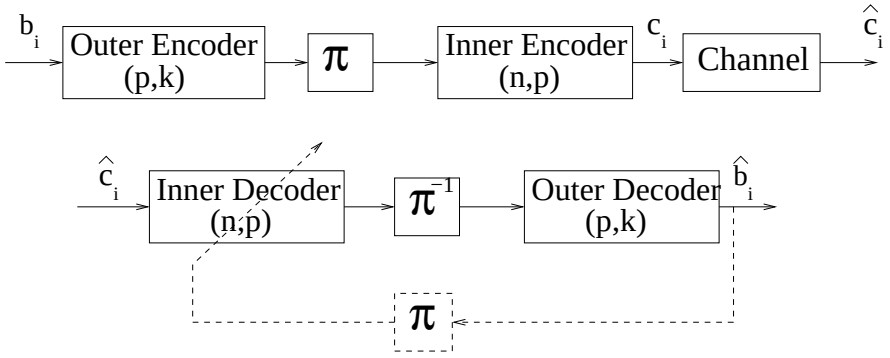


Figure 71: A Serial Concatenated Block Code (SCBC) with interleaving.

Parallel Concatenation with Interleaving & Puncturing

Figure 72 illustrates a parallel concatenated block code (PCBC) with interleaving. Without puncturing, the code rate is $R_c = \frac{mk}{m(n_1+n_2-k)}$, where $n_1, n_2 > k$. We will discuss decoding parallel concatenated codes later, when we discuss turbo codes. Note that it is possible to increase the code rate by *puncturing* (i.e. throwing away, not transmitting) some of the parity bits generated by the two block codes. This, of course, may degrade performance.

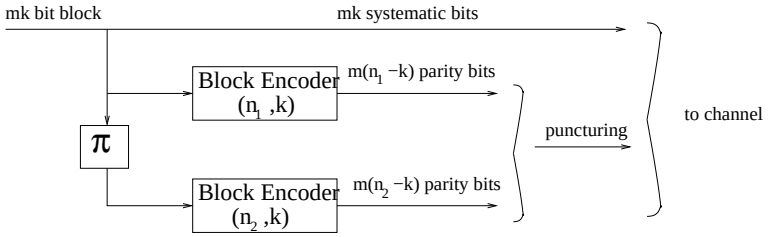


Figure 72: A Parallel Concatenated Block Code (PCBC) with interleaving.

References

[1] S. Lin and D. J. Costello, *Error Control Coding 2-nd ed.* Pearson Prentice Hall, 2004.

[2] J. Proakis and M. Salehi, *Digital Communications 5-th ed.* McGraw Hill, 2008.