Chapter 10

Error-Control Coding

Problem 10.1

The matrix of transition probabilities of a discrete memoryless channel with 2 inputs and Q outputs may be written as

$$P = \begin{bmatrix} p(0|0) & p(1|0) & p(2|0) & \cdots & p(Q-1|0) \\ p(0|1) & p(1|1) & p(2|1) & \cdots & p(Q-1|1) \end{bmatrix}$$

For a symmetric channel,

$$p(j|0) = p(Q-1-j|1), \qquad j = 0,1,\ldots,Q-1$$

Moreover, each row of the matrix **P** contains the same set of numbers, and each column of the matrix **P** contains the same set of numbers. For example, for Q=4, we may write

$$P = \begin{bmatrix} a & a & b & b \\ b & b & a & a \end{bmatrix}$$

The sum of the elements of each row of matrix **P** must add up to one. Hence, for this example,

$$2a + 2b = 1$$

The probability of receiving symbol j is

$$p(j) = p(j|0)\,p(0) + p(j|1)p(1)$$

For equally likely input symbols:

$$p(0) = p(1) = \frac{1}{2}$$

Hence,

$$p(j) = \frac{1}{2}\left[p(j\,|0) + p(Q-1-j\,|0)\right]$$

For the example of Q=4, we have

$$p(j) = \frac{1}{2}(a + b)$$

$$= \frac{1}{4}, \qquad j = 0,1,2,3$$

In general, we may write

$$p(j) = \frac{1}{Q}, \qquad j = 0,1,\ldots,Q-1$$

Problem 10.2

For a binary PSK channel, the probability density function of the correlator output in the receiver is

$$f_X(x\,|0) = \frac{1}{\sqrt{\pi N_0}} \exp\left[-\frac{1}{N_0}\left(x + \sqrt{E_b}\right)^2\right]$$

$$f_X(x\,|1) = \frac{1}{\sqrt{\pi N_0}} \exp\left[-\frac{1}{N_0}\left(x - \sqrt{E_b}\right)^2\right]$$

Let

$$y = \sqrt{\frac{2}{N_0}}\, x$$

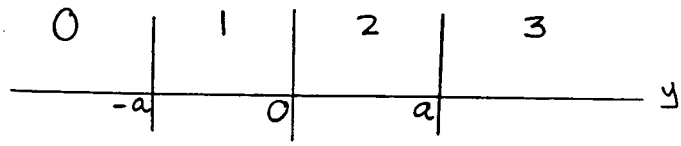
$$dy = \sqrt{\frac{2}{N_0}}\, dx$$

y pertains to a Gaussian variable of mean $\pm \sqrt{\frac{2E_b}{N_0}}$ and unit variance. We may therefore express the channel transition probability as

$$p(y|0) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(y + \sqrt{\frac{2E_b}{N_0}} \right)^2 \right]$$

$$p(y|1) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(y - \sqrt{\frac{2E_b}{N_0}} \right)^2 \right]$$

where $-\infty < y < \infty$.



$$p(0\,|0) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp\left[-\frac{1}{2}\left(y + \sqrt{\frac{2E_b}{N_0}} \right)^2 \right] dy$$

$$= \frac{1}{2}\,\mathrm{erfc}\left(\frac{a}{\sqrt{2}} - \sqrt{\frac{E_b}{N_0}} \right)$$

$$p(1\,|0) = \frac{1}{\sqrt{2\pi}} \int_{-a}^{0} \exp\left[-\frac{1}{2}\left(y + \sqrt{\frac{2E_b}{N_0}} \right)^2 \right] dy$$

$$= \frac{1}{2}\left[\mathrm{erfc}\left(-\sqrt{\frac{E_b}{N_0}} \right) - \mathrm{erfc}\left(\frac{a}{\sqrt{2}} - \sqrt{\frac{E_b}{N_0}} \right) \right]$$

$$p(2\,|0) = \frac{1}{2\pi} \int_{0}^{a} \exp\left[-\frac{1}{2}\left(y + \sqrt{\frac{2E_b}{N_0}} \right)^2 \right] dy$$

$$= \frac{1}{2}\left[\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_0}} \right) - \mathrm{erfc}\left(\frac{a}{\sqrt{2}} + \sqrt{\frac{E_b}{N_0}} \right) \right]$$

$$p(3\,|0) = \frac{1}{\sqrt{2\pi}} \int_{a}^{\infty} \exp\left[-\frac{1}{2}\left(y + \sqrt{\frac{2E_b}{N_0}} \right) \right]^2 dy$$

$$= \frac{1}{2}\,\mathrm{erfc}\left(\frac{a}{\sqrt{2}} + \sqrt{\frac{E_b}{N_0}} \right)$$

We also note that

$p(3\,|\,0) = p(0\,|\,1)$
$p(2\,|\,0) = p(1\,|\,1)$
$p(1\,|\,0) = p(2\,|\,1)$
$p(0\,|\,0) = p(3\,|\,1)$

Hence, the channel is symmetric.

Problem 10.3

From the solution to Problem 10.2, we readily note the following:

$$p(y|0) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(y + \sqrt{\frac{2E}{N_0}} \right)^2 \right], \qquad -\infty < y < \infty$$

$$p(y|1) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(y - \sqrt{\frac{2E}{N_0}} \right)^2 \right], \qquad -\infty < y < \infty$$

where E is the code symbol energy.

Problem 10.4

Message Sequence	Single-parity-check code
0 0 0	0 0 0 0
0 0 1	0 0 1 1
0 1 0	0 1 0 1
0 1 1	0 1 1 0
1 0 0	1 0 0 1
1 0 1	1 0 1 0
1 1 0	1 1 0 0
1 1 1	1 1 1 1

Problem 10.5

For the (4,1) repetition code, the parity check matrix is

$$H = \left[\begin{array}{ccc:c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right]$$

For a (7,4) Hamming code, we have

$$H = \left[\begin{array}{ccc:cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right]$$

For the Hamming code, the parity check matrix **H** is more structured than that for the repetition code. Indeed, the matrix **H** for the Hamming code includes that for the repetition code as a submatrix.

Problem 10.6

The generator matrix for the (7.4) Hamming code is

$$G = \left[\begin{array}{ccc:cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

The parity-check matrix is

$$H = \left[\begin{array}{cccc:cccc} 1 & 0 & 0 & & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & & 0 & 1 & 1 & 1 \end{array}\right]$$

Hence,

$$HG^T = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \text{mod-2}$$

Problem 10.7

(a) Viewing the matrix

$$H = \left[\begin{array}{cccc:cccc} 1 & 0 & 0 & & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & & 0 & 1 & 1 & 1 \end{array}\right]$$

as a generator matrix, we may define the code vector $\underset{\sim}{c}$ in terms of the message vector **m** as

$$\underset{\sim}{c} = \mathbf{m}\, H$$

The message word length is

$$n - k = 7-4 = 3$$

Hence, we may construct the following table

Message word	Code word	Hamming weight
0 0 0	0 0 0 0 0 0 0	0
0 0 1	0 0 1 0 1 1 1	4
0 1 0	0 1 0 1 1 1 0	4
0 1 1	0 1 1 1 0 0 1	4
1 0 0	1 0 0 1 0 1 1	4
1 0 1	1 0 1 1 1 0 0	4
1 1 0	1 1 0 0 1 0 1	4
1 1 1	1 1 1 0 0 1 0	5

(b) The minimum value of the Hamming weight defines the Hamming distance of the dual code as

$$d_{min} = 4$$

Problem 10.8

(a) For a (5,1) repetition code:

$$G = \left[\begin{array}{ccccc:c} 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right]$$

$$H = \left[\begin{array}{cccc:c} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right]$$

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

The syndrome is

$$s = e H^T$$

where **e** is the error pattern. For a single error, we thus have

Error pattern	Syndrome
0 0 0 0 1	1 1 1 1
0 0 0 1 0	0 0 0 1
0 0 1 0 0	0 0 1 0
0 1 0 0 0	0 1 0 0
1 0 0 0 0	1 0 0 0

(b) For two errors in the received word, we have

Error pattern	Syndrome
0 0 0 1 1	1 1 1 0
0 0 1 0 1	1 1 0 1
0 1 0 0 1	1 0 1 1
1 0 0 0 1	0 1 1 1
0 0 1 1 0	0 0 1 1
0 1 0 1 0	0 1 0 1
1 0 0 1 0	1 0 0 1
0 1 1 0 0	0 1 1 0
1 0 1 0 0	1 0 1 0
1 1 0 0 0	1 1 0 0

We note that the syndromes for all single-error and double-error patterns are distinct. This is intuitively satisfying since a (5,1) repetition code is capable of correcting up to two errors in the received vector

$$y = e + c$$

Problem 10.9

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

$$c(X) = m(X)\,g(X)$$

Hence, we may construct the following table:

Message word	m(X)	c(X)	Code word
0 0 0 0	0	0	0 0 0 0 0 0 0
0 0 0 1	X^3	$X^3 + X^4 + X^6$	0 0 0 1 1 0 1
0 0 1 0	X^2	$X^2 + X^3 + X^5$	0 0 1 1 0 1 0
0 1 0 0	X	$X + X^2 + X^4$	0 1 1 0 1 0 0
1 0 0 0	1	$1 + X + X^3$	1 1 0 1 0 0 0
0 0 1 1	$X^2 + X^3$	$X^2 + X^4 + X^5 + X^6$	0 0 1 0 1 1 1
0 1 1 0	$X + X^2$	$X + X^3 + X^4 + X^5$	0 1 0 1 1 1 0
1 1 0 0	$1 + X$	$1 + X^2 + X^3 + X^4$	1 0 1 1 1 0 0
0 1 0 1	$X + X^3$	$X + X^2 + X^3 + X^6$	0 1 1 1 0 0 1
1 0 1 0	$1 + X^2$	$1 + X + X^2 + X^5$	1 1 1 0 0 1 0
1 0 0 1	$1 + X^3$	$1 + X + X^4 + X^6$	1 1 0 0 1 0 1
0 1 1 1	$X + X^2 + X^3$	$X + X^5 + X^6$	0 1 0 0 0 1 1
1 1 1 0	$1 + X + X^2$	$1 + X^5 + X^6$	1 0 0 0 1 1 0
1 0 1 1	$1 + X^2 + X^3$	$1+X+X^2+X^3+X^4+X^5+X^6$	1 1 1 1 1 1 1
1 1 0 1	$1 + X + X^3$	$1 + X^2 + X^6$	1 0 1 0 0 0 1
1 1 1 1	$1 + X + X^2 + X^3$	$1+ X^3 + X^5 + X^6$	1 0 0 1 0 1 1

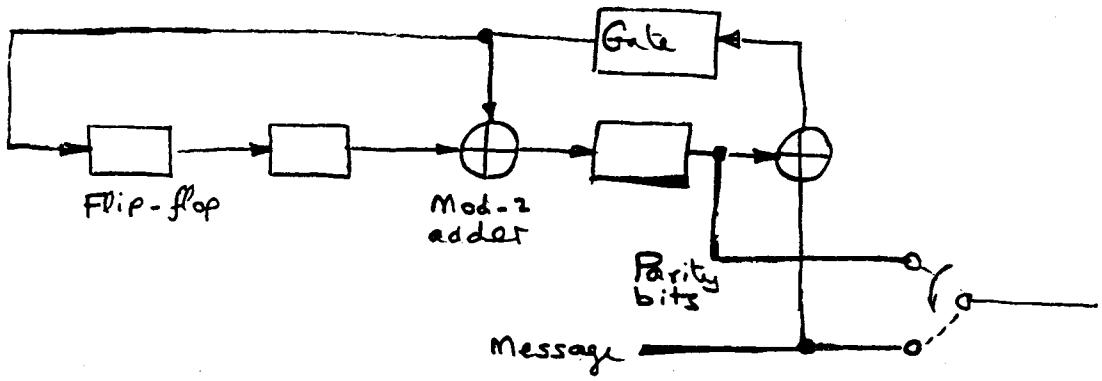
Comparing the code word to the message word, we see that the cyclic code generated by multiplying g(X) and c(X) is not a systematic code.

Problem 10.10

Consider the generator polynomial

$$g(X) = 1 + X^2 + X^3$$

The encoder corresponding to this g(X) is as follows:



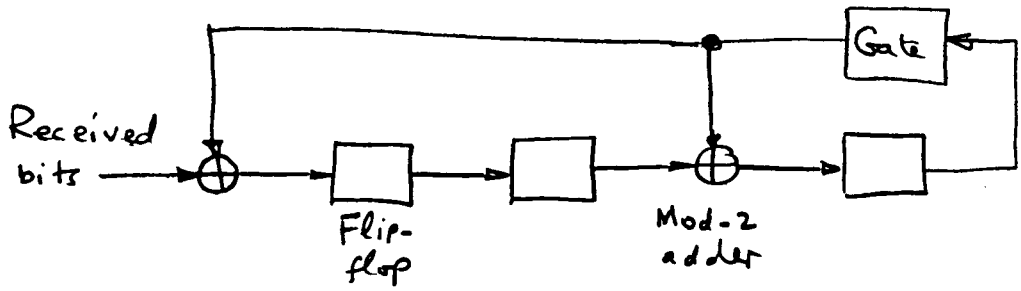
The generator matrix **G** associated with this encoder is

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

To reduce this matrix to a systematic form, we add row 1 to 2, add rows 1 and 2 to row 3, and add rows 2 and 3 to row 4:

$$G = \underbrace{\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}}_{P} \vdots \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{I_4}$$

For the syndrome calculator, we have



Given that

$$G = [P \vdots I_4]$$

$$H = [I_3 \vdots P^T]$$

we find that the parity-check matrix is

$$H = \begin{bmatrix} 1 & 0 & 0 & \vdots & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & \vdots & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & \vdots & 1 & 1 & 0 & 1 \end{bmatrix}$$

In Example 3 of the text, the message sequence (1001) was applied to the encoder and the output code word was 0111001. For the above encoder, the parity bits are 110 and the code word is then 1101001. In particular, we have

Shift	Input	Register contents
		0 0 0
1	1	1 0 1
2	0	1 1 1
3	0	1 1 0
4	1	1 1 0

If we were to make an error in the middle bit and receive 1100001, then circulating it through the syndrome calculator, we have

Shift	Input	Register contents
		0 0 0
1	1	1 0 0
2	0	0 1 0
3	0	0 0 1
4	0	1 0 1
5	0	1 1 1
6	1	0 1 0
7	1	1 0 1

From the parity check matrix we see that the syndrome calculator output 101 corresponds to the error pattern 0001000. The corrected code word is therefore 1101001.

Problem 10.11

The error polynomial is

$$e(X) = r(X) + c(X)$$

We are given

$$c(X) = X + X^2 + X^3 + X^6$$

$$r(X) = X + X^3 + X^6$$

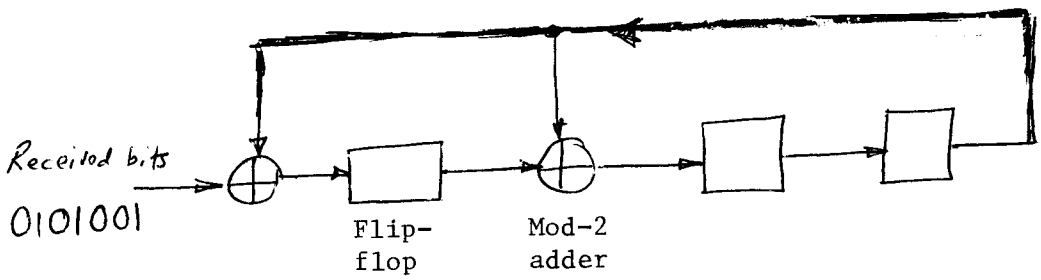
The error polynomial is therefore

$$e(X) = X^2$$

Consider next the syndrome polynomial s(X). The syndrome calculator for the generator polynomial

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

is shown in Fig. 10.11; this calculator is reproduced here for convenience of presentation:



Circulating the received bits through the syndrome calculator, we may construct the following table:

Initial state	0 0 0
	1 0 0
	0 1 0
	0 0 1
	0 1 0
	0 0 1
	0 1 0
	0 0 1

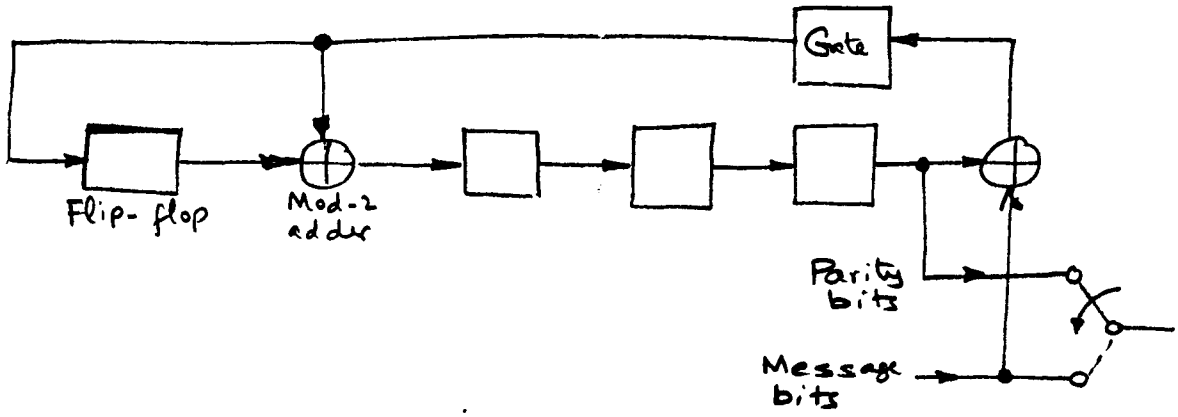
Here, the syndrome polynomial is

$$s(X) = X^2$$

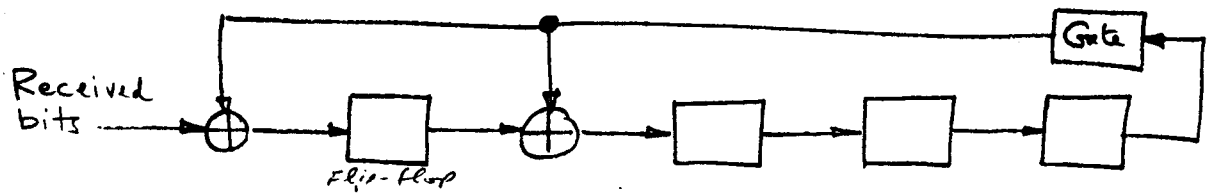
which, for the problem at hand, is the same as the error polynomial. This result demonstrates the property of the syndrome polynomial, stating that it is the same as the error polynomial when the transmission errors are confined to the parity-check bits. In Problem 11.11 the third parity-check bit is received in error.

Problem 10.12

The encoder structure is



The syndrome calculator is

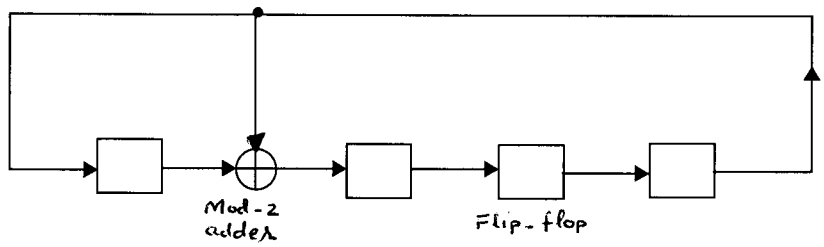


Problem 10.13

(a) A maximal-length code is the dual of the corresponding Hamming code. The generator polynomial of a (15,11) Hamming code is given as $1 + X + X^4$. We may therefore define the feedback connections of the corresponding (15,4) maximal-length code by choosing the primitive polynomial

$$h(X) = 1 + X + X^4$$

The feedback connections are therefore [4,1], which agrees with entry 3 of Table 7.1. Specifically, the encoder of the (15,4) maximal-length encoder is as follows:



(b) The generator polynomial of the (15,4) maximal-length code is

$$g(X) = \frac{1 + X^{15}}{h(X)} = \frac{1 + X^{15}}{1 + X + X^4}$$

Performing this division modulo-2, we obtain

$$g(X) = 1 + X + X^2 + X^3 + X^5 + X^7 + X^8 + X^{11}$$

(This computation is left as an exercise for the reader.) Assuming that the initial state of the encoder is 0001, we find that the output sequence is (111101011001000). Here we recognize that the length of the coded sequence is $2^4 - 1 = 15$. The output sequence repeats itself periodically every 15 bits.

Problem 10.14

(a) $n = 2^m - 1 = 31$ symbols

Hence, the number of bits per symbol in the code is

$m = 5$ bits

(b) Block length = 31 x 5 = 155 bits

(c) Minimum distance of the code is

$$\begin{aligned} d_{min} &= 2t + 1 \\ &= n-k + 1 \\ &= 31 - 15 + 1 \\ &= 17 \text{ symbols} \end{aligned}$$

(d) Number of correctable symbols is

$$\begin{aligned} t &= \frac{1}{2}(n - k) \\ &= 8 \text{ symbols} \end{aligned}$$

Problem 10.15

The encoder is realized by inspection:

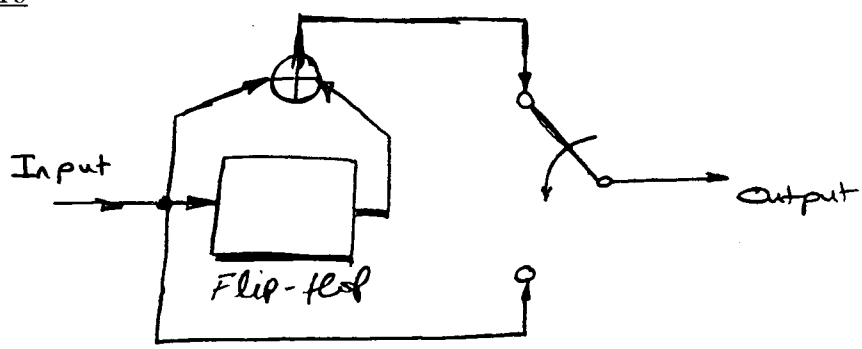
$g^{(1)} = (1,0,1)$

$g^{(2)} = (1,1,0)$

$g^{(3)} = (1,1,1)$

For the Hamming code, the parity check matrix **H** is more structured than that for the repetition code. Indeed, the matrix **H** for the Hamming code includes that for the repetition code as a submatrix.

Problem 10.16



Using this encoder, we may construct the following table by inspection:

Message	1	0	1	1	1	1	...
Output	11	10	11	01	01	01	...

Original message

The code is in fact systematic.

Problem 10.17

The generator polynomials are

$$g^{(1)}(X) = 1 + X + X^2 + X^3$$

$$g^{(2)}(X) = 1 + X + \quad X^3$$

The message polynomial is

$$m(X) = 1 + X^2 + X^3 + X^4 + \ldots$$

Hence,

$$c^{(1)}(X) = g^{(1)}(X)\, m(X)$$

$$= 1 + X + X^3 + X^4 + X^5 + \ldots.$$

$$c^{(2)}(X) = g^{(2)}(X)\, m(X)$$

$$= 1 + X + X^2 + X^3 + \quad X^6 + X^7 + \ldots$$

Hence,

$$\{c^{(1)}\} = 1,1,0,1,1,1,...$$

$$\{c^{(2)}\} = 1,1,1,1,0,0,...$$

The encoder output is therefore 11, 11, 01, 11, 10, 10.

Problem 10.18

The encoder of Fig. 10-13 (b) has three generator sequences for each of the two input paths; they are as follows (from top to bottom)

$$g_1^{(1)} = (1,1) \qquad g_1^{(2)} = (1,0), \qquad g_1^{(3)} = (1,1)$$

$$g_2^{(1)} = (0,1), \qquad g_2^{(2)} = (1,1), \qquad g_2^{(3)} = (0,0)$$

Hence,

$$g_1^{(1)}(X) = 1 + X, \qquad g_1^{(2)}(X) = 1, \qquad g_1^{(3)}(X) = 1 + X$$

$$g_2^{(1)}(X) = X, \qquad g_2^{(2)}(X) = 1 + X, \qquad g_2^{(3)}(X) = 0$$

The incoming message sequence 10111... enters the encoder two bits at a time; hence

$$m^{(1)} = 1\ 1\ ....$$

$$m^{(2)} = 0\ 1\ ....$$

The message polynomials are therefore

$$m_1(X) = 1 + X + ...$$

$$m_2(X) = X + ...$$

Hence, the output polynomials are

$$c^{(1)}(X) = g_1^{(1)}(X)\, m_1(X) + g_2^{(1)}(X)\, m_2(X)$$

$$= (1 + X)(1 + X + \ldots) + X(X + \ldots)$$

$$= 1 + \ldots.$$

$$c^{(2)}(X) = g_1^{(2)}(X)\, m_1(X) + g_2^{(2)}(X)\, m_2(X)$$

$$= (1)\,(1 + X + \ldots) + (1 + X)\,(X + \ldots)$$

$$= 1 + X + \ldots \quad + X + X^2 + \ldots$$

$$= 1 + X^2 + \ldots$$

$$c^{(3)}(X) = g_1^{(3)}(X)\, m_1(X) + g_2^{(3)}(X)\, m_2(X)$$

$$= (1 + X)\,(1 + X + \ldots) + (0)\,(X + \ldots)$$

$$= 1 + X^2 + \ldots$$

The output sequences are correspondingly as follows:

$c^{(1)} = 1, 0, \ldots$

$c^{(2)} = 1, 0, \ldots$

$c^{(3)} = 1, 0, \ldots$

The encoder output is therefore (1,1,1), (0,0,0), ...

Problem 10.19

1. If the input sequence is 00, the encoder output is 00, 00, 00, 00.

2. If the input sequence is 11, the message polynomial is

$$m(X) = 1 + X$$

The two generator polynomials are

$$g^{(1)}(X) = 1 + X + X^2$$

$$g^{(2)}(X) = 1 + X^2$$

Hence,

$$c^{(1)}(X) = (1 + X)(1 + X + X^2)$$

$$= 1 + X^3$$

$$c^{(2)}(X) = (1 + X)(1 + X^2)$$

$$= 1 + X + X^2 + X^3$$

The encoder output is 11, 01, 01, 11

3. If the input sequence is 01, the message polynomial is

$$m(X) = X$$

Hence,

$$c^{(1)}(X) = X(1 + X + X^2)$$

$$= X + X^2 + X^3$$

$$c^{(2)}(X) = X(1 + X^2)$$

$$= X + X^3$$

The encoder output is 00, 11, 10, 11

4. Finally, if the input sequence is 10, the message polynomial is

$$m(X) = 1$$

Correspondingly,

$$c^{(1)}(X) = g^{(1)}(X)$$

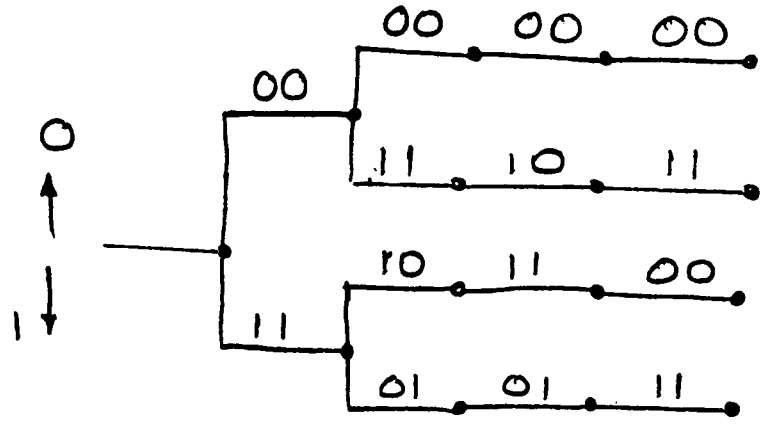
$$= 1 + X + X^2$$

$$c^{(2)}(X) = g^{(2)}(X)$$

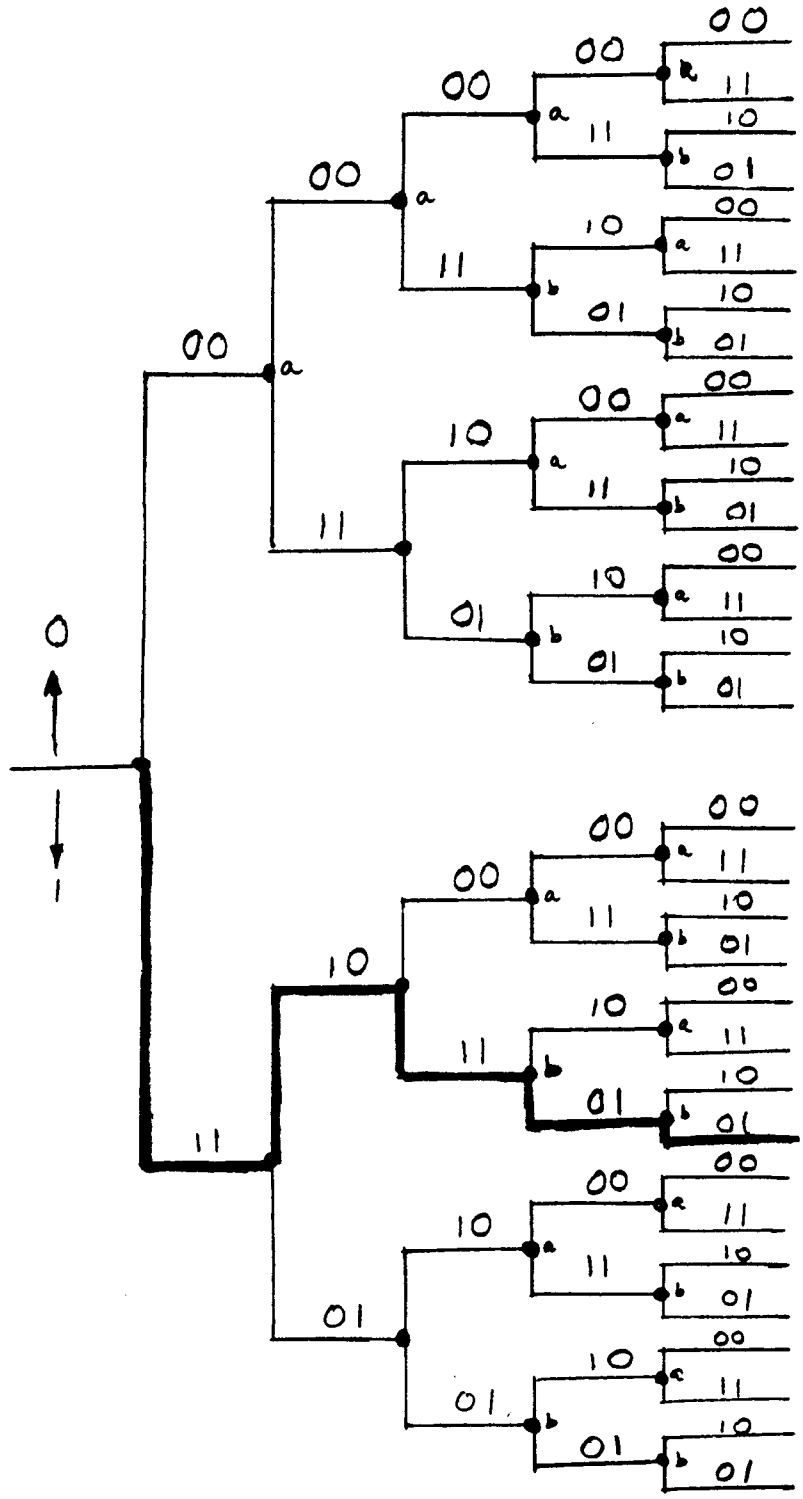
$$= 1 + X^2$$

Hence, the encoder output is 11, 10, 11, 00.

The encoder outputs calculated above are in perfect accord with the entries of the code tree:

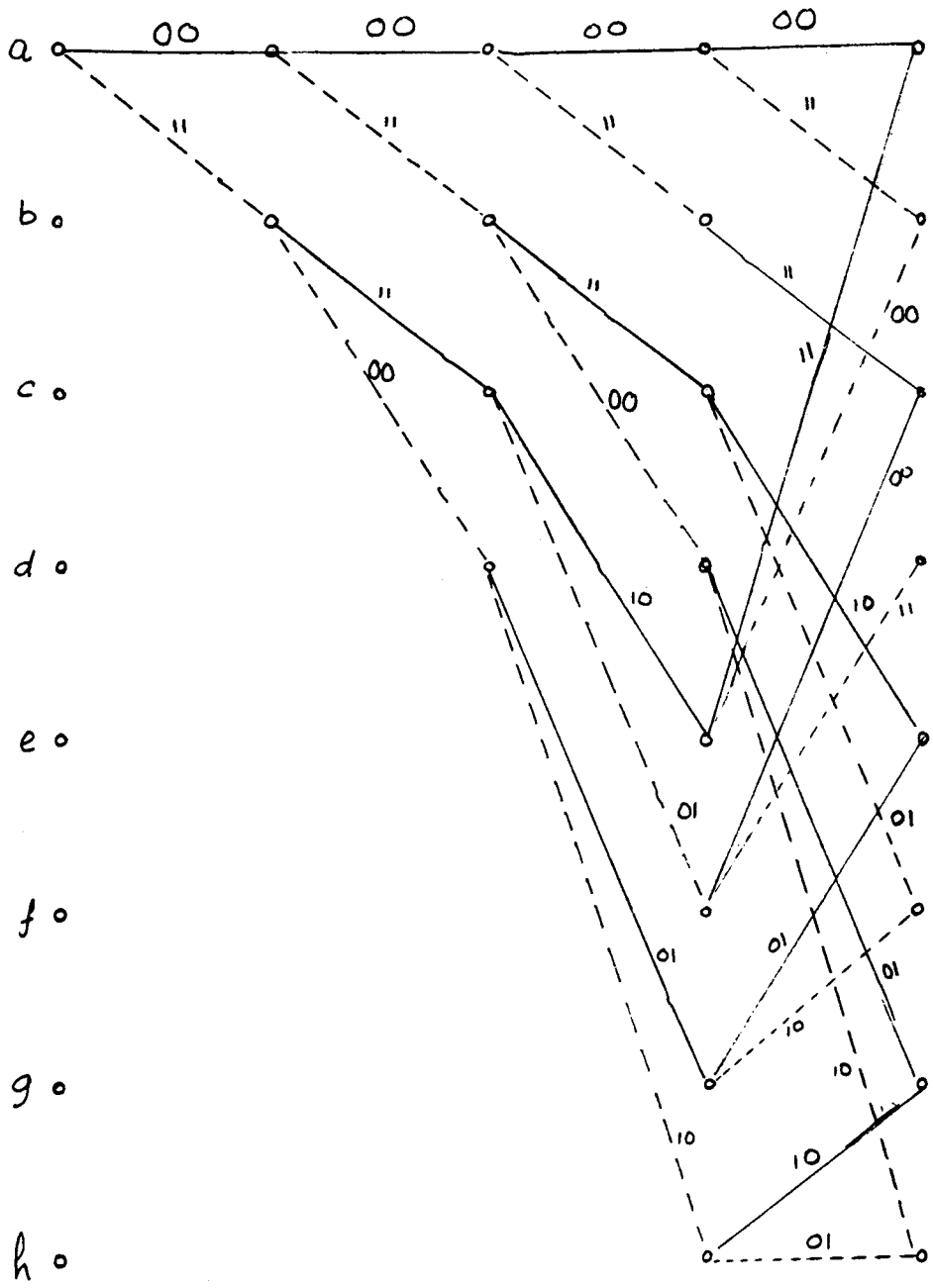


Problem 10.20



The output sequence is 11, 10, 11, 01, 01

Problem 10.21

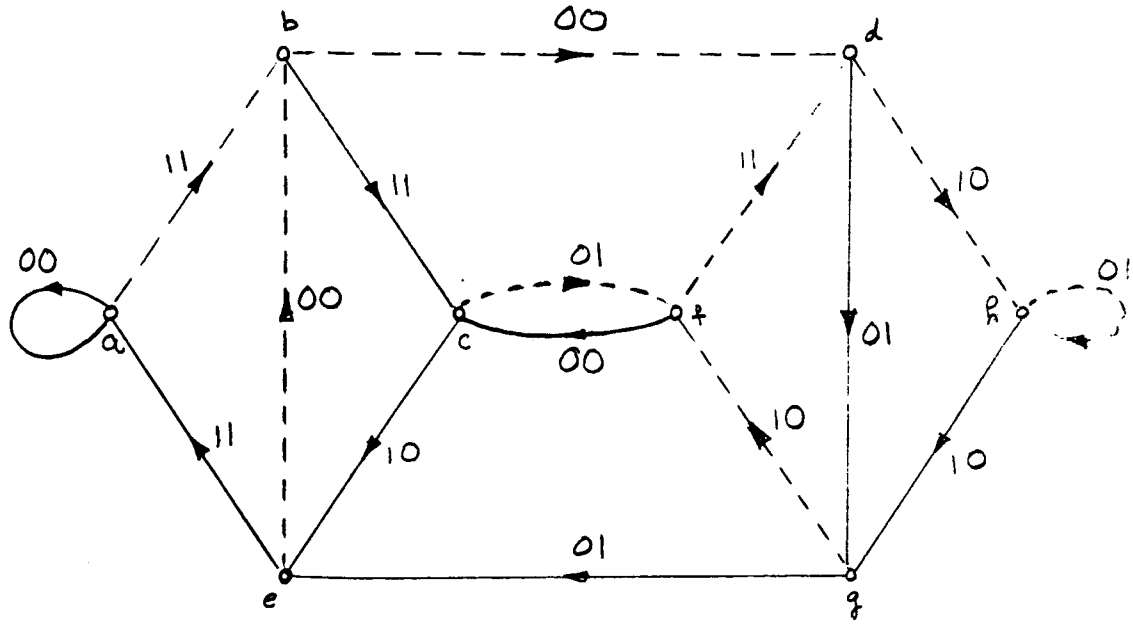


Problem 10.22

The encoder of Fig. P10-17 has eight states:

State	Register contents
a	0 0 0
b	1 0 0
c	0 1 0
d	1 1 0
e	0 0 1
f	1 0 1
g	0 1 1
h	1 1 1

The state diagram of the encoder is as follows:



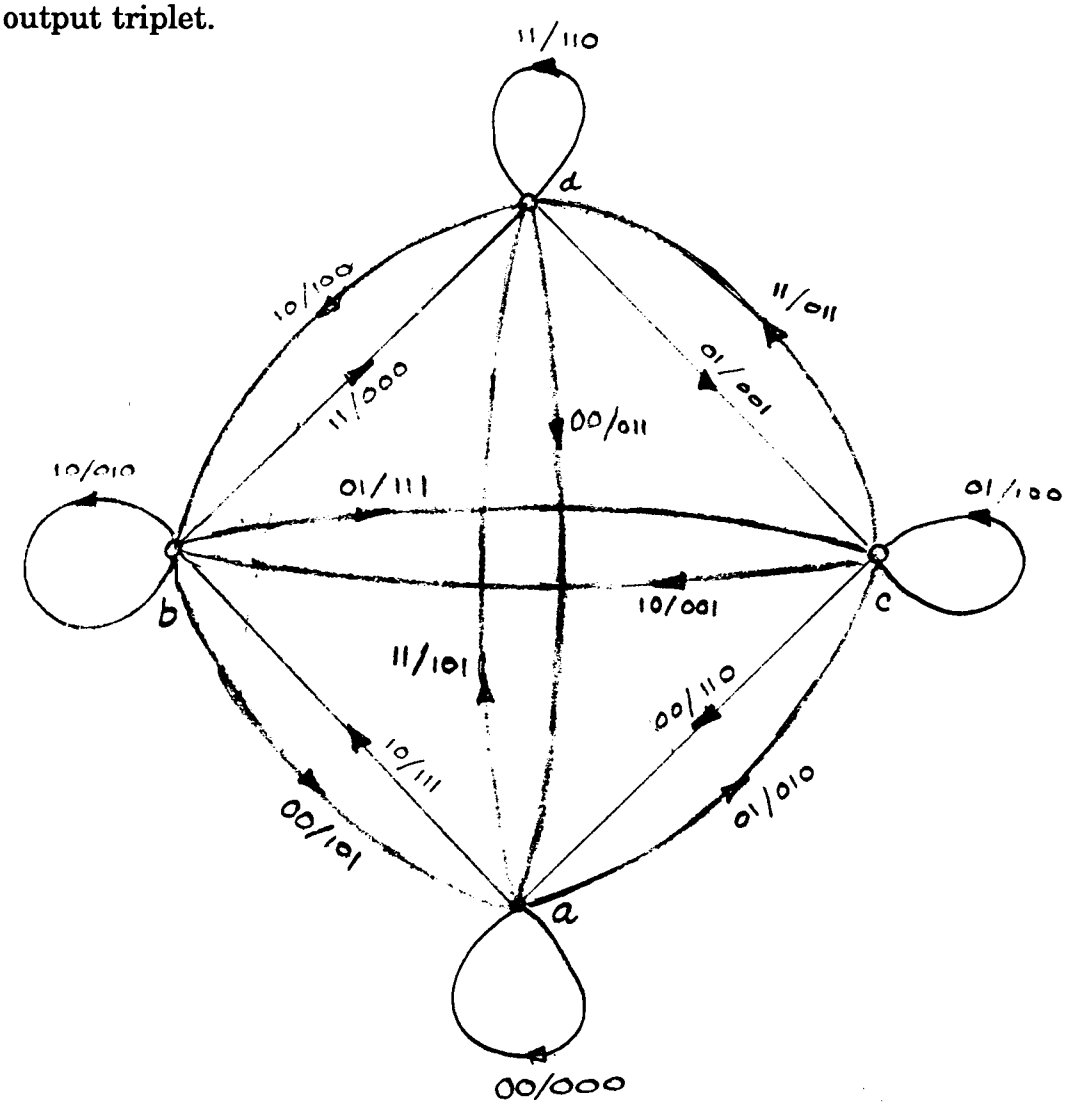
In this diagram, a solid line corresponds to an input of 0 and a dashed line corresponds to an input of 1.

Problem 10.23

(a) The encoder of Fig. 10-13b has four states:

State	Register contents
a	0,0
b	1,0
c	0,1
d	1,1

In the state diagram shown below, each branch is labeled with the input dibit followed by the output triplet.



(b) Starting from the all-zero state a, the incoming sequence 10111... produces the path a b d ... Equivalently, we have the decoded (output) sequence (111), (000), ..., which is exactly the same result calculated in Problem 10.18.

Problem 10.24

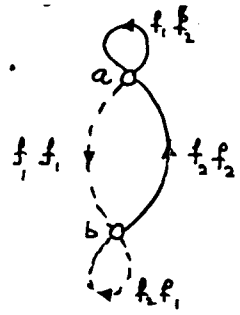
An MSK system has two distinct phase states

State	Phase,radians
a	0
b	π

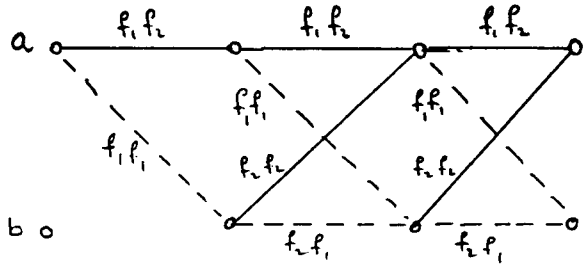
The transmission of a 1 increases the phase by $\pi/2$, whereas the transmission of a 0 decreases the phase by $\pi/2$. Correspondingly, the transmission of dibit 10 or 01 leaves the state of MSK unchanged, whereas the transmission of dibit 00 or 11 movesthe system from one state to the other. For the output, we have

Input dibit	Output frequencies
1 1	f_1 f_1
0 1	f_2 f_1
1 0	f_1 f_2
0 0	f_2 f_2

We may thus construct the following state diagram for MSK:

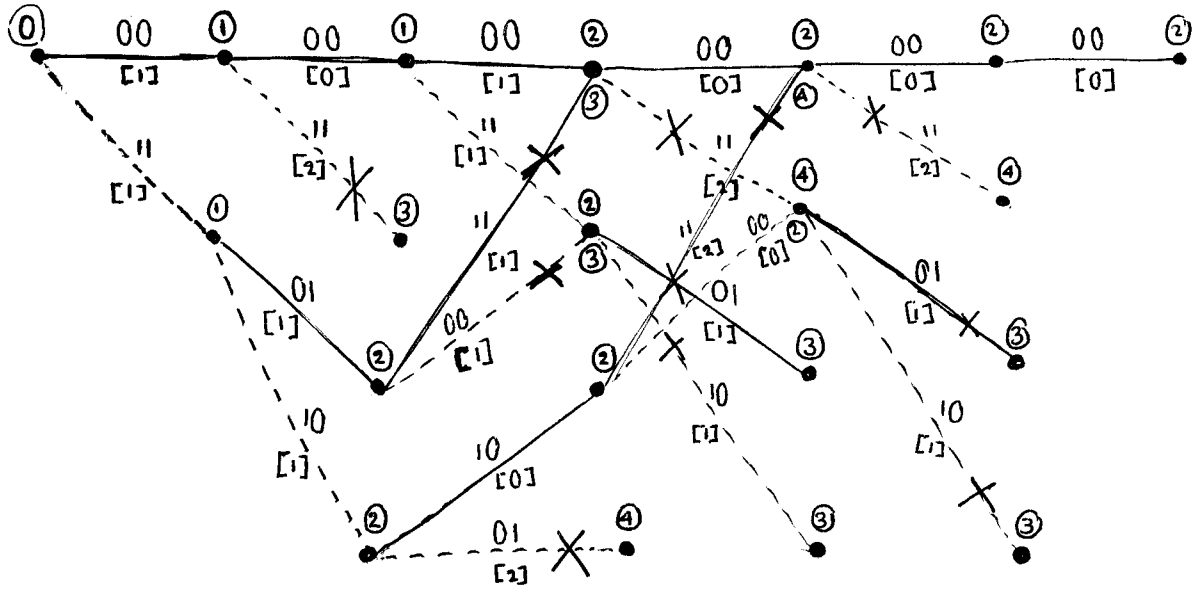


The trellis diagram for MSK is as follows:



Since $d_{min} = 5$ and the number of errors in the received sequence is 2, it should be possible to decode the correct sequence. This is readily demonstrated by applying the Viterbi algorithm.

Problem 10.25



Received bits: 10 00 10 00 00 00

Notations

- ⓝ path metric
- [n] branch metric
- — message bit 0
- ---- message bit 1
- ✕ deleted path

From the above figure we see that the decoded sequence is 00 00 00 00 00 00 ..., thereby correcting for the two errors in the received sequence.

Problem 10.26

(a) Coding gain for binary symmetric channel is

$$G_a = 10 \log_2 \left(\frac{10 \text{ x } 1/2}{2} \right)$$

$$= 10 \log_{10} 2.5$$

$$= 4 \text{ dB}$$

(b) Coding gain for additive white Gaussian noise channel is

$$G_a = 10 \log_{10} \left(10 \text{ x } \frac{1}{2} \right)$$

$$= 10 \log_{10} 5$$

$$= 7 \text{ dB}$$

Problem 10.27

The trellis of Fig. P10.27 corresponds to binary data transmitted through a dispersive channel, viewed as a finite-state (i.e., two-state) machine. There are two states representing the two possible values of the previous channel bit. Each possible path through the trellis diagram of Fig. P10.27 corresponds to a particular data sequence transmitted through the channel.

To proceed with the application of the Viterbi algorithm to the problem at hand, we first note that there are two paths of length 1 through the trellis; their squared Euclidean distances are as follows:

$$d_{1,1}^2 = (1.0 - 1.1)^2 = 0.01$$

$$d_{1,2}^2 = (1.0 - (-.9))^2 = 3.61$$

Each of these two paths is extended in two ways to form four paths of length 2; their squared Euclidean distances from the received sequence are as follows:

(a)

$$d^2_{2,1} = 0.01 + (0.0 - 1.1)^2 = 1.22$$

$$d^2_{2,2} = 3.61 + (0.0 - 0.9)^2 = 4.42$$

(b)

$$d^2_{2,3} = 0.01 + (0.0 - (-0.9))^2 = 0.82$$

$$d^2_{2,4} = 3.61 + (0.0 - (-1.1))^2 = 4.82$$

Of these four possible paths, the first and third ones (i.e., those corresponding to squared Euclidean distances $d^2_{2,1}$ and $d^2_{2,3}$) are selected as the "survivors", which are found to be in agreement. Accordingly, a decision is made that the demodulated symbol $a_0=1$.

Next, each of the two surviving paths of length 2 is extended in two ways to form four new paths of length 3. The squared Euclidean distances of these four paths from the received sequence are as follows:

(a)

$$d^2_{3,1} = 1.22 + (0.2 - 1.1)^2 = 2.03$$

$$d^2_{3,2} = 0.82 + (0.2 - 0.9)^2 = 1.31$$

(b)

$$d^2_{3,3} = 1.22 + (0.2 - (-0.9))^2 = 2.43$$

$$d^2_{3,4} = 0.82 + (0.2 - (-1.1))^2 = 2.51$$

This time, the second and third paths (i.e., those corresponding to the squared Euclidean distances $d^2_{3,2}$ and $d^2_{3,3}$) are selected as the "survivors". However, no decision can be made on the demodulated symbol a_1 as the two paths do not agree.

To proceed further, the two surviving paths are extended to form two paths of length 4. The squared Euclidean distances of these surviving paths are as follows:

(a)

$$d^2_{4,1} = 1.31 + (-1.1 - 1.1)^2 = 6.15$$

$$d^2_{4,2} = 2.43 + (-1.1 - 0.9)^2 = 6.43$$

(b)

$$d^2_{4,3} = 1.31 + (-1.1 - (-0.9))^2 = 1.35$$

$$d^2_{4,4} = 2.43 + (-1.1 - (-1.1))^2 = 2.43$$

The first and third paths are therefore selected as the "survivors", which are now found to agree in their first three branches. Accordingly, it is decided that the demodulated symbols are $a_0 = +1$, $a_1 = -1$, and $a_2 = +1$. It is of interest to note that although we could not form a decision on a_1 after the third iteration of the Viterbi algorithm, we are able to do so after the fourth iteration.

Figure 1 shows, for the problem at hand, how the trellis diagram is pruned as the application of the Viterbi algorithm progresses through the trellis of Fig. P11.5

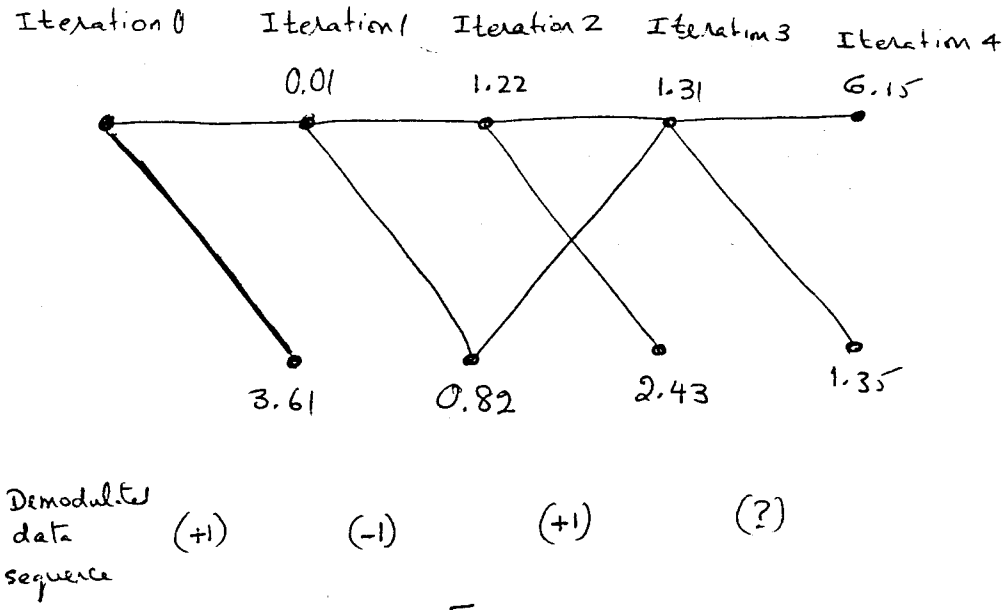
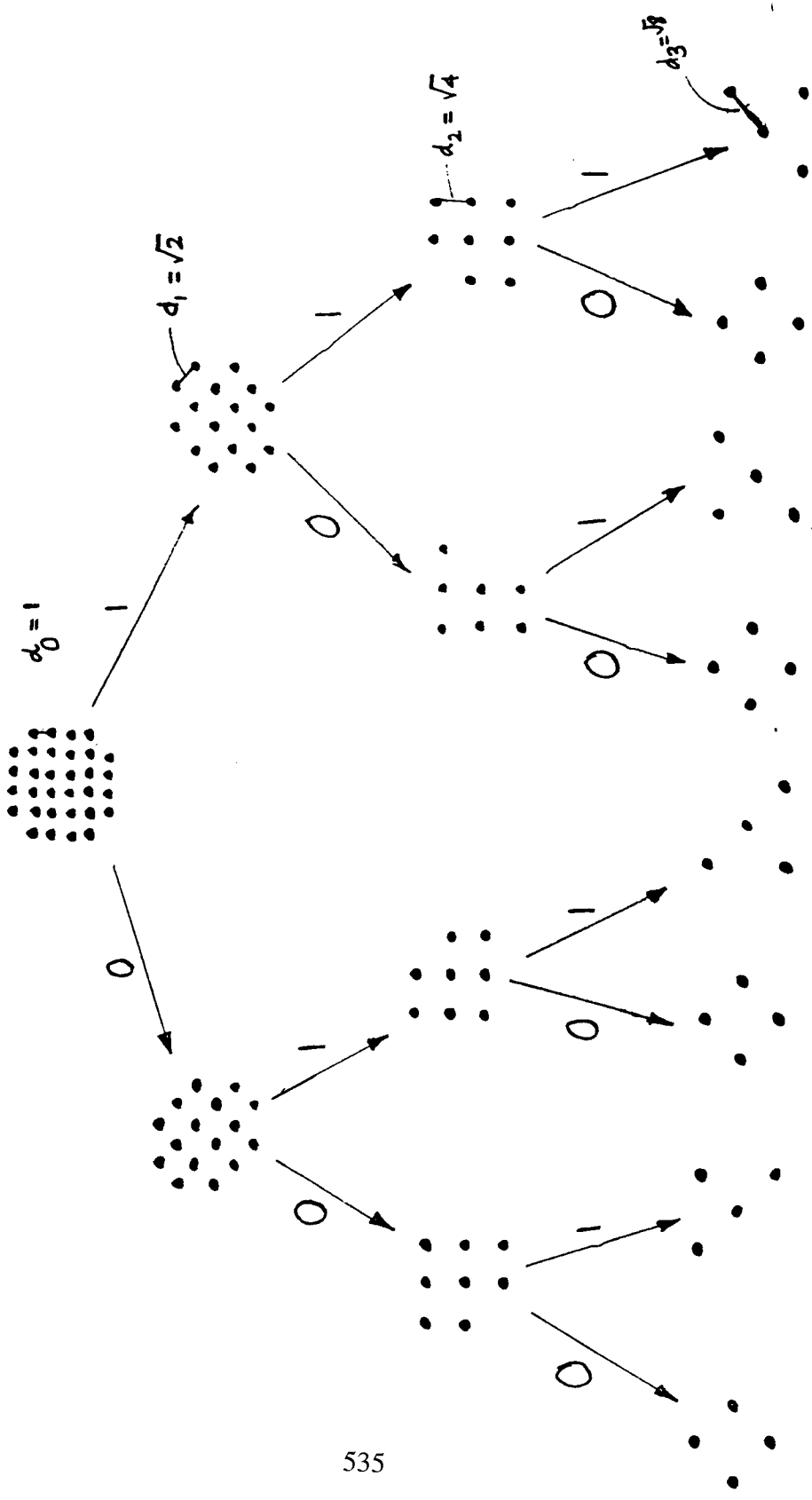


Fig. 1

(This problem is taken from R.E. Blahut, "Digital Transmission of Information", Addison-Wesley, 1990, pp. 144-149.The interested reader may consult this book for a more detailed treatment of the subject.)

Problem 10.28



Problem 10.29

(a) Without coding, the required E_b/N_0 is 12.5 dB. Given a coding gain of 5.1 dB, the required E_b/N_0 is reduced to

$$\left(\frac{E_b}{N_0}\right)_{\text{req}} = 12.5 - 5.1$$

$$= 7.4 \text{ dB}$$

For the downlink, the equation for C/N_0 is

$$\left(\frac{C}{N_0}\right)_{\text{downlink}} = \text{EIRP} + \frac{G_r}{T} - L_{\text{free-space}} + k$$

(b) By definition, the formula for receive antenna gain is

$$G_r = \frac{4\pi A_r}{\lambda^2}$$

where A_r is the receive antenna aperture and λ is the wavelength. Let $(A_r)_{\text{coding}}$ denote the receive antenna aperture that results from the use of coding. Hence

$$10\log_{10}\left(\frac{4\pi A_r}{\lambda^2}\right) - 10\log_{10}\left(\frac{4\pi (A_r)_{\text{coding}}}{\lambda^2}\right) = 5.1 \text{ dB}$$

or, equivalently,

$$10\log_{10}\left(\frac{A_r}{(A_r)_{\text{coding}}}\right) = 5.1 \text{ dB}$$

Hence,

$$\frac{A_r}{(A_r)_{\text{coding}}} = \text{antilog } 0.51 = 3.24$$

The antenna aperture is therefore reduced by a factor of 3.24 through the use of coding. Expressing this result in terms of the antenna dish diameter, d,we may write

$$\frac{\pi d^2/4}{\pi(d_{\text{coding}})^2/4} = \left(\frac{d}{d_{\text{coding}}}\right)^2 = 3.24$$

which yields

$$\frac{\text{Diameter of antenna without coding}}{\text{Diameter of antenna with coding}} = \frac{d}{d_{\text{coding}}} = \sqrt{3.24} = 1.8$$

That is, the antenna diameter is reduced by a factor of 1.8 through the use of coding.

Problem 10.30

Nonlinearity of the encoder in Fig. P10.30 is determined by adding (modulo-2) in a bit-by-bit manner a pair of sets of values of the five input bits $\{I_{1,n}, I_{2,n-1}, I_{1,n-2}, I_{2,n}, I_{2,n-1}\}$ and the associated pair of sets of values of the three output bits $Y_{0,n}$, $Y_{1,n}$ and $Y_{2,n}$,. If the result of adding these two sets of values of input bits, when it is treated as a new set of values of output-bits, does not always give a set of values of input bits identical to the result of adding the two sets of values of the aforementioned output bits, then the convolutional encoder is said to be nonlinear, For example, consider two sets of values for the sequence $\{I_{1,n}, I_{1,n-1}, I_{1,n-2}, I_{2,n}, I_{2,n-1}\}$, that are given by {0,0,1,1,1} and {0,1,0,0,0}. The associated sets of values of the three output bits $Y_{0,n}$, $Y_{1,n}$, $Y_{2,n}$, are {0,1,1} and {1,0,0}, respectively. If the 5-bit sets are passed through the Exclusive OR (i.e., mod-2 adder) bit-by-bit, the result is {0,1,1,1,1}. If the resulting set {0,1,1,1,1} is input into the encoder, then the associated output bits are {1,1,0}. However, when the sets of output bits {0,1,1} and {1,0,0} are passed through the Exclusive OR, bit-by-bit, the result is {1,1,1}. Since the two results {1,1,0} and {1,1,1} are different, it follows that the convolutional encoder of Fig. P10.30 is nonlinear.

Problem 10.31

Let the code rate of turbo code be R. We can write

$$\left(\frac{1}{R} - 1\right) = \left(\frac{1}{r_c^{(1)}} - 1\right) + \left(\frac{1}{r_c^{(2)}} - 1\right)$$

$$\frac{1}{R} = \left(\frac{1}{r_c^{(1)}}\right) + \left(\frac{1}{r_c^{(2)}} - 1\right)$$

$$= \left(\frac{q_1}{p} + \frac{q_2}{p} - 1\right)$$

$$= \frac{q_1 + q_2 - p}{p}$$

Hence

$$R = p/(q_1 + q_2 - p)$$

Problem 10.32

Figure 1 is a reproduction of the 8-state RSC encoder of Figure 10.26 used as encoder 1 and encoder 2 in the turbo encoder of Fig. 10.25 of the textbook. For an input sequence consisting of symbol 1 followed by an infinite number of symbols 0, the outputs of the RSC encoders will contain an infinite number of ones as shown in Table 1.

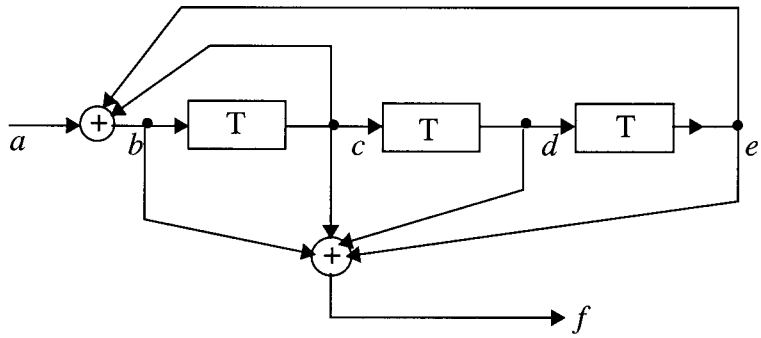


Fig. 1

$b = a \oplus c \oplus e$

$f = b \oplus c \oplus d \oplus e$

Initial conditions: $c = d = e = 0$ {empty}

(Input)	Intermediate inputs				(output)
a	b	c	d	e	f
1	1	0	0	0	1
0	1	1	0	0	0
0	1	1	1	0	1
0	0	1	1	1	1
0	1	0	1	1	1
0	0	1	0	1	0
0	0	0	1	0	1
0	1	0	0	1	0
0	1	1	0	0	0

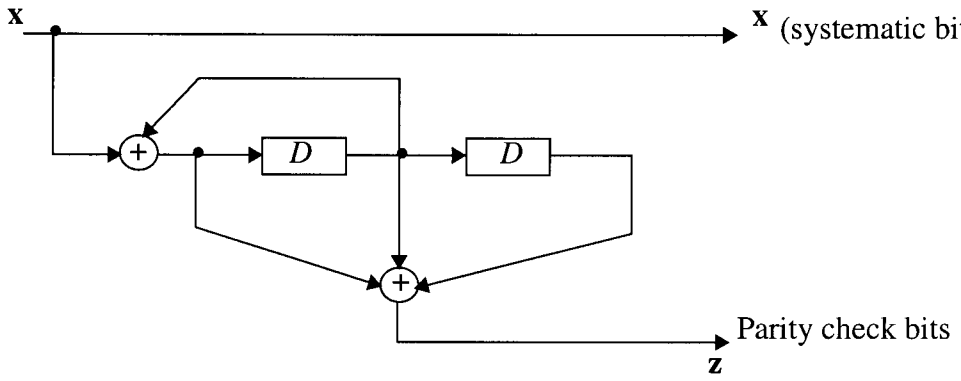
The output is 10111010011101001111...

Therefore, an all zero sequence with a single bit error (1) will cause an infinite number of channel errors.

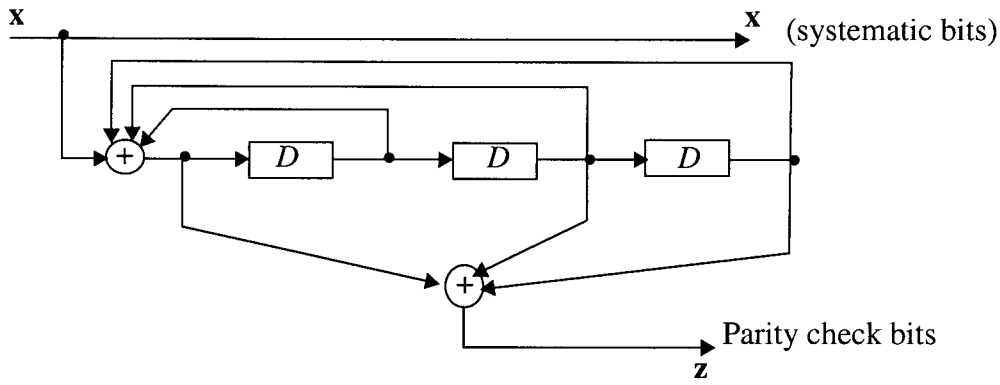
[Note: The all zero input sequence produces an all zero output sequence.]

Problem 10.33

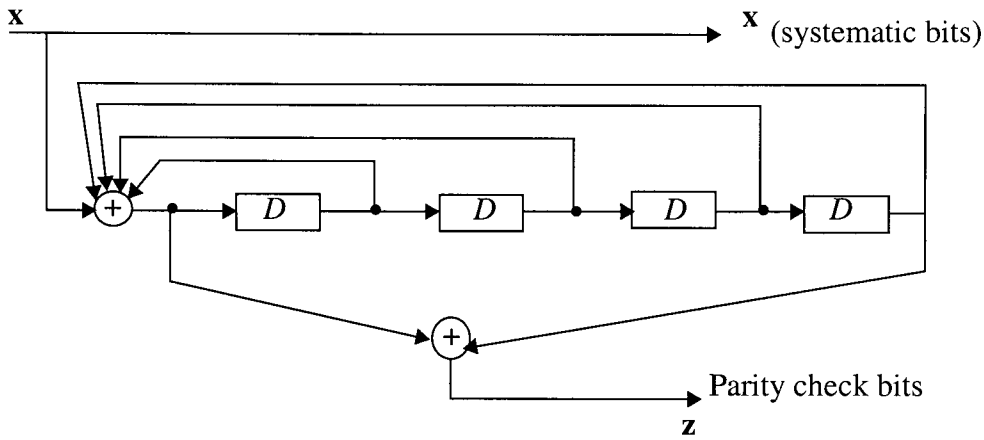
(a) 4-state encoder



8-state encoder



16-state encoder



(b) 4-state encoder

$$\mathbf{g}(D) = \left[1, \frac{1 + D + D^2}{1 + D^2}\right]$$

By definition, we have

$$\left(\frac{B(D)}{M(D)}\right) = \frac{1 + D + D^2}{1 + D^2}$$

where $B(D)$ denotes the transform of the parity sequence $\{b_i\}$ and $M(D)$ denotes the transform of the message sequence $\{m_i\}$. Hence,

$$(1 + D^2)B(D) = (1 + D + D^2)M(D)$$

The parity-check equation is given by

$$(m_i + m_{i-1} + m_{i-2}) + (b_i + b_{i-2}) = 0$$

where the addition is modulo-2.

Similarly for the 8-state encoder, we find that the parity-check equation is

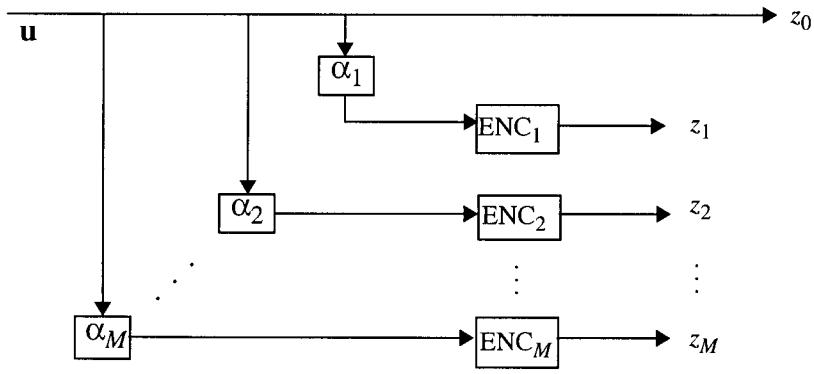
$$m_i + m_{i-2} + m_{i-3} + b_i + b_{i-1} + b_{i-2} + b_{i-3} = 0$$

For the 16-state encoder, the parity-check equation is

$$m_i + m_{i-4} + b_i + b_{i-1} + b_{i-2} + b_{i-3} + b_{i-4} = 0$$

Problem 10.34

(a) Encoder



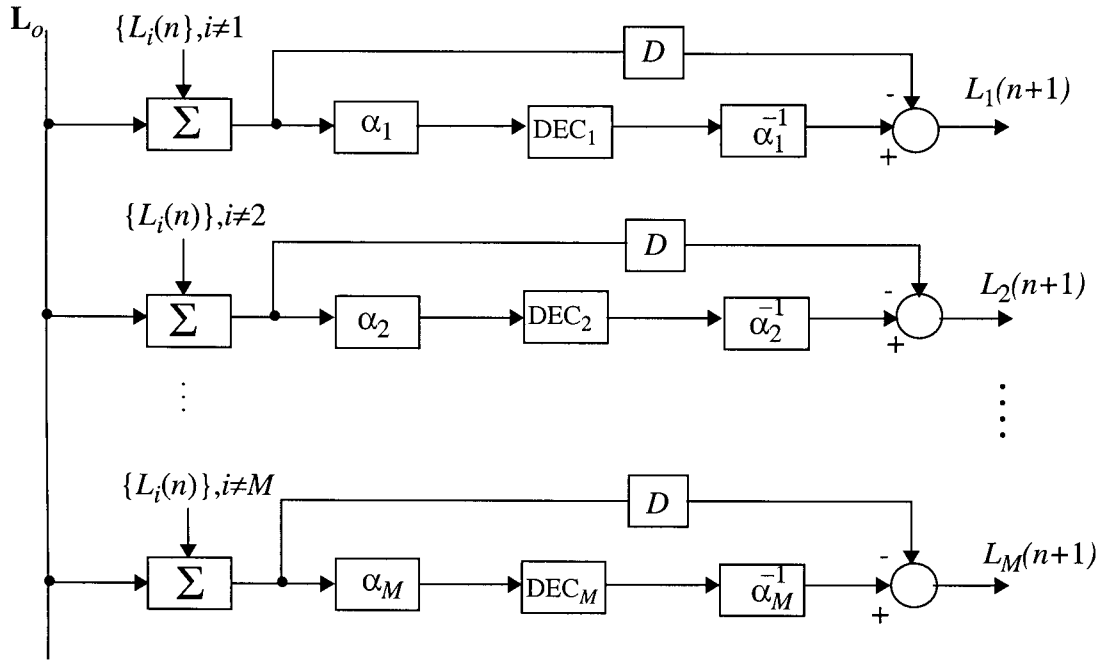
$\alpha_1, \alpha_2, \cdots, \alpha_M$ are M interleavers

$ENC_1, ENC_2, \cdots, ENC_M$ are M recursive systematic convolutional (RSC) encoders

z_o is the message sequence

$z_1, z_2, \cdots, z_M$ are the resulting M parity sequences

(b) Decoder



$\alpha_1^{-1}, \alpha_2^{-1}, \cdots, \alpha_m^{-1}$ are de-interleavers.

The generalized encoder and decoder presented here are described in Valenti (1998); see the Bibliography.

Problem 10.35

The decoding scheme used for turbo codes relies on the assumption that the bit probabilities remain independent from one iteration to the next. To maintain as much independence as possible from one iteration to the next, only extrinsic information is fed from one stage to the next, since the input and the output of the same stage will be highly correlated. However, this correlation decreases as $|t_1 - t_2|$ increases, where t_1, t_2 are any two time instants. The interleaving is utilized to spread correlation information outside of the memory of subsequent decoder stages.

Problem 10.36

The basic idea behind the turbo principle is to use soft information from one stage as input to the next stage in an iterative fashion. For a joint demodulator/decoder, this could be arranged as shown in Fig. 1.

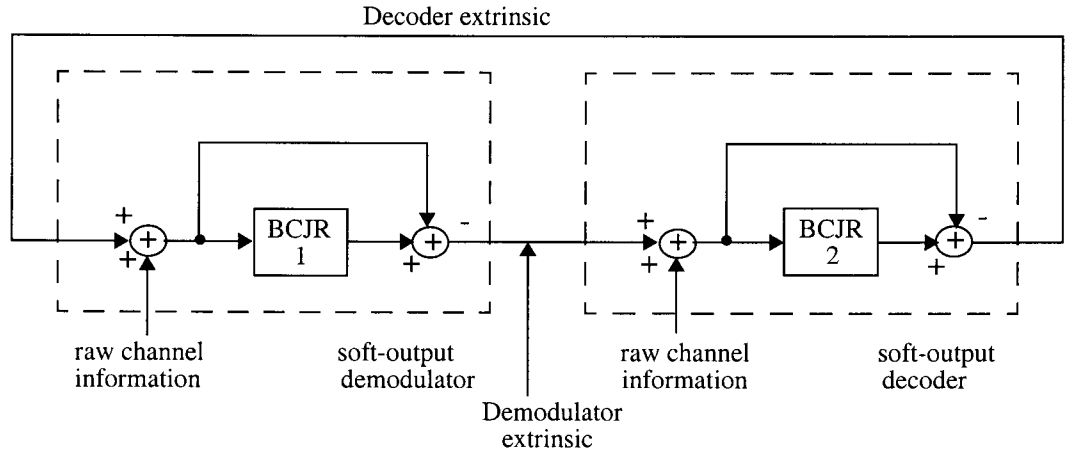


Figure 1

In this figure, BCJR 1 is a MAP decoder corresponding to the Markov model of the modulator and channel; and BCJR 2 is a MAP decoder corresponding to the Markov model of the forward error correction code. The raw channel information is fed into the soft demodulator on the first iteration; this is combined with the extrinsic information from the previous decoding stage on subsequent iterations. The extrinsic information from the soft-output demodulation stage plus the raw channel information is the input to the decoding stage. Feeding back the extrinsic information from the latter stage closes the loop. At any stage the output from the decoder can be used to estimate the data. (Figure 1 shows a symmetric implementation. Other arrangements are possible.)

Problem 10.37

Matlab codes

```
% Probelm 10.37 , CS: Haykin
% Turbo coding
%M. Sellathurai

clear all

% Block size
block_size = 400; % 200 and 400

% Convolutional code polynomial
code_polynomial = [ 1 1 1; 1 0 1 ];
[n,K]=size(code_polynomial);
m=K-1;

% Code rate for  punctured code
code_rate = 1/2;

% Number of iterations
no_of_iterations = 5;

% Number of blocks in error for termination
block_error_limit = 15;

% signal-to-noise-ratio in db
 SNRdb = [1];
 snr = 10^(SNRdb/10);

%  channel reliability value and variance of AWGN channel
 channel_reliability_value = 4*snr*code_rate;
  noise_var = 1/(2*code_rate*snr);

%initializing the error counters
block_number = 0;
block_errors(1,1:no_of_iterations) = zeros(1, no_of_iterations);
bit_errors(1,1:no_of_iterations) = zeros(1, no_of_iterations);
total_errors=0;

while block_errors(1,  no_of_iterations)< block_error_limit
   block_number=block_number+1;

% Transmitter end
% generating random data
```

```
    Data = round(rand(1, block_size-m));
% random scrambler
   [dummy, Alpha] = sort(rand(1,block_size));
% turbo-encorder output
     turbo_encoded = turbo_encorder( Data, code_polynomial, Alpha) ;

% Receiver end
% AWGN+turbo-encorder out put
     received_signal = turbo_encoded+sqrt(noise_var)*randn(1,(block_size)*2);
% demultiplexing the signals
    demul_output = demultiplexer(received_signal, Alpha );
%scaled received signal
    Datar=  demul_output *channel_reliability_value/2;

% Turbo decoder
extrinsic = zeros(1, block_size);
apriori = zeros(1, block_size);

for  iteration = 1: no_of_iterations

% First decoder
    apriori(Alpha) = extrinsic;
    LLR = BCJL1(Datar(1,:), code_polynomial, apriori);
    extrinsic = LLR - 2*Datar(1,1:2:2*(block_size)) - apriori;

% Second decoder
   apriori = extrinsic(Alpha);
   LLR = BCJL2(Datar(2,:), code_polynomial, apriori);
   extrinsic = LLR - 2*Datar(2,1:2:2*(block_size)) - apriori;

% Hard decision of information bits
  Datahat(Alpha) = (sign(LLR)+1)/2;

% Number of bit errors
         bit_errors(iteration) = length(find(Datahat(1:block_size-m)~=Data));

% Number of block errors
         if bit_errors(iteration )>0
            block_errors(iteration) = block_errors(iteration) +1;
         end
end

%Total bit errors
total_errors=total_errors+ bit_errors;

% bit error rate
```

```
            if block_errors(no_of_iterations)==block_error_limit
          BER(1:no_of_iterations)= total_errors(1:no_of_iterations)/...
          block_number/(block_size-m);
            end
end
```

```
function output = turbo_encorder( Data, code_g, Alpha)
% Turbo code encorder
% Used in Problem 10.36, CS: Haykin
%M. Sellathurai

[n,K] = size(code_g);
m = K - 1;
block_s = length(Data);

state = zeros(m,1);
y=zeros(3,block_s+m);

% encorder 1
for i = 1: block_s+m
   if  i <= block_s
      d_k = Data(1,i);
   elseif  i >  block_s
      d_k = rem( code_g(1,2:K)*state, 2 );
   end
a_k = rem( code_g(1,:)*[d_k ;state], 2 );
v_k = code_g(2,1)*a_k;

   for j = 2:K
      v_k = xor(v_k, code_g(2,j)*state(j-1));
   end;
state = [a_k;state(1:m-1)];
y(1,i)=d_k;
y(2,i)=v_k;
end

%encorder 2
% interleaving the data
for i = 1: block_s+m
   ytilde(1,i) = y(1,Alpha(i));
end

state = zeros(m,1);
% encorder 2

for i = 1: block_s+m
  d_k = ytilde(1,i);
a_k = rem( code_g(1,:)*[d_k ;state], 2 );
v_k = code_g(2,1)*a_k;
 for j = 2:K
      v_k = xor(v_k, code_g(2,j)*state(j-1));
   end;
```

```
state = [a_k; state(1:m-1)];
y(3,i)=v_k;
end
% inserting odd and even parities
 for i=1: block_s+m
       output(1,n*i-1) = 2*y(1,i)-1;
       if rem(i,2)
          output(1,n*i) = 2*y(2,i)-1;
       else
          output(1,n*i) = 2*y(3,i)-1;
       end
end
```

```
function [nxt_o, nxt_s, lst_o, lst_s] = cnc_trellis(code_g);
%used in Problem10.36.
% code trellis for  RSC;
% Mathini Sellathurai



% code properties
[n,K] = size(code_g);
m = K - 1;
no_of_states = 2^m;

for s=1: no_of_states
dec_cnt_s=s-1; i=1;

% decimal to binary state
while  dec_cnt_s >=0 & i<=m
       bin_cnt_s(i) = rem( dec_cnt_s,2) ;
      dec_cnt_s  = (dec_cnt_s-  bin_cnt_s(i))/2;
        i=i+1;
end
bin_cnt_s=bin_cnt_s(m:-1:1);

% next state when input is 0
   d_k = 0;
   a_k = rem( code_g(1,:)*[0 bin_cnt_s ]', 2 );
   v_k = code_g(2,1)*a_k;
   for j = 1:K-1
      v_k = xor(v_k, code_g(2,j+1)*bin_cnt_s(j));
   end;
nstate0 = [a_k  bin_cnt_s(1:m-1)];
y_0 = [0 v_k];


% next state when input is 1
   d_k = 1;
   a_k = rem( code_g(1,:)*[1 bin_cnt_s]', 2 );
   v_k = code_g(2,1)*a_k;
   for j = 1:K-1
      v_k = xor(v_k, code_g(2,j+1)*bin_cnt_s(j));
   end;
nstate1 = [a_k bin_cnt_s(1:m-1)];
y_1=[1 v_k];
% next output when input 0 1
nxt_o(s,:) = [y_0 y_1];
```

```
% binary to decimal state
d=2.^(m-1:-1:0);
dstate0=nstate0*d'+1; dstate1=nstate1*d'+1;
% next state when input 0 1
nxt_s(s,:) = [ dstate0  dstate1 ];

% finding the possible previous state frm the trellis
lst_s(nxt_s(s, 1), 1)=s;
lst_s(nxt_s(s, 2), 2)=s;
lst_o(nxt_s(s, 1), 1:4) =  nxt_o(s, 1:4) ;
lst_o(nxt_s(s, 2), 1:4) =  nxt_o(s, 1:4) ;

end
```

```
function output = demultiplex(Data, Alpha);
% demultiplexing the received signal
% used in problem 10.36, CS: Haykin
% Mathini Sellathurai

block_s = fix(length(Data)/2);
output=zeros(2,block_s);

for i = 1: block_s
       Dataf(i) = Data(2*i-1);
         if rem(i,2)>0
          output(1,2*i) = Data(2*i);
       else
          output(2,2*i) = Data(2*i);
       end
end

for i = 1:block_s
   output(1,2*i-1) = Dataf(i);
   output(2,2*i-1) = Dataf(Alpha(i));
end
```

```
function L = BCJL1(Datar, code_g ,apriori)
% log-BCJL (LOG-MAP algorithm) for decoder 1
% Used in Problem 10.36, CS: Haykin

% states, memory, constraint length and block size
block_s = fix(length(Datar)/2);
[n,K] = size(code_g);
m = K - 1;
no_of_states = 2^m;
infty = 1e10;
zero=1e-300;


% forward recursion
alpha(1,1) = 0;
alpha(1,2:no_of_states) = -infty*ones(1,no_of_states-1);

% code-trellis
[nxt_o, nxt_s, lst_o, lst_s] = cnc_trellis(code_g);
nxt_o = 2*nxt_o-1;
lst_o = 2*lst_o-1;

for i = 1:block_s
    for cnt_s = 1:no_of_states
      branch = -infty*ones(1,no_of_states);
      branch(lst_s(cnt_s,1)) = -Datar(2*i-1)+Datar(2*i)*...
      lst_o(cnt_s,2)-log(1+exp(apriori(i)));
      branch(lst_s(cnt_s,2)) = Datar(2*i-1)+Datar(2*i)*...
      lst_o(cnt_s,4)+apriori(i)-log(1+exp(apriori(i)));
    if(sum(exp(branch+alpha(i,:)))>zero)
        alpha(i+1,cnt_s) = log( sum( exp( branch+alpha(i,:))));
     else
    alpha(i+1,cnt_s) =-1*infty;
    end
 end
    alpha_max(i+1) = max(alpha(i+1,:));
    alpha(i+1,:) = alpha(i+1,:) - alpha_max(i+1);
end

% backward recursion
beta(block_s,1)=0;
beta(block_s,2:no_of_states) = -infty*ones(1,no_of_states-1);


for i = block_s-1:-1:1
  for cnt_s = 1:no_of_states
```

```
      branch = -infty*ones(1,no_of_states);
      branch(nxt_s(cnt_s,1)) = -Datar(2*i+1)+Datar(2*i+2)*...
      nxt_o(cnt_s,2)-log(1+exp(apriori(i+1)));
      branch(nxt_s(cnt_s,2)) = Datar(2*i+1)+Datar(2*i+2)*...
      nxt_o(cnt_s,4)+apriori(i+1)-log(1+exp(apriori(i+1)));
      if(sum(exp(branch+beta(i+1,:)))<zero)
         beta(i,cnt_s)=-infty;
      else
         beta(i,cnt_s) = log(sum(exp(branch+beta(i+1,:))));
     end
  end
beta(i,:) = beta(i,:) - alpha_max(i+1);
end

for k = 1:block_s
  for cnt_s = 1:no_of_states
     branch0 = -Datar(2*k-1)+Datar(2*k)*lst_o(cnt_s,2)-log(1+exp(apriori(k)));
     branch1 = Datar(2*k-1)+Datar(2*k)*lst_o(cnt_s,4)+apriori(k)-log(1+exp(apriori(k)));
     den(cnt_s) = exp( alpha(k,lst_s(cnt_s,1))+branch0+  beta(k,cnt_s));
     num(cnt_s) = exp( alpha(k,lst_s(cnt_s,2))+branch1+ beta(k,cnt_s));
  end
L(k) = log(sum(num)) - log(sum(den));
end
```

```
function L = BCJL1(Datar, code_g ,apriori)
% log-BCJL (LOG-MAP algorithm) for decoder 1
% Used in Problem 10.36, CS: Haykin

% states, memory, constraint length and block size
block_s = fix(length(Datar)/2);
[n,K] = size(code_g);
m = K - 1;
no_of_states = 2^m;
infty = 1e10;
zero=1e-300;


% forward recursion
alpha(1,1) = 0;
alpha(1,2:no_of_states) = -infty*ones(1,no_of_states-1);

% code-trellis
[nxt_o, nxt_s, lst_o, lst_s] = cnc_trellis(code_g);
nxt_o = 2*nxt_o-1;
lst_o = 2*lst_o-1;

for i = 1:block_s
    for cnt_s = 1:no_of_states
      branch = -infty*ones(1,no_of_states);
      branch(lst_s(cnt_s,1)) = -Datar(2*i-1)+Datar(2*i)*...
      lst_o(cnt_s,2)-log(1+exp(apriori(i)));
      branch(lst_s(cnt_s,2)) = Datar(2*i-1)+Datar(2*i)*...
      lst_o(cnt_s,4)+apriori(i)-log(1+exp(apriori(i)));
    if(sum(exp(branch+alpha(i,:)))>zero)
        alpha(i+1,cnt_s) = log( sum( exp( branch+alpha(i,:))));
     else
    alpha(i+1,cnt_s) =-1*infty;
    end
 end
    alpha_max(i+1) = max(alpha(i+1,:));
    alpha(i+1,:) = alpha(i+1,:) - alpha_max(i+1);
end

% backward recursion
beta(block_s,1)=0;
beta(block_s,2:no_of_states) = -infty*ones(1,no_of_states-1);


for i = block_s-1:-1:1
  for cnt_s = 1:no_of_states
```

```
      branch = -infty*ones(1,no_of_states);
      branch(nxt_s(cnt_s,1)) = -Datar(2*i+1)+Datar(2*i+2)*...
      nxt_o(cnt_s,2)-log(1+exp(apriori(i+1)));
      branch(nxt_s(cnt_s,2)) = Datar(2*i+1)+Datar(2*i+2)*...
      nxt_o(cnt_s,4)+apriori(i+1)-log(1+exp(apriori(i+1)));
      if(sum(exp(branch+beta(i+1,:)))<zero)
         beta(i,cnt_s)=-infty;
      else
         beta(i,cnt_s) = log(sum(exp(branch+beta(i+1,:))));
     end
  end
beta(i,:) = beta(i,:) - alpha_max(i+1);
end

for k = 1:block_s
  for cnt_s = 1:no_of_states
     branch0 = -Datar(2*k-1)+Datar(2*k)*lst_o(cnt_s,2)-log(1+exp(apriori(k)));
     branch1 = Datar(2*k-1)+Datar(2*k)*lst_o(cnt_s,4)+apriori(k)-log(1+exp(apriori(k)));
     den(cnt_s) = exp( alpha(k,lst_s(cnt_s,1))+branch0+  beta(k,cnt_s));
     num(cnt_s) = exp( alpha(k,lst_s(cnt_s,2))+branch1+ beta(k,cnt_s));
  end
L(k) = log(sum(num)) - log(sum(den));
end
```

Answer to Problem 10.37

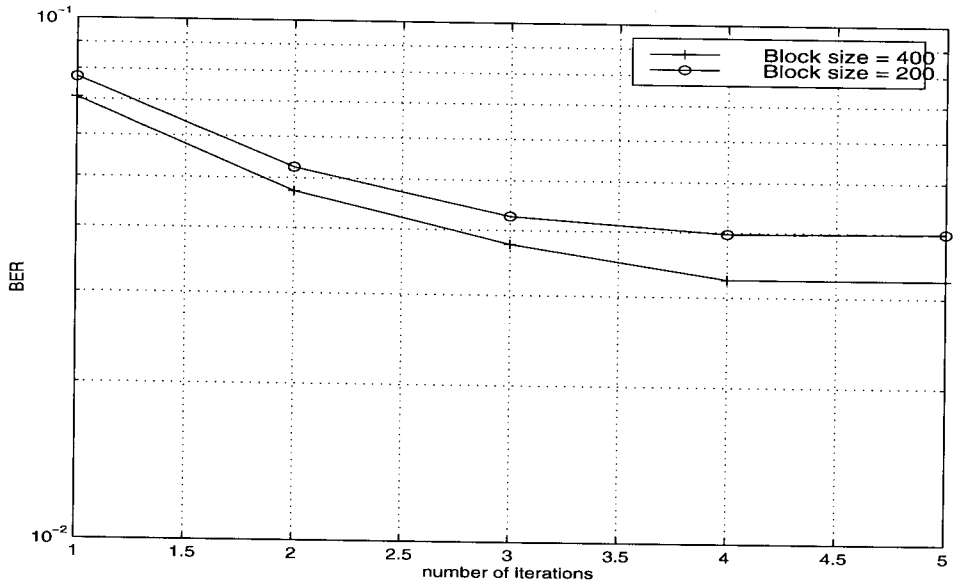


Figure 1: bit error rate Vs. the number of iterations for Block sizes: 200, and 400