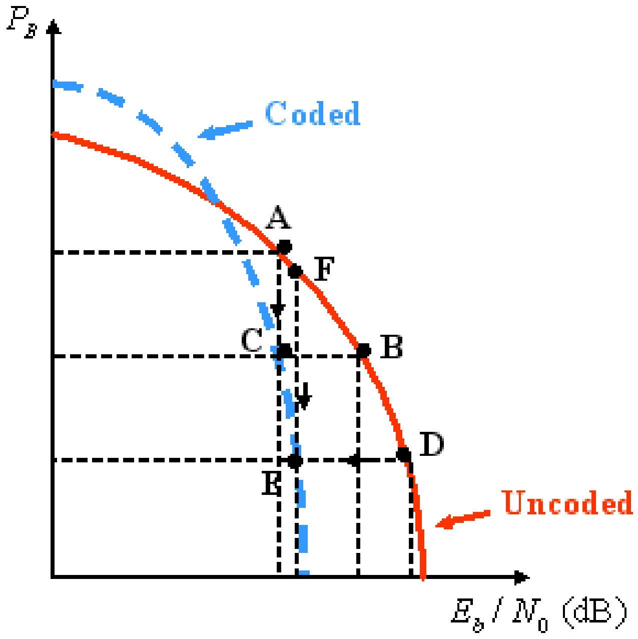
# 5. Coded Modulation Schemes

## Why using error correction coding for modulation?

As discussed before, modulated signals using M-ary signaling perform at best 9.0 dB lower than the achievable channel capacity of the physical channel. The way to close that gap is to resort to channel coding using elaborate signal sequences lying in higher-dimensional spaces, but composed from elementary modulator sets, such as M-ary QAM or M-ary PSK schemes.

**Coding gain:** For a given bit-error probability, the reduction in the Eb/N0 that can be realized through the use of code:

$$G\,[\mathrm{dB}] = \left(\frac{E_b}{N_0}\right)_u [\mathrm{dB}] - \left(\frac{E_b}{N_0}\right)_c [\mathrm{dB}]$$



Issues to consider:

- Error performance vs. bandwidth
- Power vs. bandwidth
- Data rate vs. bandwidth
- Capacity vs. bandwidth

The process of channel coding produces modulator input symbols that are interrelated in either a block-by-block or sliding-window fashion, introducing a memory and redundancy into the signaling process. The costs involved are:

(1) increased rate (bandwidth) requirement and
(2) exponentially increasing computational (realization) complexity.

Two valid questions:

1. **What is the underlying thought behind coding?**
2. **Why bother with complexity?**

The answers to both point to the real promise of Shannon’s information theory for reliable communication in noisy channels (1) at rates approaching the channel capacity and (2) to do it in an instrumentable way. Codes install two key features in message sequence blocks:

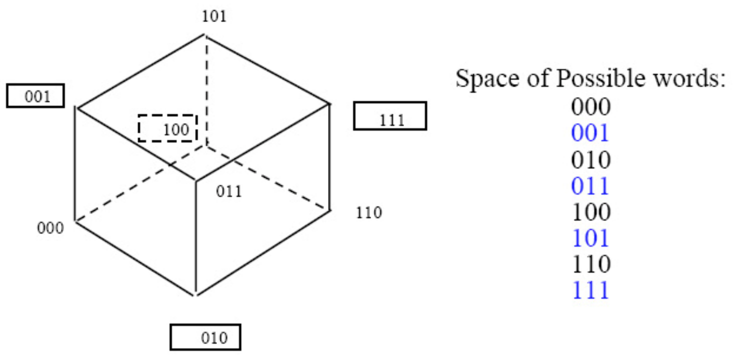
- ***Redundancy and***
- ***Memory.***

**Redundancy:** The set of allowable code sequences or codewords is often many orders smaller than the number of sequences suggested by the size of the code alphabet. Thus, the code symbols do not carry as much information per symbol as they might without coding as in the case of “parity check codes.”

**Example 5.1:** Recall the transmission of triple-bits over a BSC with a majority rule decoder studied in Chapter 1. The corresponding code pair was: {000,111}. The length of this code is $n = 3$ and the code dimension is defined as $k = \log_2 |C|$ will be $k = 2$ in this case. The rate of the code is defined by:

$$Rate = R = \frac{k}{n} \qquad \Rightarrow \quad R = 1/3\ bits/symbol$$

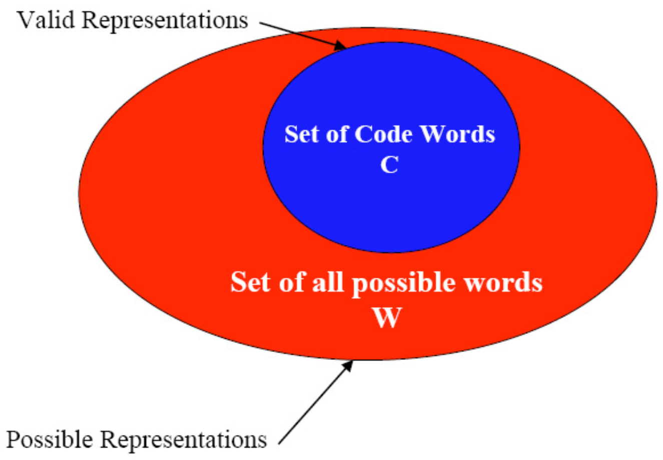
**Example 5.2:** Simple 3-bit Error Detecting Code Space:



**Boxed words = odd parity; blue words are valid code words; $d_{min} = 2$**

**Memory:** The redundancy gained by adding bits can accomplish very little unless the code symbols depend on many input symbols, which is ascribed as memory. Equivalently, information in a sequence of messages is diffused throughout a block of bits called a codeword.

## 5.1 Definitions and Classes of Codes[1]



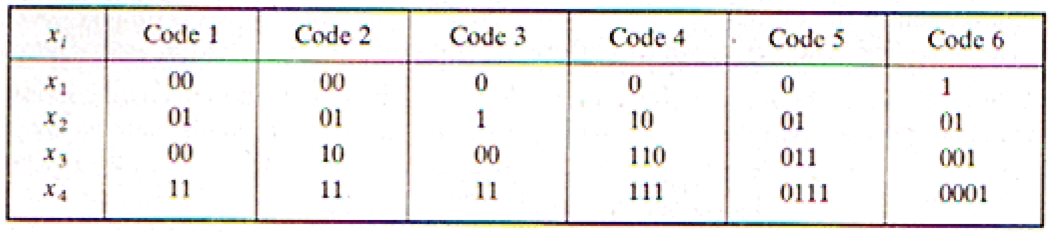
Coding techniques may be classified based on the structure behind the encoding function:

- Block Codes
- Sliding-Block Codes (Trellis Coding.)

**Block Codes:** They operate in block-by-block fashion and each codeword depends only on the current input message block. They can be further categorized as *Linear* and *Non-linear* codes.

[1] The key reference in preparing some parts of this chapter has been (1) John Proakis's book: *Digital Communications*, 4th Edition, (2) Simon Haykin's book: *Communication Systems*, 4th Edition, and (3) Bernard Sklar's book: *DigitalCommunications*, 2nd Edition. A number of tables and figures have been included with permission.

**Example 5.3:** Consider the following six codes:



| $x_i$ | Code 1 | Code 2 | Code 3 | Code 4 | Code 5 | Code 6 |
|---|---|---|---|---|---|---|
| $x_1$ | 00 | 00 | 0 | 0 | 0 | 1 |
| $x_2$ | 01 | 01 | 1 | 10 | 01 | 01 |
| $x_3$ | 00 | 10 | 00 | 110 | 011 | 001 |
| $x_4$ | 11 | 11 | 11 | 111 | 0111 | 0001 |

**Fixed-length Code:** A code with a fixed code length. (code: 1-2 are fixed length).

**Variable-length Code:** Code length is not fixed. (code: 3-6).

**Distinct Code:** A code is distinct if each codeword is distinguishable form other codewords. All codes above except code are distinguishable.

**Prefix-Code:** Code in which no codeword can be formed by adding code symbols to another codeword.

**Uniquely decodable Code:** A code is uniquely decodable if the original message can be reconstructed perfectly from the encoded binary sequence.

*A sufficient condition to ensure that a code is uniquely decodable is that no codeword is a prefix to another codeword.*

**Instantaneous Codes:** A uniquely decodable code is an instantaneous code if the end of any codeword is recognizable without examining the subsequent code symbols. All prefix codes are instantaneous codes.

**Optimal Codes:** A code is optimal if it is instantaneous and has minimum average length $L$ for a given source with a given probability distribution for the source alphabet.

**Code Efficiency** $\eta$: It is a measure of how close a particular code $L$ is to the theoretically achievable, i.e., entropy $H$.

$$\eta = \frac{H(X)}{L}$$

**Code Redundancy** $\gamma$:

$$\gamma = 1 - \eta$$

**Kraft Inequality:** Let $X$ be a source with alphabet $\{x_i; i = 1,2,\cdots,m\}$. Assume that the length of the binary codeword corresponding to a particular message $x_i$ is $k_i$. A necessary and sufficient condition for the existence of an instantaneous (prefix) binary code is bound by Kraft Inequality:

$$K = \sum_{i=1}^{m} 2^{-n_i} \leq 1$$

**Example 5.4:** Consider a signal set with letters: $\{x_i; i = 1,\cdots,4\}$ and four possible coding schemes.

| $X_i$ | Code A | Code B | Code C | Code D |
|---|---|---|---|---|
| $X_1$ | 00 | 0 | 0 | 0 |
| $X_2$ | 01 | 10 | 11 | 100 |
| $X_3$ | 10 | 11 | 100 | 110 |
| $X_4$ | 11 | 110 | 110 | 111 |

**(a)** All codes except B satisfy Kraft's inequality: Since they are all 2-bits long, we have

$$K_A = \sum_{i=1}^{4} 2^{-n_i} = \sum_{i=1}^{4} 2^{-2} = 1$$

$$K_B = \sum_{i=1}^{4} 2^{-n_i} = 1/2 + 1/4 + 1/4 + 1/8 = 9/8 > 1 \quad \Rightarrow Violation$$

$$K_C = \sum_{i=1}^{4} 2^{-n_i} = 1/2 + 1/4 + 1/8 + 1/8 = 1$$

$$K_D = \sum_{i=1}^{4} 2^{-n_i} = 1/2 + 1/8 + 1/8 + 1/8 = 7/8$$

**(b)** Codes A and D are uniquely decodable. These two codes are prefix codes and hence uniquely decodable. Since code B does NOT satisfy Kraft's inequality it is NOT. C has problems consider the bit stream:

0110110 $\Rightarrow$ two possible symbol sequences: $x_1 x_2 x_3 x_4$ OR $x_1 x_4 x_4$

**Linear Codes:** Linear codes are defined by a linear mapping over an appropriate algebraic system, such as Galois Fields, from the space of input messages to the space of output messages. This algebraic structure could allow significant simplification of encoding and decoding equipment. In other words, if linear combinations of two codewords are also legitimate codes then they are called linear codes. They are also known as the *"Parity Check Codes."*

**Distance of a code:** The number of elements in which two codewords differ.

$$d(C_i, C_j) = \sum_{l=1}^{N} C_{il} \oplus C_{jl} \, (Modulo - q) \tag{5.1}$$

Here $q$ represents the number of elements in the code. For $q=2$ we have binary case and this is called the **Hamming distance.** The minimum distance $d_{\min}$, or equivalently, $d_{\min}^{free}$ is the smallest distance for the given code set:

$$d_{\min} = Min\{d(C_i, C_j)\} \tag{5.2a}$$

**Error Detection Capability:** The number of bits can be detected is given by:

$$e = d_{\min} - 1 \tag{5.2b}$$

**Error correcting-capability** ***t*** of a code, which is defined as the maximum number of guaranteed correctable errors per codeword, is

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor \tag{5.2c}$$

The number of non-zero elements in a codeword gives weight of a code. For a binary code, the weight is basically the number of "1"s and is given by:

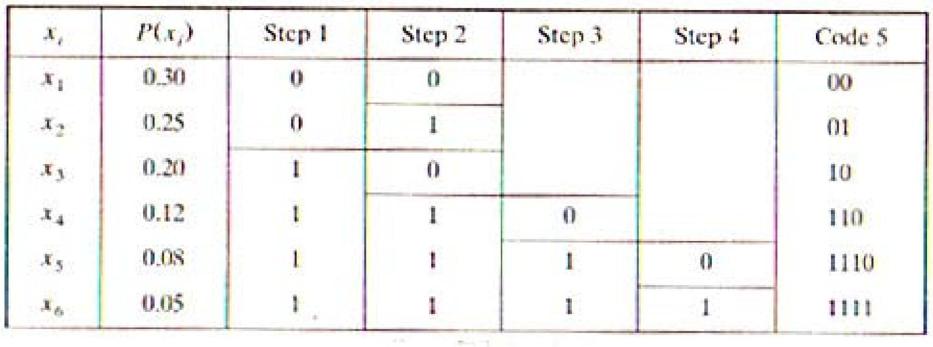
$$w(C_i) = \sum_{l=1}^{N} C_{il} \tag{5.3}$$

**Example 5.5:** Shannon-Fano Coding Algorithm:

(i) List source symbols in the order of decreasing probability.

(ii) Partition the set into two sets that are as close to equiprobable as possible and assign "0" to the upper set and "1" to the lower set.

(iii) Continue partitioning until no further subdivision is possible.



| $x_i$ | $P(x_i)$ | Step 1 | Step 2 | Step 3 | Step 4 | Code 5 |
|---|---|---|---|---|---|---|
| $x_1$ | 0.30 | 0 | 0 | | | 00 |
| $x_2$ | 0.25 | 0 | 1 | | | 01 |
| $x_3$ | 0.20 | 1 | 0 | | | 10 |
| $x_4$ | 0.12 | 1 | 1 | 0 | | 110 |
| $x_5$ | 0.08 | 1 | 1 | 1 | 0 | 1110 |
| $x_6$ | 0.05 | 1 | 1 | 1 | 1 | 1111 |

**Example 5.6:** Huffman Coding Algorithm:

(i) List source symbols in the order of decreasing probability.
(ii) Combine two symbols with lowest probability and re-order the resultant probabilities (reduction step 1). Assign "0" to the upper set and "1" to the lower set.
(iii) Continue combining until no further subdivision is possible.
(iv) Assume a tie-breaker rule for equiprobable cases. Lower index "1", upper "0" vice versa.

Consider the following six source symbols to be used as alphabet:

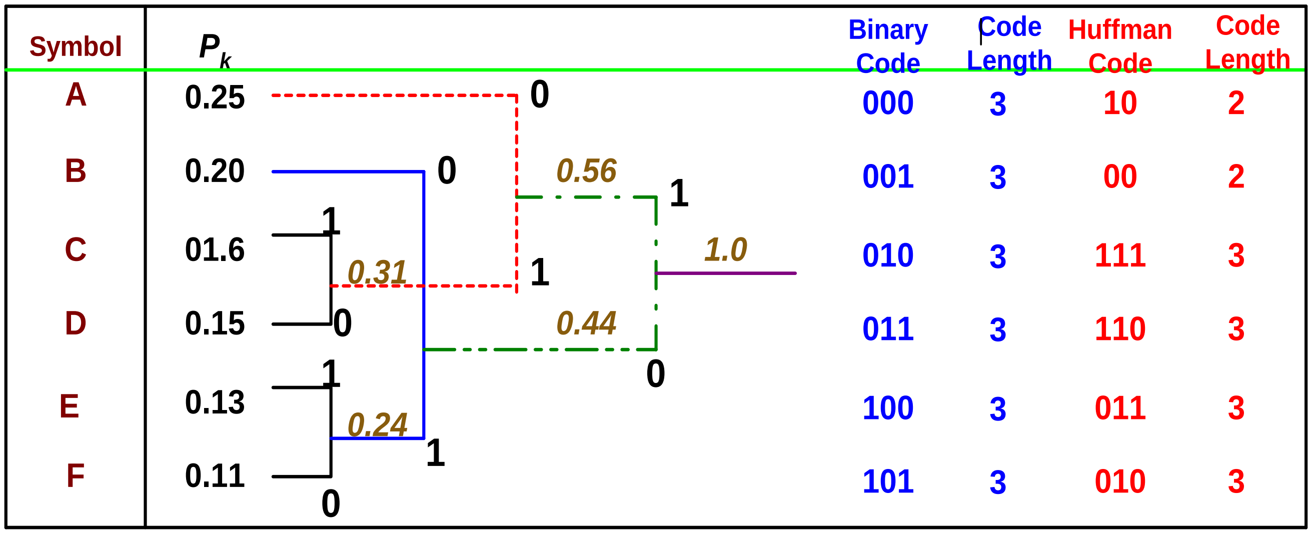
*S*={A,B,C,D,E,F} with occurrence probabilities {0.25, 0.20, 0.16, 0.15, 0.13, 0.11}. Note that sum of probabilities is 1.0 as it should be.

One way to represent this symbol set is to assign $\overline{L} = 3 - bit$ long codewords to each symbol, which might be rather away from the source entropy *H(S).*

$$H(S) = -\sum_{k=1}^{6} p_k . \log_2(p_k) = 2.5309 \quad bit/symbol \tag{1.25}$$

It is easy to see that the difference $\overline{L} - H(S) = 3.0 - 2.5309 = 0.4691\, bits/symbol$

This is almost ½ bits per symbol away from the theoretical bound *H(S).* Let us now construct a Huffman code for this set.



| Symbol | $P_k$ | Binary Code | Code Length | Huffman Code | Code Length |
|---|---|---|---|---|---|
| A | 0.25 | 000 | 3 | 10 | 2 |
| B | 0.20 | 001 | 3 | 00 | 2 |
| C | 01.6 | 010 | 3 | 111 | 3 |
| D | 0.15 | 011 | 3 | 110 | 3 |
| E | 0.13 | 100 | 3 | 011 | 3 |
| F | 0.11 | 101 | 3 | 010 | 3 |

Huffman Coding example.

The average length of this code is:

$$\overline{L}_{Huffman} = 0.25x2 + 0.20x2 + 0.16x3 + 0.15x3 + 0.13x3 + 0.11x3 = 2.55 \quad bits/symbol$$

The difference in this case is lowered to:

$$\overline{L} - H(S) = 2.55 - 2.5309 = 0.0191\, bits / symbol\ ,$$

which is almost perfect.

**A systematic code** is one in which the parity bits are appended to the end of information bits. For an *(n,k)*-code, the first *k* bits are identical to the message information, and the remaining *n-k* bits of each codeword are linear combinations of the *k* information bits.

**Cyclic Codes** exhibit a cyclic property and they are practically important subclass of linear codes. If $C = [c_{n-1}, c_{n-2}, ..., c_0]$ is a cyclic code, and then $[c_{n-2}, c_{n-3}, ..., c_0, c_{n-1}]$ is also a codeword. Due to this cyclic property, these codes possess a considerable amount of structure, which make the encoding and decoding procedures simple.

**Non-linear codes,** although not particularly important in the context of block coding, are the remaining codes.

**Trellis Coders,** in contrast, can be viewed as mapping an arbitrarily long input message to an arbitrarily long code stream without block structure. The output symbols at a particular time depends on the *state* of a finite-state machine, as well as on current inputs. The structure is a regular finite-state graph like a garden trellis. The nodes of the trellis are the labels of the state labels, which are, in turn, specified by a short block of previous inputs and the name sliding-block code is very frequently used.

**Linear Trellis Codes** are known as **Convolutional Codes**, because the code sequence can be viewed as the discrete-time convolution of the message sequence with the impulse response of the encoder. In practice, most trellis codes have thus far been the choice of design in the area of ML detection based digital communication systems.

**Non-Linear Trellis Codes** are also known as **Coset Codes** since they form standard arrays of rows (cosets) in a number of Hard-Decision Decoding schemes, which are non-linear detection techniques.

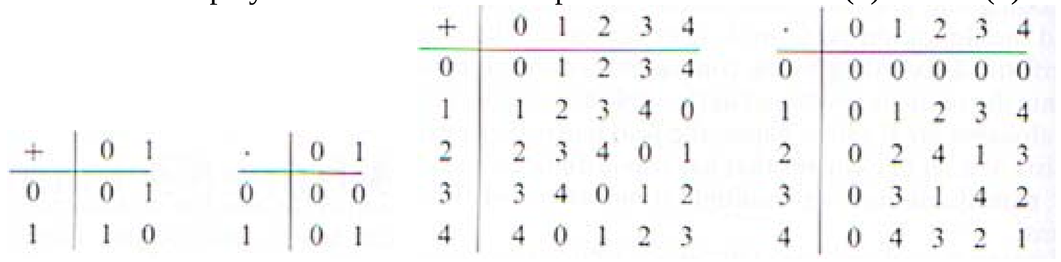
**Galois Fields:** Coding techniques make use of the mathematical constructs known as finite fields. The most common field employed in coding theory is Galois Fields $GF(2)$ and its extensions $GF(2^m)$, where *m* is an integer. Let $F$ be a finite set of elements on which two binary operations --addition and multiplication—are defined. This set is a field, in addition to two binary operations, if the following conditions are satisfied:

1. F is a commutative group under addition. The identity element with respect to addition is called the zero element.
2. The set of non-zero elements in *F* is a commutative group under multiplication. The identity element with respect to multiplication is called the unit element.
3. Multiplication is distributive over addition: $a.(b+c) = a.b + a.c$
4. The additive inverse of an element $a$ is $-a$ is the element, which forces the sum to 0.
5. The multiplicative inverse of $a$ is $a^{-1}$ and it satisfies: $a.a^{-1} = 1$

**Properties of fields:**

| | |
|---|---|
| • Property I: | $a.0 = 0 = 0.a$ |
| • Property II: | $If\ \ a \neq 0\ and\ b \neq 0\ then\ a.b \neq 0$ |
| • Property III: | $a.b = 0\ and\ a \neq 0\ imply\ b = 0$ |
| • Property IV: | $-(a.b) = (-a).b = a.(-b)$ |

**Example 5.7:** Let us display the addition and multiplication tables for GF(2) and GF(5):



| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

| · | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

| + | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 |
| 1 | 1 | 2 | 3 | 4 | 0 |
| 2 | 2 | 3 | 4 | 0 | 1 |
| 3 | 3 | 4 | 0 | 1 | 2 |
| 4 | 4 | 0 | 1 | 2 | 3 |

| · | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 | 3 | 4 |
| 2 | 0 | 2 | 4 | 1 | 3 |
| 3 | 0 | 3 | 1 | 4 | 2 |
| 4 | 0 | 4 | 3 | 2 | 1 |

**Note 1:** In binary arithmetic, modulo-2 operations are used and since $1+1=0\ implies\ 1=-1$, the subtraction is equivalent to addition. So, no need to design subtractors in realizations.

**Note 2:** Reed-Solomon codes to be discussed later make use of non-binary field and m>1. In addition to 1,0, elements represented by $\alpha^j$ are used to form:

$$F=\{0,1,\alpha,\alpha^2,...,\alpha^j,...\}=\{0,\alpha^0,\alpha^1,\alpha^2,...,\alpha^j,...\} \qquad (5.4)$$

- In order this field to contain $2^m$ element and is a closed set under multiplication we must add the following condition called the principle of irreducible polynomial:

$$\alpha^{(2^m-1)}=1=\alpha^0 \qquad (5.5)$$

The sequence of elements $F$ thus becomes the following sequence $F^*$:

$$F^*=\{0,1,\alpha,...,\alpha^{2^m-2},\alpha^{2^m-1},\alpha^{2^m},...\}=\{0,\alpha^0,\alpha^1,...,\alpha^{2^m-2},\alpha^0,\alpha,...\} \qquad (5.6)$$

**Note 3:** Each of the $2^m$ elements in Galois Field can be represented by a polynomial of degree m-1 or less. At least one of the m coefficients is non-zero and they can be denoted by:

$$\alpha^i=a_i(x)=a_{i0}+a_{i1}.x+a_{i2}.x^2+...+a_{i,m-1}.x^{m-1} \qquad (5.7)$$

**Note 4:** Addition of two elements of the field is defined as the modulo-2 addition of each of the polynomial coefficients of like powers:

$$\alpha^i+\alpha^j=(a_{i0}+a_{j0})+(a_{i1}+a_{j1}).x+...+(a_{im-1}+a_{jm-1}).x^{m-1} \qquad (5.8)$$

These two equations can be used to obtain elements in a Reed-Solomon code.

## 5.2 Linear Block Codes and Examples

A linear block code (*n*,*k*) has first *n-k* bits as parity bits and the last *k* bits as the message bits as shown below.

$$[\underbrace{b_0,b_1,\cdots,b_{n-k-1}}_{\text{Parity Bits}}\,|\,\underbrace{m_0,m_1,\cdots,m_{k-1}}_{\text{Message Bits}}]$$

**Systematic block code (*n*,*k*):** First (or last) *k* elements in the codeword are information bits.

The codewords are given by:

$$c=\begin{cases} b_i & i=0,1,...,n-k-1 \\ m_{i+k-n} & i=n-k,n-k+1,...,n-1 \end{cases} \qquad (5.9)$$

The (n-k) parity bits are linear sums of the k message bits:

$$b_i=p_{0i}.m_0+p_{1i}.m_1+...+p_{k-1,i}.m_{k-1} \qquad (5.10)$$

where the coefficients are defined by:

$$p=\begin{cases} 1 & if\ b_i\ depends\ on\ m_j \\ 0 & otherwise \end{cases} \qquad (5.11)$$

- These equations are usually written in a compact matrix form:

$$\underline{m} = [m_0, m_1, ..., m_{k-1}] \quad (5.12a)$$

$$\underline{b} = [b_0, b_1, ..., b_{n-k-1}] = \underline{m}.\underline{P} \quad (5.12b)$$

where:

$$\underline{P} = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0,n-k-1} \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & \vdots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{bmatrix} \quad (5.12c)$$

$$\underline{c} = [c_0, c_1, ..., c_{n-1}] \quad (5.12d)$$

The code is written by:

$$\underline{c} = [\underline{b} \vdots \underline{m}] = \underline{m}.[\underline{P} \vdots \underline{I}_k] \quad (5.13)$$

where $\underline{I}_k$ is the k-by-k identity matrix. Let us define the k-by-n generator matrix:

$$\underline{G} = [\underline{P} \vdots \underline{I}_k] \quad (5.14)$$

Using this generator matrix the code in (5.13) can be simplified to:

$$\underline{c} = \underline{m}.\underline{G} \quad (5.15)$$

Let $\underline{H}$ denote an (n-k)-by-n parity-check matrix:

$$\underline{H} = [\underline{I}_{n-k} \vdots \underline{P}^T] \quad (5.16)$$

We may perform the following multiplication of matrices:

$$\underline{H}\underline{G}^T = [\underline{I}_{n-k} \vdots \underline{P}^T] \begin{bmatrix} \underline{P}^T \\ \cdots \\ \underline{I}_k \end{bmatrix} = \underline{P}^T + \underline{P}^T = \underline{0} = \underline{G}.\underline{H}^T \quad (5.17)$$

Where $\underline{0}$ is a (n-k)-by-k null matrix with zero elements. Post multiplying both sides of (5.16) by the transpose of this parity-check matrix we get:

$$\underline{c}.\underline{H}^T = \underline{m}.\underline{G}.\underline{H}^T = \underline{0} \quad (5.18)$$

This last property is used in the decoding operation at the receiver.

**Syndrome Decoding:** Given $\underline{c}$ be the vector transmitter sends over a noisy channel and the received corrupted vector be $\underline{r}$:

$$\underline{r} = \underline{c} + \underline{e} \quad (5.19)$$

where $\underline{e}$ is the error vector of size 1-by-n, whose elements are defined by:

$$e_i = \begin{cases} 1 & \text{if an error has occurred in } i^{th} \text{ location} \\ 0 & \text{otherwise} \end{cases} \quad (5.20)$$

Syndrome is defined by:

$$\underline{s} = \underline{r}\underline{H}^T \quad (5.21)$$

The following two properties make syndrome essential in decoding codes operating in noisy environments.

**Property 1:** The syndrome depends only on the error pattern, not on the transmitted codeword.

$$\underline{s} = (\underline{c} + \underline{e}).\underline{H}^T = \underline{c}.\underline{H}^T + \underline{e}.\underline{H}^T = \underline{e}.\underline{H}^T \quad (5.22)$$

**Property 2:** All error patterns that differ by a codeword have the same syndrome.
For k message bits there are $2^k$ distinct codewords. Correspondingly, for any error pattern $\underline{e}$, we define vectors:

$$\underline{e}_i = \underline{c}_i + \underline{e}, \quad i = 0,1,\ldots,2^k - 1 \tag{5.23}$$

as coset vectors of the code. In other words, a coset has exactly $2^k$ elements that differ at most by a code vector and an *(n,k)* linear block code has $2^{n-k}$ possible cosets. Let us multiply both sides of (5.23) by the transpose of the parity-check matrix:

$$\underline{e}_i.\underline{H}^T = \underline{e}.\underline{H}^T + \underline{c}_i.\underline{H}^T = \underline{e}.\underline{H}^T \tag{5.24}$$

which is independent of the index i. These two properties show that the syndrome contains information about the error pattern and may therefore be used for error detection, where $(n-k)$ elements of the syndrome $\underline{s}$ are a linear combinations of $n$ elements of the error pattern $e$, as illustrated below:

| |
|---|
| $s_0 = e_0 + e_{n-k}\cdot p_{00} + e_{n-k+1}\cdot p_{10} + e_{n-k+2}\cdot p_{20} + \cdots + e_{n-1}\cdot p_{k-1,0}$ |
| $s_1 = e_1 + e_{n-k}\cdot p_{01} + e_{n-k+1}\cdot p_{11} + e_{n-k+2}\cdot p_{21} + \cdots + e_{n-1}\cdot p_{k-1,1}$ |
| $\vdots$ |
| $s_{n-k-1} = e_{n-k-1} + e_{n-k}\cdot p_{0,n-k-1} + e_{n-k+1}\cdot p_{1,n-k-1} + e_{n-k+2}\cdot p_{2,n-k-1} + \cdots + e_{n-1}\cdot p_{k-1,n-k-1}$ |

**Syndrome decoding algorithm:**

*Preliminary Step* 1: $2^k$ code vectors are placed in a row with the all-zero code vector $\underline{c}_1$ as the left-most element.

*Preliminary Step 2:* An error pattern $\underline{e}_2$ is picked and placed under $\underline{c}_1$, and a second row is formed by adding $\underline{e}_2$ to each of the remaining codewords in the first row.

*Preliminary step 3:* Step 2 is repeated until all the possible error patterns have accounted for.

$$\begin{array}{ccccccc}
\mathbf{c}_1 = \mathbf{0} & \mathbf{c}_2 & \mathbf{c}_3 & \cdots & \mathbf{c}_i & \cdots & \mathbf{c}_{2^k} \\
\mathbf{e}_2 & \mathbf{c}_2 + \mathbf{e}_2 & \mathbf{c}_3 + \mathbf{e}_2 & \cdots & \mathbf{c}_i + \mathbf{e}_2 & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_2 \\
\mathbf{e}_3 & \mathbf{c}_2 + \mathbf{e}_3 & \mathbf{c}_3 + \mathbf{e}_3 & \cdots & \mathbf{c}_i + \mathbf{e}_3 & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_3 \\
\vdots & \vdots & \vdots & & \vdots & & \vdots \\
\mathbf{e}_j & \mathbf{c}_2 + \mathbf{e}_j & \mathbf{c}_3 + \mathbf{e}_j & \cdots & \mathbf{c}_i + \mathbf{e}_j & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_j \\
\vdots & \vdots & \vdots & & \vdots & & \\
\mathbf{e}_{2^{n-k}} & \mathbf{c}_2 + \mathbf{e}_{2^{n-k}} & \mathbf{c}_3 + \mathbf{e}_{2^{n-k}} & & \mathbf{c}_i + \mathbf{e}_{2^{n-k}} & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_{2^{n-k}}
\end{array}$$

For a given channel, the probability of decoding error is minimized when the most likely error patterns are chosen as the coset leaders. For BSC, this corresponds to forming the above standard array with each coset leader having the minimum Hamming weight in its coset.

**Regular steps of the decoding procedure:**

***Step 1:*** For the received vector $\underline{r}$, compute the syndrome: $\underline{s} = \underline{r}\,\underline{H}^T$

***Step 2:*** Within the coset characterized by this syndrome, identify the coset leader by choosing the error pattern with the largest probability of occurrence; call it $\underline{e}_0$.

***Step 3:*** Compute the code vector: $\underline{c} = \underline{r} + \underline{e}_0$ as the decoded version of the received vector.

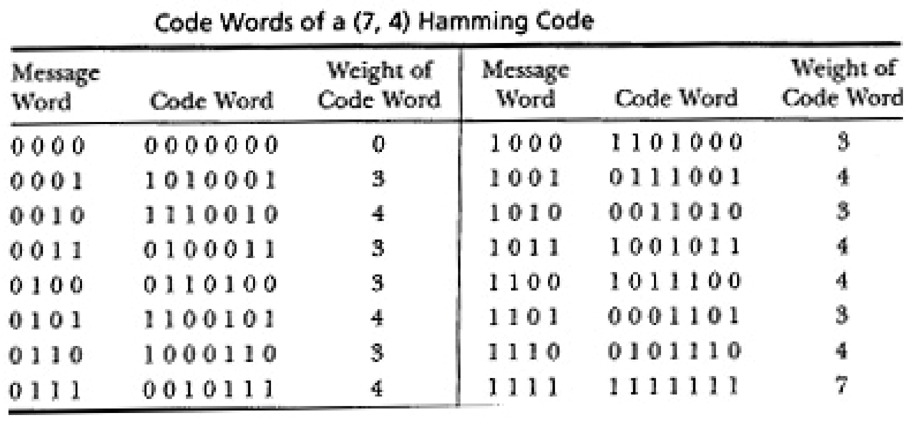
**Example 5.8:** (From Haykin Chapter 10) Hamming Codes: Consider a family of (n,k) codes that have the following parameters:

| | |
|---|---|
| Block length: | $n = 2^m - 1$ |
| Number of message bits: | $k = 2^m - m - 1$ |
| Number of parity bits: | $n - k = m$ |

Where $m \geq 3$. These codes are commonly known as the Hamming codes. Consider the specific example of (*7,4*) code, where the coding rate is *4/7*. An appropriate generator matrix and the corresponding parity-check matrix for this code for this code are given by:

$$G = \left[\underbrace{\begin{matrix}1 & 1 & 0\\ 0 & 1 & 1\\ 1 & 1 & 1\\ 1 & 0 & 1\end{matrix}}_{\mathbf{P}} \;\middle|\; \underbrace{\begin{matrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{matrix}}_{\mathbf{I}_k}\right] \qquad H = \left[\underbrace{\begin{matrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{matrix}}_{\mathbf{I}_{n-k}} \;\middle|\; \underbrace{\begin{matrix}1 & 0 & 1 & 1\\ 1 & 1 & 1 & 0\\ 0 & 1 & 1 & 1\end{matrix}}_{\mathbf{P}^T}\right]$$

Next we will tabulate 16 distinct messages together with the weight of the code for *k=4*. For a given, message, the corresponding codeword is found by using the generator matrix of (5.15), which are normally tabulated together with the Hamming weights of individual codewords. It is clear from this table that the minimum Hamming distance is 3.



Code Words of a (7, 4) Hamming Code

| Message Word | Code Word | Weight of Code Word | Message Word | Code Word | Weight of Code Word |
|---|---|---|---|---|---|
| 0000 | 0000000 | 0 | 1000 | 1101000 | 3 |
| 0001 | 1010001 | 3 | 1001 | 0111001 | 4 |
| 0010 | 1110010 | 4 | 1010 | 0011010 | 3 |
| 0011 | 0100011 | 3 | 1011 | 1001011 | 4 |
| 0100 | 0110100 | 3 | 1100 | 1011100 | 4 |
| 0101 | 1100101 | 4 | 1101 | 0001101 | 3 |
| 0110 | 1000110 | 3 | 1110 | 0101110 | 4 |
| 0111 | 0010111 | 4 | 1111 | 1111111 | 7 |

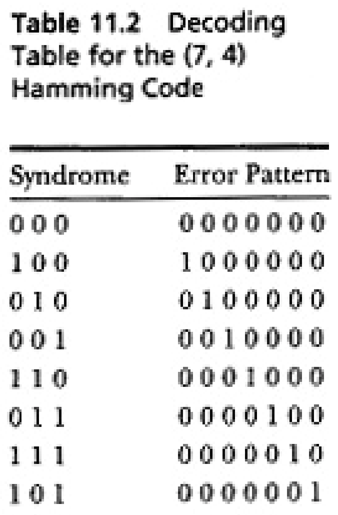


Table 11.2 Decoding Table for the (7, 4) Hamming Code

| Syndrome | Error Pattern |
|---|---|
| 000 | 0000000 |
| 100 | 1000000 |
| 010 | 0100000 |
| 001 | 0010000 |
| 110 | 0001000 |
| 011 | 0000100 |
| 111 | 0000010 |
| 101 | 0000001 |

Hamming codes have a property that we can correct up to t errors *if and only if*,

$$t \leq \lfloor 0.5(d_{\min} - 1) \rfloor \qquad (5.25)$$

In this case, we can correct single-error patterns. In other words Hamming (7,4) codes are single-error correcting binary perfect codes. If we assume that we have single-error patterns, we can formulate the seven coset leaders as listed in the second column of the second table above. As expected zero syndrome corresponds to no errors.

For instance, [1110010] is sent and the received vector is [1100010] has an error in location 3. Using (5.22), the syndrome is calculated as:

$$s = [1100010].\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = [001]$$

From this last table, we find that the corresponding coset leader (error pattern with the highest probability of error) is found to be [0010000]. Thus, adding this error pattern to the received vector produces the correct vector actually sent.

## 5.3 Cyclic Codes and Examples

Due to their well-defined structure, these codes have very efficient decoding schemes. Encoding and syndrome calculation are easily performed using feedback shift-registers. Hence, relatively long block codes can be implemented with a reasonable complexity. Finally, two of the most prominent block codes: BCH and Reed-Solomon codes are cyclic codes.

Let the n-tuple $(c_0, c_1, ..., c_{n-1})$ denote a code word of an *(n,k)* linear block code. Let *g(X)* be a polynomial of degree n-k that is a factor of $X^n + 1$, which can be expanded:

$$g(X) = 1 + \sum_{i=1}^{n-k-1} g_i.X^i + X^{n-k} \tag{5.26}$$

where each coefficient $g_i$ is 1 or 0. This polynomial has two terms with coefficient 1 separated by *n-k-1* terms. Therefore, it is called the generator polynomial of a cyclic code, where each codeword can be expressed as a polynomial product:

$$c(X) = a(X).g(X) \tag{5.27}$$

where *a(X)* is a polynomial with degree *k-1*.

Suppose we are given a generator polynomial *g(X)* and we would like to encode the message sequence $(m_0, m_1, ..., m_{k-1})$ into an *(n,k)* systematic cyclic code let us form the structure for the code with *(n-k)*-parity bits in front: $(b_0, b_1, ..., b_{n-k-1})$ followed by the message bits $(m_0, m_1, ..., m_{k-1})$.

1. Let the message polynomial be defined as:

$$m(X) = m_0 + m_1.X + ... + m_{k-1}.X^{k-1} \tag{5.28}$$

2. Multiply $X^{n-k}.m(X)$.
3. Divide the result in step 2 by the generator polynomial to obtain the remainder $b(X)$.

$$X^{n-k}.m(X) / g(X) = a(X) + \frac{b(X)}{g(X)} \tag{5.29}$$

4. Add this remainder polynomial to step 2 to obtain the code polynomial:

$$c(X) = b(X) + X^{n-k}.m(X) \tag{5.30}$$

**Parity-Check Polynomial:** A cyclic code is also uniquely defined by its parity-check polynomial:

$$h(X) = 1 + \sum_{i=1}^{k-1} h_i.X^i + X^k \tag{5.31}$$

where coefficients $h_i$ are either 0 or 1.

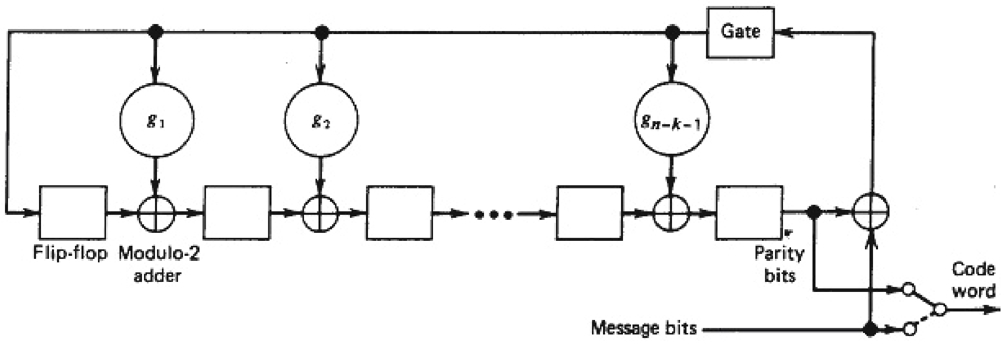
**Property:** *g(X)* and *h(X)* are factors of the polynomial: $X^n + 1$.

$$g(X).h(X) = X^n + 1 \qquad (5.32)$$

This property provides the basis for selecting the generator polynomial or the parity-check polynomial of a cyclic code.

**(n-k)-by-n Parity-Check Matrix:** $\underline{H}$ is formed from rows of *n-tuples* pertaining to the *(n-k) polynomials*: $\{X^k.h(X^{-1}), X^{k+1}.h(X^{-1}),..., X^{n-1}.h(X^{-1})\}$. Even though, it seems confusing now, it is actually a simple process, as it will be demonstrated with an example later.

- Encoder given below implements the above steps in terms of flip-flops as unit-delay elements, modulo-2 adders, branch weights, a gate and a switch. 1. Gate is ON to permit *k-message* bits to be shifted into FF. Right after, *(n-k)-bits* in SR form the parity-bits, which are the same as the coefficients of *b(X),* 2.Gate is OFF to break the feedback connections. 3. Contents of the shift registers are pumped out to the channel.



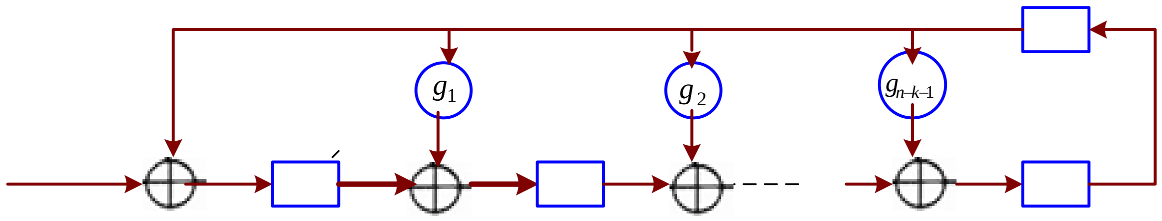
Encoder for an (n,k) cyclic Code.

- **Syndrome Computation:** Recall that the first step in the decoding of a block code is the computation of a syndrome for the received codeword. If the syndrome is zero, there are no transmission errors. Otherwise, this information can be used to correct the errors.

Let the received word be represented with a polynomial of degree n-1 or less:

$$r(x) = r_0 + r_1.X + ...r_{n-1}.X^{n-1} = q(X).g(X) + s(X) \qquad (5.33)$$

where *s(X)* is the remainder polynomial of degree n-k-1 or less, which is defined as the syndrome polynomial and its coefficients make up the *(n-k)-by-1* syndrome $\underline{s}$. The structure of this set-up is identical to the encoder structure above, except for the fact that the received bits are fed into the *(n-k)* stages of the feedback SR from the left as shown below. As soon as all the received bits have been shifted into the SR, its contents define the syndrome.



Syndrome Calculator for (n,k) cyclic Code.

**Example 5.9:** Let us revisit the *(7,4)* block code above.

1. Let us form the *4-by-7* generator matrix from *g(X):*

$$g(X) = 1 + X + X^3$$
$$X.g(X) = X + X^2 + X^4$$
$$X^2.g(X) = X^2 + X^3 + X^5$$
$$X^3.g(X) = X^3 + X^4 + X^6$$

If we use the coefficients of the terms above in a matrix form we obtain a generator matrix:

$$G^\circ = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \Rightarrow \underline{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

As it is clear from the first form the generator matrix is not systematic. However, it can be made so by adding the sum of the first two rows to the last row to have the generator matrix identical to the result above.

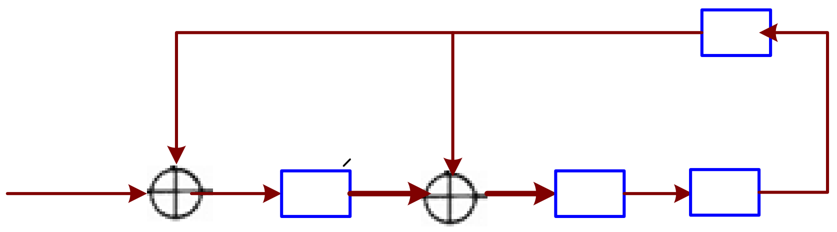
2. Let us now generate *3-by-7* parity-check matrix from the parity-check polynomial *h(X):*

$$X^4.h(X^{-1}) = 1 + X^2 + X^3 + X^4$$
$$X^5.h(X^{-1}) = X + X^3 + X^4 + X^5$$
$$X^6.h(X^{-1}) = X^2 + X^4 + X^5 + X^6$$

Using the coefficients from above we have a parity-check matrix, which is not systematic again. To put it into a systematic form, we add the third two to the first row to obtain the corresponding systematic parity-check matrix. This matrix is exactly the same as that of the previous example.

$$H^\circ = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \Rightarrow \underline{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

3. Below we present the corresponding syndrome calculator circuit for this *(7,4)* cyclic Hamming code. Let the transmitted codeword be (0111001) and the received word be (0110001) with the middle bit being in error. As the received bits are fed into SR, which were originally set to 0, its contents are modified as in the table below.

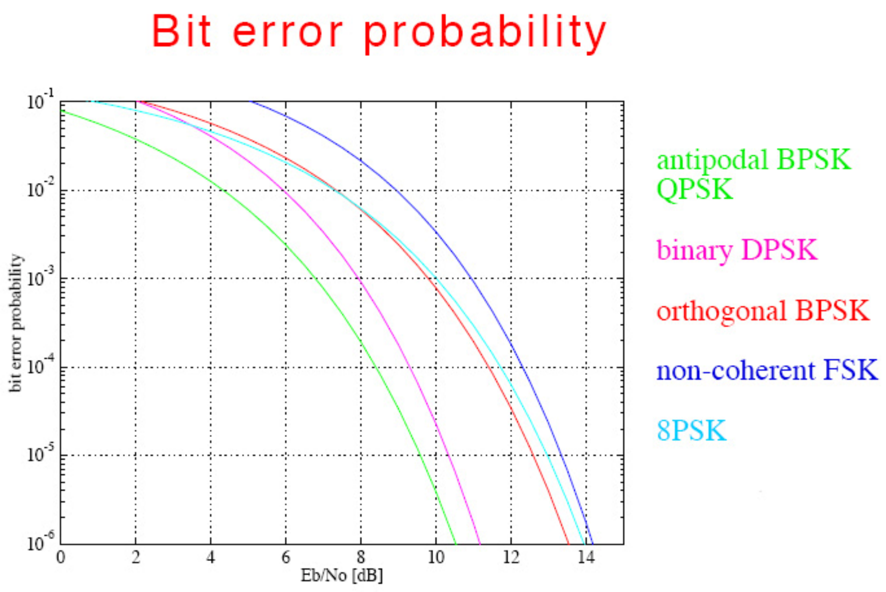


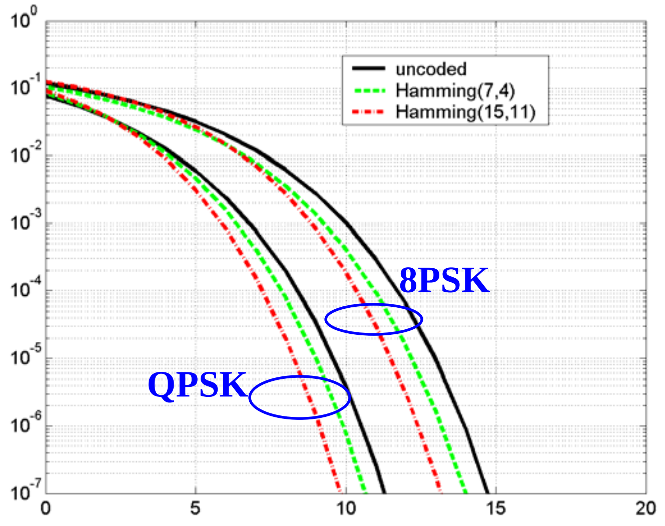
Syndrome calculator for the (*7,4*) cyclic code generated by $g(X) = 1 + X + X^3$.

| **Contents of the Syndrome Calculator for the received word: 0110001** | | |
|---|---|---|
| **Shift** | **Input Bit** | **Content of SR** |
| | | 000 (Initial State) |
| 1 | 1 | 100 |
| 2 | 0 | 010 |
| 3 | 0 | 001 |
| 4 | 0 | 110 |
| 5 | 1 | 111 |
| 6 | 1 | 001 |
| 7 | 0 | 110 |

At the end of the seventh shift, the syndrome is identified from the contents of SR as 110. From the table presented with the previous example, the error pattern corresponding to this syndrome is (0001000) indicating the error is in the middle bit.

## Performance of Hamming Coded Systems:





## 5.4 BCH and Reed-Solomon Codes and Examples

### BCH Codes:

Two of the most important and powerful classes of linear block codes are Bose, Chaudhuri, and Hocquenghem (BCH) and its subclass of non-binary Reed-Solomon (RS) codes names after their inventors. They are also known as *t-error correcting codes*. Primitive BCH codes offer design flexibility for integers $m \geq 3$ and $t \leq (2^m - 1)/2$ by the following parameters:

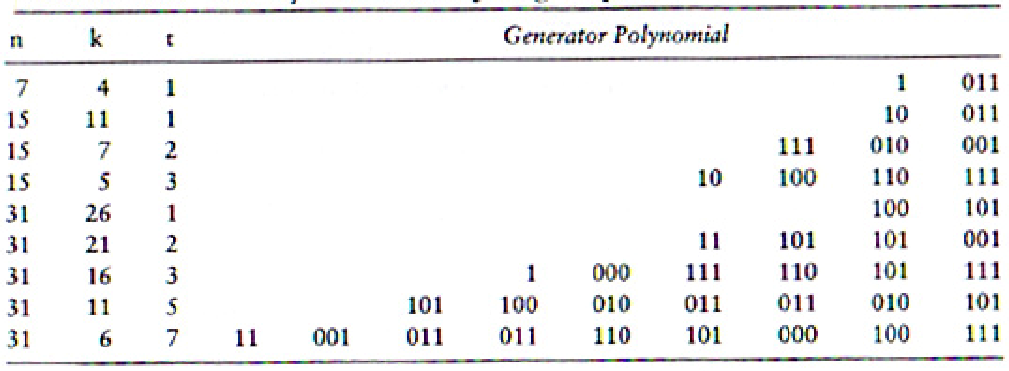
$$\textit{Block length}: \quad n = 2^m - 1 \tag{5.34a}$$

$$\textit{Message Size (bits)}: \quad k \geq n - mt \tag{5.34b}$$

$$\textit{Minimum Distance}: \quad d_{\min} = 2t + 1 \tag{5.34c}$$

As it can be seen from these parameters, each BCH code is a t-error correcting code. There are many good coding theory references on designing and decoding BCH codes. Among them, Haykin in his chapter 10 (p.653) has a nice table listing several binary BCH codes.

TABLE 10.6 *Binary BCH codes of length up to $2^5 - 1$*



| n | k | t | *Generator Polynomial* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 4 | 1 | | | | | | | | 1 | 011 |
| 15 | 11 | 1 | | | | | | | | 10 | 011 |
| 15 | 7 | 2 | | | | | | | 111 | 010 | 001 |
| 15 | 5 | 3 | | | | | | 10 | 100 | 110 | 111 |
| 31 | 26 | 1 | | | | | | | | 100 | 101 |
| 31 | 21 | 2 | | | | | | 11 | 101 | 101 | 001 |
| 31 | 16 | 3 | | | | 1 | 000 | 111 | 110 | 101 | 111 |
| 31 | 11 | 5 | | | 101 | 100 | 010 | 011 | 011 | 010 | 101 |
| 31 | 6 | 7 | 11 | 001 | 011 | 011 | 110 | 101 | 000 | 100 | 111 |

### Reed-Solomon (RS) Codes:

RS codes are very effective in channels with memory and when the set of input symbols is large, where burst-errors are likely to occur. These codes operate on multiple bits rather than individual bits, not like linear binary codes.

- An RS ($n$,$k$) code is used for encoding m-bit symbols into blocks consisting of $n = 2^m - 1$ symbols, that is, $m(2^m - 1)$ bits, where $m \geq 1$.
    - The encoder expands a block of *k* symbols to *n* symbols by adding *n-k* redundant symbols.
- When *m* is an integer power of 2, the *m-bit* symbols are called bytes. Most popular RS codes are 8-bit ones.
- An t-error correcting RS code has the following parameters:

$$\textit{Block length}: \quad n = 2^m - 1 \tag{5.35a}$$

$$\textit{Message Size (symbols)}: \quad k \tag{5.35b}$$

$$\textit{Parity-check Size (Symbols)}: n - k = 2t \tag{5.35c}$$

$$\textit{Minimum distance}: \quad d_{\min} = 2t + 1 \; \textit{symbols} \tag{5.35d}$$

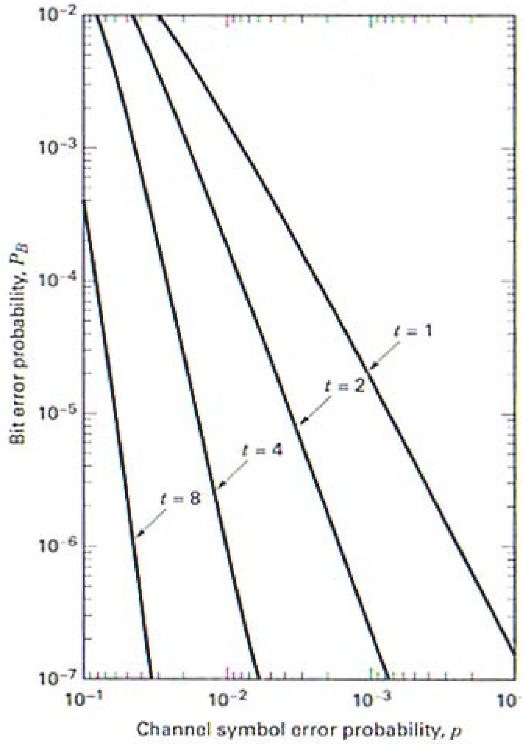
**Error Probability of RS Codes:** The BER of an RS code where the channel (usually, BSC) error probability is $p$ is approximated by:

$$P_E \approx \frac{1}{2^m - 1} \cdot \sum_{j=t+1}^{2^m - 1} j \binom{2^m - 1}{j} p^j (1-p)^{2^m - 1 - j} \tag{5.37a}$$

where $\begin{pmatrix} k \\ n \end{pmatrix}$ is the usual binary combinatorial coefficient. For a specific modulation technique this result can be turned into a BER. For instance, for M-PSK with $M = 2^m$, we have:

$$BER = P_B = \frac{2^{m-1}}{2^m - 1}.PE \tag{5.37b}$$

Following figure is included to illustrate the improvements in channel performance for 32-ary orthogonal signaling anf n=21, t-error correcting RS Coding[2].



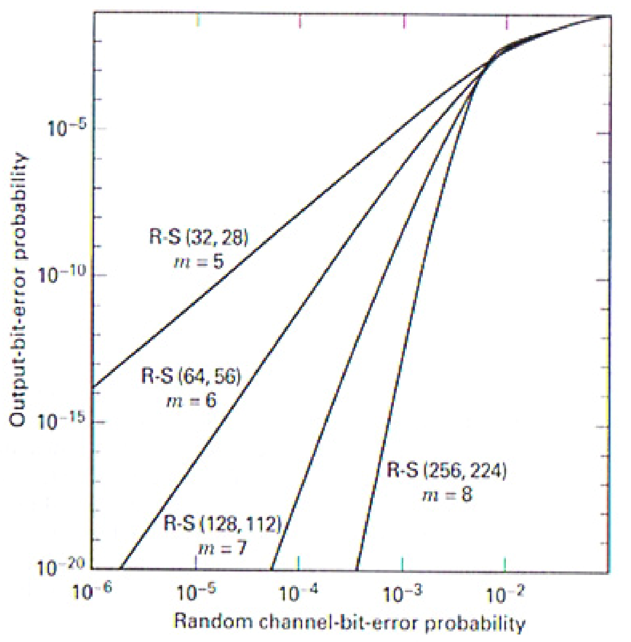


**Figure 8.4** Reed–Solomon, rate 7/8, decoder performance as a function of symbol size.

- RS codes achieve the largest possible $d_{\min}$ of any linear block code. Even though there are many GF choices, but the field $GF(64)$ is the most common one and *m=6* is the default size in many satellite communication designs. To get familiar, we start from the binary field GF(2) and extend to GF(*$2^m$*), which can be represented by a power of $\alpha$. An infinite set of elements, *F*, is obtained by starting with elements $\{0, 1, \alpha\}$, additional elements are progressively generated:

$$F = \{0, \alpha^0 = 1, \alpha, \alpha^2, \cdots, \alpha^j, \cdots\} \tag{5.36a}$$

- To obtain a finite set of elements of GF(*$2^m$*) from *F*, the condition of "closed under multiplication" is imposed, which results in fact that the elements satisfy:

$$\alpha^{2^m - 1} = 1 - \alpha \quad \text{and} \quad \alpha^{2^m + n} = \alpha^{2^m - 1}.\alpha^{n+1} = \alpha^{n+1} \tag{5.36b}$$

$$F = \{0, \alpha^0 = 1, \alpha, \alpha^2, \cdots, \alpha^{2^m - 2}, \alpha^0, \alpha, \alpha^2, \cdots\} \tag{5.36c}$$

- Addition in GF(*$2^m$*): It is the modulo-2 sum of each of the polynomial coefficients involved.

[2] This section has an excellent coverage in Sklar's Book, Chapter 8. The result and the associated graph was published originally in J.P. Odenwalder, *Error Control Coding Handbook*, M/A COM LINKABIT, Inc, San Diego, CA., July 15, 1976 also appeared in T.C. Bartee, *Data Communications*, H.W. Sams Co. 1981.

**Primitive Polynomials:** RS codes are generated from primitive polynomials. An irreducible polynomial $f(X)$ of degree $m$ is primitive if the smallest positive integer $n$ for which divides $X^m + 1$. An example is GF($2^3$)= GF(8) will be $f(X) = 1 + X + X^3$. The mapping for this polynomial is shown below.

| | | **Field Elements** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Basis Elements** | | 0 | $\alpha^0$ | $\alpha^1$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ | $\alpha^7$ |
| | $X^0$ | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| | $X^1$ | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| | $X^2$ | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

Recall that GF($2^3$)= GF(8) has 8 elements in the field, and hence solving for the roots of $f(X) = 0$, will result in three distinct roots for the given primitive polynomial $f(X) = 1 + X + X^3$. Hence, this 8-order polynomial must have 8 roots and they lie in the extension field of GF(8). Similarly, we will need to work with other primitive polynomials for higher orders as shown in the table below.

| $m$ | $f(X)$ |
|---|---|
| 3 | $1 + X + X^3$ |
| 4 | $1 + X + X^4$ |
| 5 | $1 + X^2 + X^5$ |
| 6 | $1 + X + X^6$ |
| 7 | $1 + X^3 + X^7$ |
| 8 | $1 + X^2 + X^3 + X^4 + X^8$ |

**Example 5.10:** Using the results until now GF($2^3$)= GF(8) will have

$f(\alpha) = 1 + \alpha + \alpha^3 = 0$ ; with $\alpha^3 = -1 - \alpha$ and using +1=-1 $\Rightarrow \alpha^3 = \alpha + 1$

Similarly,

$$\Rightarrow \alpha^4 = \alpha.(\alpha^3) = \alpha.(\alpha + 1) = \alpha^2 + \alpha^1$$
$$\Rightarrow \alpha^5 = \alpha.(\alpha^4) = \alpha.(\alpha^2 + \alpha^1) = \alpha^3 + \alpha^2 = 1 + \alpha + \alpha^2$$
$$\Rightarrow \alpha^6 = \alpha.(\alpha^5) = \alpha.(1 + \alpha + \alpha^2) = \alpha^3 + \alpha^2 + \alpha = 1 + \alpha^2$$
$$\Rightarrow \alpha^7 = \alpha.(\alpha^6) = \alpha.(1 + \alpha^2) = \alpha^3 + \alpha = 1 = \alpha^0$$

Thus the elements of GF($2^3$)= GF(8) will be

$$GF(8) = \{0, \alpha^0, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\} \tag{5.38}$$

This equation is frequently illustrated with the Linear Feedback Shift Register (LFSR) circuit. Addition and multiplication tables are also shown below.

**Figure 8.8** Extension field elements can be represented by the contents of a binary linear feedback shift register (LFSR) formed from a primitive polynomial.

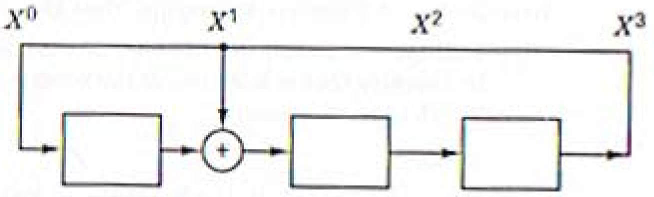
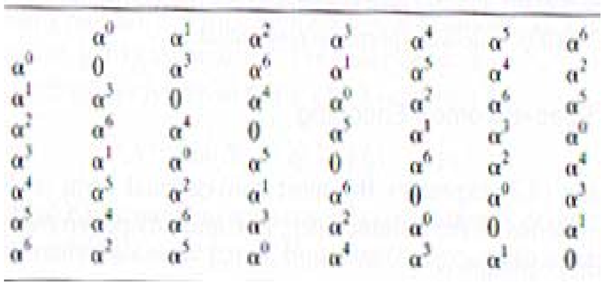
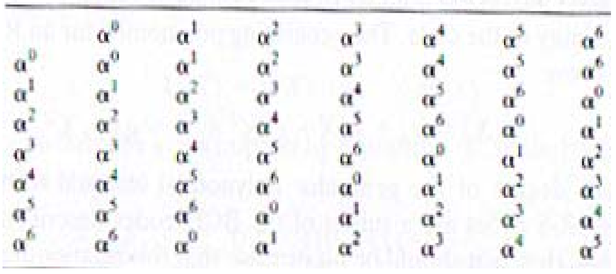


TABLE 8.2 Addition Table for GF(8) with $f(X) = 1 + X + X^3$



| | $\alpha^0$ | $\alpha^1$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ |
|---|---|---|---|---|---|---|---|
| $\alpha^0$ | 0 | $\alpha^3$ | $\alpha^6$ | $\alpha^1$ | $\alpha^5$ | $\alpha^4$ | $\alpha^2$ |
| $\alpha^1$ | $\alpha^3$ | 0 | $\alpha^4$ | $\alpha^0$ | $\alpha^2$ | $\alpha^6$ | $\alpha^5$ |
| $\alpha^2$ | $\alpha^6$ | $\alpha^4$ | 0 | $\alpha^5$ | $\alpha^1$ | $\alpha^3$ | $\alpha^0$ |
| $\alpha^3$ | $\alpha^1$ | $\alpha^0$ | $\alpha^5$ | 0 | $\alpha^6$ | $\alpha^2$ | $\alpha^4$ |
| $\alpha^4$ | $\alpha^5$ | $\alpha^2$ | $\alpha^1$ | $\alpha^6$ | 0 | $\alpha^0$ | $\alpha^3$ |
| $\alpha^5$ | $\alpha^4$ | $\alpha^6$ | $\alpha^3$ | $\alpha^2$ | $\alpha^0$ | 0 | $\alpha^1$ |
| $\alpha^6$ | $\alpha^2$ | $\alpha^5$ | $\alpha^0$ | $\alpha^4$ | $\alpha^3$ | $\alpha^1$ | 0 |

TABLE 8.3 Multiplication Table for GF(8) with $f(X) = 1 + X + X^3$



| | $\alpha^0$ | $\alpha^1$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ |
|---|---|---|---|---|---|---|---|
| $\alpha^0$ | $\alpha^0$ | $\alpha^1$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ |
| $\alpha^1$ | $\alpha^1$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ | $\alpha^0$ |
| $\alpha^2$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ | $\alpha^0$ | $\alpha^1$ |
| $\alpha^3$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ | $\alpha^0$ | $\alpha^1$ | $\alpha^2$ |
| $\alpha^4$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ | $\alpha^0$ | $\alpha^1$ | $\alpha^2$ | $\alpha^3$ |
| $\alpha^5$ | $\alpha^5$ | $\alpha^6$ | $\alpha^0$ | $\alpha^1$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ |
| $\alpha^6$ | $\alpha^6$ | $\alpha^0$ | $\alpha^1$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ |

**Encoding Stage:** A Reed-Solomon coding system uses the following six polynomials in its structure:

| | | |
|---|---|---|
| 1. | Raw Information Polynomial | = *d(x)* |
| 2. | Parity Polynomial | = *p(x)* |
| 3. | Codeword Polynomial | = *c(x)* |
| 4. | Generator Polynomial | = *g(x)* |
| 5. | Quotient Polynomial | = *q(x)* |
| 6. | Remainder Polynomial | = *r(x)* |

In terms of these polynomials, an encoded RS polynomial is simply:

$$c(X) = d(X) + p(X) = \sum_{i=0}^{n-1} c_i . x^i \tag{5.39}$$

where $(c_0, c_1, ..., c_{n-1})$ is a codeword *IFF* it is a multiple of the generator polynomial *g(X)*, which is of the form:

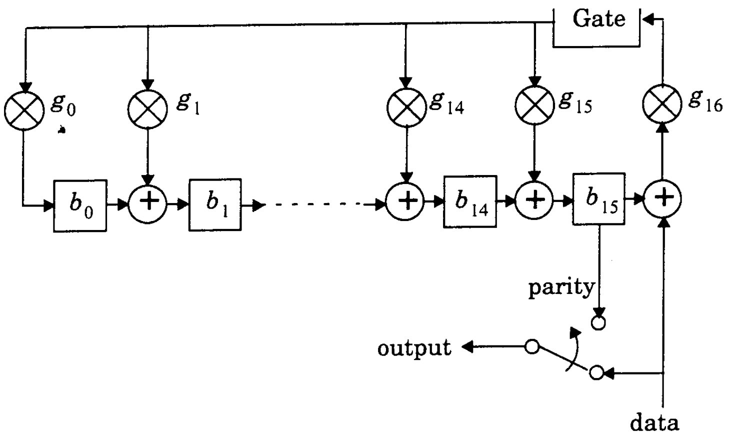
$$g(X) = (x + \alpha).(x + \alpha)...(x + \alpha^{2t}) = \sum_{i=0}^{2t} g_i . x^i \tag{5.40}$$

A common method for encoding an RS code is to derive *p(X)* by diving *d(X)* by *g(X)*, which yields an irrelevant quotient polynomial *q(X)* and a relevant remainder polynomial *r(X):*

$$c(X) = p(X) + g(X).q(X) + r(X) \tag{5.41a}$$

If the parity polynomial is defined as being equal to the negatives of the coefficients of *r(X)* then we have (5.41a) simplifies and a block diagram for this implementation is shown below.

$$c(X) = g(X).q(X) \tag{5.41b}$$



In this block diagram, each "+" represents an exclusive-OR of two m-bit numbers, each "X" represents a multiplication of two m-bit numbers under *GF(2^m^)* and each m-bit register contains an m-bit number $b_i$. Initially, all registers are "0", the switch is in "data" position. Code symbols $c_{n-1}, c_{n-2}, ..., c_{n-k}$ are sequentially shifted in to the circuit. As soon as the last code element enters, the switch goes to "Parity" position, the gate is OFF to cut-off the feedback loop. At the same instant, registers $b_0, b_1, ..., b_{2t-1}$ contain the parity symbols $p_0, p_1, ..., p_{2t-1}$. These are sequentially shifted to output.

- ***SR Decoding:*** Let the transmitted codeword and the corrupted receiver input code be:

$$c(X) = v_0 + ... + v_{n-1}.x^{n-1}; \qquad r(X) = r_0 + ... + r_{n-1}.x^{n-1} \tag{5.42}$$

and the error pattern defined as the difference polynomial between the two:

$$e(X) = r(X) - c(X) = e_0 + e_{n-1}.x^{n-1} \tag{5.43}$$

Let the 2t-partial syndromes: $\{S_0,.S_2,..,S_{2t}\}$ be defined as:

$$S = r(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i) \tag{5.44}$$

which depends only on the error pattern. The first term in middle equality is zero due to the fact that each codeword is a multiple root of *g(X)* and $c(\alpha^i) = 0$. To locate and correct errors we need to develop an error locator polynomial.

Suppose *e(X)* contains *k* errors, and $k \le t$, at locations: $x^{j_1}, x^{j_2}, ..., x^{j_k}$ and let the error magnitude at each location be $e_{j_i}$.

$$e(X) = e_{j_1}.x^{j_1} + e_{j_2}.x^{j_2} + ... + e_{j_k}.x^{j_k} \tag{5.45}$$

Now, define the set of error locator numbers

$$\beta_i = \alpha^{j_i} \quad for \quad i = 1,2,...,k \tag{5.46}$$

Then the set of 2t-partial syndromes define the following system of equations, also known as the linear normal-equations:

$$\begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t} \end{bmatrix} = \begin{bmatrix} e_{j1}.\beta_1 + e_{j2}.\beta_2 + \cdots + e_{jk}.\beta_k \\ e_{j1}.\beta_{j1}^2 + e_{j2}.\beta_{j2}^2 + \cdots + e_{jk}.\beta_k^2 \\ \vdots \\ e_{j1}.\beta_{j1}^{2t} + e_{j2}.\beta_{j2}^{2t} + \cdots + e_{jk}.\beta_k^{2t} \end{bmatrix} \tag{5.47}$$

Any algorithm that solves (5.47) is a Reed-Solomon Decoding Algorithm.

- **Decoding Algorithms:** Typical RS Decoders employ five distinct steps:

**Step 1: Calculate 2t partial syndromes** as the remainder polynomial obtained from the received code polynomial and evaluating it at $x = \alpha^i$:

$$r(X) = q(X).(x + \alpha^i) + rem \Rightarrow r(\alpha^i) = q(\alpha^i).(\alpha^i + \alpha^i) + rem = rem = S_i \tag{5.48}$$

Evaluation of (5.48) can be implemented very efficiently in software by arranging the function so that we have a recursive form:

$$r(\alpha^i) = (\ldots((r_{n-1}.\alpha^i + r_{n-2})\alpha^i + r_{n-3})\alpha^i + \cdots)\alpha^i + r_0 \tag{5.49}$$

**Step 2: Error-locator polynomial computation:** In this stage Berlekamp-Massey Algorithm is normally used. Let us define a new polynomial called "error-locator polynomial" in terms of the coefficient set in (5.47):

$$\sigma(X) = (1+\beta_1.x)....(1+\beta_k.x^k) = \sigma_0 + \sigma_1.x + \ldots + \sigma_k.x^k \tag{5.50}$$

and the relations between the coefficients of the last polynomial and the error-location numbers:

$$\begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_k \end{bmatrix} = \begin{bmatrix} 1 \\ \beta_1 + \beta_2 + \cdots + \beta_k \\ \beta_1.\beta_2 + \beta_2.\beta_3 + \cdots + \beta_{k-1}.\beta_k \\ \vdots \\ \beta_1.\beta_2.\beta_3 \cdots \beta_k \end{bmatrix} \tag{5.51}$$

Partial syndromes are related to these to yield “Newton’s Identities in vector notation:

$$\begin{bmatrix} S_1 + \sigma_1 \\ S_2 + \sigma_1.S_1 + 2\sigma_2 \\ S_3 + \sigma_1.S_2 + \sigma_2.S_1 + 3\sigma_3 \\ \vdots \\ S_k + \sigma_1.S_{k-1} + \cdots + \sigma_{k-1}.S_1 + k\sigma_k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5.52}$$

The most frequently used technique for solving (5.52) is known as the Berlecamp-Massey algorithm and it is covered in almost all coding theory textbooks.

**Step3: Actual error-location computation:** Berlecamp-Massey algorithm does not yield the actual locations of errors in a received word. Chien Search algorithm is used in literature to calculate these specific error locations from the error polynomial.

**Step4: Magnitude error computation at each location.**

**Step5: Error Correction:** Knowing both the error locations and the magnitudes of the error in each location, the errors, up to t of them, are corrected using a software error correction procedure.

## 5.5 Convolutional Codes with Examples

Convolutional codes are fundamentally different from the previous classes of codes, in that a continuous sequence of message bits is mapped into a continuous sequence of encoder output bits. It is well-known in the literature and practice that these codes achieve a larger coding gain than that with block coding with the same complexity. A convolutional encoder:

- encodes the entire data stream, into a single codeword.
- does not need to segment the data stream into blocks of fixed size (*Convolutional codes are often forced to block structure by periodic truncation*).
- is a state-machine with memory consisting of an M-stage shift register with prescribed connections to n modulo-2 adders, and a multiplexer that serializes the outputs of the adders.
- A convolutional code is specified by three parameters $(n, k, K)$ or $(k/n, K)$ where
  1. $R_C = k/n$ is the coding rate, determining the number of data bits per coded bit.
  2. K is the constraint length of the encoder a where the encoder has K-1 memory elements.

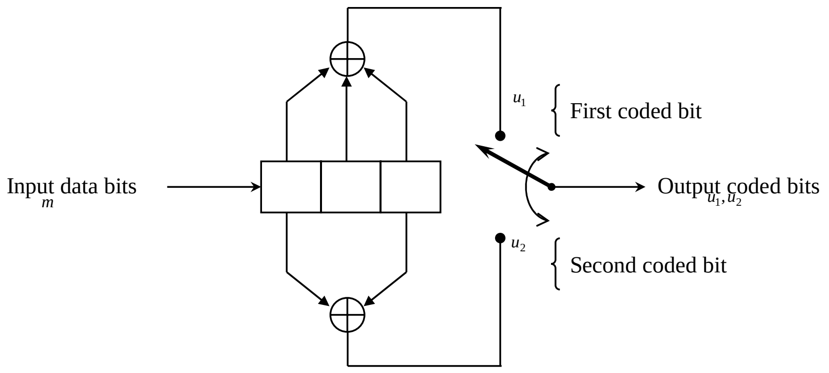
**Some definitions:**

1. **Code Rate:** A k-bit message sequence produces a coded output sequence of length $n$ bits and the code rate is defined by:

$$R_C = k/n \tag{5.53}$$

**2. Constraint Length:** K, the number of shifts over which a single message bit can influence the encoder output. For instance, in an encoder with an M-stage SR, the memory of the encoder equals M message bits, and $K = M + 1$ shifts are required for a message bit to enter and clear out. Hence, the constraint length of this coder is K. Below we have the most popular convolutional coder structure use din the field.

**Example 5.11:** Convolutional encoder (rate ½, K=3)
It is a machine with 3 shift-registers where the first one takes the incoming data bit and the rest, form the memory of the encoder.



**Connection Vectors:** We normally represent the encoder in terms of *n* connection vectors, one for each modulo-2 adder. For encoder above, they are:

$$Path\,1: \quad g_1 = 111 \text{ and } Path\,2: \quad g_1 = 101$$

**Impulse Response:** It is the response of the encoder to a single bit that moves through it.

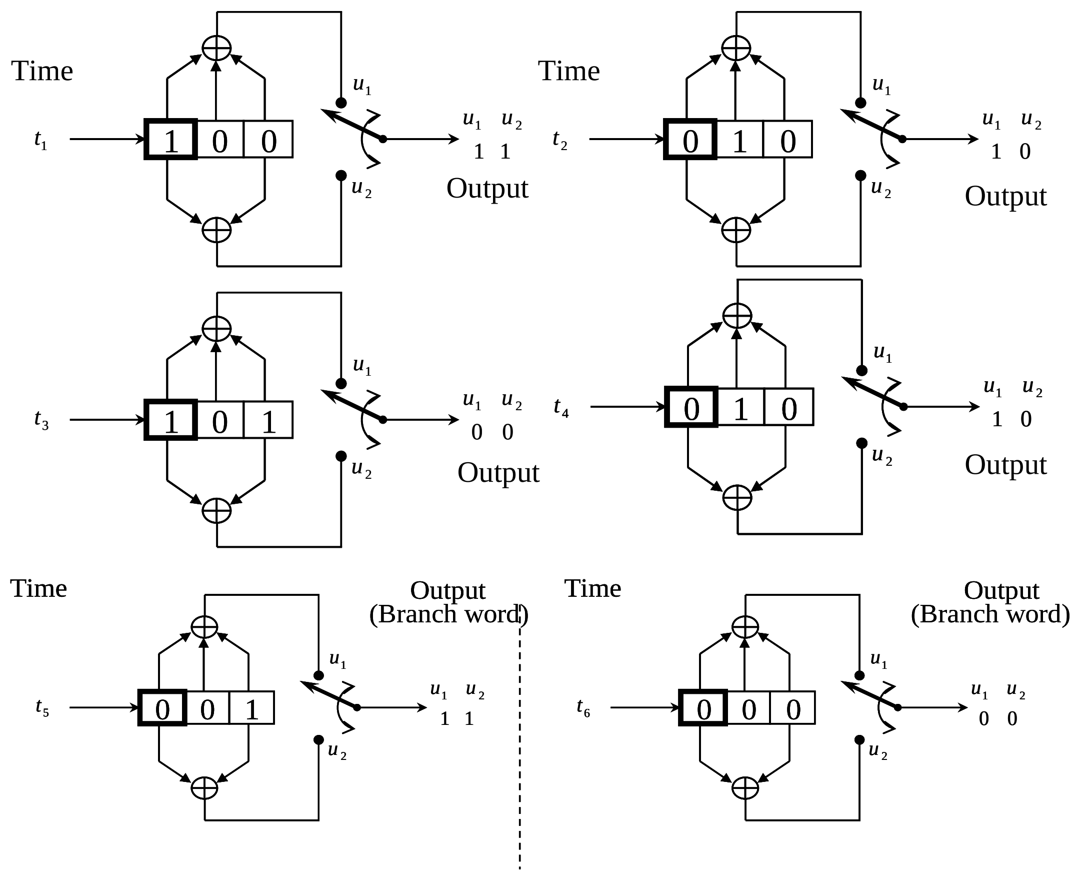
| Register Contents | Branch Words | |
|---|---|---|
| | $u_1$ | $u_2$ |
| 100 | 1 | 1 |
| 010 | 1 | 0 |
| 001 | 1 | 1 |

Input Sequence : 1 1 1
Output Sequence: 11 10 11

The output sequence for the input "one" is called the impulse response. The response for a three bit long message is simply the superposition of individual responses as shown below:

| **Input *m*** | **Output** | | | | |
|---|---|---|---|---|---|
| 1 | 11 | 10 | 11 | | |
| 0 | | 00 | 00 | 00 | |
| 1 | | | 11 | 10 | 11 |
| **Mod-2 Sum** | 11 | 10 | 00 | 10 | 11 |

This is shown in an illustrated manner below (from Sklar Figure 7.4)



**Result:**

$$\begin{tikzcd}\mathbf{m} = (101) \arrow[r] & \boxed{\text{Encoder}} \arrow[r] & \mathbf{U} = (11\ \ 10\ \ 00\ \ 10\ \ 11)\end{tikzcd}$$

**Effective code rate:**

- Initialize the memory before encoding the first bit (all-zero)
- Clear out the memory after encoding the last bit (all-zero); i.e. a tail of zero-bits is appended to data bits.
- m is the number of data bits and *k=1* is assumed:

$$\begin{tikzcd}\boxed{\text{data}}\,\boxed{\text{tail}} \arrow[r] & \boxed{\text{Encoder}} \arrow[r] & \boxed{\text{codeword}}\end{tikzcd}$$

$$R_{eff} = \frac{m}{n(m+K-1)} < R_c$$

**Generator Polynomial:** If D is the unit-delay variable, then the generator polynomial for the ith path is defined by:

$$g^i(D) = g_0^i + g_1^i D + g_2^i D^2 + \cdots + g_M^i D^M \qquad (5.54)$$

Then the complete encoder is described by an appropriate set of polynomials.

**Example 5.11 (Redone):** Consider the rate ½ coder above with two paths 1 and 2.

- Impulse response of path 1: (1,1,1) and the corresponding generator polynomial is given by:

$$g^1(D) = 1 + D + D^2$$

Similarly, the impulse response for path 2: (101) results in a generator:

$$g^2(D) = 1 + D^2$$

- Let us assume the input sequence is : 10011, which can be written in terms of a polynomial:

$$m(D) = 1 + D^3 + D^4$$

- As in the FFT, the convolution in the time-domain is transformed into multiplication in the D-domain. We can write output polynomials and output sequences for paths 1&2 in the form:

$$c^1(D) = g^1(D).m(D) = (1 + D + D^2)(1 + D^3 + D^4) = 1 + D + D^2 + D^3 + D^6 = 1111001$$

$$c^2(D) = g^2(D).m(D) = (1 + D^2)(1 + D^3 + D^4) = 1 + D^2 + D^3 + D^4 + D^5 + D^6 = 1011111$$

Finally, the output is obtained by multiplexing these two sequences:

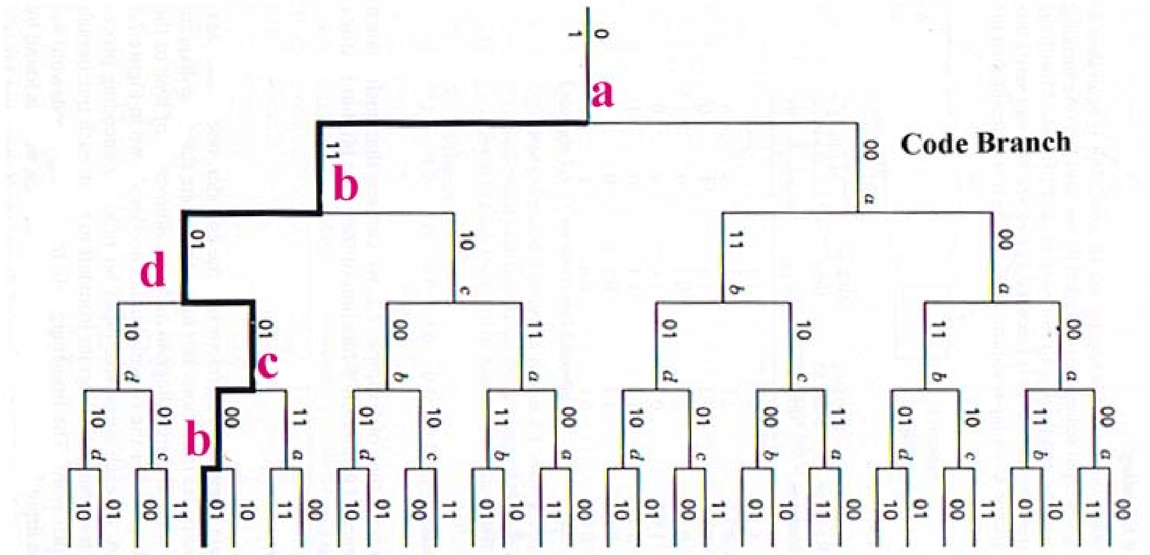
$$\underline{c} = (11,10,11,11,01,01,11)$$

**Note 1:** The message sequence of length L=5 bits produced an encoded sequence of length n(L+K+-1)= 14 bits.
**Note 2 :** We need to append a terminating sequence of K-1=2 zeros to the last input of the message sequence to restore the shift register to its zero initial state. These terminating sequence of K-1 zeros is called the *tail of the message*.

## Code Tree, Trellis and State Diagram Representations:

Structural properties of convolutional codes are normally portrayed in graphical form by using one of three equivalent diagrams.

1. **Code Tree** is a complete tree. That is, every node of a binary tree has two off-springs. (The case is similar for M-ary trees). An input “0” specifies the upper branch or left-child of a bifurcation, whereas “1” is for the lower branch or right-child. A specific path in the tree is traced from left-to-right in accordance with the input message sequence. Each branch of the input tree is labeled by the n digits of the output associated therewith.

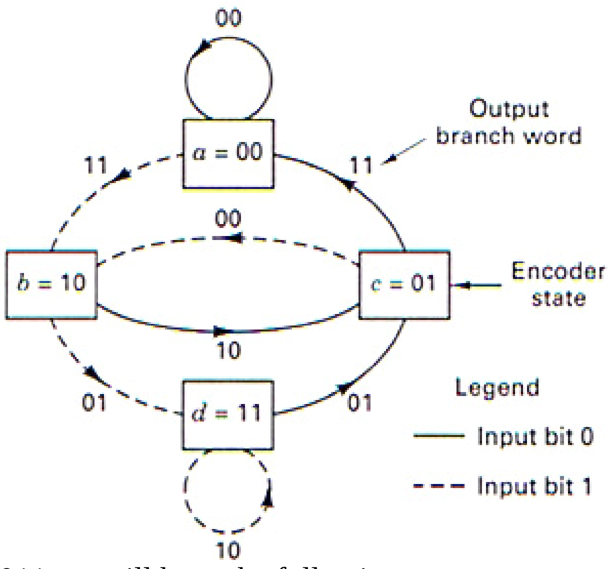


**Example 5.11 (Redone):** As it can be seen from the tree diagram, the input sequence (10011) when applied to the ½ rate coder in the example we have been studying for some time now, we find the output as (11,10,11,11,01), which agrees with the first five pairs of bits in the output sequence.

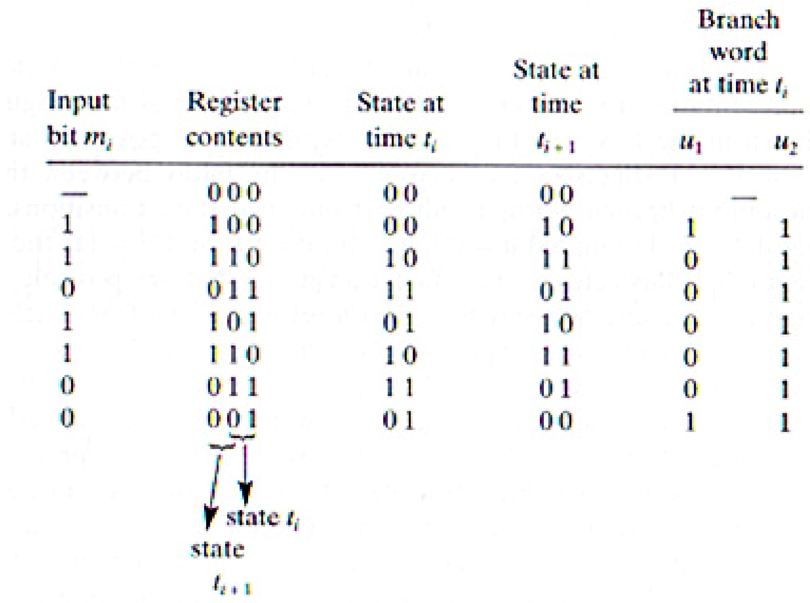
- We can also observe that the tree becomes repetitive after the first three branches. For instance, two nodes labeled "a" are identical, so are all the other node pairs. This is due to the fact that the encoder has memory M=K-1=2 message bits. Hence, when the third bit enters the encoder, the first bit is shifted out from the SR. Consequently, the pairs of nodes labeled "a" generate the same code symbols.

Therefore, we can collapse the tree into a form called "*trellis*." Hence, a more instructive structure of "trellis" description of convolutional codes emerges. Before we do that it is logical to present the state diagram for this encoder, which will be used in building the trellis later.

2. **State Diagram:** State machine representation for a rate ½ K=3 coder of the previous example is shown below.



For the message: $m = 11011$ we will have the following output sequence generated:



| Input bit $m_i$ | Register contents | State at time $t_i$ | State at time $t_{i+1}$ | Branch word at time $t_i$ $u_1$ | $u_2$ |
|---|---|---|---|---|---|
| — | 0 0 0 | 0 0 | 0 0 | — | |
| 1 | 1 0 0 | 0 0 | 1 0 | 1 | 1 |
| 1 | 1 1 0 | 1 0 | 1 1 | 0 | 1 |
| 0 | 0 1 1 | 1 1 | 0 1 | 0 | 1 |
| 1 | 1 0 1 | 0 1 | 1 0 | 0 | 1 |
| 1 | 1 1 0 | 1 0 | 1 1 | 0 | 1 |
| 0 | 0 1 1 | 1 1 | 0 1 | 0 | 1 |
| 0 | 0 0 1 | 0 1 | 0 0 | 1 | 1 |

state $t_i$
state $t_{i+1}$

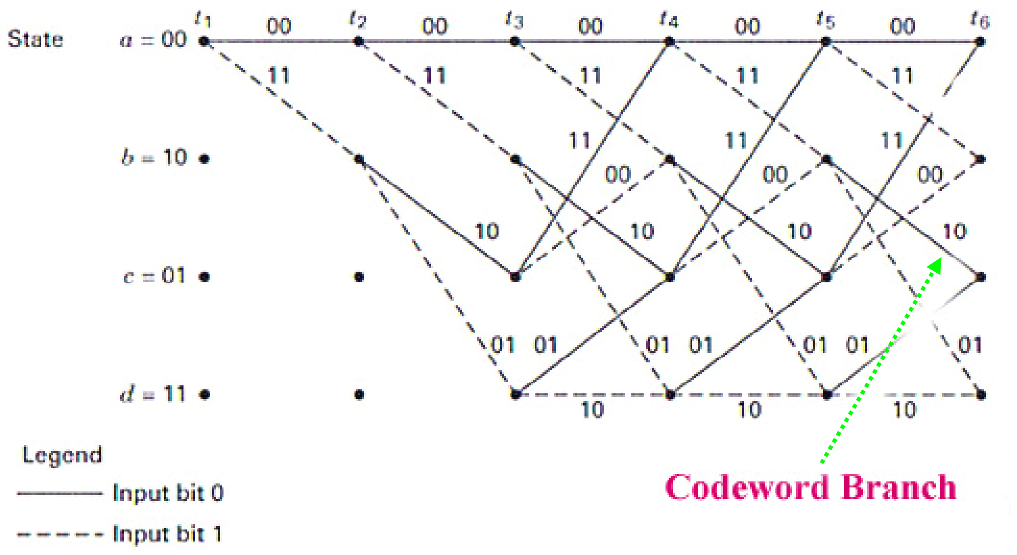
The resultant output sequence including K-1=2 zeros to flush out the register will be:

**U**= 11 01 01 00 01 01 11

3. **Code Trellis:**

Trellis structures use the following convention:

- A code branch produced by an input "0" is drawn as a solid line, whereas a code produced by an input "1" is a dashed line.
- Each input (message) sequence corresponds to a specific path along the trellis. For instance, the message (10011) produces the output coded sequence of di-bits (11,10,11,11,01), that agrees with our previous results.



A trellis is strikingly more instructive than a tree since it brings out explicitly the finite-state machine structure as we have labeled them at the extreme left of the above figure. Here {a,b,c,d} are the state labels of our machine. Some terminology:

- **State:** Each (K-1)-bits stored in the encoder's shift registers represent identifies a state. Given that the current bit is $m_j$ at time j then the portion of the message sequence containing the most recent K-bits are written as: $(m_{j-K+1},\cdots,m_{j-1},m_j)$. In the case of rate ½ encoder with K-1=2, we have four distinct states {a,b,c,d} and the corresponding state-diagram is very simple.
- **Levels (Depths):** A trellis contains (L+K)-levels, where L is the length of the incoming message sequence, and K is the constraint length of the code. The levels of a trellis are labeled as j=0,1,2,…,L+K-1. The first (K-1)-levels correspond to the encoder's departure from the initial state and the last (K-1)-levels correspond to the encoder's return to the same state. It is worth pointing out that not all the states can be reached in these two portions of a trellis. However, in the central portion of a trellis, all the states of an encoder are reachable.

## ML Decoding and Viterbi Algorithm:

Let $\underline{m}$ be a message vector, the code vector going into a discrete memoryless channel and $\underline{r}$ be the corresponding received vector. The decoder makes an estimate $\hat{\underline{m}}$ of the message. Since there is one-to-one correspondence, the decoder may equivalently produce an estimate $\hat{\underline{c}}$ of the code vector. We can now reformulate the ML decoder principle:

**ML Decoding Rule:** Choose the estimate $\hat{\underline{c}}$, for which the log-likelihood function $\log_e(P(\underline{r}\,|\,\underline{c})$ is maximum. If the channel is BSC then this rule simplifies to the following case:

Choose the estimate $\underline{\hat{c}}$ that minimizes the Hamming Distance between the received vector $\underline{r}$ and the transmitted code vector $\underline{c}$. In this case, ML decoder is again a minimum distance decoder. The Viterbi Algorithm implements this rule best.

**Viterbi Algorithm:**

The equivalence between ML and minimum distance decoding for a BSC implies that we may decode a convolutional code by choosing a path in the code tree or trellis whose coded sequence differs from the received sequence in the fewest number of places. Let us consider the case of rate ½ and K=3 decoder of the previous example. At level j=3, there are two paths entering any of the four possible nodes in the trellis and they are identical onward from that point! So, the task of minimum distance decoder is to make a decision at that point as to which of these two paths to retain, without loss of performance. Similarly, we need to do the same for j=4 and on. Viterbi Algorithm is designed for that task by computing a metric (usually, the Hamming distance) for every possible path in the trellis. The paths retained are called *survivor or active paths*. In the case of multiple paths entering a state with identical metric, we employ a tie-braking rule.

More comprehensive treatment of Viterbi Algorithm and its applications to convolutional and trellis coded modulation will be presented as a student project later in the semester.

## 5.6 Trellis Coded Modulation and Examples of Ungerboeck Codes

Until now, encoding is performed separately from modulation in the transmitter and likewise for detection and decoding in the receiver. Moreover, error control provided by sending additional bits as parity-check information lowers the baud rate per channel bandwidth. To attain more effective use of the available bandwidth and power, combined coding and modulation is shown to be the way. By doing so, we redefine the coding process as *the process of imposing certain patterns on the transmitted signal*. Trellis codes for band-limited channels result in a combined entity called “Trellis-Coded Modulation (TCM)” due to the sentinel work by Ungerboeck in late seventies. It has three basic features:

1. Number of signal points in the constellation is larger than what is required for; the additional points allow redundancy for forward error-control (FEC) coding without sacrificing bandwidth.
2. Convolutional coding is used for introducing a certain dependency among sequences of points.
3. Soft-decision decoding is performed in the receiver using trellis structures.

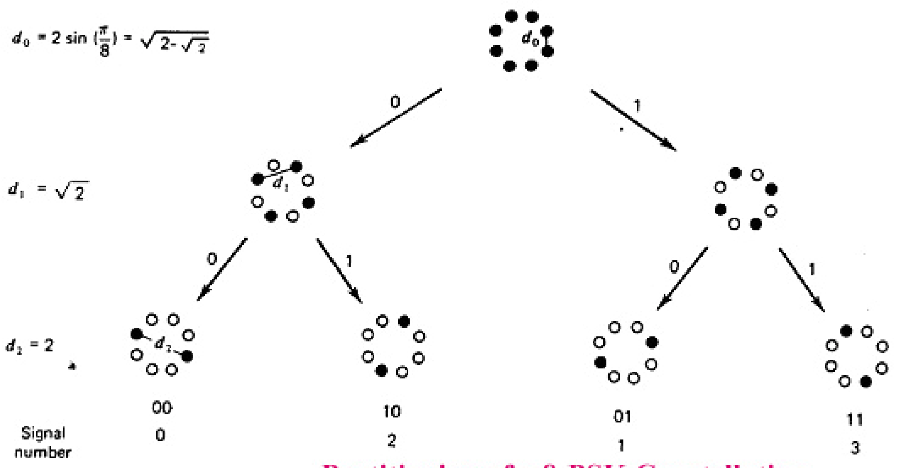
**TASK:** In an AWGN regime, ML decoding of trellis codes is to find that particular path through the trellis with Minimum -Squared Euclidean (MSE) distance to the received sequence.

- This mse distance restricts the choice of modulation schemes to BPSK and QPSK. This is due to the fact that the maximizing Hamming distance is NOT equivalent to MSE for other codes.
- Even though, a more general treatment is possible, the systems designed are traditionally confined to two-dimensional constellations as described by the following algorithm.

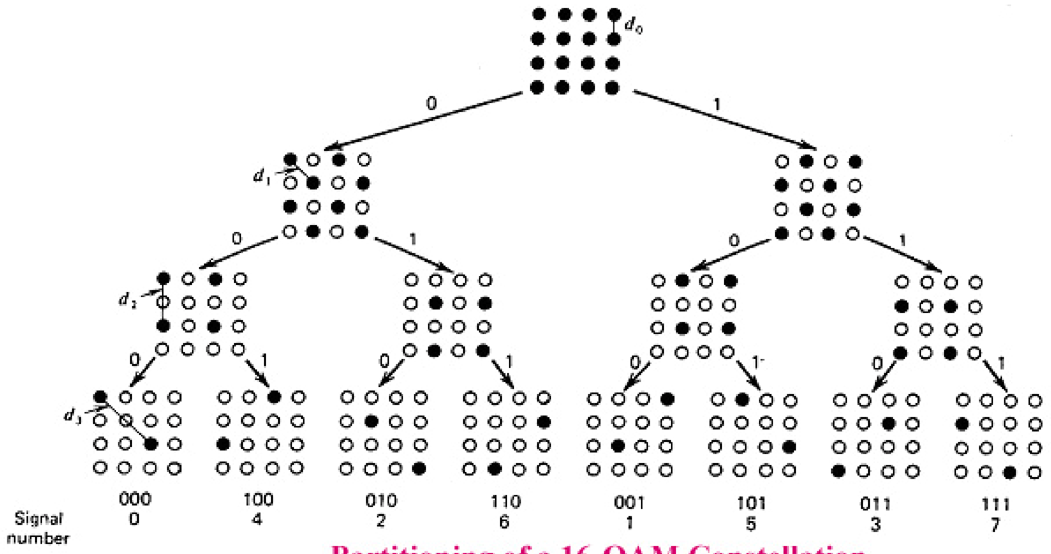
**Partitioning:**The approach used for designing this type of trellis codes involves partitioning an M-ary constellation successively into 2,4,8,… subsets with size M/2, M/4,… and having progressively larger increasing MSE distance between their respective signal points, thereby more efficient coded modulation for band-limited channels is possible. Below we have two partitioning, one for 8-PSK and another one for 16-QAM with respective MSE distances in an increasing order:

$$d_0 = 2\sin(\pi/8) < d_1 = \sqrt{2} < d_2 = 2 \qquad for \quad 8-PSK \tag{5.55a}$$

$$d_0 < d_1 = \sqrt{2d_0} < d_2 = 2d_0 < d_3 = 2\sqrt{2d_0} \qquad for \quad 16-QAM \tag{5.55b}$$



Partitioning of a 8-PSK Constellation



Partitioning of a 16-QAM Constellation

This subject will also be a part of a student presentation.

**Asymptotic Coding Gain:** These coders are measured in terms of their Asymptotic Coding Gains defined by:

$$G_a = 20\log_{10}\left(\frac{d_{free}}{d_{ref}}\right) \tag{5.56}$$

where $d_{free}$ is the free Euclidean distance of the code and $d_{ref}$ is the minimum Euclidean distance of an uncoded modulation scheme operating with the same signal energy per bit.

**Example 5.12**: Find the coding gain for the TCM for 8-PSK for 2 bits/symbol and rate-1/2 Ungerboeck Coder. From the partitioning diagram, we see that each branch of this coder corresponds to a subset of two antipodal signal points. The free Euclidean distance can be no longer than $d_2 = 2$ and

$$d_{free} = d_2 = 2$$

The minimum Euclidean distance of an uncoded 4-PSK (QPSK) viewed as a reference operating with the same energy per bit is simply:

$$d_{ref} = \sqrt{2}$$

Therefore, the coding gain is:

$$G_a = 20\log_{10}(\frac{2}{\sqrt{2}}) = 3.0\ dB.$$

Below we present a table of asymptotic coding gain of Ungerboeck 8-PSK codes wrt QPSK.

Asymptotic Coding Gain of Ungerboeck 8-PSK Codes compared with Uncoded 4-PSK.

| Number of states | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|
| Coding Gain dB | 3.0 | 3.6 | 4.1 | 4.6 | 4.8 | 5.0 | 5.4 | 5.7 |