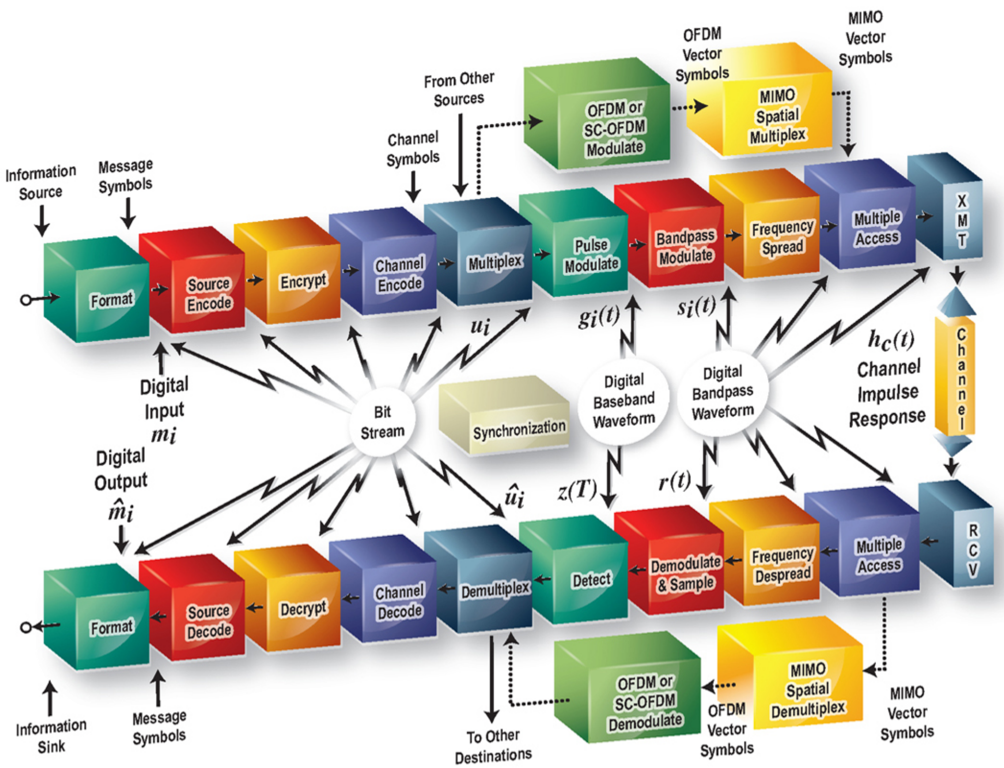
THIRD EDITION

Digital Communications

Fundamentals and Applications



Bernard Sklar and fred harris

DIGITAL COMMUNICATIONS

Fundamentals and Applications

Third Edition

Bernard Sklar

fred harris

Pearson

by using Equation (6.37); the entries of any given row (coset) of the standard array have the same syndrome. The correction of a corrupted codeword proceeds by computing its syndrome and locating the error pattern that corresponds to that syndrome. Finally, the error pattern is modulo-2 added to the corrupted codeword yielding the corrected output. Equation (6.49), repeated below, indicates that error-detection and error-correction capabilities can be traded, provided that the distance relationship

$$d_{\min} \geq \alpha + \beta + 1$$

prevails. Here, α represents the number of bit errors to be corrected, β represents the number of bit errors to be detected, and $\beta \geq \alpha$. The trade-off choices available for the (8, 2) code example are as follows:

Detect (β)	Correct (α)
2	2
3	1
4	0

This table shows that the (8, 2) code can be implemented to perform only error correction, which means that it first detects as many as $\beta = 2$ errors and then corrects them. If some error correction is sacrificed so that the code will only correct single errors, then the detection capability is increased so that all $\beta = 3$ errors can be detected. Finally, if error correction is completely sacrificed, the decoder can be implemented so that all $\beta = 4$ errors can be detected. In the case of error detection only, the circuitry is very simple. The syndrome is computed, and an error is detected whenever a nonzero syndrome occurs.

For correcting single errors, the decoder can be implemented with gates [4], similar to the circuitry in Figure 6.11, where a received code vector **r** enters at two places. In the top part of the figure, the received digits are connected to exclusive-OR gates, which yield the syndrome. For any given received vector, the syndrome is obtained from Equation (6.35) as

$$\mathbf{S}_i = \mathbf{r}_i \mathbf{H}^T \qquad i = 1, \cdots, 2^{n-k}$$

Using the $\mathbf{H}^T$ values for the (8, 2) code, the wiring between the received digits and the exclusive-OR gates in a circuit similar to the one in Figure 6.11 must be connected to yield

$$\mathbf{S}_i = [r_1 \; r_2 \; r_3 \; r_4 \; r_5 \; r_6 \; r_7 \; r_8] \begin{bmatrix} 1\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,0\,1 \\ 0\,0\,1\,1\,1\,1 \\ 1\,1\,1\,1\,0\,0 \end{bmatrix}$$

Each of the s_j digits ($j = 1, \ldots, 6$) making up syndrome $\mathbf{S}_i$ ($i = 1, \ldots, 64$) is related to the input-received code vector in the following way:

$$s_1 = r_1 + r_8 \qquad s_2 = r_2 + r_8 \qquad s_3 = r_3 + r_7 + r_8$$
$$s_4 = r_4 + r_7 + r_8 \qquad s_5 = r_5 + r_7 \qquad s_6 = r_6 + r_7$$

To implement a decoder circuit similar to Figure 6.11 for the (8, 2) code necessitates that the eight received digits be connected to six modulo-2 adders yielding the syndrome digits as described above. Additional modifications to the figure need to be made accordingly.

If the decoder is implemented to correct only single errors—that is, $\alpha = 1$ and $\beta = 3$—this is tantamount to drawing a line under coset 9 in Figure 6.14, and error correction takes place only when one of the eight syndromes associated with a single error appears. The decoding circuitry (similar to Figure 6.11) then transforms the syndrome to its corresponding error pattern. The error pattern is then modulo-2 added to the "potentially" corrupted received vector, yielding the corrected output. Additional gates are needed to test for the case in which the syndrome is nonzero and there is no correction designed to take place. For single-error correction, such an event happens for any of the syndromes numbered 10 through 64. This outcome is then used to indicate an error detection.

If the decoder is implemented to correct single and double errors, which means that $\beta = 2$ errors are detected and then corrected, this is tantamount to drawing a line under coset 37 in the standard array of Figure 6.14. Even though this (8, 2) code is capable of correcting some combination of triple errors corresponding to the coset leaders 38 through 64, a decoder is most often implemented as a *bounded distance* decoder, which means that it corrects all combinations of errors up to and including t errors but no combinations of errors greater than t. The unused error-correction capability can be applied toward some error-detection enhancement. As before, the decoder can be implemented with gates similar to those shown in Figure 6.11.

6.6.5 The Standard Array Provides Insight

In the context of Figure 6.14, the (8, 2) code satisfies the Hamming bound. That is, from the standard array, it is recognizable that the (8, 2) code can correct all combinations of single and double errors. Consider the following question: Given that transmission takes place over a channel that always introduces errors in the form of a burst of 3-bit errors and thus there is no interest in correcting single or double errors, wouldn't it be possible to set up the coset leaders to correspond to only triple errors? It is simple to see that in a sequence of 8 bits, there are $\binom{8}{3} = 56$ ways to make triple errors. If we only want to correct all these 56 combinations of triple errors, there is sufficient room (sufficient number of cosets) in the standard array, since there are 64 rows. Will that work? No, it will not. For any code, the overriding parameter for determining error-correcting capability is $d_{\min}$. For the (8, 2) code, $d_{\min} = 5$ dictates that only 2-bit error correction is possible.

How can the standard array provide some insight as to why this scheme won't work? In order for a group of x-bit error patterns to enable x-bit error correction, the

Syndromes		Standard array			
000000	1.	00000000	11110001	00111110	11001111
111100	2.	00000001	11110000	00111111	11001110
001111	3.	00000010	11110011	00111100	11001101
000001	4.	00000100	11110101	00111010	11001011
000010	5.	00001000	11111001	00110110	11000111
000100	6.	00010000	11100001	00101110	11011111
001000	7.	00100000	11010001	00011110	11101111
010000	8.	01000000	10110001	01111110	10001111
100000	9.	10000000	01110001	10111110	01001111
110011	10.	00000011	11110010	00111101	11001100
111101	11.	00000101	11110100	00111011	11001010
111110	12.	00001001	11111000	00110111	11000110
111000	13.	00010001	11100000	00101111	11011110
110100	14.	00100001	11010000	00011111	11101110
101100	15.	01000001	10110000	01111111	10001110
011100	16.	10000001	01110000	10111111	01001110
001110	17.	00000110	11110111	00111000	11001001
001101	18.	00001010	11111011	00110100	11000101
001011	19.	00010010	11100011	00101100	11011101
000111	20.	00100010	11010011	00011100	11101101
011111	21.	01000010	10110011	01111100	10001101
101111	22.	10000010	01110011	10111100	01001101
000011	23.	00001100	11111101	00110010	11000011
000101	24.	00010100	11100101	00101010	11011011
001001	25.	00100100	11010101	00011010	11101011
010001	26.	01000100	10110101	01111010	10001011
100001	27.	10000100	01110101	10111010	01001011
000110	28.	00011000	11101111	00100110	11010111
001010	29.	00101000	11011001	00010110	11100111
010010	30.	01001000	10111001	01110110	10000111
100010	31.	10001000	01111001	10110110	01000111
001100	32.	00110000	11000001	00001110	11111111
010100	33.	01010000	10100001	01101110	10011111
100100	34.	10010000	01100001	10101110	01011111
011000	35.	01100000	10010001	01011110	10101111
101000	36.	10100000	01010001	10011110	01101111
110000	37.	11000000	00110001	11111110	00001111
110010	38.	00000111	11100010	00111001	11101000
110111	39.	00010011	11100010	00101101	11011100
111011	40.	00100011	11010010	00011101	11101100
100011	41.	01000011	10110010	01111101	10001100
010011	42.	10000011	01110010	10111101	01001100
111111	43.	00001101	11111100	00110011	11000010
111001	44.	00010101	11100100	00101011	11011010
110101	45.	00100101	11010100	00011011	11101010
101101	46.	01000101	10110100	01111011	10001010
011101	47.	10000101	01110100	10111011	01001010
011110	48.	01000110	10110111	01111000	10001001
101110	49.	10000110	01110111	10111000	01001001
100101	50.	10010100	01100101	10101010	01011011
011001	51.	01100100	10010101	01011010	10101011
110001	52.	11000100	00110101	11111010	00001011
011010	53.	01101000	10011001	01010110	10100111
010110	54.	01011000	10101001	01100110	10010111
100110	55.	10011000	01101001	10100110	01010111
101010	56.	10101000	01011001	10010110	01100111
101001	57.	10100100	01010101	10011010	01101011
100111	58.	10100010	01010011	10011100	01101101
010111	59.	01100010	10010011	01011100	10101101
010101	60.	01010100	10100101	01101010	10011011
011011	61.	01010010	10100011	01101100	10011101
110110	62.	00101001	11011000	00010111	11100110
111010	63.	00011001	11101000	00100111	11010110
101011	64.	10010010	01100011	10101100	01011101

Figure 6.14 The syndromes and the standard array for the (8, 2) code.

entire group of weight-x vectors must be coset leaders; that is, they must only occupy the leftmost column. In Figure 6.14, it can be seen that all weight-1 and weight-2 vectors appear in the leftmost column of the standard array, and nowhere else. Even if we forced all weight-3 vectors into row numbers 2 through 57, we would find that some of these vectors would have to reappear elsewhere in the array (which violates a basic property of the standard array). In Figure 6.14 a shaded box is drawn around every one of the 56 vectors having a weight of 3. Look at the coset leaders representing 3-bit error patterns in rows 38, 41–43, 46–49, and 52 of the standard array. Now look at the entries of the same row numbers in the rightmost column, where shaded boxes indicate other weight-3 vectors. Do you see the ambiguity that exists for each of the rows listed above, and why it is not possible to correct all 3-bit error patterns with this (8, 2) code? Suppose the decoder receives the weight-3 vector 1 1 0 0 1 0 0 0, located at row 38 in the rightmost column. This flawed codeword could have arisen in one of two ways: One would be that codeword 1 1 0 0 1 1 1 1 was sent and the 3-bit error pattern 0 0 0 0 0 1 1 1 perturbed it; the other would be that codeword 0 0 0 0 0 0 0 0 was sent and the 3-bit error pattern 1 1 0 0 1 0 0 0 perturbed it.

6.7 CYCLIC CODES

Binary cyclic codes are an important subclass of linear block codes. The codes are easily implemented with feedback shift registers; the syndrome calculation is easily accomplished with similar feedback shift registers; and the underlying algebraic structure of a cyclic code lends itself to efficient decoding methods. An (n, k) linear code is called a *cyclic code* if it can be described by the following property. If the n-tuple $\mathbf{U} = (u_0, u_1, u_2, \dots, u_{n-1})$ is a codeword in the subspace S, then $\mathbf{U}^{(1)} = (u_{n-1}, u_0, u_1, u_2, \dots, u_{n-2})$ obtained by an end-around shift, is also a codeword in S. Or, in general, $\mathbf{U}^{(i)} = (u_{n-i}, u_{n-i+1}, \dots u_{n-1}, u_0, u_1, \dots, u_{n-i-1})$, obtained by i end-around or cyclic shifts, is also a codeword in S.

The components of a codeword $\mathbf{U} = (u_0, u_1, u_2, \dots, u_{n-1})$ can be treated as the coefficients of a polynomial $\mathbf{U}(X)$ as follows:

$$\mathbf{U}(X) = u_0 + u_1X + u_2X^2 + \cdots + u_{n-1}X^{n-1} \tag{6.54}$$

The polynomial function $\mathbf{U}(X)$ can be thought of as a "placeholder" for the digits of the codeword $\mathbf{U}$; that is, an n-tuple vector is described by a polynomial of degree $n - 1$ or less. The presence or absence of each term in the polynomial indicates the presence of a 1 or 0 in the corresponding location of the n-tuple. If the u_{n-1} component is nonzero, the polynomial is of degree $n - 1$. The usefulness of this polynomial description of a codeword will become clear as we discuss the algebraic structure of the cyclic codes.

6.7.1 Algebraic Structure of Cyclic Codes

Expressing the codewords in polynomial form, the cyclic nature of the code manifests itself in the following way. If $\mathbf{U}(X)$ is an $(n - 1)$-degree codeword polynomial, then $\mathbf{U}^{(i)}(X)$, the remainder resulting from dividing $X^i\mathbf{U}(X)$ by $X^n + 1$, is also a codeword; that is

$$\frac{X^i\mathbf{U}(X)}{X^n + 1} = \mathbf{q}(X) + \frac{\mathbf{U}^{(i)}(X)}{X^n + 1} \tag{6.55a}$$

or, multiplying through by $X^n + 1$,

$$X^i\mathbf{U}(X) = \mathbf{q}(X)(X^n + 1) + \underbrace{\mathbf{U}^{(i)}(X)}_{\text{remainder}} \tag{6.55b}$$

which can also be described in terms of modulo arithmetic as

$$\mathbf{U}^{(i)}(X) = X^i\mathbf{U}(X) \text{ modulo } (X^n + 1) \tag{6.56}$$

where x modulo y is defined as the remainder obtained from dividing x by y. Let us demonstrate the validity of Equation (6.56) for the case of $i = 1$:

$$\mathbf{U}(X) = u_0 + u_1X + u_2X^2 + \cdots + u_{n-2}X^{n-2} + u_{n-1}X^{n-1}$$
$$X\mathbf{U}(X) = u_0X + u_1X^2 + u_2X^3 + \cdots + u_{n-2}X^{n-1} + u_{n-1}X^n$$

We now add and subtract u_{n-1}; or, since we are using modulo-2 arithmetic, we add u_{n-1} twice, as follows:

$$X\mathbf{U}(X) = \underbrace{u_{n-1} + u_0X + u_1X^2 + u_2X^3 + \cdots + u_{n-2}X^{n-1}}_{\mathbf{U}^{(1)}(X)} + u_{n-1}X^n + u_{n-1}$$
$$= \mathbf{U}^{(1)}(X) + u_{n-1}(X^n + 1)$$

Since $\mathbf{U}^{(1)}(X)$ is of degree $n - 1$, it cannot be divided by $X^n + 1$. Thus, from Equation (6.55a), we can write

$$\mathbf{U}^{(1)}(X) = X\mathbf{U}(X) \text{ modulo } (X^n + 1)$$

By extension, we arrive at Equation (6.56):

$$\mathbf{U}^{(i)}(X) = X^i\mathbf{U}(X) \text{ modulo } (X^n + 1)$$

Example 6.7 Cyclic Shift of a Code Vector

Let $\mathbf{U} = 1\ 1\ 0\ 1$, for $n = 4$. Express the codeword in polynomial form, and using Equation (6.56), solve for the third end-around shift of the codeword.

Solution

$$\mathbf{U}(X) = 1 + X + X^3 \qquad \text{(polynomial is written low order to high order);}$$
$$X^i\mathbf{U}(X) = X^3 + X^4 + X^6, \qquad \text{where } i = 3.$$

Divide $X^3\mathbf{U}(X)$ by $X^4 + 1$, and solve for the remainder using polynomial division:

$$\begin{array}{r@{}l} & X^2 + 1 \\ X^4 + 1\,\big) & X^6 + X^4 + X^3 \\ & X^6 \qquad\qquad\quad + X^2 \\ \hline & \quad X^4 + X^3 + X^2 \\ & \quad X^4 \qquad\qquad\quad + 1 \\ \hline & \qquad X^3 + X^2 + 1 \qquad \text{remainder } \mathbf{U}^{(3)}(X) \end{array}$$

Writing the remainder low order to high order—$1 + X^2 + X^3$—the codeword $\mathbf{U}^{(3)} = 1\ 0\ 1\ 1$ is three cyclic shifts of $\mathbf{U} = 1\ 1\ 0\ 1$. Remember that for binary codes, the addition operation is performed modulo-2, so that $+1 = -1$, and we consequently do not show any minus signs in the computation.

6.7.2 Binary Cyclic Code Properties

We can generate a cyclic code by using a *generator polynomial* in much the way that we generated a block code using a generator matrix. The generator polynomial $\mathbf{g}(X)$ for an (n, k) cyclic code is unique and is of the form

$$\mathbf{g}(X) = g_0 + g_1X + g_2X^2 + \cdots + g_pX^p \tag{6.57}$$

where g_0 and g_p must equal 1. Every codeword polynomial in the subspace is of the form $\mathbf{U}(X) = \mathbf{m}(X)\mathbf{g}(X)$, where $\mathbf{U}(X)$ is a polynomial of degree $n - 1$ or less. Therefore, the message polynomial $\mathbf{m}(X)$ is written as

$$\mathbf{m}(X) = m_0 + m_1X + m_2X^2 + \cdots + m_{n-p-1}X^{n-p-1} \tag{6.58}$$

There are 2^{n-p} codeword polynomials, and there are 2^k code vectors in an (n, k) code. Since there must be one codeword polynomial for each code vector,

$$n - p = k$$

or

$$p = n - k$$

Hence, $\mathbf{g}(X)$, as shown in Equation (6.57), must be of degree $n - k$, and every codeword polynomial in the (n, k) cyclic code can be expressed as

$$\mathbf{U}(X) = (m_0 + m_1X + m_2X^2 + \cdots + m_{k-1}X^{k-1})\mathbf{g}(X) \tag{6.59}$$

$\mathbf{U}$ is said to be a valid codeword of the subspace S *if, and only if,* $\mathbf{g}(X)$ divides into $\mathbf{U}(X)$ without a remainder.

A generator polynomial $\mathbf{g}(X)$ of an (n, k) cyclic code is a factor of $X^n + 1$; that is, $X^n + 1 = \mathbf{g}(X)\mathbf{h}(X)$. For example,

$$X^7 + 1 = (1 + X + X^3)(1 + X + X^2 + X^4)$$