5. Coded Modulation Schemes

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.

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:

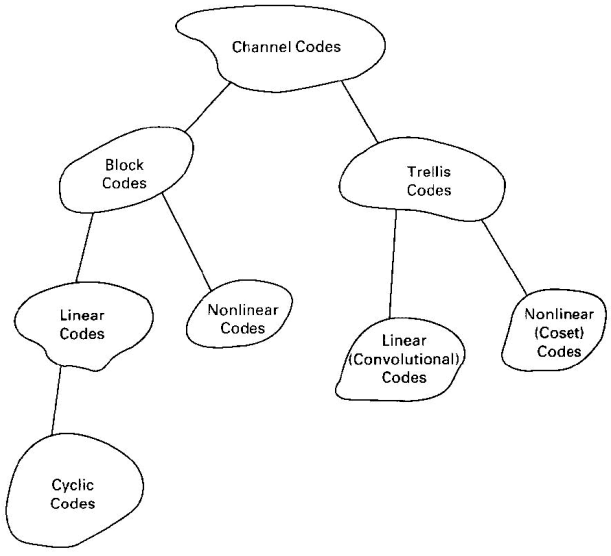
- ***Redundancy and***
- ***Memory.***

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."

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

Coding techniques may be classified based on the structure behind the encoding function: Block Codes or Sliding-Block Codes (Trellis Coding.)



These notes are © Huseyin Abut, February 2004.

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. **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.

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

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:

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

- 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:

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

Note 1: 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.

Note 2: 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.

2. **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 certain 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.

a. **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.

These notes are © Huseyin Abut, February 2004.

b. **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: $-(a.b) = (-a).b = a.(-b)$

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 a^j are used to form:

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

- 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:

$$a^{(2^m-1)} = 1 = a^0 \quad (5.5)$$

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

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

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:

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

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:

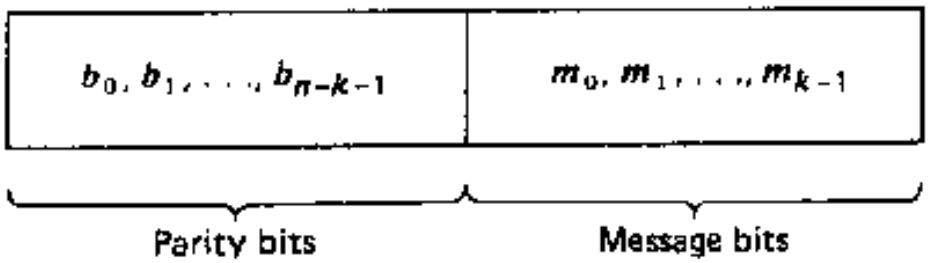
$$a^i + a^j = (a_{i0} + a_{j0}) + (a_{i1} + a_{j1}).x + ... + (a_{im-1} + a_{jm-1}).x^{m-1} \quad (5.8)$$

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

These notes are © Huseyin Abut, February 2004.

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.



The codewords are given by:

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

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

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

where the coefficients are defined by:

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

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

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

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

where:

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

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

The code is written by:

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

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

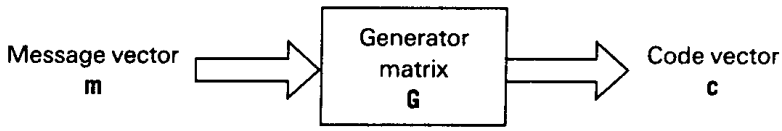
$$\underline{G} = [\underline{P} : \underline{I}_k] \quad (5.14)$$

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

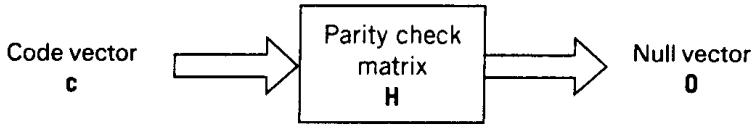
$$\underline{c} = \underline{m}.\underline{G} \quad (5.15)$$

This code generation step can be summarized with a block diagram.

These notes are © Huseyin Abut, February 2004.



(a)



(b)

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

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

We may perform the following multiplication of matrices:

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

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:

$$\underline{c}.\underline{H}^T = \underline{m}.\underline{G}.\underline{H}^T = \underline{0} \tag{5.18}$$

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 $\underline{r}$:

$$\underline{r} = \underline{c} + \underline{e} \tag{5.19}$$

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

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

Syndrome is defined by:

$$\underline{s} = \underline{r}\underline{H}^T \tag{5.21}$$

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.

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

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:

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

These notes are © Huseyin Abut, February 2004.

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:

$$\underline{e}_i.\underline{H}^T = \underline{e}.\underline{H}^T + \underline{c}_i.\underline{H}^T = \underline{e}.\underline{H}^T \qquad (5.24)$$

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.

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.

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

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 5.1: 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:

These notes are © Huseyin Abut, February 2004.

$$\mathbf{G} = \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] = [\underbrace{\mathbf{P}}_{} \quad \underbrace{\mathbf{I}_k}_{}]$$

$$\mathbf{H} = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] = [\underbrace{\mathbf{I}_{n-k}}_{} \quad \underbrace{\mathbf{P}^T}_{}]$$

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.

Table 11.1 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 Defined in Table 11.1

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*,

$$t \le \lfloor 0.5(d_{\min} - 1) \rfloor \tag{5.25}$$

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:

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

These notes are © Huseyin Abut, February 2004.

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. Let the n-tuple $(c_0, c_1, \ldots, c_{n-1})$ 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:

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

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:

$$c(X) = a(X).g(X) \qquad (5.27)$$

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 $(m_0, m_1, \ldots, m_{k-1})$ into an *(n,k)* systematic cyclic code let us form the structure for the code with *(n-k)*-parity bits in front: $(b_0, b_1, \ldots, b_{n-k-1})$ followed by the message bits $(m_0, m_1, \ldots, m_{k-1})$.

1. Let the message polynomial be defined as:
$$m(X) = m_0 + m_1 . X + \ldots + m_{k-1} . X^{k-1} \qquad (5.28)$$
2. Multiply $X^{n-k} . m(X)$.
3. Divide the result in step 2 by the generator polynomial to obtain the remainder $b(X)$.
$$X^{n-k} . m(X) / g(X) = a(X) + \frac{b(X)}{g(X)} \qquad (5.29)$$
4. Add this remainder polynomial to step 2 to obtain the code polynomial:
$$c(X) = b(X) + X^{n-k} . m(X) \qquad (5.30)$$

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

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

where coefficients h_i are either 0 or 1.

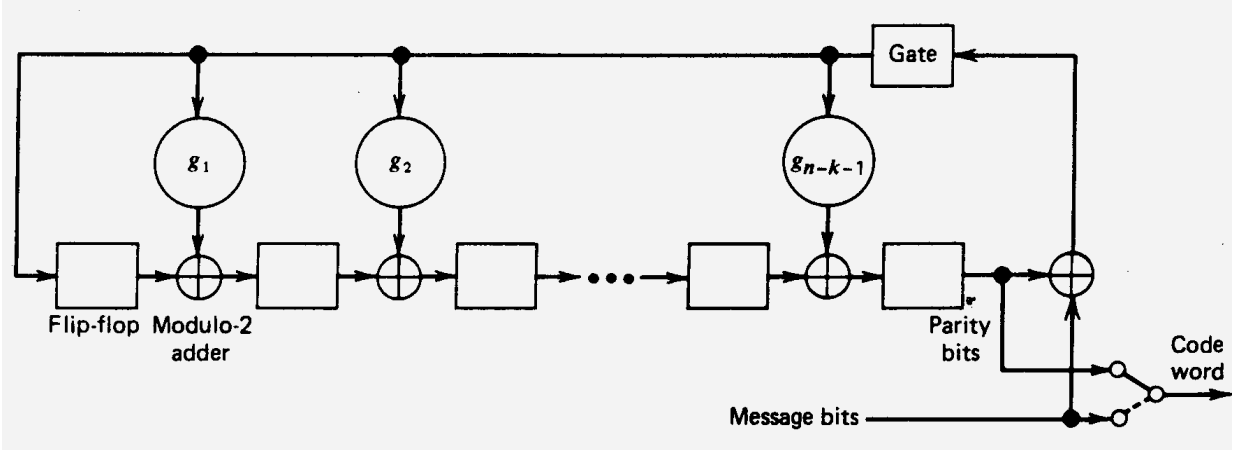
Property: *g(X)* and *h(X)* are factors of the polynomial: $X^n + 1$.

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

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

These notes are © Huseyin Abut, February 2004.

- **(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.



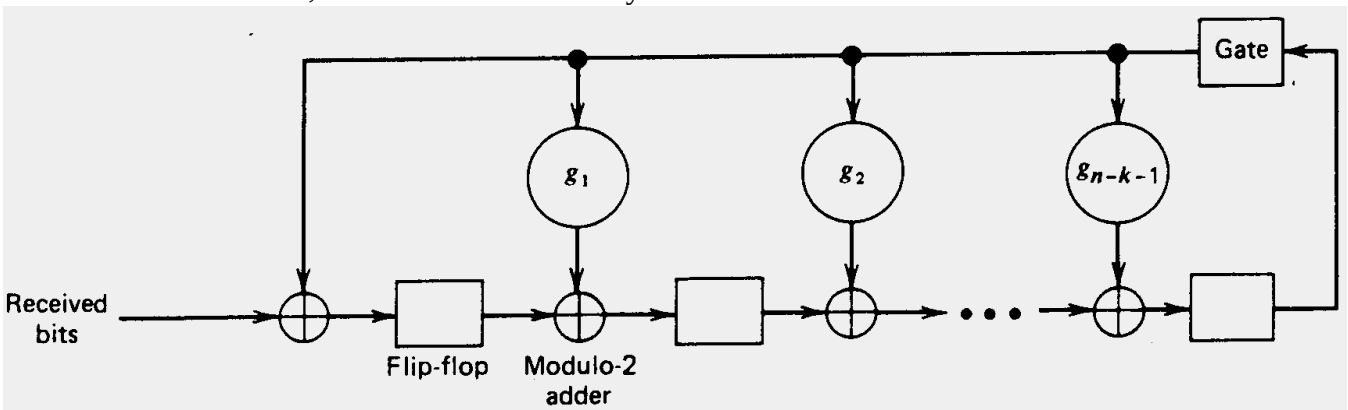
Encoder for an (n,k) cyclic Code.

- **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.

These notes are © Huseyin Abut, February 2004.

Example 5.2: 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.

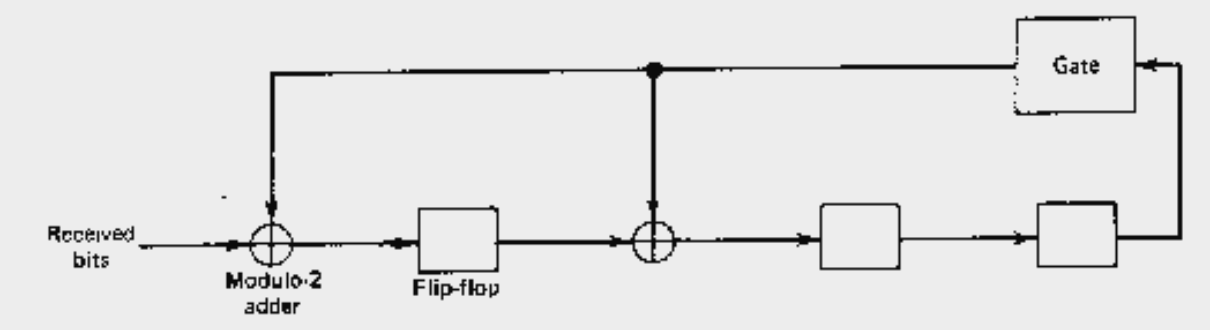
2. Let us now generate *3-by-7* parity-check matrix from the parity-check polynomial *h(X):*

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

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 $g(X) = 1 + X + X^3$.

These notes are © Huseyin Abut, February 2004.

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.

These notes are © Huseyin Abut, February 2004.

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. 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.

Reed-Solomon (RS) Codes:

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 $m \geq 1$.
- 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 thus, *m=6* is the default size in many designs.
- 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:

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

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

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

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

Example 5.3: The generator polynomial has the form:

$$g(X) = 1 + X + X^m \Rightarrow \textit{for } m = 6 \quad g(X) = 1 + X + X^6 \tag{5.36}$$

In order to map symbols to field elements, as shown in the table below, the generating primitive polynomial is set to zero:

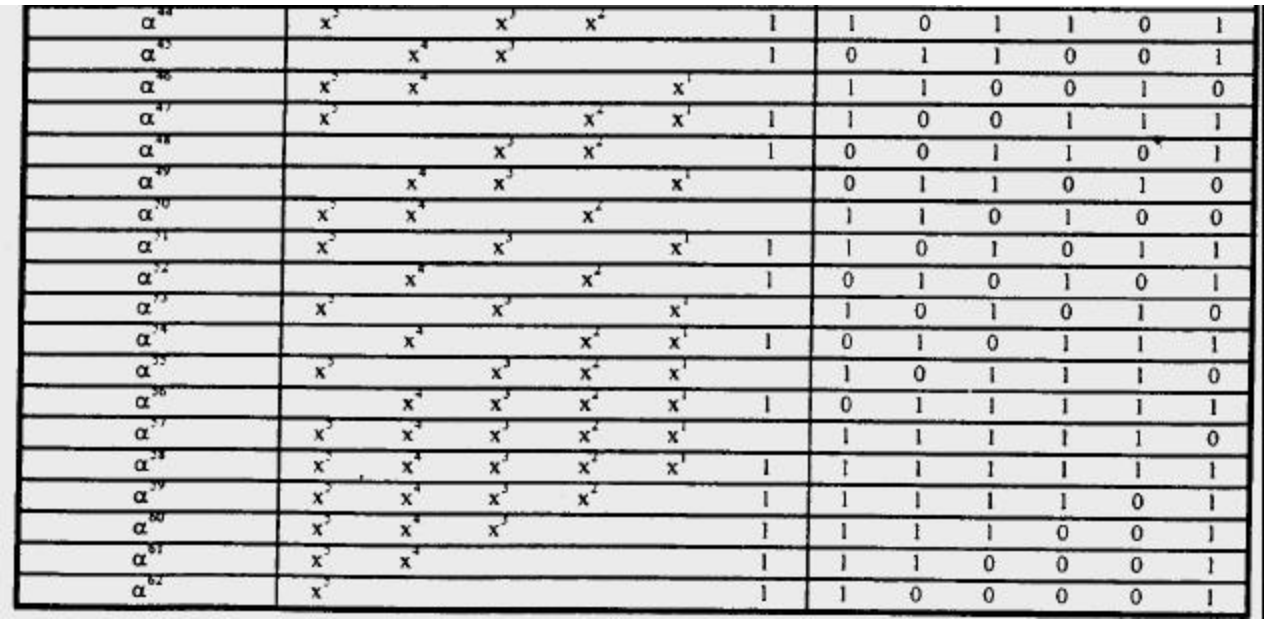
$$g(a) = 0 \quad \Rightarrow a^6 + a + 1 = 0 \tag{5.37}$$

These notes are © Huseyin Abut, February 2004.

Three Representations of the Elements of GF(64)

Power Representation	Polynomial Representation	6-Bit Symbol Representation
$0 = \alpha^{63}$	0	0 0 0 0 0 0
$1 = \alpha^{0}$	1	0 0 0 0 0 1
α	x^1	0 0 0 0 1 0
α^2	x^2	0 0 0 1 0 0
α^3	x^3	0 0 1 0 0 0
α^4	x^4	0 1 0 0 0 0
α^5	x^5	1 0 0 0 0 0
α^6	$x^1 + 1$	0 0 0 0 1 1
α^7	$x^2 + x^1$	0 0 0 1 1 0
α^8	$x^3 + x^2$	0 0 1 1 0 0
α^9	$x^4 + x^3$	0 1 1 0 0 0
α^{10}	$x^5 + x^4$	1 1 0 0 0 0
α^{11}	$x^5 + x^1 + 1$	1 0 0 0 1 1
α^{12}	$x^2 + 1$	0 0 0 1 [illegible] 1
α^{13}	$x^3 + x^1$	0 0 1 0 1 0
α^{14}	$x^4 + x^2$	0 1 0 1 0 0
α^{15}	$x^5 + x^3$	1 0 1 0 0 0
α^{16}	$x^4 + x^1 + 1$	0 1 0 0 1 1
α^{17}	$x^5 + x^2 + x^1$	1 0 0 1 1 0
α^{18}	$x^3 + x^2 + x^1 + 1$	0 0 1 1 1 1
α^{19}	$x^4 + x^3 + x^2 + x^1$	0 1 1 1 1 0
α^{20}	$x^5 + x^4 + x^3 + x^2$	1 1 1 1 0 0
α^{21}	$x^5 + x^4 + x^3 + x^1 + 1$	1 1 1 0 1 1
α^{22}	$x^5 + x^4 + x^2 + 1$	1 1 0 1 0 1
α^{23}	$x^5 + x^3 + 1$	1 0 1 0 0 1
α^{24}	$x^4 + 1$	0 1 0 0 0 1
α^{25}	$x^5 + x^1$	1 0 0 0 1 0
α^{26}	$x^2 + x^1 + 1$	0 0 0 1 1 1
α^{27}	$x^3 + x^2 + x^1$	0 0 1 1 1 0
α^{28}	$x^4 + x^3 + x^2$	0 1 1 1 0 0
α^{29}	$x^5 + x^4 + x^3$	1 1 1 0 0 0
α^{30}	$x^5 + x^4 + x^1 + 1$	1 1 0 0 1 1
α^{31}	$x^5 + x^2 + 1$	1 0 0 1 0 1
α^{32}	$x^3 + 1$	0 0 1 0 0 1
α^{33}	$x^4 + x^1$	0 1 0 0 1 0
α^{34}	$x^5 + x^2$	1 0 0 1 0 0
α^{35}	$x^3 + x^1 + 1$	0 0 1 0 1 1
α^{36}	$x^4 + x^2 + x^1$	0 1 0 1 1 0
α^{37}	$x^5 + x^3 + x^2$	1 0 1 1 0 0
α^{38}	$x^4 + x^3 + x^1 + 1$	0 1 1 0 1 1
α^{39}	$x^5 + x^4 + x^2 + x^1$	1 1 0 1 1 0
α^{40}	$x^5 + x^3 + x^2 + x^1 + 1$	1 0 1 1 1 1
α^{41}	$x^4 + x^3 + x^2 + 1$	0 1 1 1 0 1
α^{42}	$x^5 + x^4 + x^3 + x^1$	1 1 1 0 1 0
α^{43}	$x^5 + x^4 + x^2 + x^1 + 1$	1 1 0 1 1 1

These notes are © Huseyin Abut, February 2004.



Note 1: Multiplication is achieved by $Modulo\_(2^m - 1)$ addition of the elementary operands.
Note 2: Addition corresponds to $Modulo\_2$ adding the coefficients of the polynomial representation of the elements.

Example 5.4:

$$a^{27} + a^{5} = (001110)_2 \, XOR (100000)_2 = (101110)_2 = a^{55}$$
$$a^{19} + a^{62} = (011110)_2 \, XOR (100001)_2 = (111111)_2 = a^{58}$$

which are identical to the corresponding entries in the above table.

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:

$$c(X) = d(X) + p(X) = \sum_{i=0}^{n-1} c_i . x^i \qquad (5.38)$$

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:

$$g(X) = (x + a).(x + a)...(x + a^{2t}) = \sum_{i=0}^{2t} g_i . x^i \qquad (5.39)$$

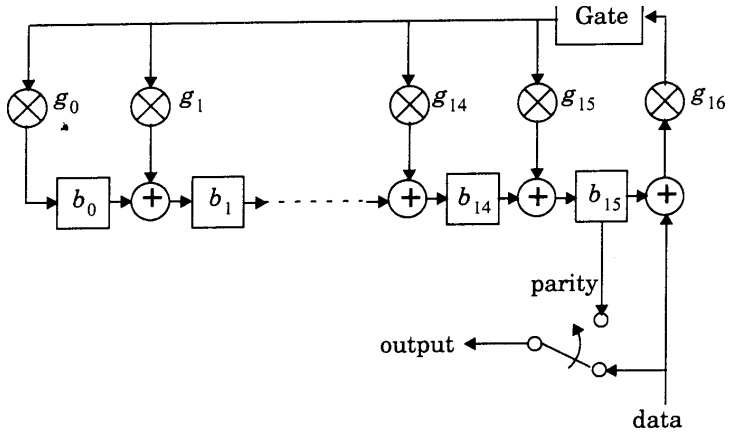
A common method for encoding an RS code is to derive *p(X)* by diving *d(X)* by *g(X)*, which yields an irrelevant quotient polynomial *q(X)* and a relevant remainder polynomial *r(X):*

These notes are © Huseyin Abut, February 2004.

$$c(X) = p(X) + g(X).q(X) + r(X) \tag{5.40}$$

If the parity polynomial is defined as being equal to the negatives of the coefficients of *r(X)* then we have (5.40) simplifies and a block diagram for this implementation is shown below.

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



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:

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

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 $c(a^i) = 0$. 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

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

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

These notes are © Huseyin Abut, February 2004.

$$\begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t} \end{bmatrix} = \begin{bmatrix} e_{j_1}.b_1 + e_{j_2}.b_2 + \cdots + e_{j_k}.b_k \\ e_{j_1}.b_{j_1}^{2} + e_{j_2}.b_{j_2}^{2} + \cdots + e_{j_k}.b_k^{2} \\ \vdots \\ e_{j_1}.b_{j_1}^{2t} + e_{j_2}.b_{j_2}^{2t} + \cdots + e_{j_k}.b_k^{2t} \end{bmatrix} \tag{5.47}$$

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 $x = a^i$:

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

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

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

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):

$$s(X) = (1 + b_1.x)....(1 + b_k.x^k) = s_0 + s_1.x + \ldots + s_k.x^k \tag{5.50}$$

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

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_k \end{bmatrix} = \begin{bmatrix} 1 \\ b_1 + b_2 + \cdots + b_k \\ b_1.b_2 + b_2.b_3 + \cdots + b_{k-1}.b_k \\ \vdots \\ b_1.b_2.b_3 \cdots b_k \end{bmatrix} \tag{5.51}$$

Partial syndromes are related to these to yield "Newton's Identities in vector notation:

$$\begin{bmatrix} S_1 + s_1 \\ S_2 + s_1.S_1 + 2s_2 \\ S_3 + s_1.S_2 + s_2.S_1 + 3s_3 \\ \vdots \\ S_k + s_1.S_{k-1} + \cdots + s_{k-1}.S_1 + ks_k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5.52}$$

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.

These notes are © Huseyin Abut, February 2004.

In Appendix of this chapter, you will find a number of neat material from ECC Homepage of Prof. Robert Morelos-Zaragosa.

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. The encoder operating at a rate 1/n bits/symbol, may be viewed as a finite-state machine that consists of an M-stage shift register with prescribed connections to n modulo-2 adders, and a multiplexer that serializes the outputs of the adders.

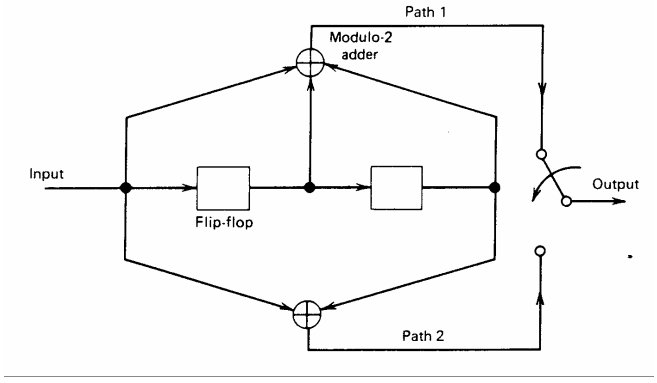
Some definitions:

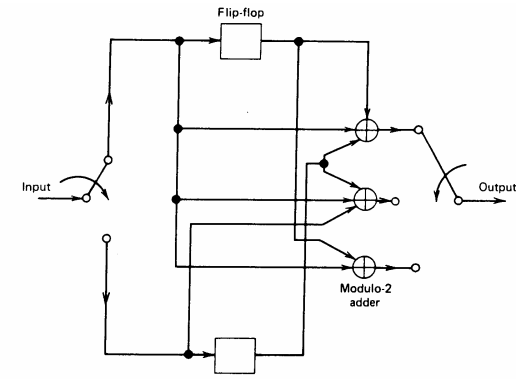
1. **Code Rate:** An L-bit message sequence produces a coded output sequence of length $n(L+M)$ bits and the code rate is defined by:

$$r = \frac{L}{n(L+M)} \; bis/symbol \; where \; L >> M \qquad (5.53)$$

2. Constraint Length: 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 two popular convolutional code setups.

Example 5.5: In the first example, encoder operates on one bit at a time with n=2 and K=3. The coding rate is simply ½. On the other hand, the second encoder has two-bit inputs and three bit outputs with k=2, n=3 and the two shift registers have K=2 each. The code rate is 2/3 in this case.





3. **Impulse Response:** Each path connecting output to the input is characterized in terms of its impulse response, defined as the response of that path to a symbol "1" applied to its input, with each FF in the encoder set initially to the zero state. Hence, a generator sequence $(g_0^i, g_1^i, \cdots g_M^i)$ is used for showing the impulse response of the i^{th} path, in which each term is "0" or "1".
4. **Generator Polynomial:** If D is the unit-delay variable, then the generator polynomial for the i^{th} path is defined by:

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

These notes are © Huseyin Abut, February 2004.

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

Example 5.6: 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*.

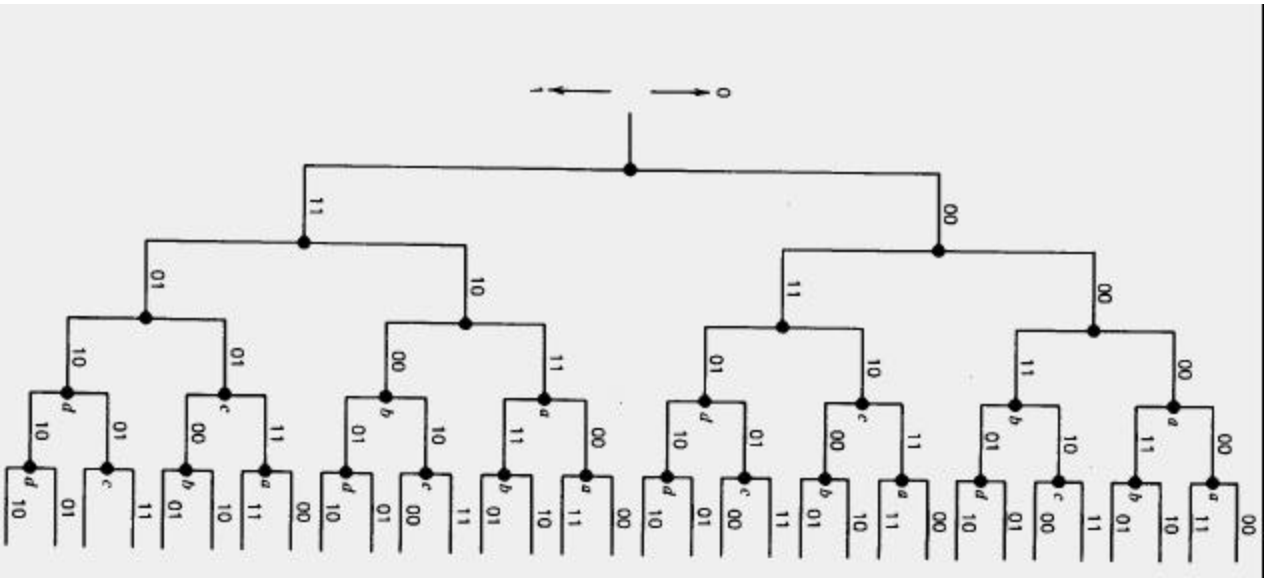
Code Tree, Trellis and State Diagram Representations:

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.

- As it can be seen from the tree diagram, the input sequence (10011) when applied to the ½ rate coder in the example, 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.

These notes are © Huseyin Abut, February 2004.



Code Tree for the Convolutional Encoder

Therefore, we can collapse the tree into a form called "*trellis*." Hence, a more instructive structure of "trellis" description of convolutional codes emerges.

2. **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.

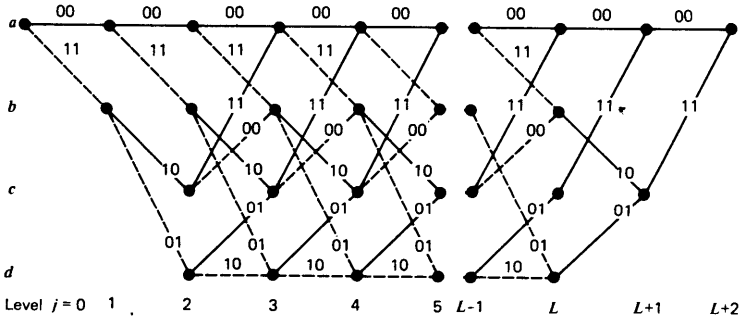


Figure 11.15 Trellis for the convolutional encoder

State Table for the Convolutional Encoder

State	Binary Description
a	00
b	10
c	01
d	11

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

These notes are © Huseyin Abut, February 2004.

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.

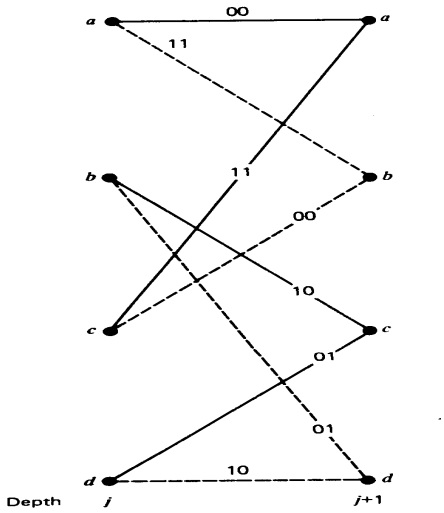


Figure 11.16a A portion of the central part of the trellis for the encoder of Fig. 11.13a.

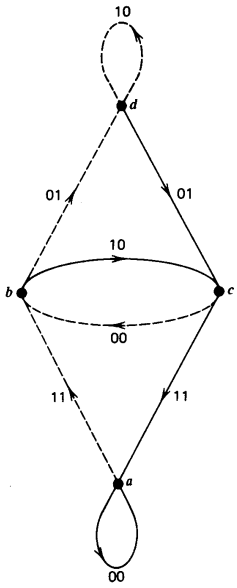


Figure 11.16b State diagram of the convolutional encoder of Fig. 11.13a.

- Let us consider the portion of a trellis at times j and j+1 as shown above. Here left nodes are the possible states of the coder and the right nodes represent the next states. By using this, we obtain the *state-diagram* of this encoder as shown in the right, where the nodes stand for the four possible states of this coder. Start from the state a, follow a solid branch if the input is "0" and a dashed branch if it is a "1". For the input sequence (10011), we follow: abcabd, and therefore, the output sequence is (11,10,11,11,01) as in previous two cases.

3. 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 $\log_e(P(\underline{r} \mid \underline{c})$ is maximum. If the channel is BSC then this rule simplifies to the following case:

These notes are © Huseyin Abut, February 2004.

Choose the estimate $\hat{\underline{c}}$ 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.

Steps of Viterbi Algorithm:

1. Initialization: Label the left-most state of the trellis as "0", i.e., the all-zero state at level 0.

2. Computation step j+1: Let $j = 0,1,2,\cdots,$ and suppose that we have performed the following two things in the previous *step j*:

(a) All survivor paths have been identified.

(b) The survivor path and its metric have been stored for each state of the trellis.

Now, at level j+1:

1. Compute the metric for all the paths entering each state of the trellis by adding the metric of the incoming branches to the metric of the connecting survivor path from level j.
2. For each state, identify the path with the lowest metric as the survivor of step j+1, thereby updating the computation.

3. Final Step: Continue computation till the forward search through the trellis reaches the termination node, i.e., all-zero state. At this time, make a decision on the minimum distance, or ML path. Then, like a block decoder, the sequence of symbols associated with that path is released to the destination as the decoded version of the received sequence.

4. Modification: When the received sequence is very long, the storage requirement of the Viterbi Decoder becomes impractically high.

(a) The path memory is truncated by assuming a *decoding window of length l*, and the algorithm stops always after *l steps*.

(b) The decision is made based on the best path. The symbol associated with the *FIRST* branch on that path is released to the user.

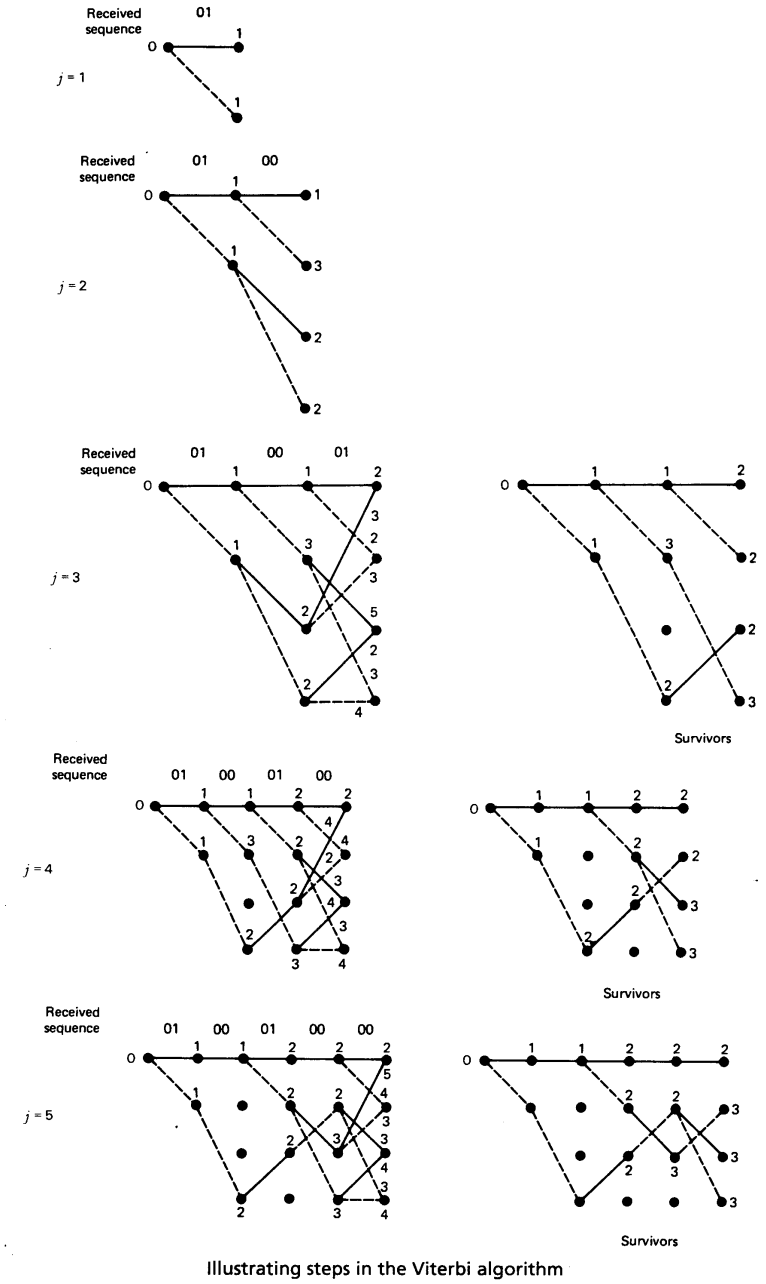
(c) Drop the symbol associated with the last branch of the path and move the decoding window one-time interval.

The decoding decision made this way is no longer truly ML, but they can be made almost as good provided the decoding window is long enough. Experiments show that results are impressive if the decoding window length l is on the order of (5) times the constraint length K of the encoder.

Example 5.7: Let us consider the encoder with rate ½ and K=3 again. Assume that the encoder has generated an all-zero sequence (000000000….) and the received sequence is (0100010000…) with (2) errors due to noise in the BSC channel.

These notes are © Huseyin Abut, February 2004.

Question: Can we correct this double-error pattern?
Below, we show the results of applying VA for levels j=1,2,3,4,5.



Illustrating steps in the Viterbi algorithm

When we examine the survivors for j=5, we see that the all-zero path has the smallest metric=8 when compared to metrics=9,10,10,11. This shows that the choice of VA agrees exactly with the input sequence.

Example 5.8: Let us consider the same encoder with rate ½ and K=3 again. But this time, the received sequence is (1100010000…) with (3) errors due to noise in the BSC channel. The results of applying VA for levels j=1,2,3,4,5 are given below.

These notes are © Huseyin Abut, February 2004.

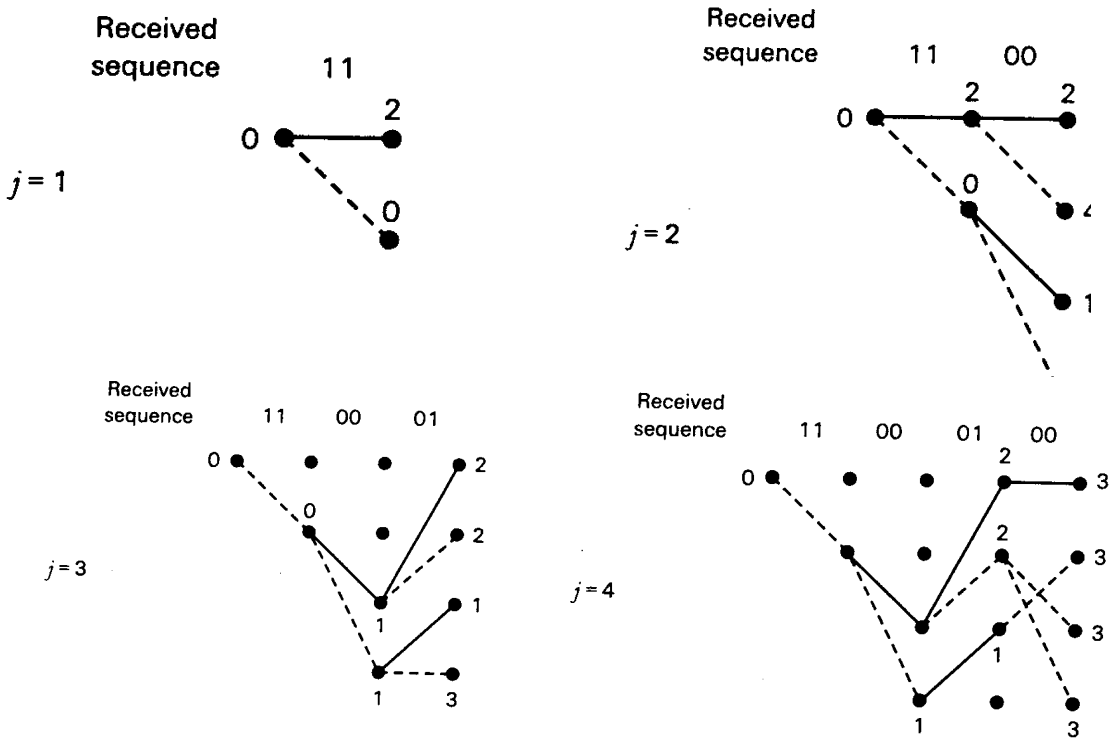


Figure 11.18 Illustrating breakdown of the Viterbi algorithm

In this case, the correct path has been eliminated by the level $j = 3$ and a triple error is NOT correctable by the VA applied to a rate ½ and constraint length $K = 3$ convolution or trellis coder. Exception to this rule is a triple-error pattern spread over a time span significantly longer one constraint length, in which case it is *very likely* to be correctable.

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:

These notes are © Huseyin Abut, February 2004.

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.

Step 1: 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:

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

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

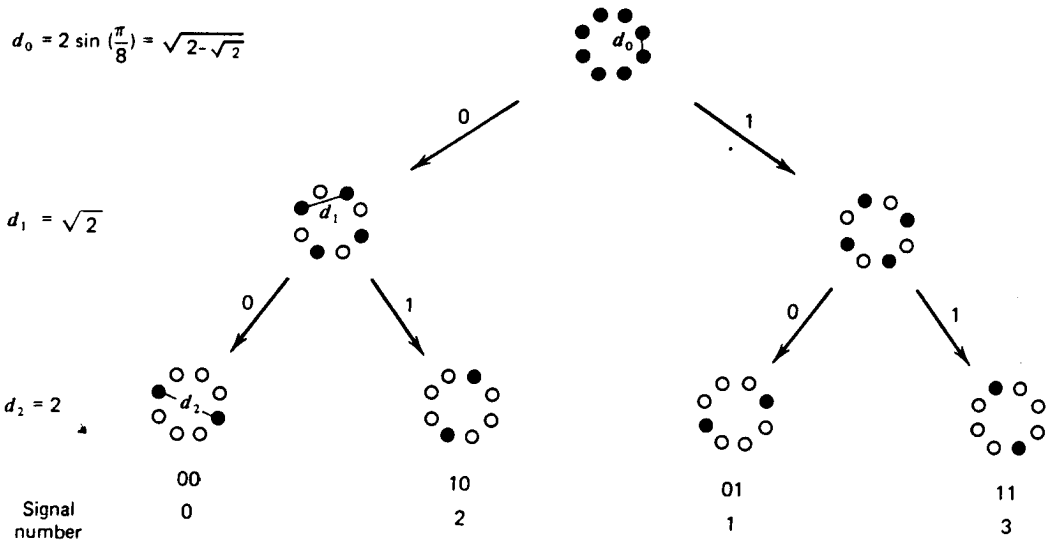


Figure 11.20 Partitioning of 8-PSK constellation.

These notes are © Huseyin Abut, February 2004.

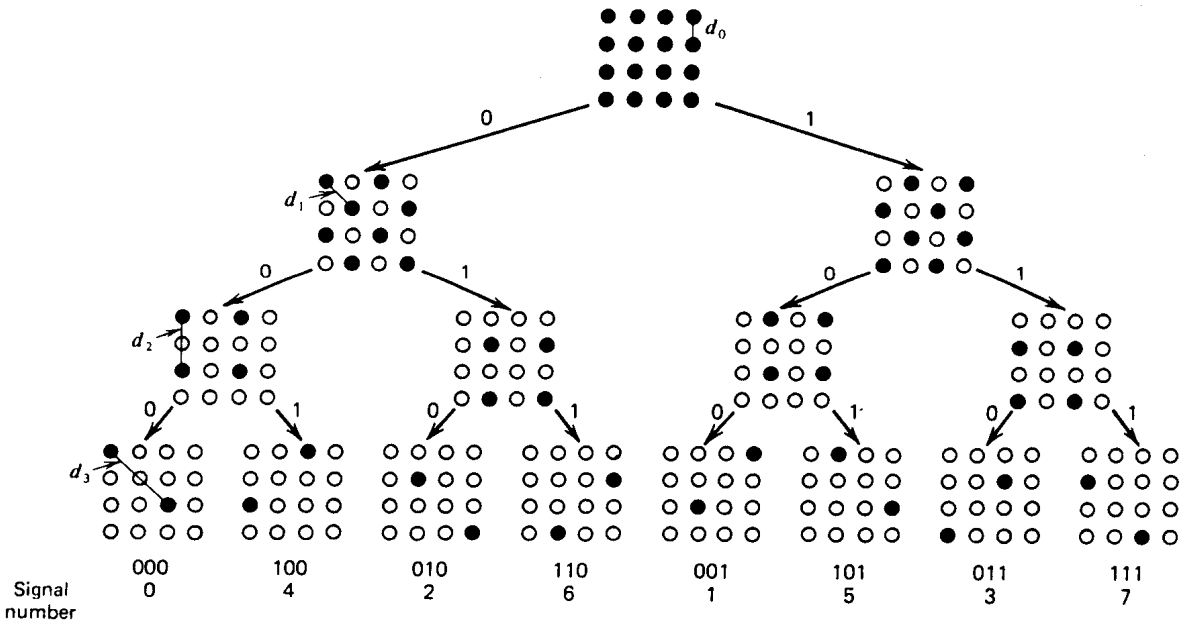


Figure 11.21 Partitioning of 16-QAM constellation.

Step 2: Design of Ungerboeck Codes:

Based on the subsets resulting from the successive partitioning of a 2-D constellation and to send n-bits/symbol with quadrature modulation (in-phase and quadrature components), we start with a 2-D constellation of 2^{n+1} signal points, where we employ a circular grid for M-PSK and a rectangular grid for M-QAM. One or two incoming bits per symbol enter into a rate-½ convolutional coder or a rate-2/3 binary convolutional coder, respectively, which results in 2 or 3 coded bits/symbol. Here we determine the selection of a particular subset from an Ungerboeck Trellis family.

Step 3: Decoding of Ungerboeck Codes by the VA:

Since the modulator has memory, the use of the VA is the preferred choice for ML sequence detection at the receiver in the following manner:

1. Each branch of an Ungerboeck trellis code corresponds to a subset rather than an individual signal point.
2. First step in detection is to determine the signal point within each subset that has MSE to the received signal point. The signal point and the MSE distance metric is used thereafter for the branch in question and the VA then proceeds in the usual manner as described previously.

These notes are © Huseyin Abut, February 2004.

Example 5.9: TCM for 8-PSK for 2 bits/symbol and rate-1/2 Ungerboeck Coder.

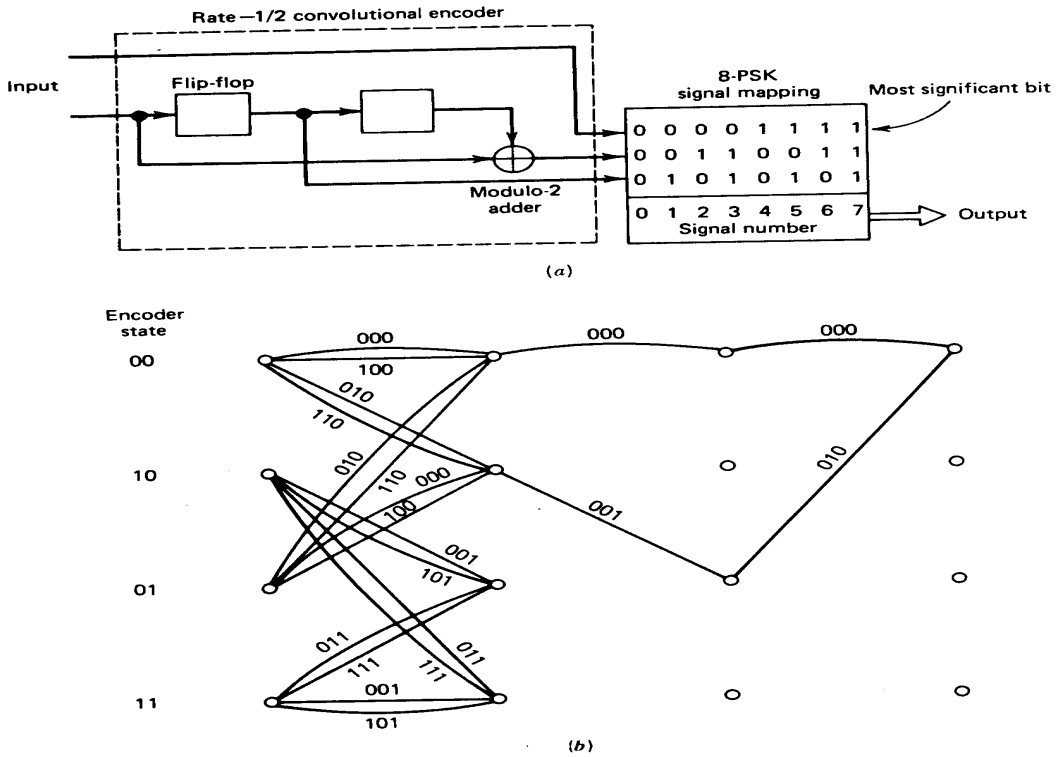


Figure 11.22 (*a*) Four-state Ungerboeck code for 8-PSK. (*b*) Trellis.

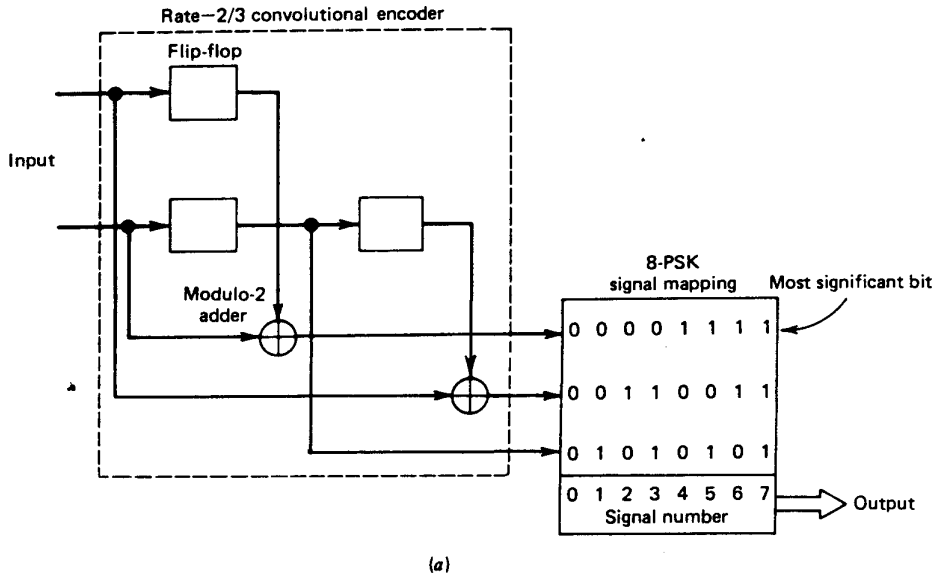
This coder has a 4-state finite-state architecture.

- Most significant bit of the incoming binary word is left uncoded and hence,
- Each branch of the trellis corresponds to two different output values of the 8-PSK modulator, or equivalently, to one of the four 2-point subset of 8-PSK partitioning.
- In addition, the trellis shows the MSE distance path.

Example 5.10: TCM for 8-PSK for 2 bits/symbol and rate-2/3 Ungerboeck Coder.

Here the trellis will have (8) states and both bits of the incoming binary word are encoded. Therefore, each branch of the trellis corresponds to a specific output value of this 8-PSK modulator. The MSE distance path is also depicted. It is important to note that in the previous case, the state of the encoder was defined by the contents of the two-stage SR. Whereas, it is defined by the content of the single-stage (top) SR followed by that of the two-stage (bottom) SR.

These notes are © Huseyin Abut, February 2004.



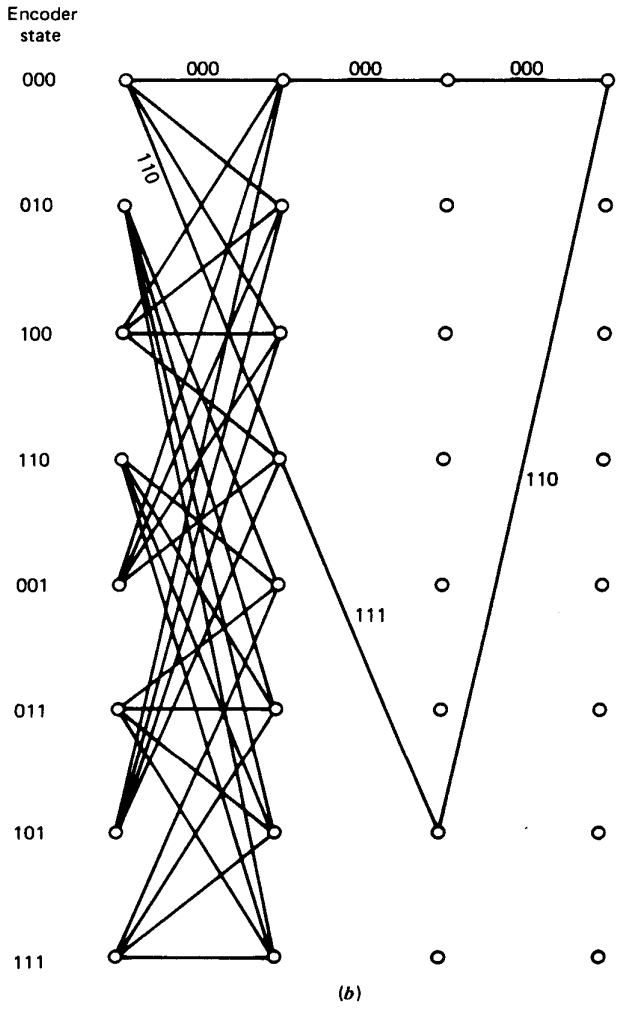


Figure 11.23 (*a*) Eight-state Ungerboeck code for 8-PSK. (*b*) Trellis.

These notes are © Huseyin Abut, February 2004.

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

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

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 5.11: 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	3.6	4.1	4.6	4.8	5	5.4	5.7

APPENDIX

ECC Homepage Introductory Material: http:///imailab.iis.u-tokyo.ac.jp/~robert/codes.html
Copyright (c) 1996, 1997. Robert Morelos-Zaragoza. All rights reserved worldwide
****

Welcome!

In this page you will find free programs written in C language that implement encoding and decoding routines of popular error correcting codes (ECC), such as Reed-Solomon codes, BCH codes, the binary Golay code, a binary Goppa code, a Viterbi decoder and more. Some of these programs I have found over the years (since 1988), while I was a Ph.D. student in Hawaii. Others were either "hacked" or authored by me.

Note that no effort has been made to `optimize' most of the algorithms used in the programs below. The algorithms work well, but by no means should be used as a basis for an implementation. All these programs are free to use for academic and personal purposes. Use them at your own discretion!

If you have implemented algorithms for ECC in C/C++ and would like to share them with the world, please email them. I will post the pointers to them in this page. I am planning to add more programs, and maintain this page, as my research activities allow.

For more information on error correcting codes, and before you ask me a very basic question, I recommend the following books:

1. S. Lin and D. J. Costello, Jr., Error Control Coding: Fundamentals and Applications, Prentice Hall: Englewood Cliffs, NJ, 1983.
2.W.W. Peterson and E.J. Weldon, Jr., Error-Correcting Codes, 2nd edition, MIT Press: Cambridge, Mass., 1972.
3.F.J. MacWilliams and N.J.A. Sloane, The Theory of Error-Correcting Codes, North-Holland: New York, NY, 1977.

List of programs and links

1. Reed-Solomon (RS) codes: This program uses a good implementation of finite (Galois) field arithmetic and the Berlekamp-Masssey (BM) algorithm, as explained in Lin and Costello's book. (Simon Rockliff, 1991)
2. Reed-Solomon errors-and-erasures decoder: Based on the above program to handle errors and erasures, plus other features. Check it out. Note: The program does not seem to work well with shortened codes and codes over GF(2^m), m<8. (Morelos-Zaragoza and Thirumoorthy, 1995)
3. Another Reed-Solomon errors-and-erasures decoder: From Phil Karn, it is nicely written and a greatly improved version of the program above. Phil has done a terrific job and is working now on decoding shortened RS codes. You can get the package here. (Phil Karn, 1996).
4. BCH codes: Enter only the length and error correcting capability. The program computes the generator polynomial of any binary BCH code, plus encoding and decoding using the BM algorithm. (Morelos-Zaragoza, 1994).
5. Binary (48,36,5) BCH code. This BCH code is used in control channels for cellular TDMA in the U.S. Since this code has only two-error correcting capability, fast decoding is done by pre-solving a system of two equations (the syndromes) in two unknowns (the error positions), see MacWilliams and Slone's book, chapter 3. (Morelos-Zaragoza, 1994).
6. Binary (31,21,5) BCH code. This BCH code is used in the POCSAG protocol specification for pagers. The program is identical to the one above, except for the parameters. (Morelos-Zaragoza, 1997).
7. Golay (23,12,7) code. Fast encoding and decoding by software with look-up tables. The program uses a 16K-by-16 bit encoding table and an 8K-by-32 bit decoding table. (Morelos-Zaragoza, 1994).
8. A Goppa code. Encoding/Decoding of a (1024,654,75) Goppa code (originally written with a public key cryptographic scheme in mind). This program is a compact implementation of Goppa codes with parameters m=10, t=37 for 32-bit machines. Decoding method due to N. Patterson, ``Algebraic Decoding of Goppa Codes," IEEE Trans. Info.Theory, 21 (1975), 203-207. (Anonymous, as far as I know)
9. CRC-32. Computes the CRC value of a file, as used in ZMODEM or PKZIP. From a coding theory perspective, this is relatively simple. The code in question is just a Hamming code, with 32 redundant bits. (Craig Bruce, 1994)

These notes are © Huseyin Abut, February 2004.

10. ecc-1.2.1.tar (106496 bytes) . You can find this using archie. Routines to encode and decode a file using a (255,249,7) RS code. (Paul Flaherty, 1993)
11. Turbo-codes home page at JPL. Do I plan to work on Turbo codes? Maybe in the future...
12. viterbi.tar. Viterbi decoding of a (de-facto standard) rate-1/2 m=7 convolutional code. (Phil Karn, 1995) Note: A newer version is now available from Phil's home page. If you go there, you will also find other interesting programs, such as an implementation of the Fano decoding algorithm (3/96). He has recently put an interesting page: "Wireless Digital Communication: A View Based on Three Lessons Learned" by Dr. Andrew Viterbi (5/96).
13. galois.tar. Encoding/decoding for BCH/RS codes. It looks interesting, but unfortunately I have not found the time to test it. (Bart De Canne, 1994)
14. Simulate a BCH, Reed-Solomon or Reed-Muller Code on the Web! This one comes from The Netherlands. It needs some work, but it is nice if, say, you need to know the generator polynomial of a BCH or RS code, without programming/compiling, etc. (Richie B, 1996)
15. A block coded QPSK modulation for unequal error protection (UEP). This program was used to simulate the performance of a coding scheme proposed in my Ph.D. thesis for UEP over an AWGN channel. For more details, see R.H. Morelos-Zaragoza and S. Lin, ``QPSK Block Modulation Codes for Unequal Error Protection," IEEE Transactions on Information Theory, Vol. 41, No. 2, pp. 576-581, March 1995. (Morelos-Zaragoza, 1993)
16. Routines in C for a Generalized BCH codec/performance analyzer. BCH codes over GF(Q^s) can be evaluated for error performance over an AWGN channel, where Q is any prime or its power, and s is a positive integer. You must specify the mapping from GF(Q^s) to a 2-D constellation. Selectable Berlekamp-Massey decoder or Peterson-Gorenstein-Zierler decoder. (Kuntal Sampat, 1996)
17. Linear code bound. How good is a code? What is the lower and upper bounds on the minimum distance of a linear block code given its length and dimension? The answer to this question may be found on-line! (remkor@win.tue.nl, 1995)
18. CD-ROM Recording Software. Various programs for recording in SCSI based CD-ROM recorders on DOS/Windows'95 base systems. (Jeff Arnold, 1996)
19. Erasure-correcting codes. An implementation of a block code for erasure correction in network communication protocols. The encoder/decoder runs quite fast (up to several MB/s on a Pentium). (Luigi Rizzo, 1996)

Other pointers (send yours!):

http://www.4i2i.com/ J. Bierbrauer's home page

This page was last updated on January 16, 1997. Robert Morelos-Zaragoza.

****

These notes are © Huseyin Abut, February 2004.