Lecture 12

Cyclic Codes

A code is called a cyclic code if the cyclic shift of each codeword is another codeword. That is if

$$c_0, c_1 \cdots c_{n-1}$$

is a codeword $c_{n-1}, c_0, c_1, \ldots, c_{n-2}$ is also a codeword.

The advantage of cyclic codes is that they can be treated algebraically. In specific terms they can be represented as algebraic polynomials and generate by polynomial multiplication.

A codeword $c_0, c_1, \cdots c_{n-1}$ is represented as

$$C(X) = c_0 + c_1 X + \cdots + c_{n-1} X^{n-1}$$

Note that with X substituted by 2, we have the decimal representation of a binary vector.

For cyclic Codes the Codeword polynomial $c(x)$ is equal to $c(x) = a(x)\, g(x)$

where,

$$g(x) = g_0 + g_1 x + \cdots + g_{n-k} x^{n-k}$$

is a polynomial of degree $n-k$ called the <u>generator</u> <u>polynomial</u> that divide $x^n + 1$.

Using the generator polynomial to find the Codewords (<u>enCoding</u>)

Note that

$$\underline{c} = (c_0, c_1, \cdots, c_{n-1}) = (b_0, b_1, \cdots, b_{n-k+1}, m_0, \cdots m_{k-1}$$

or

$$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1} = b_0 + b_1 x + \cdots b_{n-k+1} x^{n-k+1} + m_0 x^{n-k} + \cdots + m_{k-1} x^{n-1}$$

or

$$c(x) = b(x) + x^{n-k} m(x)$$

where $m(x) = m_0 + m_1 x + \cdots + m_{k-1} x^{k-1}$ is the message polynomial and $b(x) = b_0 + b_1 x + \cdots + b_{n-k+1} x^{n-k+1}$ is the parity polynomial.

let $c(x) = a(x)\, g(x)$ then,

$$x^{n-k} m(x) = a(x)\, g(x) + b(x)$$

That is, $b(x)$ is the remainder resulting from dividing $x^{n-k} m(x)$ by $g(x)$.

So, the encoding procedure is as follows:

1) Shift $m(x)$ by $n-k$ bits, i.e., multiply it by x^{n-k}.

2) Divide $x^{n-k} m(x)$ by $g(x)$, the remainder $b(x)$ is the parity polynomial.

3) Add $b(x)$ to $x^{n-k} m(x)$ to form $c(x)$.

Example: The generator polynomial for (7,4) Hamming Code is $g(x) = x^3 + x + 1$.

Find the Codeword for the message $\underline{m} = [0101]$

$m(x) = x^3 + x$

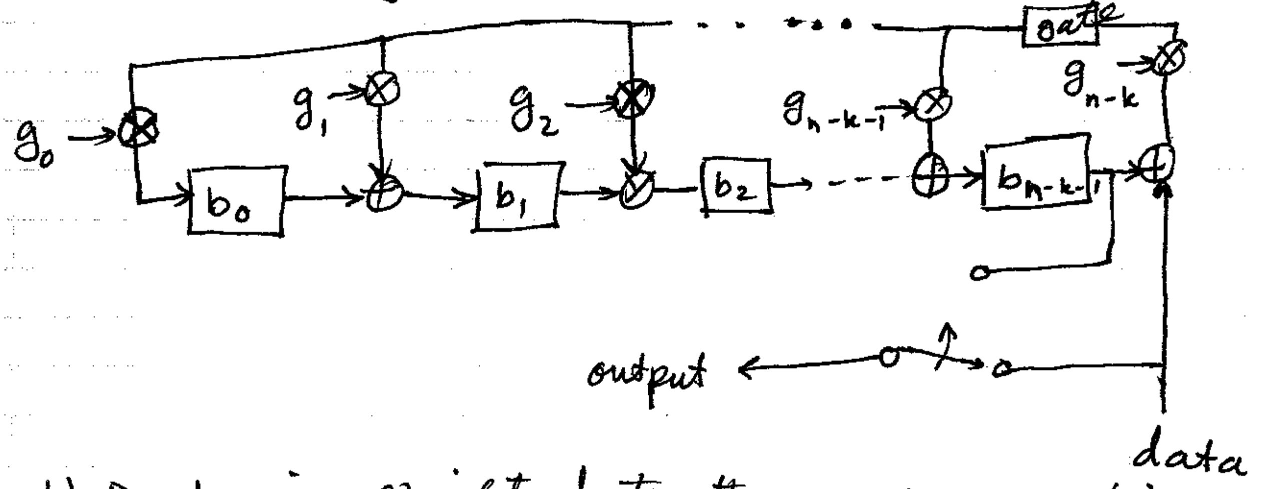
$x^{n-k}\, m(x) = x^{7-4}(x^3 + x) = x^3(x^3 + x) = x^6 + x^4$

$$
\begin{array}{l|l}
\cancel{x^6}+\cancel{x^4} & x^3+x+1 \\
\hline
\cancel{x^6}+\cancel{x^4}+x^3 & x^3+1 \\
\hline
\cancel{x^3} & \\
\cancel{x^3}+x+1 & \\
\hline
x+1 &
\end{array}
$$

So, $c(x) = b^{n-k} m(x) + b(x) = x^6 + x^4 + x + 1$

$\Rightarrow \quad c = [1\ 1\ 0\ 0\ 1\ 0\ 1]$

Circuit Diagram of the encoder:

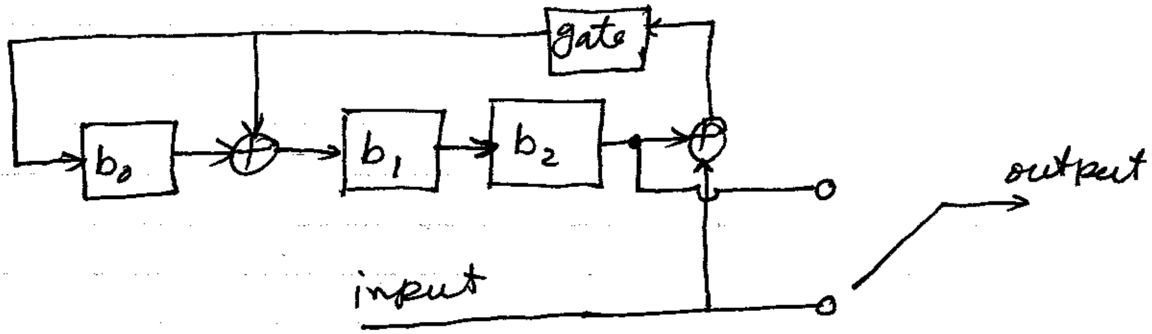


1) Data is shifted to the register.

- It is both shifted n-k places and divided by g(x), after entering all k data bits the parities $b_0, b_1, \dots b_{n-k-1}$ are formed. Switch is in lower position so message bits are transmitted.

Now, the gate is opened so that the parities do not change and switch is flipped up so that the parities are transmitted.

For the (7,4) Hamming code the encoder is:



Other cyclic Codes:

Hamming codes in general:

Block Length $n = 2^m - 1$

Number of message bits: $k = 2^m - m - 1$

Number of parity bits: $n - k = m$

We saw the example of (7,4) Code correspondi to $m=3 \Rightarrow n = 2^3 - 1 = 7$ and $k = 2^3 - 3 - 1 = 4$.

For $m = 4 \Rightarrow n = 15$ and $k = 11 \Rightarrow (15,11)$ Code.

Hamming Codes all have $d_{min} = 3$ so correct one bit of error.

BCH Codes have length $n=2^m-1$, but the information bits and parity bits, k and $n-k$, respectively can be chosen so that the error correcting capability be varied.

Reed-Solomon (RS) Codes:

These codes are non-binary. They consist of $n = 2^m - 1$ symbol and each symbol is m bits long. The number of information symbols k can be varied. The maximum number of symbols that an (n,k) RS code can correct is $\lfloor \frac{n-k}{2} \rfloor$, i.e.,

$$t \leqslant \lfloor \frac{n-k}{2} \rfloor$$

As an example take $m = 8$. This means that each symbol is 8 bits (a byte). The total length of a codeword is $n=2^m-1=255$

So the code is $(255, k)$ where k can be selected anywhere from 1 to 255 to make the Code stronger or weaker. Say, if we let $k=239$, we have a (255, 239) Code. This Code takes 239 bytes (8 bit Symbols), i.e., $239 \times 8 = 1912$ and adds $255-239=16$ parity bits and create a 255 bytes (2040 bits) Codeword. This Code can correct $\frac{255-239}{2} = 8$ bytes of error. The good thing about RS Code is that it corrects a symbol (several bits) So, it is suitable for burst errors (errors occuring Consecutively). For example, the above Code (255, 239) Can correct any Consecutive 7 bytes, i.e., any burst of upto $7 \times 8 = 56$ bits

Performance of Coded Systems

When we use error Control Coding schemes we hope that we get lower bit error rate than uncoded System. The difference between a Coded System and an uncoded version at a given $\frac{E_b}{N_0}$ is called the Coding gain.

To clarify the idea consider a simple example using BPSK and (7,4) Hamming Code.

Example: Find the performance of a System using BPSK modulation and (7,4) Code at $\frac{E_b}{N_0} = 5\,dB$. Compare with uncoded case.

Solution: $\frac{E_b}{N_0} = 5\,dB \Rightarrow \frac{E_b}{N_0} = 10^{0.5} = 3.16$

$$P_{unc}(e) = Q\left(\sqrt{2\frac{E_b}{N_0}}\right) = Q\left(\sqrt{2\times 3.16}\right) \simeq 6\times 10^{-3}$$

For coded Case for each 4 bits we send 7 bits so energy is reduced by $\frac{4}{7}$. So

$$\frac{E_c}{N_0} = \frac{4}{7}\times 3.16 = 1.807$$

So, the error probability before decoding (before correcting errors) is:

$$p = Q\left(\sqrt{2\frac{E_c}{N_0}}\right) = Q(1.9) = 0.0287$$

The code can correct 1 error in a codeword (in 7 bits). So, the probability of error after decoding is the probability that more than one error exists in 7 bits.

$$P_{cw}(e) = \sum_{i=2}^{7} \binom{7}{i} p^i (1-p)^{7-i}$$

$$= 1 - \left(\sum_{i=0}^{1} \binom{7}{i} p^i (1-p)^{7-i}\right)$$

$$= 1 - (1-p)^7 - 7p(1-p)^6 = 0.0157$$

This is the probability that a codeword is decoded erroneously. Assume that, on the average, the probability that a bit is in error is half this, i.e., one the average half the bits are in error the

$$P_b = 0.00785$$

which is almost the same as uncoded case.

Even when $\frac{E_b}{N_0} = 10\ dB \rightarrow \frac{E_b}{N_0} = 10$

$$P_{unc} = Q(\sqrt{20}) = Q(4.47) = 4\times10^{-6}$$

and with coding $\frac{E_c}{N_0} = 10\times\frac{4}{7} = 5.7$

and $P = Q(\sqrt{2\times5.7}) = 3.4\times10^{-4}$

and

$$P_B = \frac{1}{2}\left[1-p^{7}-(1-p)^{6}p\right] = 1.2\times10^{-6}$$

which is only about one third of uncoded case. The following figure shows the performanc of Hamming (7,4) Code and some other codes.

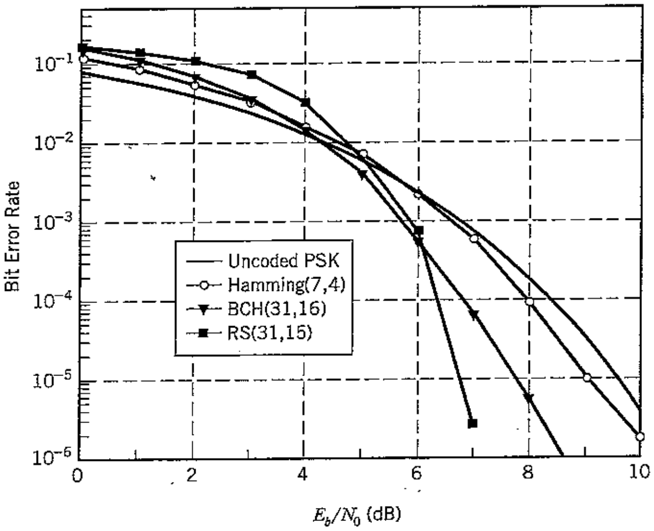


FIGURE 10.25 Simulated BER performance of uncoded and FEC-encoded PSK transmitted over a Gaussian channel.

Now Consider using Hamming (15,11) Code. This Code Corrects 1 bit of error in 15 bits.

This may be a weaker code, i.e., correcting one in 15 instead of 1 in 7 errors, but it has less parity and the E_b is only reduced by $\frac{11}{15}$ instead of $\frac{4}{7}$. So,

$$\frac{E_c}{N_0} = 10 \times \frac{11}{15} = 7.33$$

$$P = Q(\sqrt{2 \times 7.33}) = Q(3.83) = 7 \times 10^{-5}$$

and the bit error probability after decoding is:

$$P_B = \frac{1}{2}\left[1 - P^{15} - P(1-P)^{14}\right] = 2.57 \times 10^{-7}$$

which is more than one order of magnitude better than the uncoded bit error rate (BER) of 4×10^{-6}.

Shortened block Codes:

Sometimes, in a system, we may want to have smaller code length in order to reduce latency or to adapt to data at the input of the encoder. In such case an (n,k) code can be shortened by putting L zero's at the end of $k-L$ information bits (or symbols). After adding $n-k$ parities, we have an (n,k) code whose last L bits (or symbols) are zero. Since the transmitter and receiver know that these bits are zero, it is not needed to transmit them. So, we have a code with length $n-L$ with $k-L$ information symbols.

For example in Digital Video Broadcasting (DVB) Standard, each packet is 188 bytes (MPEG packets). Only 16 bytes of parity is needed in most situations. So, a $(255,239)$ Reed-Solomon Code is selected. However, since the information length is 188 and not 239, we null $239-188=51$ bytes and we get $(204,188)$ shortened code.

Convolutional Codes

In the case of block Codes, a usually long block of data is fed to the encoder and a block of encoded data consisting of the message (the input) and the parities is output.

In the case of Convolutional Codes a few (most often one) bit is fed to the encoder and n bits are generated based on the input bit(s) and a few previous bits.

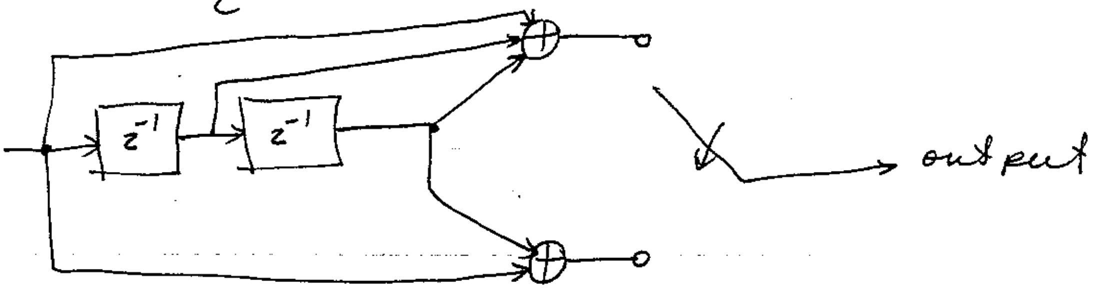
Assume that the system has k bits input an n bits output and M memory cells to store the previous inputs.

Assume that a block of L symbols, i.e., kL bits is fed to the encoder. for each k-bit input, we have an n-bit output i.e., nL output bits, we need also flush the memory by feeding M null (zero) symbols. So the rate of the code will be

$$\frac{kL}{n(L+M)} \cong \frac{k}{n}$$

Example

A rate $\frac{1}{2}$ code

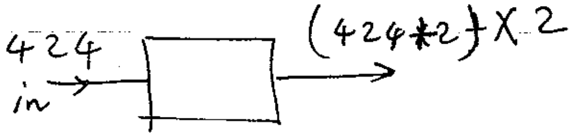


if the initial state of shift registers is 0 0 then the input sequence 1001 produces:

$$11, 10, 11, 11, \underbrace{10, 11}_{\text{Tail}},$$

actual rate here is $= \frac{4}{12} = \frac{1}{3}$ not $\frac{1}{2}$

but usually more bits enter encoder ~~a~~ at a given time interval. Say, if you have ATM cell (53 bytes = 424 bits) then



$$\text{rate} = \frac{424}{(424+2)\times 2} \approx \frac{1}{2} \quad (0.4976)$$

12-15

generator polynomial

$\underline{g}_\phi = (1\ 1\ 1)$ / we can also show this as $g_\phi(D) = 1 + D + D^2$

$\underline{g}_2 = (1\ 0\ 1)$ / $g_2(D) = 1 + D^2$

two vectors (in general one vector for each input-output pair)

a 1 in location l of mth vector represents connection between the output of the shift register l to m th output.

Constraint length K = the number of shift register bits (2+1 latch)

number of state $S = 2^{K-1}$, e.g., for $K = 3$, $S = 4$

Using generator polynomial we can generate output by finding

$$c_1(D) = m(D)\, g_1(D)$$

$$c_2(D) = m(D)\, g_2(D)$$

let input be $\underline{m} = (1\ 1\ 0\ 0\ 1) \Rightarrow m(D) = D^4 + D^3 + 1$

then $c_1(D) = (D^4 + D^3 + 1)(D^2 + D + 1) = D^6 + D^3 + D^2 + D + 1$

or $\underline{c}_1 = (1\ D\ D\ 1\ 1\ 1\ 1)$

and $c_2(D) = (D^4 + D^3 + 1)(D^2 + 1) = D^6 + D^5 + D^4 + D^3 + D^2 + 1$

or $\underline{c}_2 = (1\ 1\ 1\ 1\ 1\ 0\ 1)$

So the output will be (after multiplexing):

11, 10, 10, 11, 11, 01, 11

A typical example:

A de factor industrial standard code is a rate $\frac{1}{2}$ or $\frac{1}{3}$ code with K = 7 and

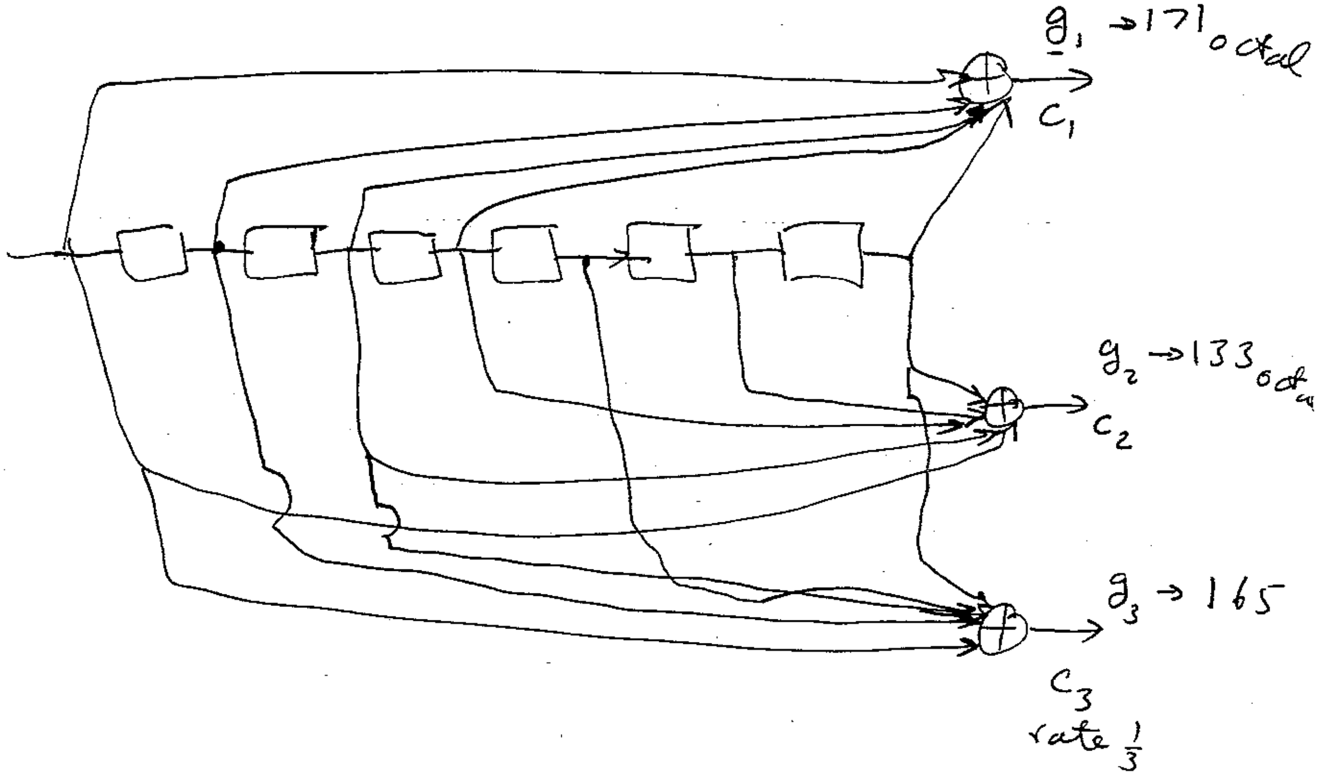
$\underline{g}_1 = (171)_{octal}$, $\underline{g}_2 = (133)$, $\underline{g}_3 = 165$

Check: http://people.qualcomm.com/karn/papers/cnc_coding.htm

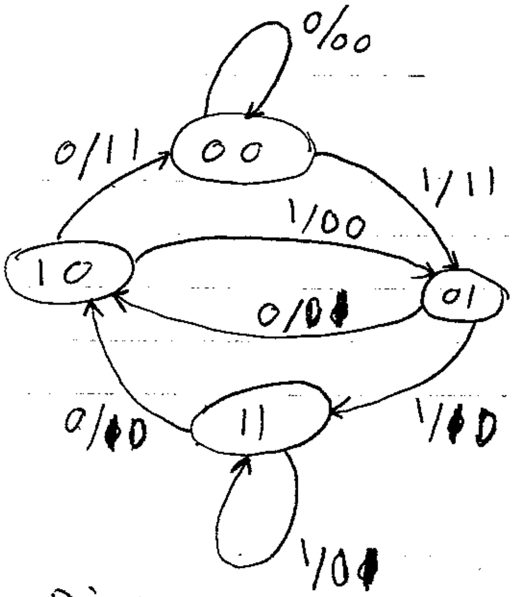
http://www.istari-design.com/vitnote.htm

(speeding up Software ~~implemen~~ Simulation).

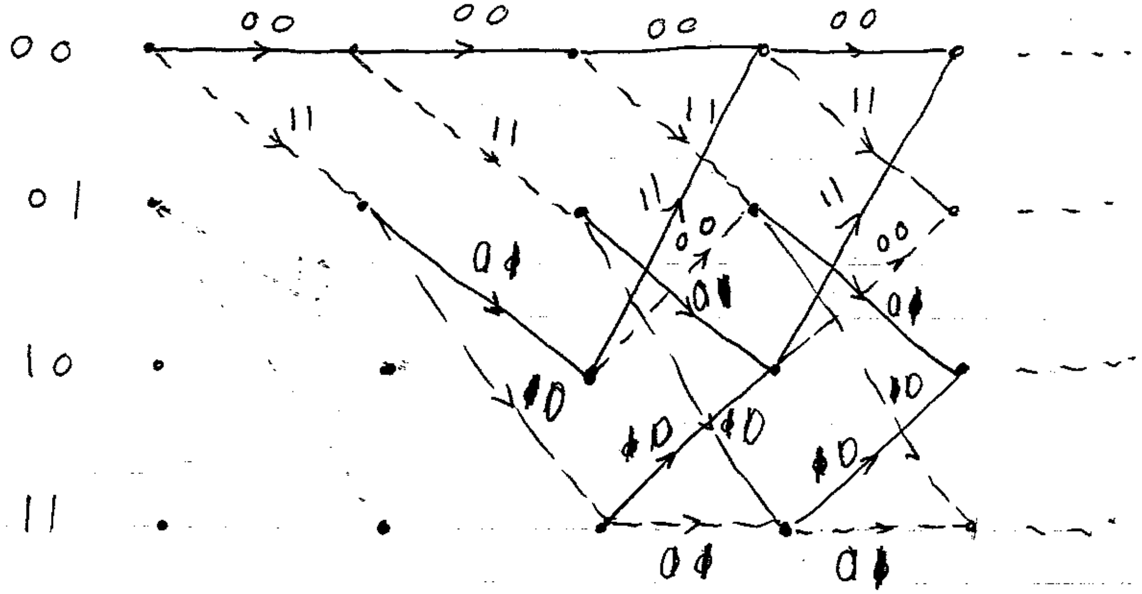
Q1900 / http://www.qualcomm.com/ProdTech/asic/vtdec.h



Finite state representation



Trellis Diagram



Viterbi Decoding Algorithm (VA)

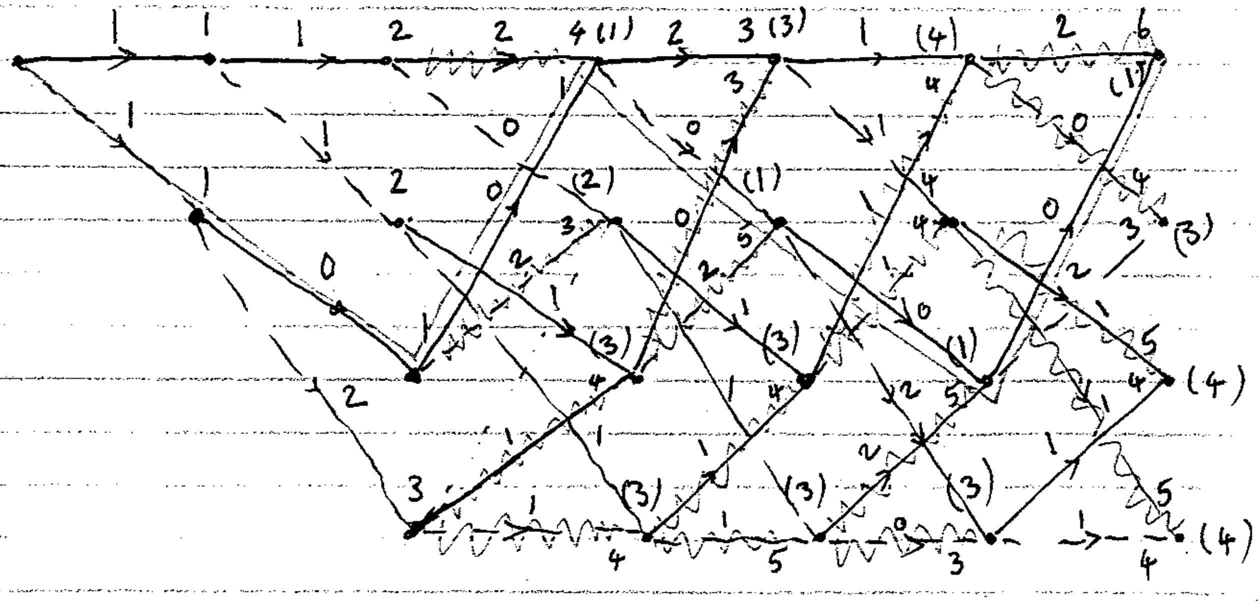
VA tries to find the sequence closest to the received Symbols. If it works on original samples at the output of the channel

~~(the sat.~~
(the sampled output of the matched filter) it is called soft decision Viterbi decoder.

If VA works on the bits at the output of the demodulator, we have hard decision (≈ 3 dB penalty in comparison with soft decision decoding

Example: Assume we had the message 1001 and we sent $11, 01, 11, 11, \underbrace{01, 11}_{\text{Tail}}$ and received

10, 01, 11, 11, 01, 11

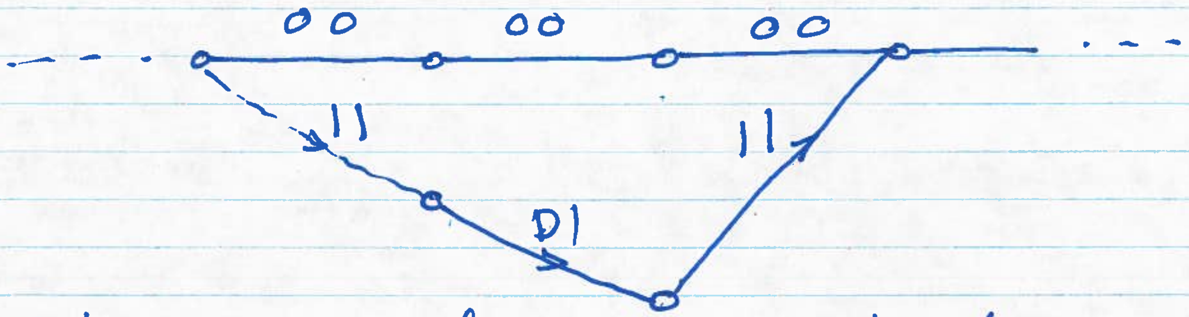


$\underbrace{1 \quad 0 \quad 0 \quad 1}$ 0 0

Performance of Convolutional Codes

The performance of convolutional codes is determined by the structure of the trellis, i.e., how close (in Hamming distance sense) are the pathes diverging from one node and converging at a later time.

Diverging and converging paths create an error event. For example, in the example given before an error path (or error event) may look like this.



That is, we may have transmitted 0,0,0 starting from state 00 which corresponds to 00, 00, 00, but have detected 11, 01, 11 which represent 1 0 0. Note that the Hamming distance between the two paths is 5 and the number of errors made if this switch occurs is 1 bit (only error in the first bit).

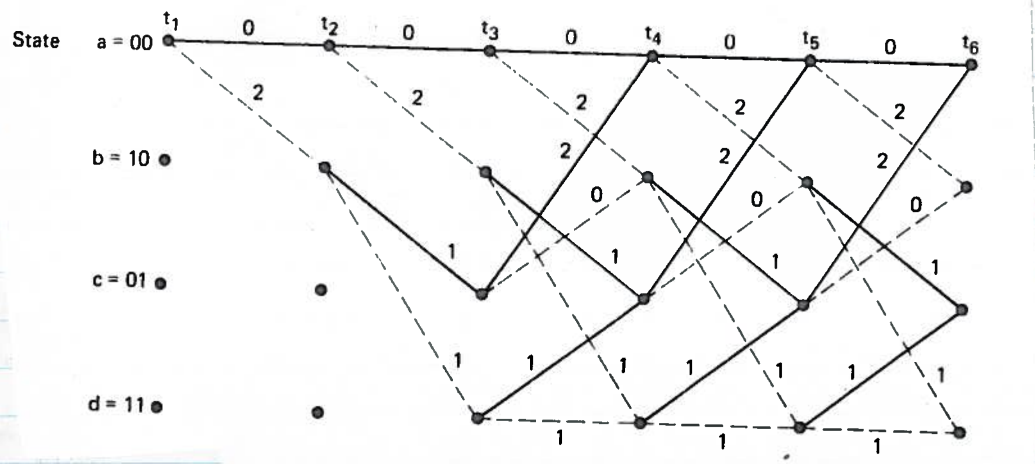
The distance between different paths or equivalently the weighth of the error events and the number of error events is what determines the performance of ~~err~~ Convolutional Codes.

In particular the shortest weighth path is of importance and has the dominating role on the error correcting capability of the Convolutional Codes, This is called d_{free} and plays similar role as d_{min} in block Codes.

One way to find the weighth distribution of Convolutional Codes and, in particular, d_{free}, is to derive the so called Transfer function of the Code.

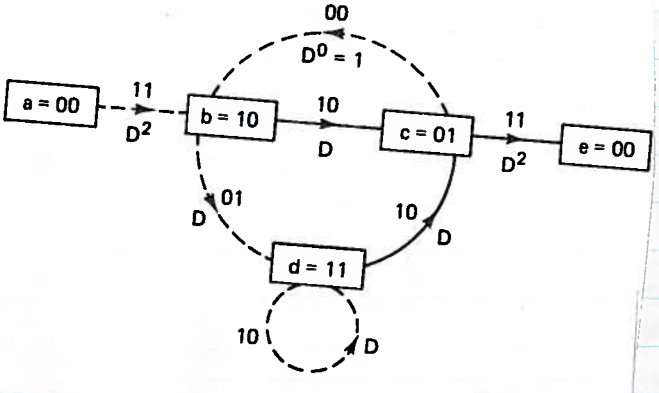
To do this, the trellis is redrawn as a circuit with nodes representing the states of the trellis and each labelled by X^i where i is the weighth of the label of the edge connecting the two nodes on the trellis.

For example, for the following Trellis,



Trellis diagram, labeled with distances from the all-zeros path.

The circuit representing the state diagram is



State diagram, labeled according to distance from the all-zeros path.

Assigning Labels a, b, c, d and e to the nodes (states), we have the following relations:

$$X_b = D^2X_a + X_c$$
$$X_c = DX_b + DX_d$$
$$X_d = DX_b + DX_d$$
$$X_e = D^2X_c$$

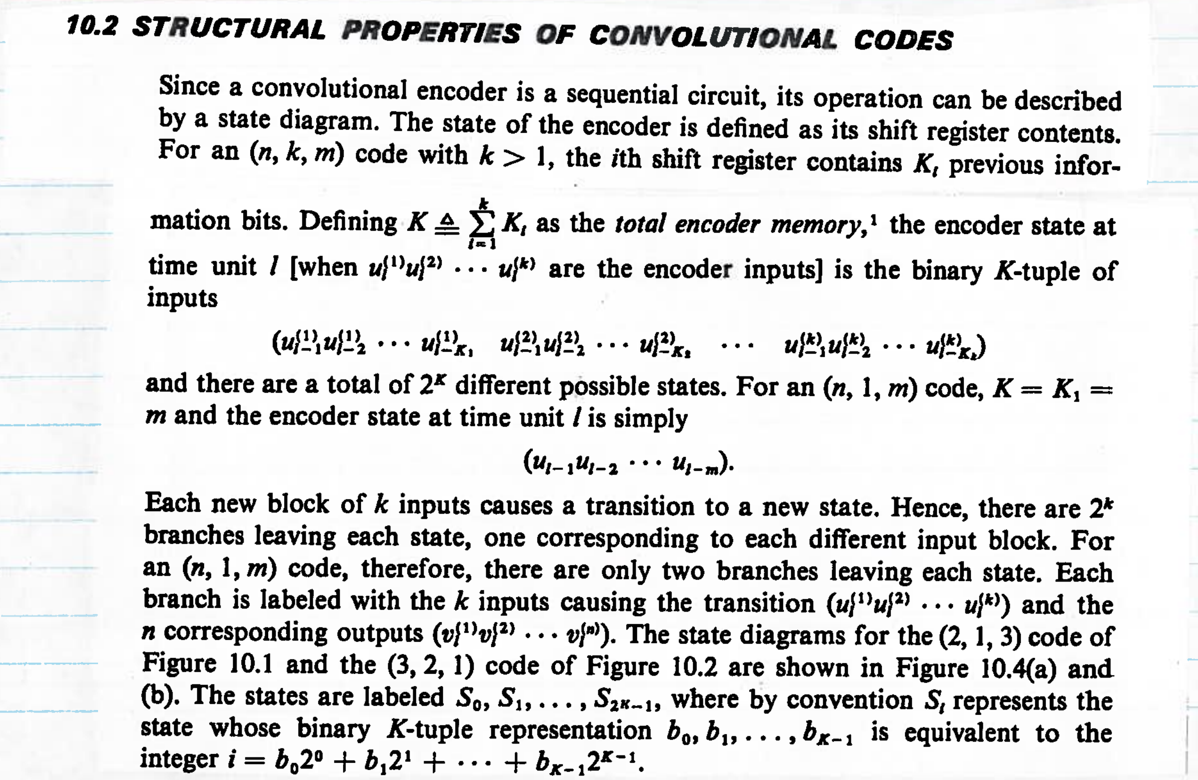
Solving these, we get

$$T(D) = \frac{X_e}{X_a} = \frac{D^5}{1-2D} = D^5 + 2D^6 + 4D^7 + \cdots$$

So, it has one error event of weighth 5, 2 with eveighth 6, ...

The $d_{free} = 5$

It is not possible to do the above derivation for more complex Codes. For Codes with a larger number of States, methods from circuit theory is used. Following is a few pages from the text by Shu Lin and Costello Showing the method.



10.2 STRUCTURAL PROPERTIES OF CONVOLUTIONAL CODES

Since a convolutional encoder is a sequential circuit, its operation can be described by a state diagram. The state of the encoder is defined as its shift register contents. For an (n, k, m) code with $k > 1$, the ith shift register contains K_i previous information bits. Defining $K \triangleq \sum_{i=1}^{k} K_i$ as the *total encoder memory*,[1] the encoder state at time unit l [when $u_l^{(1)}u_l^{(2)} \cdots u_l^{(k)}$ are the encoder inputs] is the binary K-tuple of inputs

$$(u_{l-1}^{(1)}u_{l-2}^{(1)} \cdots u_{l-K_1}^{(1)} \quad u_{l-1}^{(2)}u_{l-2}^{(2)} \cdots u_{l-K_2}^{(2)} \quad \cdots \quad u_{l-1}^{(k)}u_{l-2}^{(k)} \cdots u_{l-K_k}^{(k)})$$

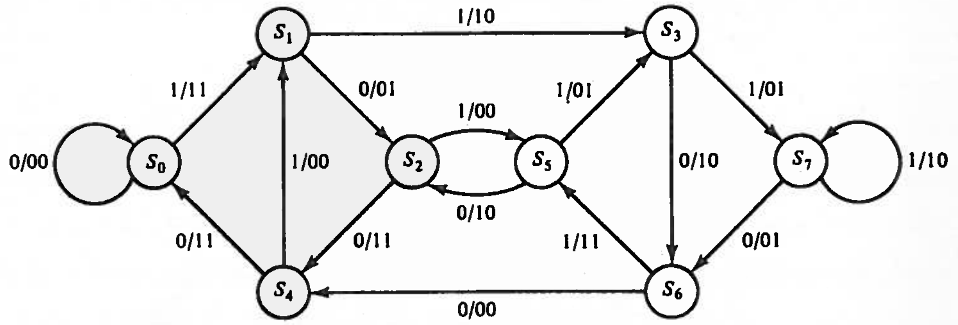
and there are a total of 2^K different possible states. For an $(n, 1, m)$ code, $K = K_1 = m$ and the encoder state at time unit l is simply

$$(u_{l-1}u_{l-2} \cdots u_{l-m}).$$

Each new block of k inputs causes a transition to a new state. Hence, there are 2^k branches leaving each state, one corresponding to each different input block. For an $(n, 1, m)$ code, therefore, there are only two branches leaving each state. Each branch is labeled with the k inputs causing the transition $(u_l^{(1)}u_l^{(2)} \cdots u_l^{(k)})$ and the n corresponding outputs $(v_l^{(1)}v_l^{(2)} \cdots v_l^{(n)})$. The state diagrams for the (2, 1, 3) code of Figure 10.1 and the (3, 2, 1) code of Figure 10.2 are shown in Figure 10.4(a) and (b). The states are labeled $S_0, S_1, \ldots, S_{2^K-1}$, where by convention S_i represents the state whose binary K-tuple representation $b_0, b_1, \ldots, b_{K-1}$ is equivalent to the integer $i = b_0 2^0 + b_1 2^1 + \cdots + b_{K-1} 2^{K-1}$.

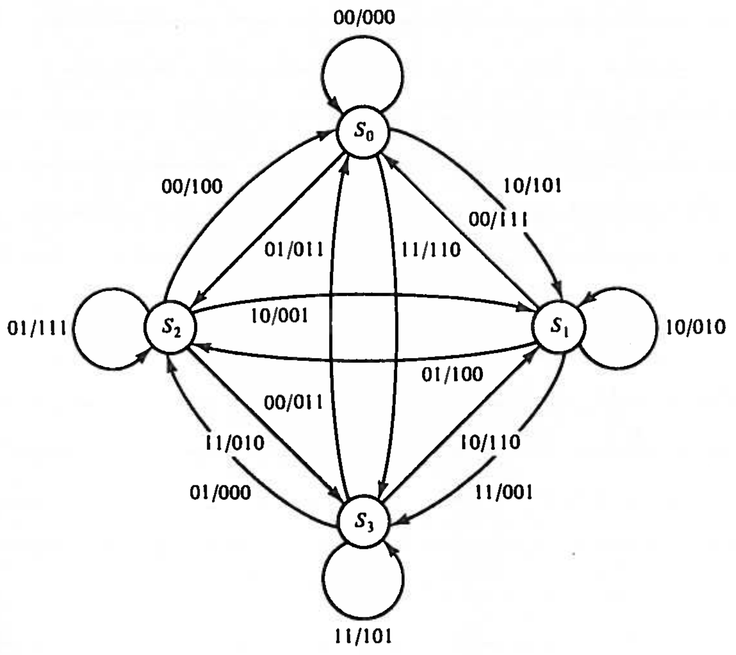
Assuming that the encoder is initially in state S_0 (the all-zero state), the code word corresponding to any given information sequence can be obtained by following the path through the state diagram determined by the information sequence and noting the corresponding outputs on the branch labels. Following the last nonzero information block, the encoder is returned to state S_0 by a sequence of m all-zero blocks appended to the information sequence. For example, in Figure 10.4(a), if $\mathbf{u} = (1\ 1\ 1\ 0\ 1)$, the code word $\mathbf{v} = (1\ 1,\ 1\ 0,\ 0\ 1,\ 0\ 1,\ 1\ 1,\ 1\ 0,\ 1\ 1,\ 1\ 1)$.

The state diagram can be modified to provide a complete description of the Hamming weights of all nonzero code words (i.e., a weight distribution function for the code). State S_0 is split into an initial state and a final state, the self-loop around state S_0 is deleted, and each branch is labeled with a *branch gain* X^i, where



(a)

Figure 10.4 Encoder state diagrams: (a) (2, 1, 3) code.



(b)

Figure 10.4 (*cont.*) (b) (3, 2, 1) code.

[1]Note the distinction between the total encoder memory K and the encoder memory order K is the sum of all the shift register lengths, whereas m is the maximum length of any shift register.

i is the weight of the n encoded bits on that branch. Each path connecting the initial state to the final state represents a nonzero code word that diverges from and remerges with state S_0 exactly once. Code words that diverge from and remerge with state S_0 more than once can be considered as a sequence of shorter code words. The *path gain* is the product of the branch gains along a path, and the weight of the associated code word is the power of X in the path gain. The modified state diagrams for the codes of Figures 10.1 and 10.2 are shown in Figure 10.5(a) and (b).

Example 10.9

(a) The path representing the state sequence $S_0S_1S_3S_7S_6S_5S_2S_4S_0$ in Figure 10.5(a) has path gain $X^2 \cdot X^1 \cdot X^1 \cdot X^1 \cdot X^2 \cdot X^1 \cdot X^2 \cdot X^2 = X^{12}$, and the corresponding code word has weight 12.

(b) The path representing the state sequence $S_0S_1S_3S_2S_0$ in Figure 10.5(b) has path gain $X^2 \cdot X^1 \cdot X^0 \cdot X^1 = X^4$, and the corresponding code word has weight 4.

The weight distribution function of a code can be determined by considering the modified state diagram as a signal flow graph and applying Mason's gain formula [5] to compute its "generating function"

$$T(X) = \sum_i A_i X^i, \tag{10.22}$$

where A_i is the number of code words of weight i. In a signal flow graph, a path connecting the initial state to the final state which does not go through any state twice is called a *forward path*. Let F_i be the gain of the ith forward path. A closed

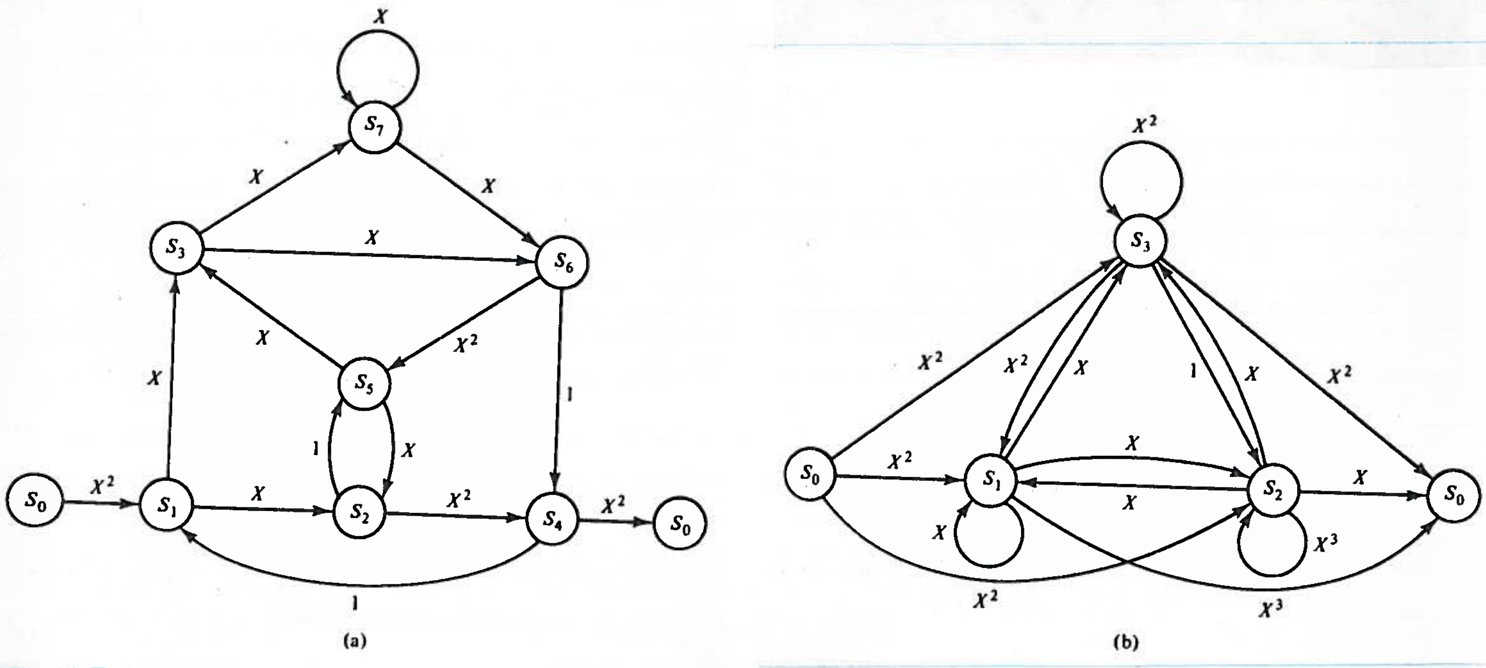


Figure 10.5 Modified encoder state diagrams: (a) (2, 1, 3) code; (b) (3, 2, 1) code.

12-23

path starting at any state and returning to that state without going through any other state twice is called a *loop*. Let C_i be the gain of the ith loop. A set of loops is *nontouching* if no state belongs to more than one loop in the set. Let $\{i\}$ be the set of all loops, $\{i', j'\}$ be the set of all pairs of nontouching loops, $\{i'', j'', l''\}$ be the set of all triples of nontouching loops, and so on. Then define

$$\Delta = 1 - \sum_i C_i + \sum_{i',j'} C_{i'}C_{j'} - \sum_{i'',j'',l''} C_{i''}C_{j''}C_{l''} + \cdots, \qquad (10.23)$$

where $\sum_i C_i$ is the sum of the loop gains, $\sum_{i',j'} C_{i'}C_{j'}$ is the product of the loop gains of two nontouching loops summed over all pairs of nontouching loops, $\sum_{i'',j'',l''} C_{i''}C_{j''}C_{l''}$ is the product of the loop gains of three nontouching loops summed over all triples of nontouching loops, and so on. Finally, Δ_i is defined exactly like Δ, but only for that portion of the graph not touching the ith forward path; that is, all states along the ith forward path, together with all branches connected to these states, are removed from the graph when computing Δ_i. Mason's formula for computing the generating function $T(X)$ of a graph can now be stated as

$$T(X) = \frac{\sum_i F_i \Delta_i}{\Delta}, \qquad (10.24)$$

where the sum in the numerator is over all forward paths. Two illustrative examples will clarify the procedure.

Example 10.10

(a) There are 11 loops in the graph of Figure 10.5(a):

Loop 1:	$S_1S_3S_7S_6S_5S_2S_4S_1$	$(C_1 = X^8)$
Loop 2:	$S_1S_3S_7S_6S_4S_1$	$(C_2 = X^3)$
Loop 3:	$S_1S_3S_6S_5S_2S_4S_1$	$(C_3 = X^7)$
Loop 4:	$S_1S_3S_6S_4S_1$	$(C_4 = X^2)$
Loop 5:	$S_1S_2S_5S_3S_7S_6S_4S_1$	$(C_5 = X^9)$
Loop 6:	$S_1S_2S_5S_3S_6S_4S_1$	$(C_6 = X^8)$
Loop 7:	$S_1S_2S_4S_1$	$(C_7 = X^3)$
Loop 8:	$S_2S_5S_2$	$(C_8 = X)$
Loop 9:	$S_3S_7S_6S_5S_3$	$(C_9 = X^5)$
Loop 10:	$S_3S_6S_5S_3$	$(C_{10} = X^4)$
Loop 11:	S_7S_7	$(C_{11} = X)$.

There are 10 pairs of nontouching loops:

Loop pair 1:	(loop 2, loop 8)	$(C_2C_8 = X^4)$
Loop pair 2:	(loop 3, loop 11)	$(C_3C_{11} = X^8)$
Loop pair 3:	(loop 4, loop 8)	$(C_4C_8 = X^3)$
Loop pair 4:	(loop 4, loop 11)	$(C_4C_{11} = X^3)$
Loop pair 5:	(loop 6, loop 11)	$(C_6C_{11} = X^9)$
Loop pair 6:	(loop 7, loop 9)	$(C_7C_9 = X^8)$
Loop pair 7:	(loop 7, loop 10)	$(C_7C_{10} = X^7)$

12-25

Loop pair 8: (loop 7, loop 11) $(C_7C_{11} = X^4)$
Loop pair 9: (loop 8, loop 11) $(C_8C_{11} = X^2)$
Loop pair 10: (loop 10, loop 11) $(C_{10}C_{11} = X^5)$.

There are two triples of nontouching loops:

Loop triple 1: (loop 4, loop 8, loop 11) $(C_4C_8C_{11} = X^4)$
Loop triple 2: (loop 7, loop 10, loop 11) $(C_7C_{10}C_{11} = X^8)$.

There are no other sets of nontouching loops. Therefore,

$$\begin{aligned}\Delta = 1 &- (X^8 + X^3 + X^7 + X^2 + X^4 + X^3 + X^3 + X + X^5 + X^4 + X)\\ &+ (X^4 + X^8 + X^3 + X^3 + X^4 + X^8 + X^7 + X^4 + X^2 + X^5)\\ &- (X^4 + X^8)\\ &= 1 - 2X - X^3.\end{aligned}$$

There are seven forward paths in the graph of Figure 10.5(a):

Forward path 1: $S_0S_1S_3S_7S_6S_5S_2S_4S_0$ $(F_1 = X^{12})$
Forward path 2: $S_0S_1S_3S_7S_6S_4S_0$ $(F_2 = X^7)$
Forward path 3: $S_0S_1S_3S_6S_5S_2S_4S_0$ $(F_3 = X^{11})$
Forward path 4: $S_0S_1S_3S_6S_4S_0$ $(F_4 = X^6)$
Forward path 5: $S_0S_1S_2S_5S_3S_7S_6S_4S_0$ $(F_5 = X^8)$
Forward path 6: $S_0S_1S_2S_5S_3S_6S_4S_0$ $(F_6 = X^7)$
Forward path 7: $S_0S_1S_2S_4S_0$ $(F_7 = X^7)$.

Forward paths 1 and 5 touch all states in the graph, and hence the subgraph not touching these paths contains no states. Therefore,

$$\Delta_1 = \Delta_5 = 1.$$

The subgraph not touching forward paths 3 and 6 is shown in Figure 10.6(a), and hence

$$\Delta_3 = \Delta_6 = 1 - X.$$

The subgraph not touching forward path 2 is shown in Figure 10.6(b), and hence

$$\Delta_2 = 1 - X.$$

The subgraph not touching forward path 4 is shown in Figure 10.6(c), and hence

$$\Delta_4 = 1 - (X + X) + (X^2) = 1 - 2X + X^2.$$

The subgraph not touching forward path 7 is shown in Figure 10.6(d), and hence

$$\Delta_7 = 1 - (X + X^4 + X^5) + (X^5) = 1 - X - X^4.$$

The generating function for this graph is then given by

$$\begin{aligned}T(X) &= \frac{\begin{array}{l}X^{12}\cdot 1 + X^7(1 - X) + X^{11}(1 - X)\\ \qquad + X^6(1 - 2X + X^2) + X^8\cdot 1 + X^7(1 - X) + X^7(1 - X - X^4)\end{array}}{1 - 2X - X^3}\\ &= \frac{X^6 + X^7 - X^8}{1 - 2X - X^3}\\ &= X^6 + 2X^7 + 5X^8 + 11X^9 + 25X^{10} + \cdots.\end{aligned}$$

$T(X)$ provides a complete description of the weight distribution of all nonzero code words that diverge from and remerge with state S_0 exactly once. In this case there is one such code word of weight 6, three of weight 7, five of weight 8, and so on.

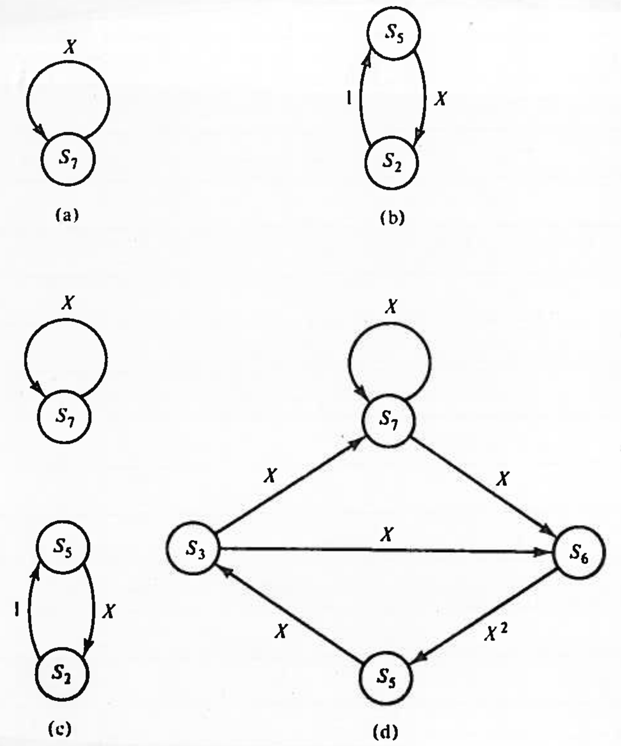


Figure 10.6 Subgraphs for computing Δ_l.

Additional information about the structure of a code can be obtained using the same procedure. If the modified state diagram is augmented by labeling each branch corresponding to a nonzero information block with Y^j, where j is the weight of the k information bits on that branch, and labeling every branch with Z, the generating function is given by

$$T(X, Y, Z) = \sum_{i,j,l} A_{i,j,l} X^i Y^j Z^l. \tag{10.25}$$

The coefficient $A_{i,j,l}$ denotes the number of code words with weight i, whose associated information sequence has weight j, and whose length is l branches. The augmented state diagram for the code of Figure 10.1 is shown in Figure 10.7.

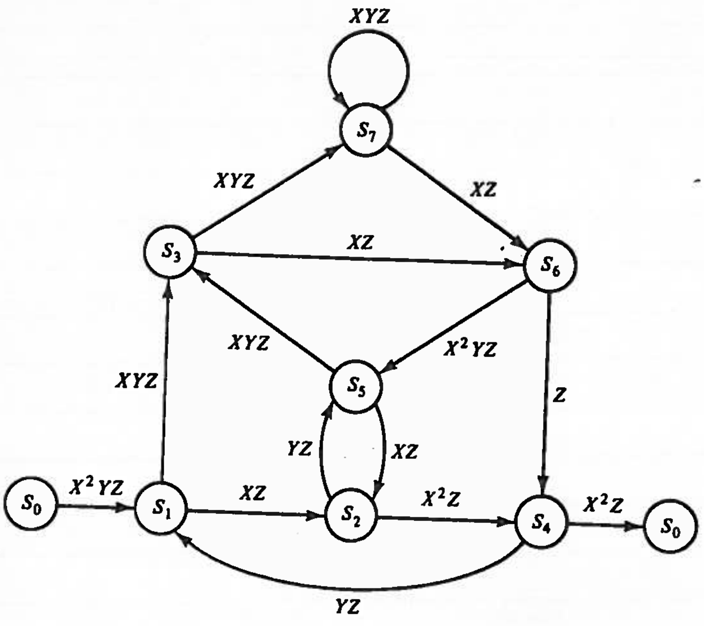


Figure 10.7 Augmented state diagram for the (2, 1, 3) code.

Example 10.11

For the graph of Figure 10.7,

$$\begin{aligned}\Delta = 1 &- (X^8Y^4Z^7 + X^3Y^3Z^5 + X^7Y^3Z^6 + X^2Y^2Z^4 + X^4Y^4Z^7 \\ &+ X^3Y^3Z^6 + X^3YZ^3 + XYZ^2 + X^5Y^3Z^4 + X^4Y^2Z^3 + XYZ) \\ &+ (X^4Y^4Z^7 + X^8Y^4Z^7 + X^3Y^3Z^6 + X^3Y^3Z^5 + X^4Y^4Z^7 \\ &+ X^8Y^4Z^7 + X^7Y^3Z^6 + X^4Y^2Z^4 + X^2Y^2Z^3 + X^5Y^3Z^4) \\ &- (X^4Y^4Z^7 + X^8Y^4Z^7) \\ = 1 &+ XY(Z + Z^2) - X^2Y^2(Z^4 - Z^3) - X^3(YZ^3 - Y^3Z^6) \\ &- X^4Y^2(Z^3 - Z^4) - X^8(Y^3Z^6 - Y^4Z^7) - X^9Y^4Z^7\end{aligned}$$

and

$$\begin{aligned}\sum_i F_i\Delta_i = X^{12}Y^4Z^8\cdot 1 &+ X^7Y^3Z^6(1 - XYZ^2) + X^{11}Y^3Z^7(1 - XYZ) \\ &+ X^6Y^2Z^5(1 - XY(Z + Z^2) + X^2Y^2Z^3) + X^8Y^4Z^8\cdot 1 \\ &+ X^7Y^3Z^7(1 - XYZ) + X^7YZ^4(1 - XYZ - X^4Y^2Z^3) \\ = X^6Y^2Z^5 &+ X^7YZ^4 - X^8Y^2Z^5.\end{aligned}$$

Hence, the generating function is

$$\begin{aligned}T(X, Y, Z) &= \frac{X^6Y^2Z^5 + X^7YZ^4 - X^8Y^2Z^5}{\Delta} \\ &= X^6Y^2Z^5 + X^7(YZ^4 + Y^3Z^6 + Y^3Z^7) \\ &\quad + X^8(Y^2Z^6 + Y^4Z^7 + Y^4Z^8 + 2Y^4Z^9) + \cdots.\end{aligned}$$

This implies that the code word of weight 6 has length 5 branches and an information sequence of weight 2, one code word of weight 7 has length 4 branches and information sequence weight 1, another has length 6 branches and information sequence weight 3, the third has length 7 branches and information sequence weight 3, and so on.