MODULE 5

CHAPTER 1: SOME IMPORTANT CYCLIC CODES

STRUCTURE

- Golay codes
- BCH codes

OBJECTIVE

- Discuss about BCH codes and Golay codes.

5.1 GOLAY CODES

Golay code is a (**23, 12**) perfect binary code that is capable of correcting any combination of three or fewer random errors in a block of **23** bits. It is a perfect code because it satisfies the Hamming bound with the equality sign for **t** = 3 as:

$$\left[\binom{23}{0}+\binom{23}{1}+\binom{23}{2}+\binom{23}{3}\right]2^{12}=2^{23}$$

The code has been used in many practical systems. The generator polynomial for the code is obtained from the relation $\mathbf{(X^{23}+1) = (X+1)\, g_1(X)\, g_2(X)}$, where:

$\mathbf{g_1(X) = 1 + X^2 + X^4 + X^5 + X^6 + X^{10} + X^{11}}$ and $\mathbf{g_2(X) = 1 + X + X^5 + X^6 + X^7 + X^9 + X^{11}}$

The encoder can be implemented using shift registers using either $\mathbf{g_1(X)}$ or $\mathbf{g_2(X)}$ as the divider polynomial. The code has a minimum distance, $\mathbf{d_{min} = 7}$. The extended Golay code, a (**924, 12**) code has $\mathbf{d_{min} = 8}$. Besides the binary Golay code, there is also a perfect ternary **(11, 6)** Golay code with $\mathbf{d_{min} = 5}$.

5.2 BCH CODES

BCH codes can also be used for burst error correction. If correction of single bursts is the only requirement, implementation would be much simpler. The additional error correcting ability can be used to error detection (any error pattern that the code would correct, if fully utilized, it could certainly still detect) or it can be used to correct bursts beyond its guaranteed burst correcting capability.

5.2.1 Decoding:

Burst error correcting cyclic codes can be decoded with an extremely simple version of the error trapping decoder. Fig 5.1 illustrates a scheme for decoding an (**n, k**) cyclic code with burst error correcting ability '**b**' (The following description assumes a Binary code. Defining suitable **q**-ary logic gates it can be easily generalized for **q** – ary codes). To understand the operation of the circuit we proceed as below.

- Assume that a burst of length '**b**' or less has occurred.
- The circuit computes a shifted version of the syndrome of the error pattern which is contained in the received word.
- If the burst is confined to the '**b**' high-order positions of the received word, then the '**b**' right most stages of the syndrome generator contain the burst and the (**n-k-b**) left stages will contain zeros.

- The logical OR gate detects this situation. Since its output is now **0**, the feedback path in the syndrome generator is opened and the path to the binary adder is closed.
- Upon shifting the syndrome generator and buffer register, the syndrome, which is the error burst, is added to the received word, thereby correcting it.
- If the burst has occurred elsewhere in the word, it would be corrected in exactly the same manner. For, after shifting, the burst will occupy the '**b**' high-order positions of the buffer. At this time the '**b**' high-order bits of the syndrome will be identical to the burst as before.
- This decoding procedure does not correct "bursts" that occupy only the '**i**' high-order positions of the word and (**b-i**) low-order positions. Such bursts are correctable with a burst-**b**-correcting cyclic code, however. It is possible, even simple; to do this if a syndrome generator which does not pre multiply the received word by $\mathbf{X^{n-k}}$ is used.

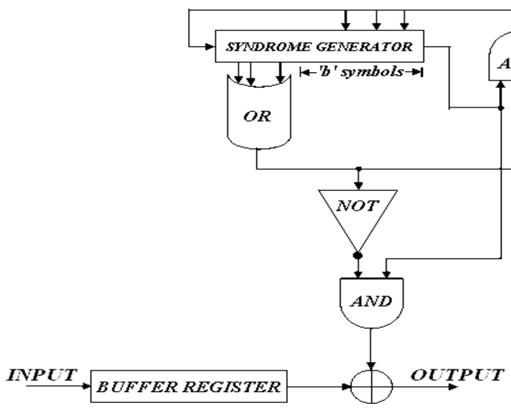


Fig 5.1 A decoder for burst error correcting cyclic codes

Example 5.1: Interleaver for a BCH code.

Consider a (**15, 7**) **BCH** code generated by $\mathbf{g(X) = 1+X+X^2+X^4+X^8}$. For this code $\mathbf{d_{min}=5}$, $\mathbf{t} = \frac{d_{mim}-1}{2} = 2$ .With **λ** =**5**, we can construct a (**75, 35**) interleaved code with a burst error correcting capability of **b**= **λt=10.** A **35**-bit message block is divided into five **7**-bit message blocks and five code words of length **15** are generated using **g(X)**.These code words are arranged as **5**-rows of a **5×15** matrix. The columns of the matrix are transmitted in the sequence shown as a **75**-bit long code vector.

$$\leftarrow \textit{Each rows is } 15-\textit{bit code word} \rightarrow$$

1	**6**	**11**	**….**	**31**	**(36)**	**…..**	**(66)**	**71**
2	**7**	**12**	**….**	**(32)**	**(37)**	**…..**	**67**	**72**
3	**8**	**13**	**….**	**(33)**	**(38)**	**…..**	**68**	**73**
4	**(9)**	**14**	**….**	**(34)**	**39**	**…..**	**69**	**74**
5	**10**	**15**	**….**	**(35)**	**40**	**……**	**(70)**	**75**

Fig 5.2 Block Interleaver for a (15, 7) BCH code.

To illustrate the burst and random error correcting capabilities of this code, we have put the bit positions **9**, **3**2 to **38**, **66** and **70** in parenthesis, indicating errors occurred in these positions .The de-interleaver now feeds the rows of Fig 5.1 to the decoder. Clearly each row has a maximum of two errors and since the (**15, 7**) **BCH** code, from which the rows were constructed, is capable of correcting up to two errors per row. Hence the error pattern shown in parenthesis in the Figure can be corrected. The isolated errors in bit positions **9**, **66** and **70** may be thought of as random errors while the cluster of errors in bit positions **32** to **38** as a burst error.

Example 5.2: The polynomial: $\mathbf{G(X) = (1+X+ X^4)(1+X^7) = 1+X+X^4+X^7+X^8+X^{11}}$, generates a binary fire code of length $\mathbf{n=7(2^4-1)=105}$, which corrects any single burst of length **4 or less**. It has **11** check symbols and **105-11=94** information symbols. The encoder for the code is shown in Fig 5.3(a) and the decoder in Fig 5.3(b).

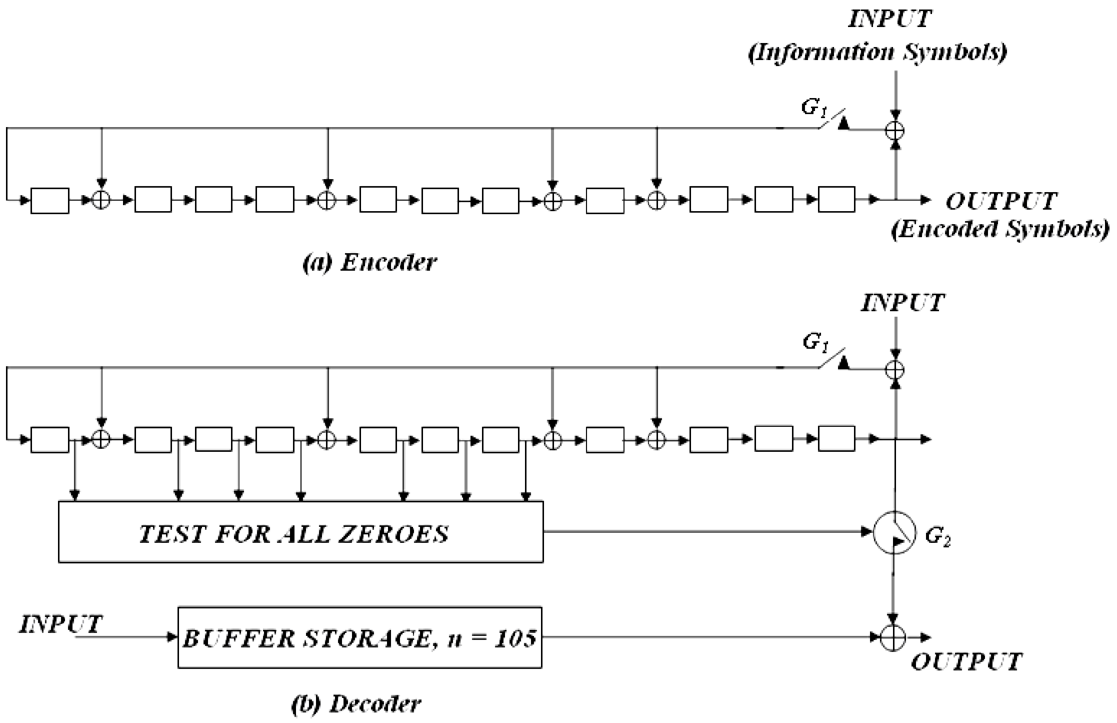


Fig 5.3 Encoder/Decoder for a (105,94)

- The decoder has a local encoder.
- The entire received vector is read into the buffer and simultaneously in to the syndrome generator with $\mathbf{G_1}$ **closed** and $\mathbf{G_2}$ **open**.
- After the syndrome bits have been formed, the received bits are read out from the buffer bit by bit and the syndrome bits are also shifted with flushing zero input.
- When allzero bits appear in the first **7** syndrome positions, the test circuit indicates the allzero pattern, at the same time the error pattern must be in the last **5** positions.
- The erroneous symbols are now ready to leave the buffer and they are corrected by the last **4** syndrome bits by addition while they are shifted bit by bit with gates $\mathbf{G_2}$ **closed** and $\mathbf{G_1}$ **open**.

It may be desirable to shorten a burst error correcting code either because (i) bursts occur too frequently or (ii) the total length or number of information bits is constrained by other system requirements. If a code of suitable natural length is not available or it cannot be found, an available code can be shortened simply by making some high-order information bits completely ‘**0**’ and omitting them. By doing this, the encoding and parity check calculations are not affected, since the leading **0’s** do not affect them. However, such shortening of codes do affect the error correction procedure. Suppose we want to form a shortened cyclic code of length **(n-s)** from an existing cyclic code of natural length **n**. This requires omitting ‘**s**’ information symbols from the original **n**. Then, to have an unaltered correction procedure, we must first shift the ‘**s**’ omitted information bits which are assumed to be **0’**s before reading the actual received code word out of the buffer. One possible simplification could be achieved by an automated pre-multiplication by ‘X^s **mod g(X)**’.

OUTCOMES

- To know how the decoding can be done.
- How the interleaving of coded data can be obtained for burst error correction.

REFERENCE

- **nptel**.ac.in/Clarify_doubts.php?subjectId=117106031&lectureId=16
- elearning.**vtu**.ac.in/EC63.html
- www.inference.phy.cam.ac.uk/itprnn/book.pdf

CHAPTER 2 CONVOLUTIONAL CODES

STRUCTURE

- Connection Pictorial Representation
- Convolutional Encoding – Time domain approach
- Encoding of Convolutional Codes; Transform Domain Approach
- Systematic Convolutional Codes
- Structural Properties of Convolutional Codes
- Maximum Likely-hood decoding of Convolutional Codes
- Sequential Decoding

OBJECTIVE

- Discuss about Convolution codes which is used to generate encoded sequences for input sequence on bit by bit basis.
- Discuss about the decoding of convolutional codes with the help of Viterbi algorithm.

INTRODUCTION

In block codes, a block of **n**-digits generated by the encoder depends only on the block of **k**-data digits in a particular time unit. These codes can be generated by combinatorial logic circuits. In a convolutional code the block of **n**-digits generated by the encoder in a time unit depends on not only on the block of **k**-data digits with in that time unit, but also on the preceding '**m**' input blocks. An (**n, k, m**) convolutional code can be implemented with **k**-input, **n**-output sequential circuit with input memory **m**. Generally, **k** and **n** are small integers with **k < n** but the memory order **m** must be made large to achieve low error probabilities. In the important special case when **k = 1**, the information sequence is not divided into blocks but can be processed continuously.

Similar to block codes, convolutional codes can be designed to either detect or correct errors. However, since the data are usually re-transmitted in blocks, block codes are better suited for error detection and convolutional codes are mainly used for error correction.

Convolutional codes were first introduced by Elias in 1955 as an alternative to block codes. This was followed later by Wozen Craft, Massey, Fano, Viterbi, Omura and others. A detailed discussion and survey of the application of convolutional codes to practical communication channels can be found in Shu-Lin & Costello Jr., J. Das etal and other standard books on error control coding.

To facilitate easy understanding we follow the popular methods of representing convolutional encoders starting with a connection pictorial - needed for all descriptions followed by connection vectors.

5.1 CONNECTION PICTORIAL REPRESENTATION

The encoder for a (rate **1/2**, **K** = **3**) or (**2, 1, 2**) convolutional code is shown in Fig.5.1. Both sketches shown are one and the same. While in Fig.5.1 (a) we have shown a 3-bit register, by noting that the content of the third stage is simply the output of the second stage, the circuit is modified using only two shift register stages. This modification, then, clearly tells us that" the memory requirement **m = 2**. For every bit inputted the encoder produces two bits at its output. Thus the encoder is labeled (n, k, m)→ (**2, 1, 2**) encoder.

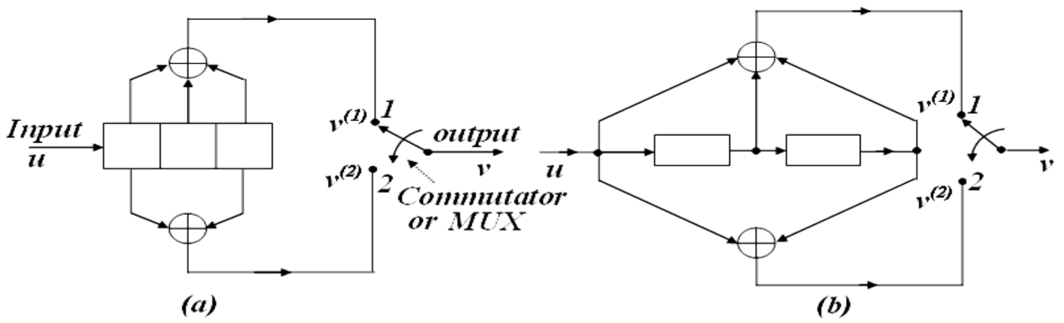


Fig 5.1 A (2,1,2) Encoder (a) Representation using 3-bit shift register (b) Equivalent representation requires only two shift register stages

At each input bit time one bit is shifted into the left most stage and the bits that were present in the registers shifted to the right by one position. Output switch (commutator /MUX) samples the output of each X-OR gate and forms the code symbol pairs for the bits introduced. The final code is obtained after flushing the encoder with "**m**" zero's where '**m**'- is the memory order (In Fig.8.1, **m = 2**). The sequence of operations performed by the encoder of Fig.5.1 for an input sequence **u = (101)** are illustrated diagrammatically in Fig 5.2.

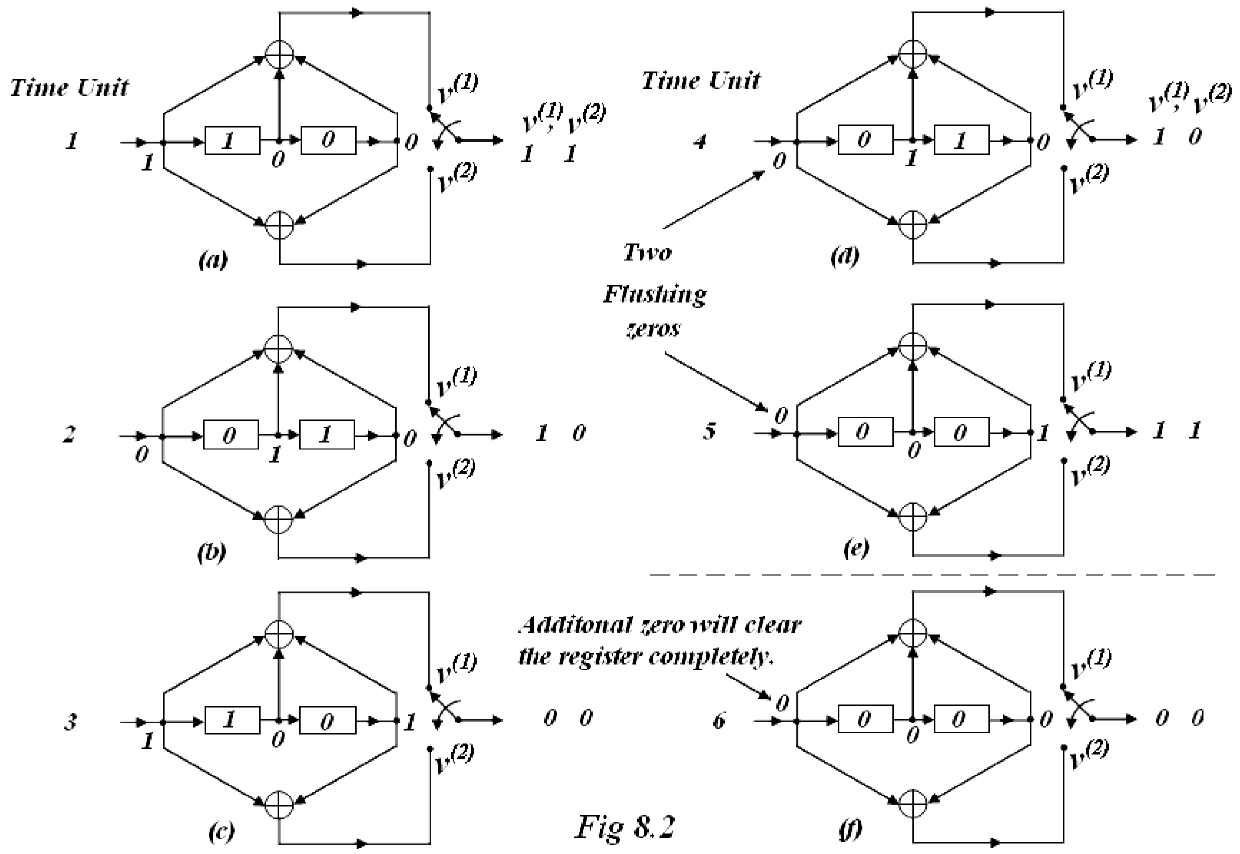


Fig 5.2

From Fig 5.2, the encoding procedure can be understood clearly. Initially the registers are in Re-set mode i.e. (**0, 0**). At the first time unit the input bit is **1**. This bit enters the first register and pushes out its previous content namely '**0**' as shown, which will now enter the second register and pushes out its previous content. All these bits as indicated are passed on to the X-OR gates and the output pair (**1, 1**) is obtained.The same steps are repeated until time unit **4**, where zeros are introduced to clear the register contents producing two more output pairs. At time unit **6**, if an additional '**0**' is introduced the encoder is re-set and the output pair (**0, 0**) obtained. However, this step is not absolutely necessary as the next bit, whatever it is, will flush out the content of the second register. The '**0**' and the '**1**' indicated at the output of second register at time unit **5** now vanishes. Hence after (**L+m**) = 3 + 2 = 5 time units, the output sequence will read **v** = (**11, 10, 00, 10, 11**). (Note: **L** = length of the input sequence). This then is the code word produced by the encoder. It is very important to remember that "**Left most symbols represent earliest transmission**".

As already mentioned the convolutional codes are intended for the purpose of error correction. However, it suffers from the 'problem of choosing connections' to yield good distance properties. The selection of connections indeed is very complicated and has not been solved yet. Still, good codes have been developed by computer search techniques for all **constraint lengths** less than

20. Another point to be noted is that the convolutional codes do not have any particular block size. They can be periodically truncated. Only thing is that they require **m**-zeros to be appended to the end of the input sequence for the purpose of '**clearing**' or '**flushing**' or '**re-setting**' of the encoding shift registers off the data bits. These added zeros carry no information but have the effect of reducing the code rate below (**k/n**). To keep the code rate close to (**k/n**), the truncation period is generally made as long as practical. The encoding procedure as depicted pictorially in Fig 5.2 is rather tedious. We can approach the encoder in terms of "Impulse response" or "generator sequence" which merely represents the response of the encoder to a single '**1**' bit that moves through it.

5.2 Convolutional Encoding – Time domain approach:

The encoder for a (**2, 1, 3**) code is shown in Fig. 8.3. Here the encoder consists of **m=3** stage shift register, **n=2** module-2 adders (X-OR gates) and a multiplexer for serializing the encoder outputs. Notice that module-2 addition is a linear operation and it follows that all convolution encoders can be implemented using a "**linear feed forward shift register circuit**".

The "information sequence' $\mathbf{u = (u_1, u_2, u_3 \ldots\ldots)}$ enters the encoder one bit at a time starting from $\mathbf{u_1}$. As the name implies, a convolutional encoder operates by performing convolutions on the information sequence. Specifically, the encoder output sequences, in this case $\mathbf{v^{(1)} = \{v_1^{(1)}, v_2^{(1)}, v_3^{(1)} \ldots\}}$ and $\mathbf{v^{(2)} = \{v_1^{(2)}, v_2^{(2)}, v_3^{(2)} \ldots\}}$ are obtained by the discrete convolution of the information sequence with the encoder "impulse responses'. The impulse responses are obtained by determining the output sequences of the encoder produced by the input sequence $\mathbf{u = (1, 0, 0, 0\ldots)}$.The impulse responses so defined are called '**generator sequences**' of the code. Since the encoder has a **m**-time unit memory the impulse responses can last at most **(m+ 1)** time units (That is a total of **(m+ 1)** shifts are necessary for a message bit to enter the shift register and finally come out) and are written as:

$$\mathbf{g^{(i)} = \{g_1^{(i)}, g_2^{(i)}, g_3^{(i)} \ldots g_{m+l}^{(i)}\}.}$$

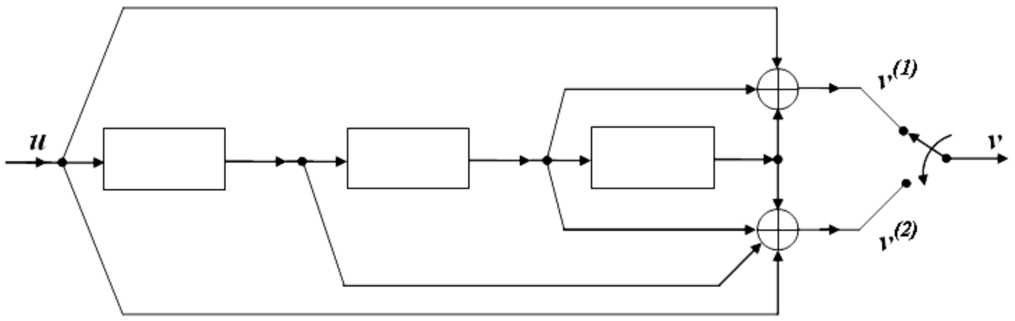


Fig 5.3 (2,1,3) binary encoder

For the encoder of Fig.5.3, we require the two impulse responses,

$\mathbf{g^{(1)} = \{g_1^{(1)}, g_2^{(1)}, g_3^{(1)}, g_4^{(1)}\}}$ and

$\mathbf{g^{(2)} = \{g_1^{(2)}, g_2^{(2)}, g_3^{(2)}, g_4^{(2)}\}}$

By inspection, these can be written as: $\mathbf{g^{(1)} = \{1, 0, 1, 1\}}$ and $\mathbf{g^{(2)} = \{1, 1, 1, 1\}}$

Observe that the generator sequences represented here is simply the 'connection vectors' of the encoder. In the sequences a '**1**' indicates a connection and a '**0**' indicates no connection to the corresponding X - OR gate. If we group the elements of the generator sequences so found in to pairs, we get the overall impulse response of the encoder, Thus for the encoder of Fig 5.3, the over-all impulse response' will be:

v = (11, 01, 11, 11)

The encoder outputs are defined by the convolution sums:

$$\mathbf{v}^{(1)} = \mathbf{u} * \mathbf{g}^{(1)} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (5.1\text{ a})$$

$$\mathbf{v}^{(2)} = \mathbf{u} * \mathbf{g}^{(2)} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (5.1\text{ b})$$

Where * denotes the 'discrete convolution', which implies:

$$v_l^{(j)} = \sum_{i=0}^{m} u_{l-i} \cdot g_{i+1}^{(j)}$$
$$= u_l\, g_1^{(j)} + u_{l-1}\, g_2^{(j)} + u_{l-2}\, g_3^{(j)} + \ldots\ldots + u_{l-m}\, g_{m+1}^{(j)} \quad \ldots\ldots\ldots\ldots\ldots \quad (5.2)$$

for **j = 1, 2** and where $u_{l-i} = 0$ for all $l < i$ and all operations are modulo - **2**. Hence for the encoder of Fig 5.3, we have:

$$v_l^{(1)} = u_l + u_{l-2} + u_{l-3}$$

$$v_l^{(2)} = u_l + u_{l-1} + u_{l-2} + u_{l-3}$$

This can be easily verified by direct inspection of the encoding circuit. After encoding, the two output sequences are multiplexed into a single sequence, called the "**code word**" for transmission over the channel. The code word is given by:

$$\mathbf{v} = \{v_1^{(1)}\, v_1^{(2)},\, v_2^{(1)}\, v_2^{(2)},\, v_3^{(1)}\, v_3^{(2)} \ldots\}$$

Example 5.1:

Suppose the information sequence be **u = (10111)**. Then the output sequences are:

$$\mathbf{v}^{(1)} = (1\,0\,1\,1\,1) * (10\,1\,1)$$
$$= (1\,0\,0\,0\,0\,0\,0\,1),$$

$$\mathbf{v}^{(2)} = (1\,0\,1\,1\,1) * (1\,1\,1)$$
$$= (1\,1\,0\,1\,1\,1\,0\,1),$$

and the code word is

v = (11, 01, 00, 01, 01, 01, 00, 11)

The discrete convolution operation described in Eq (8.2) is merely the addition of shifted impulses. Thus to obtain the encoder output we need only to shift the overall impulse response by **'one branch word'**, multiply by the corresponding input sequence and then add them. This is illustrated in the table below:

INPUT	OUT PUT
1	1 1 0 1 1 1 1 1
0	0 0 0 0 0 0 0 0 -----one branch word shifted sequence
1	1 1 0 1 1 1 1 1 ---Two branch word shifted
1	1 1 0 1 1 1 1 1
1	1 1 0 1 1 1 1 1
Modulo -2 sum	1 1 0 1 0 0 0 1 0 1 0 1 0 0 1 1

The Modulo-2 sum represents the same sequence as obtained before. There is no confusion at all with respect to indices and suffices! Very easy approach - super position or linear addition of shifted impulse response - demonstrates that the convolutional codes are linear codes just as the block codes and cyclic codes. This approach then permits us to define a **'Generator matrix'** for the convolutional encoder. Remember, that interlacing of the generator sequences gives the overall impulse response and hence they are used as the rows of the matrix. The number of rows equals the number of information digits. Therefore the matrix that results would be "**Semi-Infinite**". The second and subsequent rows of the matrix are merely the shifted versions of the first row -They are each shifted with respect to each other by "**One branch word**". If the information sequence **u** has a finite length, say **L**, then **G** has **L** rows and **n × (m +L)** columns (or **(m +L)** branch word columns) and **v** has a length of **n × (m +L)** or a length of **(m +L)** branch words. Each branch word is of length '**n**'. Thus the Generator matrix **G**, for the encoders of type shown in Fig 8.3 is written as:

$$G = \begin{bmatrix} g_1^{(1)}g_1^{(2)} & g_2^{(1)}g_2^{(2)} & g_3^{(1)}g_3^{(2)} & g_4^{(1)}g_4^{(2)} & \cdots & \cdots & \cdots \\ & g_1^{(1)}g_1^{(2)} & g_2^{(1)}g_2^{(2)} & g_3^{(1)}g_3^{(2)} & g_4^{(1)}g_4^{(2)} & \cdots & \cdots \\ & & g_1^{(1)}g_1^{(2)} & g_2^{(1)}g_2^{(2)} & g_3^{(1)}g_3^{(2)} & g_4^{(1)}g_4^{(2)} & \cdots \end{bmatrix} \quad \ldots.. \quad (5.3)$$

(Blank places are zeros.)

The encoding equations in Matrix form is:

$$\mathbf{v} = \mathbf{u}\,.\mathbf{G} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots. \qquad (5.4)$$

Example 5.2:

For the information sequence of Example 5.1, the G matrix has 5 rows and 2(3 +5) =16 columns and we have

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

Performing multiplication, **v = u G** as per Eq (5.4), we get: **v = (11, 01, 00, 01, 01, 00, 11)** same as before.

As a second example of a convolutional encoder, consider the (**3, 2, 1**) encoder shown in Fig.8.4. Here, as **k =2**, the encoder consists of two **m = 1** stage shift registers together with **n = 3** modulo -2 adders and two multiplexers. The information sequence enters the encoder **k = 2** bits at a time and can be written as $\mathbf{u} = \{u_1^{(1)}\, u_1^{(2)}, u_2^{(1)}\, u_2^{(2)}, u_3^{(1)}\, u_3^{(2)} \ldots\}$ or as two separate input sequences:

$\mathbf{u}^{(1)} = \{u_1^{(1)}, u_2^{(1)}, u_3^{(1)} \ldots\}$ and $\mathbf{u}^{(2)} = \{u_1^{(2)}, u_2^{(2)}, u_3^{(2)} \ldots\}$.

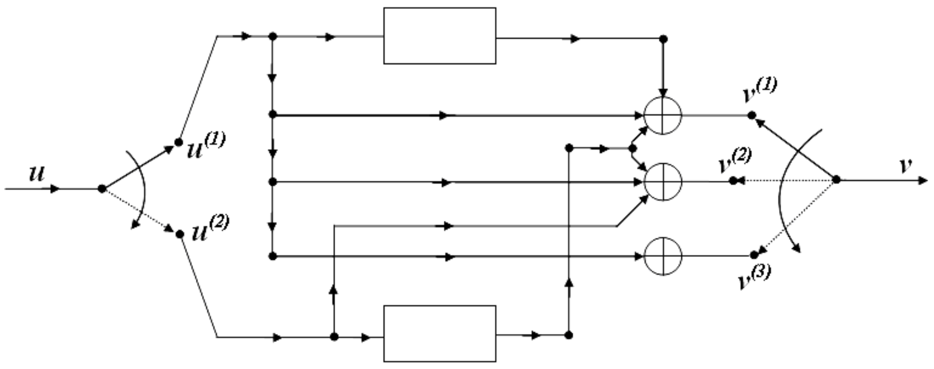


Fig 5.4 (3,2,1) Convolution encoder

There are three generator sequences corresponding to each input sequence. Letting $\mathbf{g}_i^{(j)} = \{g_{i,1}^{(j)}, g_{i,2}^{(j)}, g_{i,3}^{(j)} \ldots g_{i,m+1}^{(j)}\}$ represent the generator sequence corresponding to input **i** and output **j**. The generator sequences for the encoder are:

$$\mathbf{g}_1^{(1)} = (1, 1), \mathbf{g}_1^{(2)} = (1, 0), \mathbf{g}_1^{(3)} = (1, 0)$$

$$\mathbf{g}_2^{(1)} = (0, 1), \mathbf{g}_2^{(2)} = (1, 1), \mathbf{g}_2^{(3)} = (0, 0)$$

The encoding equations can be written as:

$\mathbf{v}^{(1)} = \mathbf{u}^{(l)} * \mathbf{g}_1^{(1)} + \mathbf{u}^{(2)} * \mathbf{g}_2^{(1)}$ ………………….. (5.5 a)

$\mathbf{v}^{(2)} = \mathbf{u}^{(1)} * \mathbf{g}_1^{(2)} + \mathbf{u}^{(2)} * \mathbf{g}_2^{(2)}$ ………………….. (5.5 b)

$\mathbf{v}^{(3)} = \mathbf{u}^{(l)} * \mathbf{g}_1^{(3)} + \mathbf{u}^{(2)} * \mathbf{g}_2^{(3)}$ ………………… (5.5 c)

The convolution operation implies that:

$$\mathbf{v}_l^{(1)} = \mathbf{u}_l^{(1)} + \mathbf{u}_{l-1}^{(1)} + \mathbf{u}_{l-1}^{(2)}$$

$$\mathbf{v}_l^{(2)} = \mathbf{u}_l^{(1)} + \mathbf{u}_l^{(2)} + \mathbf{u}_{l-1}^{(2)}$$

$$\mathbf{v}_l^{(3)} = \mathbf{u}_l^{(1)}$$

as can be seen from the encoding circuit.

After multiplexing, the code word is given by:

$$\mathbf{v} = \{ \mathbf{v}_1^{(1)} \mathbf{v}_1^{(2)} \mathbf{v}_1^{(3)}, \mathbf{v}_2^{(1)} \mathbf{v}_2^{(2)} \mathbf{v}_2^{(3)}, \mathbf{v}_3^{(1)} \mathbf{v}_3^{(2)} \mathbf{v}_3^{(3)} \ldots \}$$

Example 5.3:

Suppose **u** = (**1 1 0 1 1 0**). Hence $\mathbf{u}^{(1)}$ = (**1 0 1**) and $\mathbf{u}^{(2)}$ = (**1 1 0**). Then

$$\mathbf{v}^{(1)} = (1\ 0\ 1) * (1,1) + (1\ 1\ 0) *(0,1) = (1\ 0\ 0\ 1)$$

$$\mathbf{v}^{(2)} = (1\ 0\ 1) * (1,0) + (1\ 1\ 0) *(1,1) = (0\ 0\ 0\ 0)$$

$$\mathbf{v}^{(3)} = (1\ 0\ 1) * (1,0) + (1\ 1\ 0) *(0,0) = (1\ 0\ 1\ 0)$$

$$\therefore\ \mathbf{v} = (1\ 0\ 1,\ 0\ 0\ 0,\ 0\ 0\ 1,\ 1\ 0\ 0).$$

The generator matrix for a (**3, 2, m**) code can be written as:

$$G = \begin{bmatrix} g_{11}^{(1)}g_{11}^{(2)}g_{11}^{(3)} & g_{12}^{(1)}g_{12}^{(2)}g_{13}^{(3)} & \cdots & g_{1,m+1}^{(1)}g_{1,m+1}^{(2)}g_{1,m+1}^{(3)} & \\ g_{21}^{(1)}g_{21}^{(2)}g_{21}^{(3)} & g_{22}^{(1)}g_{22}^{(2)}g_{22}^{(3)} & \cdots & g_{2,m+1}^{(1)}g_{2,m+1}^{(2)}g_{2,m+1}^{(3)} & \\ \ddots & g_{11}^{(1)}g_{11}^{(2)}g_{11}^{(3)} & g_{12}^{(1)}g_{12}^{(2)}g_{12}^{(3)} & \cdots & \\ \ddots & g_{21}^{(1)}g_{21}^{(2)}g_{21}^{(3)} & g_{22}^{(1)}g_{22}^{(2)}g_{22}^{(3)} & \cdots & \\ \ddots & \ddots & \ddots & \ddots & \end{bmatrix} \quad \ldots\ldots \quad (5.6)$$

The encoding equations in matrix form are again given by **v = u G**. observe that each set of **k = 2** rows of **G** is identical to the preceding set of rows but shifted by **n = 3** places or one branch word to the right.

Example 5.4:

For the Example 5.3, we have

$$\mathbf{u} = \{u_1^{(1)}\ u_1^{(2)},\ u_2^{(1)}\ u_2^{(2)},\ u_3^{(1)}\ u_3^{(2)}\} = (1\ 1,\ 0\ 1,\ 1\ 0)$$

The generator matrix is:

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

***Remember that the blank places in the matrix are all zeros.**

Performing the matrix multiplication, **v = u G**, we get: **v = (101,000,001,100),** again agreeing with our previous computation using discrete convolution.

This second example clearly demonstrates the complexities involved, when the number of input sequences are increased beyond k > 1, in describing the code. In this case, although the encoder contains k shift registers all of them need not have the same length. If $\mathbf{k_i}$ is the length of the **i**-th shift register, then we define the encoder "**memory order, m**" by

$$m \triangleq \max_{1\le i\le k} k_i \qquad \text{………………} \qquad (5.7)$$

(i.e. the maximum length of all **k**-shift registers)

An example of a (**4, 3, 2**) convolutional encoder in which the shift register lengths are **0**, **1** and **2** is shown in Fig 5.5.

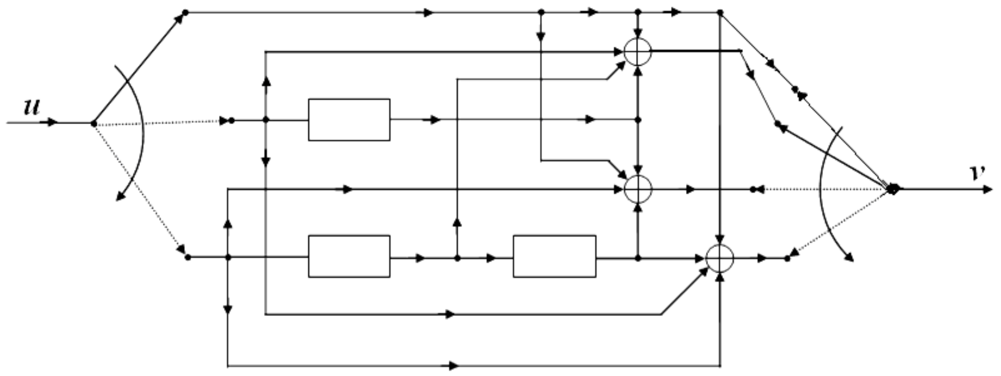


Fig 5.5 (4,3,2) Binary convolution code encoder

Since each information bit remains in the encoder up to (**m + 1**) time units and during each time unit it can affect any of the **n**-encoder outputs (which depends on the shift register connections) it follows that "**the maximum number of encoder outputs that can be affected by a single information bit**" is

$$n_A \triangleq n(m+1) \qquad \text{……………………} \qquad (5.8)$$

'$\mathbf{n_A}$' is called the 'constraint length" of the code. For example, the constraint lengths of the encoders of Figures 5.3, 5.4 and 5.5 are 8, 6 and 12 respectively. Some authors have defined the constraint length (For example: Simon Haykin) as the number of shifts over which a single message bit can influence the encoder output. In an encoder with an **m**-stage shift register, the "**memory**" of the encoder equals **m**-message bits, and the constraint length $\mathbf{n_A = (m + 1)}$. However, we shall adopt the definition given in Eq (5.8).

The number of shifts over which a single message bit can influence the encoder output is usually denoted as **K.** For the encoders of Fig 5.3, 5.4 and 5.5 have values of **K** = **4, 2** and **3** respectively. The encoder in Fig 8.3 will be accordingly labeled as a '**rate 1/2, K = 4'** convolutional encoder. The term **K** also signifies the number of branch words in the encoder's impulse response.

Turning back, in the general case of an (**n, k, m**) code, the generator matrix can be put in the form:

$$G = \begin{bmatrix} G_1 & G_2 & G_3 & \cdots & G_m & G_{m+1} & & & \\ & G_1 & G_2 & \cdots & G_{m-1} & G_m & G_{m+1} & & \\ & & G_1 & \cdots & G_{m-2} & G_{m-1} & G_m & G_{m+1} & \\ & & \ddots & & \ddots & \ddots & \ddots & \ddots & \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots \quad (5.9)$$

Where each $\mathbf{G_i}$ is a (**k × n**) sub matrix with entries as below:

$$G_i = \begin{bmatrix} g_{1,i}^{(1)} & g_{1,i}^{(2)} & \cdots & g_{1,i}^{(n)} \\ g_{2,i}^{(1)} & g_{2,i}^{(2)} & \cdots & g_{2,i}^{(n)} \\ \vdots & \vdots & \vdots & \vdots \\ g_{k,i}^{(1)} & g_{k,i}^{(2)} & \cdots & g_{k,i}^{(n)} \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (5.10)$$

Notice that each set of **k**-rows of **G** are identical to the previous set of rows but shifted **n**-places to the right. For an information sequence $\mathbf{u = (u_1, u_2\ldots)}$ where $\mathbf{u_i = \{u_i^{(1)}, u_i^{(2)}\ldots u_i^{(k)}\}}$, the code word is $\mathbf{v = (v_1, v_2\ldots)}$ where $\mathbf{v_j = (v_j^{(1)}, v_j^{(2)} \ldots . v_j^{(n)})}$ and $\mathbf{v = u\ G}$. Since the code word is a linear combination of rows of the **G** matrix it follows that an (**n, k, m**) convolutional code is a linear code.

Since the convolutional encoder generates **n**-encoded bits for each **k**-message bits, we define **R = k/n** as the "**code rate**". However, an information sequence of finite length **L** is encoded into a code word of length **n ×(L +m)**, where the final **n×m** outputs are generated after the last non zero information block has entered the encoder. That is, an information sequence is terminated with all zero blocks in order to clear the encoder memory. The terminating sequence of **m**-zeros is called the "**Tail of the message**". Viewing the convolutional-code as a linear block code, with generator matrix **G**, then the block code rate is given by **kL/n(L +m)** - the ratio of the number of message bits to the length of the code word. If **L >> m**, then**, L/ (L +m)** ≈ 1 and the block code rate of a convolutional code and its rate when viewed as a block code would appear to be same. Infact, this is the normal mode of operation for convolutional codes and accordingly we shall not distinguish between the rate of a convolutional code and its rate when viewed as a block code. On the contrary, if '**L'** were small, the effective rate of transmission indeed is **kL/n (L + m)** and will be below the block code rate by a fractional amount:

$$\frac{k/n - kL/n(L+m)}{k/n} = \frac{m}{L+m} \qquad \text{……………………..} \qquad (5.11)$$

and is called "**fractional rate loss**". Therefore, in order to keep the fractional rate loss at a minimum (near zero), ‘**L**’ is always assumed to be much larger than ‘**m**’. For the information 'sequence of Example 8.1, we have **L = 5**, **m =3** and **fractional rate loss = 3/8 = 37.5%**. If **L** is made 1000, the fractional rate loss is only **3/1003≈ 0.3%**.

5.3 Encoding of Convolutional Codes; Transform Domain Approach:

In any linear system, we know that the time domain operation involving the convolution integral can be replaced by the more convenient transform domain operation, involving polynomial multiplication. Since a convolutional encoder can be viewed as a 'linear time invariant finite state machine, we may simplify computation of the adder outputs by applying appropriate transformation. As is done in cyclic codes, each 'sequence in the encoding equations can' be replaced by a corresponding polynomial and the convolution operation replaced by polynomial multiplication. For example, for a (**2, 1, n$_i$**) code, the encoding equations become:

$$\mathbf{v^{(1)}(X) = u(X)\, g^{(l)}(X)} \qquad \text{……………..} \qquad (5.12a)$$

$$\mathbf{v^{(2)}(X) = u(X)\, g^{(2)}(X)} \qquad \text{……………....} \qquad (5.12b)$$

Where $\mathbf{u(X) = u_1 + u_2X + u_3X^2 + \ldots}$ is the information polynomial,

$\mathbf{v^{(1)}(X) = v_1^{(1)} + v_2^{(1)}X + v_3^{(1)} X^2 + \ldots..}$, and

$\mathbf{v^{(2)}(X) = v_1^{(2)} + v_2^{(2)}X + v_3^{(2)} X^2 + \ldots..}$

are the encoded polynomials.

$\mathbf{g^{(1)}(X) = g_1^{(1)} + g_2^{(1)} X + g_3^{(1)} X^2 + \ldots..}$, and

$\mathbf{g^{(2)}(X) = g_1^{(2)} + g_2^{(2)} X + g_3^{(2)} X^2 + \ldots..}$

are the “generator polynomials” of' the code; and all operations are modulo-2. After multiplexing, the code word becomes:

$$\mathbf{v(X) = v^{(1)}(X^2) + X\, v^{(2)}(X^2)} \qquad \text{…………………} \qquad (5.13)$$

The indeterminate '**X**' can be regarded as a “**unit-delay operator**”, the power of **X** defining the number of time units by which the associated bit is delayed with respect to the initial bit in the sequence.

Example 5.5:

For the (**2, 1, 3**) encoder of Fig 8.3, the impulse responses were: $\mathbf{g^{(1)} = (1,0,1,1)}$, and $\mathbf{g^{(2)} = (1,1,1,1)}$

The generator polynomials are: $\mathbf{g^{(l)}(X) = 1 + X^2 + X^3}$, and $\mathbf{g^{(2)}(X) = 1 + X + X^2 + X^3}$

For the information sequence $\mathbf{u = (1, 0, 1, 1, 1)}$; the information polynomial is: $\mathbf{u(X) = 1+X^2+X^3+X^4}$

The two code polynomials are then:

$$\mathbf{v^{(1)}(X) = u(X)\, g^{(l)}(X) = (1 + X^2 + X^3 + X^4)(1 + X^2 + X^3) = 1 + X^7}$$

$$\mathbf{v^{(2)}(X) = u(X)\, g^{(2)}(X) = (1 + X^2 + X^3 + X^4)(1 + X + X^2 + X^3) = 1 + X + X^3 + X^4 + X^5 + X^7}$$

From the polynomials so obtained we can immediately write:

$\mathbf{v^{(1)} = (1\ 0\ 0\ 0\ 0\ 0\ 0\ 1)}$, and $\mathbf{v^{(2)} = (1\ 1\ 0\ 1\ 1\ 1\ 0\ 1)}$
Pairing the components we then get the code word **v = (11, 01, 00, 01, 01, 01, 00, 11).**

We may use the multiplexing technique of Eq (5.13) and write:

$\mathbf{v^{(l)}(X^2) = 1 + X^{14}}$ and $\mathbf{v^{(2)}(X^2) = 1+X^2+X^6+X^8+X^{10}+X^{14}}$; $\mathbf{Xv^{(2)}(X^2) = X + X^3 + X^7 + X^9 + X^{11} + X^{15}}$;

and the code polynomial is: $\mathbf{v(X) = v^{(1)}(X^2) + X\, v^{(2)}(X^2) = 1 + X + X^3 + X^7 + X^9 + X^{11} + X^{14} + X^{15}}$

Hence the code word is: **v = (1 1, 0 1, 0 0, 0 1, 0 1, 0 1, 0 0, 1 1);** this is exactly the same as obtained earlier.

The generator polynomials of an encoder can be determined directly from its circuit diagram. Specifically, the co-efficient of $\mathbf{X^l}$ is a '**1**' if there is a "**connection**" from the **l**-th shift register stage to the input of the adder of interest and a '**0**' otherwise. Since the last stage of the shift register in an (**n, l**) code must be connected to at least one output it follows that at least one generator polynomial should have a degree equal to the shift register length '**m**', i.e.

$$m = \max_{1 \le j \le n} \left\{ \deg\ g^{(j)}(X) \right\} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (5.14)$$

In an (**n, k**) code, where **k > 1**, there are **n**-generator polynomials for each of the **k**-inputs, each set representing the connections from one of the shift registers to the **n**-outputs. Hence, the length $\mathbf{K_l}$ of the **l**-th shift register is given by:

$$K_l = \max_{1 \le j \le n} \left\{ \deg\ g_l^{(j)}(X) \right\},\ 1 \le l \le k \qquad \ldots\ldots\ldots\ldots \qquad (5.15)$$

Where $\mathbf{g_l^{(j)}(X)}$ is the generator polynomial relating the **l**-th input to the **j**-th output and the encoder memory order **m** is:

$$m = \max_{1 \le l \le k} K_l = \max_{\substack{1 \le j \le \\ 1 \le l \le k}} \left\{ \deg \; g_l^{(j)}(X) \right\} \qquad \ldots\ldots\ldots\ldots \qquad (5.16)$$

Since the encoder is a linear system and **u $^{(l)}$ (X)** represents the **l**-th input sequence and **v $^{(j)}$ (X)** represents the **j**-th output sequence the generator polynomial **g$_l$ $^{(j)}$ (X)** can be regarded as the 'encoder transfer function' relating the input - **l** to the output – **j.** For the **k**-input, **n-** output linear system there are a total of **k×n** transfer functions which can be represented as a (**k × n**) "**transfer function matrix**".

$$G(X) = \begin{bmatrix} g_1^{(1)}(X), & g_1^{(2)}(X), & \cdots & g_1^{(n)}(X) \\ g_2^{(1)}(X), & g_2^{(2)}(X), & \cdots & g_2^{(n)}(X) \\ \vdots & \vdots & \vdots & \vdots \\ g_k^{(1)}(X), & g_k^{(2)}(X), & \cdots & g_k^{(n)}(X) \end{bmatrix} \qquad \ldots\ldots\ldots\ldots\ldots \qquad (5.17)$$

Using the transfer function matrix, the encoding equations for an (**n, k, m**) code can be expressed as

$$\mathbf{V(X) = U(X)\, G(X)} \qquad \ldots\ldots\ldots\ldots \qquad (5.18)$$

U(X) = [u $^{(1)}$ (X), u $^{(2)}$ (X)...u $^{(k)}$ (X)] is the **k**-vector, representing the information polynomials, and. **V(X) = [v $^{(1)}$ (X), v $^{(2)}$ (X) … v $^{(n)}$ (X)]** is the **n**-vector representing the encoded sequences. After multiplexing, the code word becomes:

$$\mathbf{v(X) = v^{(1)}(X^n) + X\, v^{(2)}(X^n) + X^2\, v^{(3)}(X^n) + \ldots + X^{n-1}\, v^{(n)}(X^n)} \qquad \ldots\ldots\ldots\ldots \qquad (5.19)$$

Example 5.6:

For the encoder of Fig 5.4, we have:

$\mathbf{g_1^{(1)}(X) = 1 + X}$, $\quad \mathbf{g_2^{(1)}(X) = X}$

$\mathbf{g_1^{(2)}(X) = 1}$, $\quad \mathbf{g_2^{(2)}(X) = 1 + X}$

$\mathbf{g_1^{(3)}(X) = 1}$, $\quad \mathbf{g_2^{(3)}(X) = 0}$

$$\therefore \; G(X) = \begin{bmatrix} 1+X & 1 & 1 \\ X & 1+X & 0 \end{bmatrix}$$

For the information sequence **u $^{(1)}$ = (1 0 1), u $^{(2)}$ = (1 1 0)**, the information polynomials are:

$\mathbf{u^{(1)}(X) = 1 + X^2,\ u^{(2)}(X) = 1 + X}$

Then $\quad \mathbf{V(X) = [v^{(1)}(X), v^{(2)}(X), v^{(3)}(X)]}$

$$= [1 + X^2, 1 + X] \begin{bmatrix} 1+X & 1 & 1 \\ X & 1+X & 0 \end{bmatrix} = [1 + X^3, 0, 1+X^2]$$

Hence the code word is:

$$\mathbf{v(X) = v^{(1)}(X^3) + Xv^{(2)}(X^3) + X^2v^{(3)}(X^3)}$$

$$= \mathbf{(1 + X^9) + X\ (0) + X^2(1 + X^6)}$$

$$= \mathbf{1 + X^2 + X^8 + X^9}$$

$$\therefore\ \mathbf{v = (1\ 0\ 1,\ 0\ 0\ 0,\ 0\ 0\ 1,\ 1\ 0\ 0).}$$

This is exactly the same as that obtained in Example 5.3.From Eq (5.17) and (5.18) it follows that:

$$\boldsymbol{v^{(j)}(X) = \sum_{i=1}^{k} u^{(i)}(X) g_i^{(j)}(X)}$$

And using Eq (8.19) we have:

$$\boldsymbol{v(X) = \sum_{j=1}^{n} X^{j-1} v^{(j)}(x^n)}$$

$$\boldsymbol{= \sum_{j=1}^{n} X^{j-1} \sum_{i=1}^{k} u^{(i)}(X^n) g_i^{(j)}(X^n)}$$

$$\therefore\quad \mathbf{v(X)} = \boldsymbol{\sum_{i=1}^{k} u^{(i)}(X^n) g_i(X)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (5.20)$$

Where $\boldsymbol{g_i(X) = \sum_{j=1}^{n} X^{j-1} g_i^{(j)}(X^n)}$

$$\boldsymbol{= g_i^{(1)}(X^n) + Xg_i^{(2)}(X^n) + X^2 g_i^{(3)}(X^n) + \cdots + X^{n-1} g_i^{(n)}(X^n)} \quad \ldots\ldots\ldots\ldots.. \quad (5.21)$$

is called the "**composite generator polynomial**" relating the **i-th** input sequence to **v(X)**.

Example 5.7:

From Example 5.6, we have:

$$\mathbf{g_1(X) = g_1^{(l)}(X^3) + Xg_1^{(2)}(X^3) + X^2g_l^{(3)}(X^3) = 1 + X + X^2 + X^3}$$

$$\mathbf{g_2(X) = g_2^{(1)}(X^3) + Xg_2^{(2)}(X^3) + X^2g_2^{(3)}(X^3) = X + X^3 + X^4}$$

For the input sequence $\mathbf{u^{(1)}(X) = 1 + X^2}$, $\mathbf{u^{(2)}(X) = 1 + X}$, we have

$\mathbf{v(X) = u^{(1)}(X^3)\, g_1(X) + u^{(2)}(X^3)\, g_2(X) = 1 + X^2 + X^8 + X^9}$. This is exactly the same as obtained before.

5.4 Systematic Convolutional Codes:

In a systematic code, the first k-output sequences are exact replica of the k-input sequences i.e.

$$\mathbf{v^{(i)} = u^{(i)},\ i = 1,2,\ldots\ldots k} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (5.22)$$

and the generating sequences satisfy:

$$\boldsymbol{g_i^{(j)} = \delta_{i,j} \quad i = 1,2,3\ldots.k} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (5.23)$$

where $\delta_{i,j}$ is the Kronecker delta, having values: $\delta_{i,j} = 1 \ldots$ **if j = i**
= 0 … if j ≠ i

The generator matrix for such codes is given by

$$G = \begin{bmatrix} I & P_1 & O & P_2 & O & P_3 & \cdots & O & P_{m+1} & & & \\ \ddots & \ddots & I & P_1 & O & P_2 & \cdots & O & P_m & O & P_{m+1} & \\ \ddots & \ddots & \ddots & \ddots & I & P_1 & \cdots & O & P_{m-1} & O & P_m & O \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots \quad (5.24)$$

Where **I** is a **k × k** identity (unit) matrix, **O** is the **k × k** all zero matrix and $\mathbf{P_i}$ is a **k × (n - k)** matrix given by:

$$P_i = \begin{bmatrix} g_{1,i}^{(k+1)} & g_{1,i}^{(k+2)} & \cdots & g_{1,i}^{(n)} \\ g_{2,i}^{(k+1)} & g_{2,i}^{(k+2)} & \cdots & g_{2,i}^{(n)} \\ \vdots & \vdots & \vdots & \vdots \\ g_{k,i}^{(k+1)} & g_{k,i}^{(k+2)} & \cdots & g_{k,i}^{(n)} \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (5.25)$$

Further, the transfer function matrix of the code is given by:

$$G(X) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & g_1^{(k+1)}(X) & \cdots & g_1^{(n)}(X) \\ 0 & 1 & 0 & \cdots & 0 & g_2^{(k+1)}(X) & \cdots & g_2^{(n)}(X) \\ 0 & 0 & 1 & \cdots & 0 & g_3^{(k+1)}(X) & \cdots & g_3^{(n)}(X) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & g_k^{(k+1)}(X) & \cdots & g_k^{(n)}(X) \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (5.26)$$

The first **k**-output sequences = Input sequences → Information sequences
Last **(n-k)** sequences → parity sequences.

Number of sequences required to specify a general **(n, k, m)** code = **kn.**

For a systematic code we require only **k × (n-k)** sequences. Thus systematic codes form a sub class of the set of all possible convolutional codes. Any code not satisfying Eq (5.22) to Eq (5.26) is said to be "non systematic".

Example 5.8:

Consider a (**2, 1, 3**) systematic code whose encoder is shown in Fig 5.6.

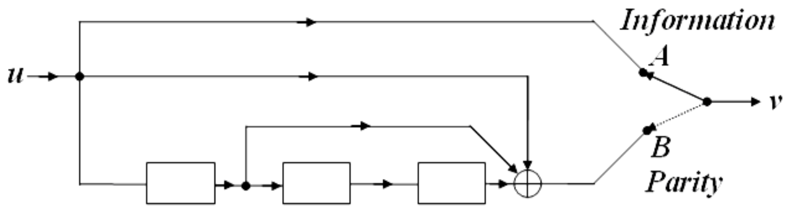


Fig 8.6 (2,1,3) systematic encoder

The generator sequences are: $\mathbf{g^{(1)} = (1\ 0\ 0\ 0)}$ and $\mathbf{g^{(2)} = (1\ 1\ 0\ 1)}$**;** and the generator matrix is:

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

The transfer function matrix is:

$$\mathbf{G(X) = [1, 1 + X + X^3]}$$

For an input sequence $\mathbf{u(X) = 1{+}X{+}X^3}$, the information sequence is:

$$\mathbf{v^{(1)}(X) = u(X)\, g^{(1)}(X) =(1 + X + X^3)}$$

and the parity sequence is:

$$\mathbf{v^{(2)}(X) = u(X)\, g^{(2)}(X) = (1+ X + X^3)\, (1 + X + X^3) = (1 + X^2 + X^6)}$$

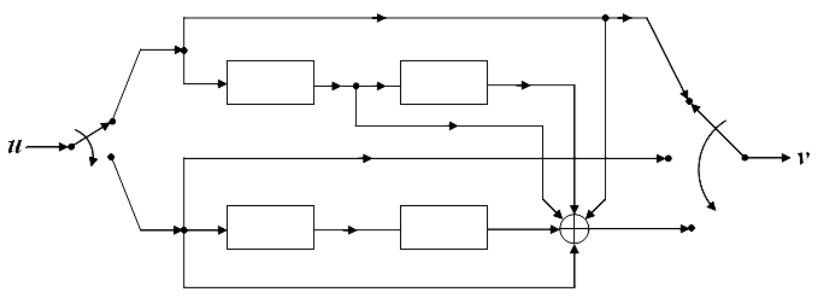
One advantage of systematic codes is that encoding is much simpler than for the non systematic codes - because less hardware is required. For example, the (**2, 1, 3**) encoder of Fig 5.6 needs only one modulo-2 adder while that of Fig 5.5 requires three such adders. Notice also the total number of inputs to the adders required. Further, for systematic (**n, k, m**) codes with **K > n – k**, encoding schemes that normally require fewer than **K**-shift registers exist as illustrated in the following simple example.

Example 5.9:

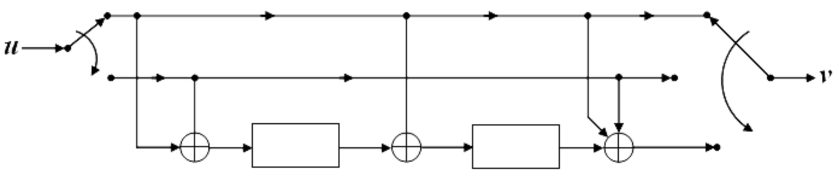
Consider a systematic (**3, 2, 2**) code with the transfer function matrix

$$G(X) = \begin{bmatrix} 1 & 0 & 1+X+X^2 \\ 0 & 1 & 1+X^2 \end{bmatrix}$$

The straight forward realization requires a total of $\mathbf{K{=}K_1{+}K_2{=}2{+}2{=}4}$ shift registers and is shown in Fig 5.7(a). However, since the parity sequences are generated by: $\mathbf{v^{(3)}(X) = u^{(1)}(X).\ g_1^{(3)}(X) + u^{(2)}(X)\ g_2^{(3)}(X)}$, an alternative realization as shown in Fig 5.7(b) can be obtained.



(a) Straight forward Realization



(b) Alternative Realization

Fig 5.7 Realization of encoder for example 5.9

In majority of situations, the straight forward realization is the most efficient. However, in the case of systematic codes simpler realizations usually exist as shown in Example 5.9.

Another advantage of' systematic codes is that no inverting circuit is needed for recovering information from the code word. For information recovery from a non systematic code, inversion is required in the form of an (**n** × **k**) matrix $\mathbf{G^{-1}(X)}$ such that

$$\mathbf{G(X).G^{-1}(X) = I_k X^l} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (5.27)$$

for some $\mathbf{l \geq 0}$ and $\mathbf{I_k}$ is the $\mathbf{(k \times k)}$ unit matrix. Then it follows that:

$$\mathbf{V(X).G^{-1}(X) = U(X)\ G(X)\ G^{-1}(X) = U(X).X^l} \quad \ldots\ldots\ldots\ldots\ldots \quad (5.28)$$

and the information sequence can be recovered with an **l**-time unit delay from the code word by letting **V(X)** to be the input to the **n-input, k-output** linear sequential circuit whose transfer function matrix **is** $\mathbf{G^{-1}(X)}$.

For an **(n, l, m)** code the transfer function matrix **G(X)** will have a "feed forward" inverse $\mathbf{G^{-1}(X)}$ of delay **l** units if and only if :

$$\mathbf{G.C.D\ [g^{(1)}(X),\ g^{(2)}(X)\ \ldots g^{(n)}(X)] = X^l} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (5.29)$$

for some $\mathbf{l \geq 0}$; where **G.C.D** denotes the '**greatest common divisor**'. For an (**n, k, m**) code with **k > 1**, let $\mathbf{\Delta_i(X)}$, **i=1, 2** … $\binom{n}{k}$ be the determinants of the $\binom{n}{k}$ distinct **k × k** sub-matrices of the transfer function matrix **G(X)**.Then a feed forward inverse of delay **l**-units exists if an only if:

$$\mathbf{GGD\ [\Delta_i(X),}\ \ i=1, 2 \ldots \binom{n}{k}\ \mathbf{] = X^l} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (5.3$$

Example 5.10:

For the (**2, 1, 3**) encoder of Fig 5.3, we have, from Example 8.5, the generator matrix as

$$\mathbf{G(X) = [1 + X^2 + X^3, 1 + X + X^2 + X^3)}$$

Its inverse can be computed as:

$$G^{-1}(X) = \begin{bmatrix} 1 + X + X^2 \\ X + X^2 \end{bmatrix}$$

and the implementation of the inverse is shown in Fig 5.8.

Example 5.11:

For the (**3, 2, 1**) encoder of Fig 5.4, the generator matrix as found in Example 5.6 is:

$$G(X) = \begin{bmatrix} 1 + X & 1 & 1 \\ X & 1 + X & 0 \end{bmatrix}$$

The determinants of the (**2 × 2**) sub matrices are **1+ X + X^2, X** and **1 + X**. Their **GCD** is 1. A feed forward inverse with no delay exists and can be computed as:

$$G^{-1}(X) = \begin{bmatrix} 1 + X & X \\ X & 1 + X \\ X + X^2 & 1 + X^2 \end{bmatrix} \qquad \text{I}$$

Implementation of this inverse is shown in Fig 5.9.

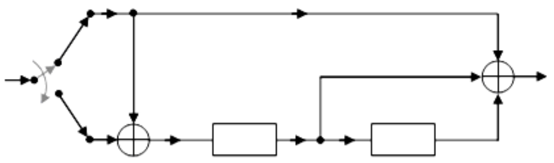


Fig 5.8 Feed forward encoder of (2,1,3) code

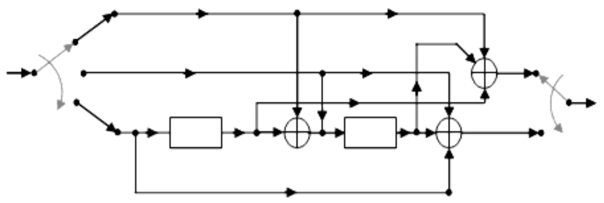


Fig 5.9 Feed forward encoder of (3,2,1) code

To understand what happens when a feed forward inverse does not exist consider an example of a (**2, 1, 2**) encoder with generator matrix

$$\mathbf{G(X) = [1+X, 1 + X^2]}$$

Since the **GCD** of **g$^{(1)}$ (X)** and **g$^{(2)}$ (X)** is (**1+ X**) (not of the form **X^l**), a feed forward inverse does not exist. Suppose the input sequence is:

$u(X)=\frac{1}{1+X}=1+X^2+X^3+...$ Then the output sequences are: **$v^{(1)}(X) = 1$, $v^{(2)}(X) = 1 + X$.**

That is, the code word contains only three nonzero bits even though the information sequence has infinite weight. If this code word is transmitted over a BSC and the three nonzero bits are changed to zeros by the channel noise, the received sequence will be all zeros. A maximum likely hood decoder (**MLD**) will then produce the all-zero code word as its estimate, since this a valid code word and it agrees exactly with the received sequence. Thus, the estimated information sequence will be $\hat{u}(X)=0$, implying an infinite number of decoding errors caused by a finite number (only three in this case) of channel errors. Clearly this is a very undesirable situation and the code is said to be subject to "**Catastrophic error propagation**" and the code is called a "**catastrophic code**".

Equations (5.29) and (5.30) can be shown to be necessary and sufficient conditions for a code to be '**non-catastrophic**'. Hence any code for which a feed forward inverse exists is non-catastrophic. **Another advantage of systematic codes is that they are always non-catastrophic**.

8.5 STATE DIAGRAMS:

The state of an encoder is defined as its shift register contents. For an (**n, k, m**) code with **k > 1, i**-th shift register contains '**K**$_i$' previous information bits. Defining $K=\sum_{i=1}^{k}K_i$ as the total encoder - memory (**m** - represents the memory order which we have defined as the maximum length of any shift register), the encoder state at time unit **T**', when the encoder inputs are, $\{u_l^{(1)}, u_l^{(2)} \ldots u_l^{(k)}\}$, are the binary **k**-tuple of inputs:

$\{u_{l-1}^{(1)}\ u_{l-2}^{(1)}, u_{l-3}^{(1)} \ldots u_{l-k}^{(1)}; u_{l-1}^{(2)}, u_{l-2}^{(2}, u_{l-3}^{(2)} \ldots u_{l-k}^{(2)}; \ldots ; u_{l-1}^{(k)}\ u_{l-2}^{(k)}, u_{l-3}^{(k)} \ldots u_{l-k}^{(k)}\}$,

and there are a total of 2^k different possible states. For a (**n, 1, m**) code, **K = K$_1$ = m** and the encoder state at time unit **l** is simply $\{u_{l-1}, u_{l-2} \ldots u_{l-m}\}$.

Each new block of **k**-inputs causes a transition to a new state. Hence there are 2^k branches leaving each state, one each corresponding to the input block. For an (**n, 1, m**) code there are only two branches leaving each state. On the state diagram, each branch is labeled with the **k**-inputs causing the transition and the **n**-corresponding outputs. The state diagram for the convolutional encoder of Fig 8.3 is shown in Fig 8.10. A **state table** would be, often, more helpful while drawing the state diagram and is as shown.

State table for the (2, 1, 3) encoder of Fig 5.3

State	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Binary Description	000	100	010	110	001	101	011	111

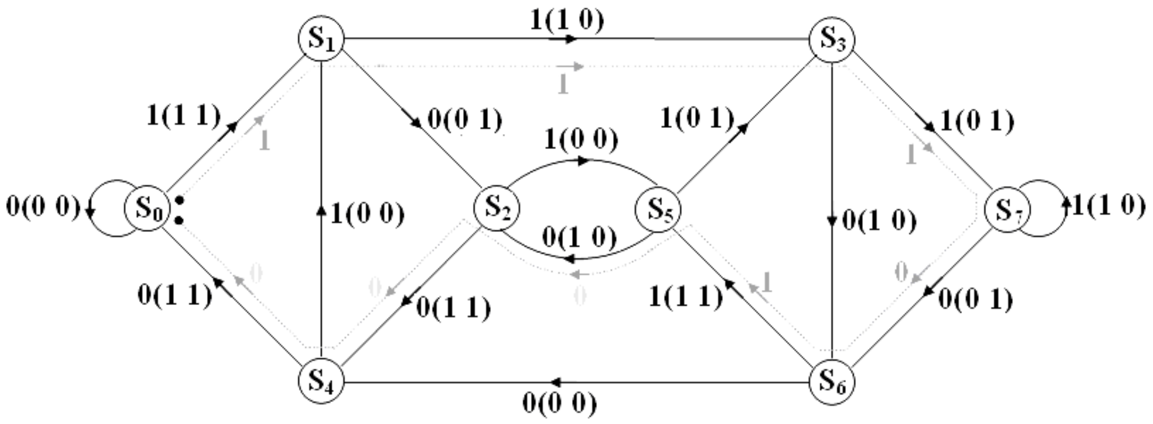


Fig 5.10 State diagram of encoder of Fig 5.3

Recall (or observe from Fig 8.3) that the two out sequences are:

$$\mathbf{v}^{(1)} = \mathbf{u}_l + \mathbf{u}_{l-2} + \mathbf{u}_{l-3} \quad \text{and}$$
$$\mathbf{v}^{(2)} = \mathbf{u}_l + \mathbf{u}_{l-1} + \mathbf{u}_{l-2} + \mathbf{u}_{l-3}$$

Till the reader, gains some experience, it is advisable to first prepare a transition table using the output equations and then translate the data on to the state diagram. Such a table is as shown below:

State transition table for the encoder of Fig 5.3

Previous State	Binary Description	Input	Next State	Binary Description	u_l u_{l-1} u_{l-2} u_{l-3}	Output
S_0	0 0 0	0	S_0	0 0 0	0 0 0 0	0 0
		1	S_1	1 0 0	1 0 0 0	1 1
S_1	1 0 0	0	S_2	0 1 0	0 1 0 0	0 1
		1	S_3	1 1 0	1 1 0 0	1 0
S_2	0 1 0	0	S_4	0 0 1	0 0 1 0	1 1
		1	S_5	1 0 1	1 0 1 0	0 0
S_3	1 1 0	0	S_6	0 1 1	0 1 1 0	1 0
		1	S_7	1 1 1	1 1 1 0	0 1
S_4	0 0 1	0	S_0	0 0 0	0 0 0 1	1 1
		1	S_1	1 0 0	1 0 0 1	0 0
S_5	1 0 1	0	S_2	0 1 0	0 1 0 1	1 0
		1	S_3	1 1 0	1 1 0 1	0 1
S_6	0 1 1	0	S_4	0 0 1	0 0 1 1	0 0
		1	S_5	1 0 1	1 0 1 1	1 1
S_7	1 1 1	0	S_6	0 1 1	0 1 1 1	0 1
		1	S_7	1 1 1	1 1 1 1	1 0

For example, if the shift registers were in state $\mathbf{S_5}$, whose binary description is **101**, an input '**1**' causes this state to change over to the new state $\mathbf{S_3}$ whose binary description is **110** while producing an output (**0 1**). Observe that the inputs causing the transition are shown first, followed by the corresponding output sequences shown with in parenthesis.

Assuming that the shift registers are initially in the state $\mathbf{S_0}$ (the all zero state) the code word corresponding to any 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 block, the encoder is returned to state $\mathbf{S_0}$ by a sequence of **m**-all-zero block appended to the information sequence. For example, in Fig 5.10, if **u = (11101),** the code word is **v = (11, 10, 01, 01, 11, 10, 11, 10)** the path followed is shown in thin gray lines with arrows and the input bit written along in thin gray. The **m = 3** zeros appended are indicated in gray which is much lighter compared to the information bits.

Apart from obtaining the output sequence for a given input sequence, the state diagram can be modified to provide a complete description of the Hamming weights of all nonzero code words. (That is, the state diagram is useful in determining a weight- distribution for the code).

This is achieved as follows: The state $\mathbf{S_0}$ is split into an initial and a final state. The self loop around $\mathbf{S_0}$ is discarded. Each branch is labeled with a '**branch gain**', '$\mathbf{X^{i}}$', where '**i**' is the weight (number of ones) of the **n**-encoded bits on that branch. Each path that connects the initial state to the final state which diverges from and remerges with state $\mathbf{S_0}$**,** exactly once, represents a nonzero code word.

Those code words that diverge from and remerge with $\mathbf{S_0}$ more than once can be regarded 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. As an lustration, let us consider the modified state diagram for the (**2, 1, 3**) code of Fig 5.3 as shown in Fig 5.11 and another version of the same as shown in Fig 5.12.

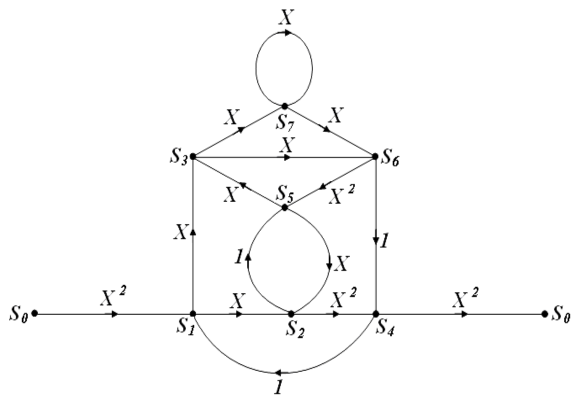


Fig 5.11 Modified state diagram for the (2,1,3) code

Example 5.12: A (2, 1, 2) Convolutional Encoder:

Consider the encoder shown in Fig 5.15. **We shall use this example for discussing further graphical representations viz. Trees, and Trellis.**

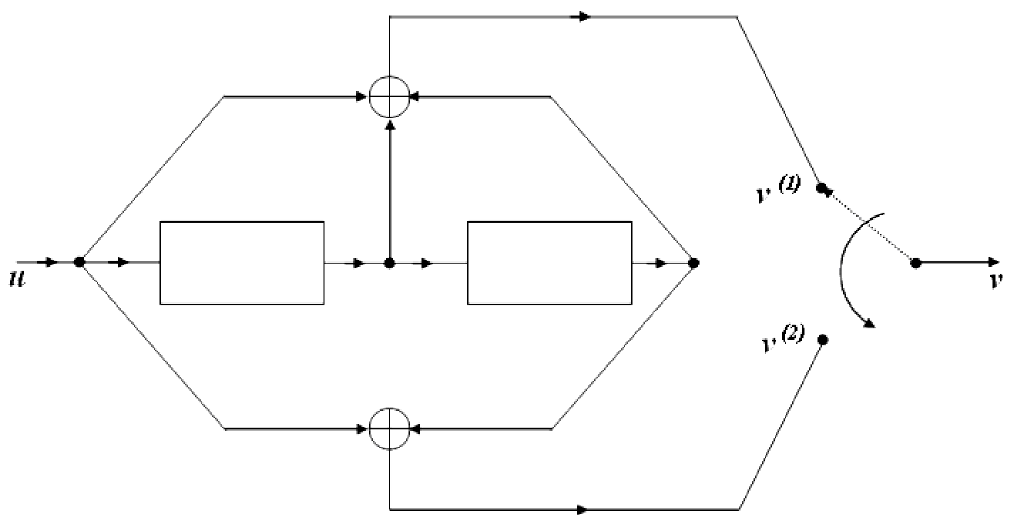


Fig 5.15 (2,1,2) convolution encoder

For this encoder we have: $\mathbf{v}_l^{(1)} = \mathbf{u}_l + \mathbf{u}_{l-1} + \mathbf{u}_{l-2}$ and $\mathbf{v}_l^{(2)} = \mathbf{u}_l + \mathbf{u}_{l-2}$
The state transition table is as follows.

State transition table for the (2, 1, 2) convolutional encoder of Example 5.12

Previous state	**Binary description**	**Input**	**Next State**	**Binary description**	$\mathbf{u}_l$ $\mathbf{u}_{l-1}$ $\mathbf{u}_{l-2}$	**Output**
S0	**0 0**	**0**	**S0**	**0 0**	**0 0 0**	**0 0**
		1	**S1**	**1 0**	**1 0 0**	**1 1**
S1	**1 0**	**0**	**S2**	**0 1**	**0 1 0**	**1 0**
		1	**S3**	**1 1**	**1 1 0**	**0 1**
S2	**0 1**	**0**	**S0**	**0 0**	**0 0 1**	**1 1**
		1	**S1**	**1 0**	**1 0 1**	**0 0**
S3	**1 1**	**0**	**S2**	**0 1**	**0 1 1**	**0 1**
		1	**S3**	**1 1**	**1 1 1**	**1 0**

The state diagram and the augmented state diagram for computing the 'complete path enumerator function' for the encoder are shown in Fig 5.16.

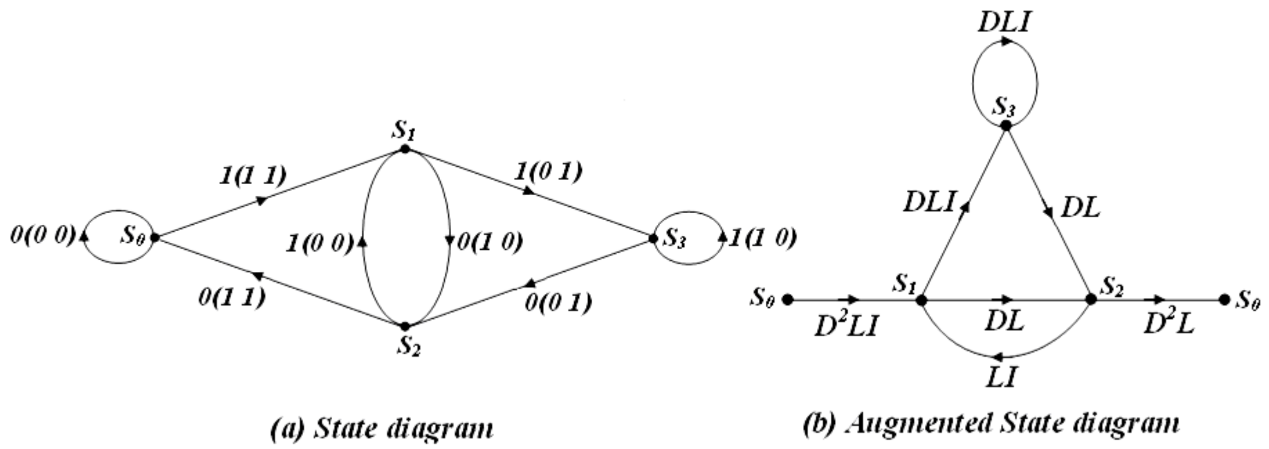


Fig 5.16 State diagram for the (2,1,2) encoder

There are three loops in the augmented state diagram:

$\mathbf{S_3 \rightarrow S_3: l_1 = DLI, S_1 \rightarrow S_2 \rightarrow S_1: l_2 = DL^2I, S_1 \rightarrow S_3 \rightarrow S_2 \rightarrow S1{:}l3 = D^2L^3I^2}$

The loops $\mathbf{l_1}$ and $\mathbf{l_2}$ are non-touching and their gain product is: $\mathbf{l_1 \times l_2 = D^2L^3I^2}$

$$\therefore \Delta = 1 - (l_1 + l_2 + l_3) - l_1 l_2$$

$$= 1 - DLI\,(1 + L)$$

There are two forward paths: $\mathbf{F_1 \Rightarrow S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_0}$. **Path gain =** $\mathbf{D^5L^3I}$

$\mathbf{F_2 \Rightarrow S_0 \rightarrow S_1 \rightarrow S_3 \rightarrow S_2 \rightarrow S_0}$, **Path gain =** $\mathbf{D^6L^4I^2}$

The loop $\mathbf{l_1}$ does not touch the forward path $\mathbf{F_1}$. $\therefore \mathbf{\Delta_1 = 1 - l_1 = 1 - DLI}$.

All the three loops touch the forward path $\mathbf{F_2}$. $\therefore \mathbf{\Delta_2 = 1}$

Now use Mason's gain formula to get:

$$T(D,L,I) = \frac{D^5L^3I(1-DLI) + D^6L^4I^2}{1 - DLI(1+L)} = \frac{D^5L^3I}{1-DLI(1+L)}$$
$$= D^5L^3I + D^6L^4I^2(1+L) + D^7L^5I^3(1+L)^2 + \ldots$$

Thus there is one code word of weight **5** that has length **3** branches and an information sequence of weight **1**, Two code words of weight **6**, of which one has length **4** branches and an information sequence of weight **2**, and the other has length **5** branches and an information sequence of weight **2** and so on.

8.6 TREE AND TRELLIS DIAGRAMS:

Let us now consider other graphical means of portraying convolutional codes. The state diagram can be re-drawn as a 'Tree graph'. The convention followed is: If the input is a '**0**', then the upper path is followed and if the input is a '**1**', then the lower path is followed. A vertical line is called a '**Node**' and a horizontal line is called '**Branch**'. The output code words for each input bit are shown on the branches. The encoder output for any information sequence can be traced through the tree paths. The tree graph for the (**2, 1, 2**) encoder of Fig 5.15 is shown in Fig 5.18. The state transition table can be conveniently used in constructing the tree graph.

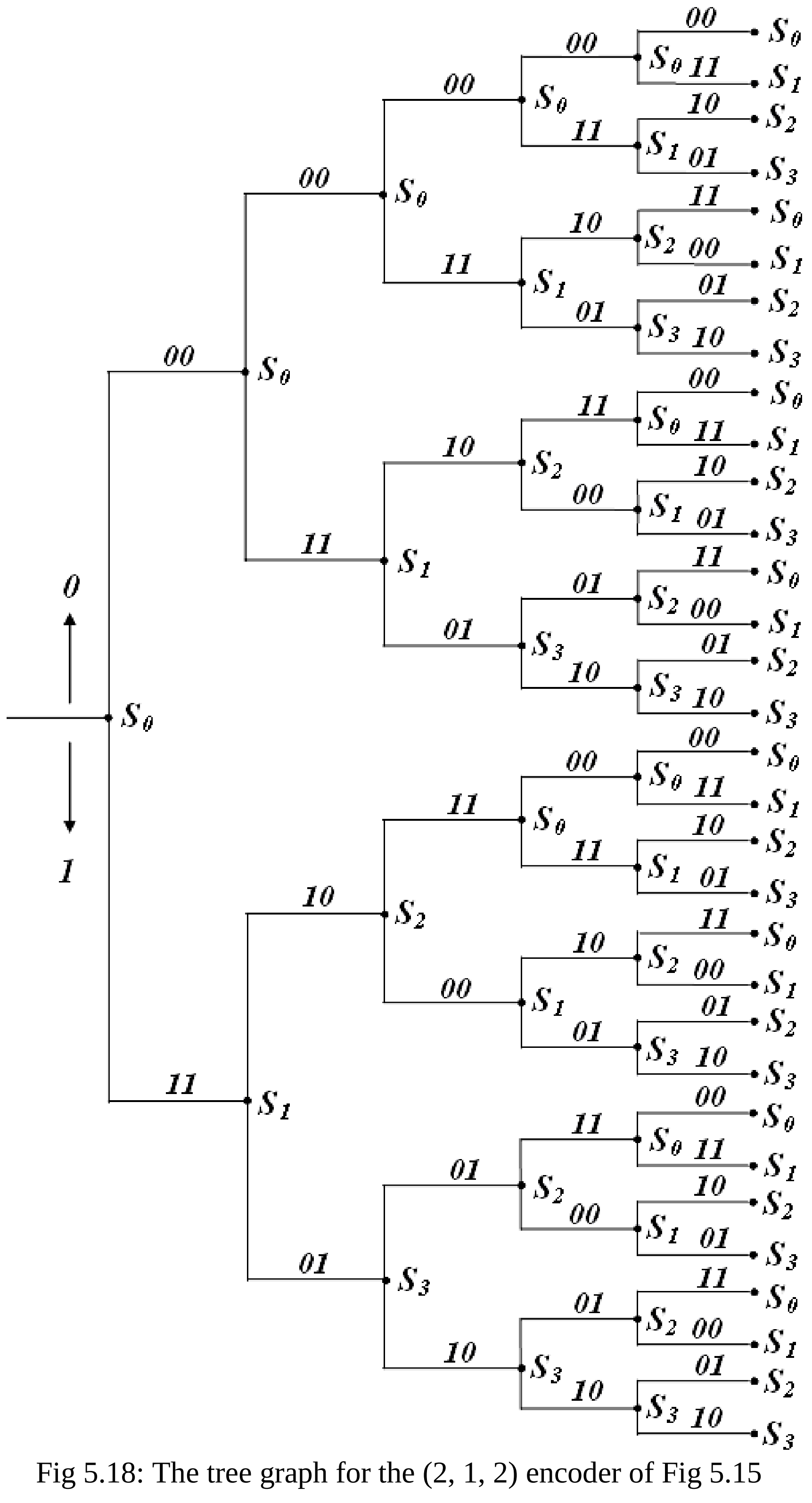


Fig 5.18: The tree graph for the (2, 1, 2) encoder of Fig 5.15

Following the procedure just described we find that the encoded sequence for an information sequence (**10011**) is (**11, 10, 11, 11, 01**) which agrees with the first **5** pairs of bits of the actual encoded sequence. Since the encoder has a memory = **2** we require two more bits to clear and re-set the encoder. Hence to obtain the complete code sequence corresponding to an information sequence of length **kL**, the tree graph is to extended by **n(m-l)** time units and this extended part is called the

"**Tail of the tree**", and the **2kL** right most nodes are called the "**Terminal nodes**" of the tree. Thus the extended tree diagram for the (**2, 1, 2**) encoder, for the information sequence (**10011**) is as in Fig 5.19 and the complete encoded sequence is (**11, 10, 11, 11, 01, 01, 11**).

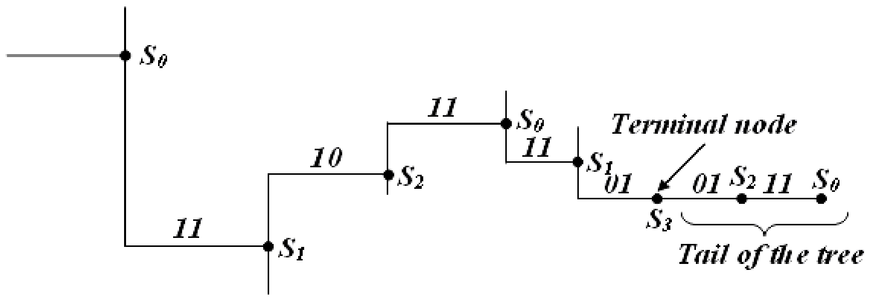
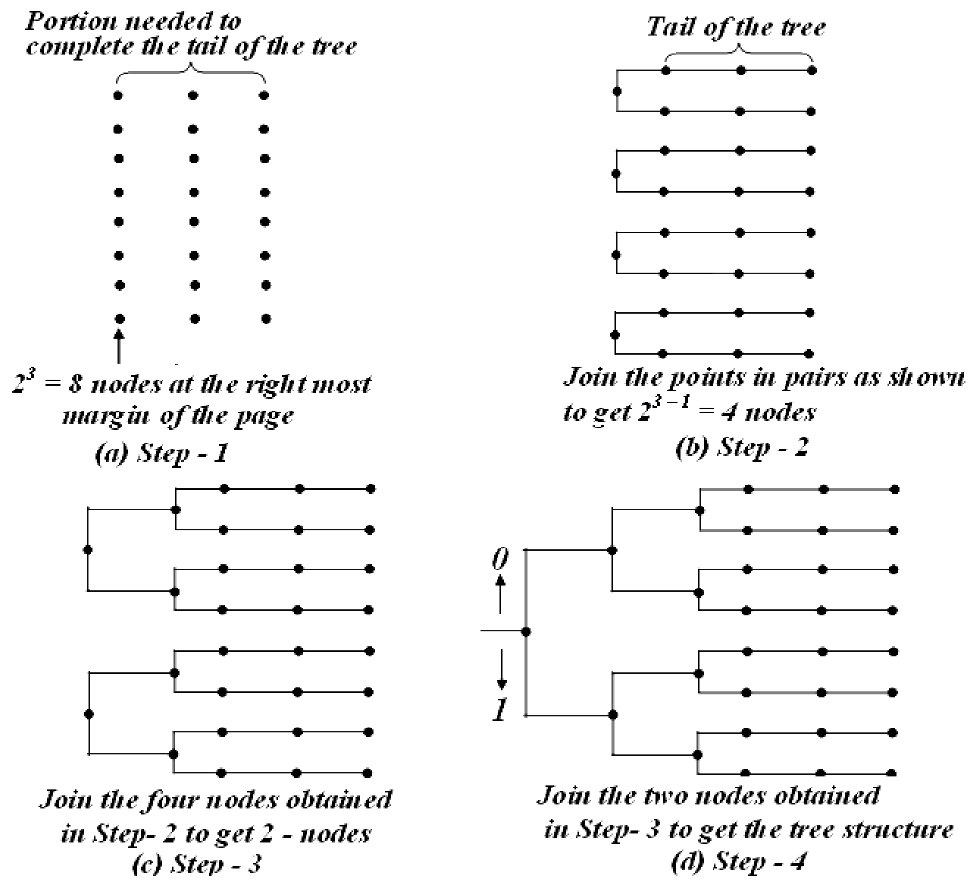


Fig 5.19 Illustration of the "Tail of the tree"

At this juncture, a very important clue for the student in drawing tree diagrams neatly and correctly, without wasting time appears pertinent. As the length of the input sequence **L** increases the number of right most nodes increase as $\mathbf{2^L}$. Hence for a specified sequence length, **L**, compute $\mathbf{2^L}$. Mark $\mathbf{2^L}$ equally spaced points at the rightmost portion of your page, leaving space to complete the **m** tail branches. Join two points at a time to obtain $\mathbf{2^{L-1}}$ nodes. Repeat the procedure until you get only one node at the left most portion of your page. The procedure is illustrated diagrammatically in Fig 5.20 for **L = 3.** Once you get the tree structure, now you can fill in the needed information either looking back to the state transition table or working out logically.



Portion needed to complete the tail of the tree
$2^3 = 8$ nodes at the right most margin of the page
(a) Step - 1
Tail of the tree
Join the points in pairs as shown to get $2^{3-1} = 4$ nodes
(b) Step - 2
Join the four nodes obtained in Step- 2 to get 2 - nodes
(c) Step - 3
0
1
Join the two nodes obtained in Step- 3 to get the tree structure
(d) Step - 4

Fig 5.20 Procedure for drawing neat tree diagram

From Fig 5.18, observe that the tree becomes "**repetitive**' after the first three branches.

Beyond the third branch, the nodes labeled **S_0** are identical and so are all the other pairs of nodes that are identically labeled. Since the encoder has a memory **m = 2,** it follows that when the third information bit enters the encoder, the first message bit is shifted out of the register. Consequently, after the third branch the information sequences (**$000u_3u_4$---**) and (**$100u_3u_4$---**) generate the same code symbols and the pair of nodes labeled **S_0** may be joined together. The same logic holds for the other nodes.

Accordingly, we may collapse the tree graph of Fig 5.18 into a new form of Fig 5.21 called a "**Trellis**". It is so called because Trellis is a tree like structure with re-merging branches (You will have seen the trusses and trellis used in building construction).

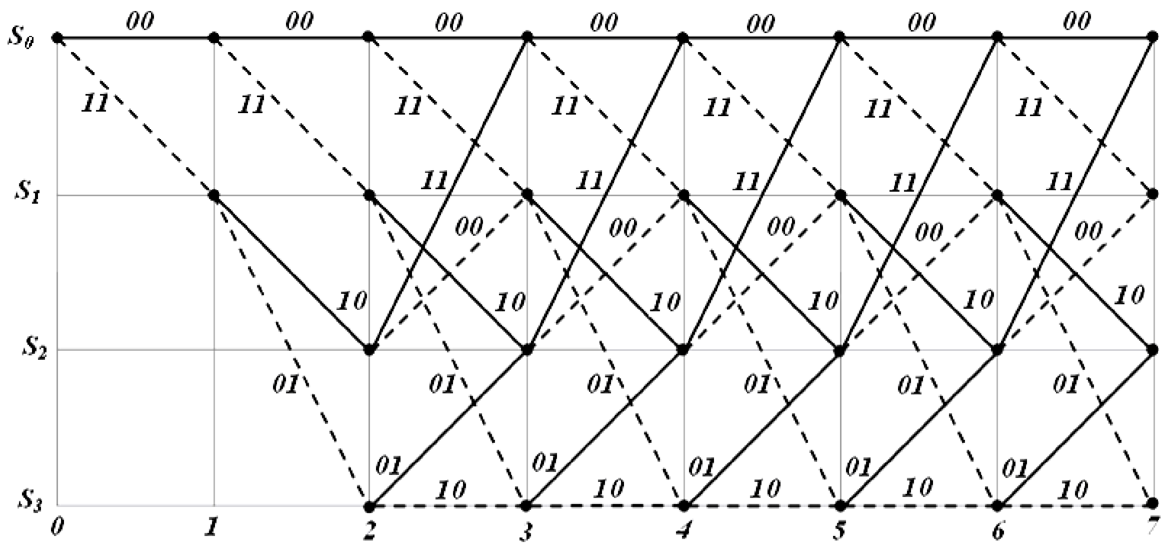


Fig 5.21 Trellis diagram for encoder of fig 5.15

The Trellis diagram contain (**L + m + 1**) time units or levels (or depth) and these are labeled from **0** to (**L + m**) (**0 to 7** for the case with **L = 5** for encoder of Fig 5.15 as shown in Fig 5.21.
The following observations can be made from the Trellis diagram

1. There are no fundamental paths at distance **1**, **2** or **3** from the all zero path.

2. There is a single fundamental path at distance **5** from the all zero path. It diverges from the all-zero path three branches back and it differs from the all-zero path in the single input bit.

3. There are two fundamental paths at a distance **6** from the all zero path. One path diverges from the all zero path four branches back and the other five branches back. Both paths differ from the all zero path in two input bits. The above observations are depicted in Fig 8.24(a).

4. There are four fundamental paths at a distance **7** from the all-zero path. One path diverges from the all zero path five branches back, two other paths six branches back and the fourth path diverges seven branches back as shown in Fig 8.24(b). They all differ from the all zero path in three input bits. This information can be compared with those obtained from the complete path enumerator function found earlier.

8.7 THE VITERBI ALGORITHM:

The Viterbi algorithm, when applied to the received sequence **r** from a **DMC** finds the path through the trellis with the largest metric. At each step, it compares the metrics of all paths entering each state and stores the path with the largest metric called the "**survivor**" together with its metric.

The Algorithm:

Step: 1. Starting at level (i.e. time unit) **j = m**, compute the partial metric for the single path entering each node (state). Store the path (the survivor) and its metric for each state.

Step: 2. Increment the level **j** by **1**. Compute the partial metric for all the paths entering a state by adding the branch metric entering that state to the metric of the connecting survivor at the preceding time unit. For each state, store the path with the largest metric (the survivor), together with its metric and eliminate all other paths.

Step: 3. If **j < (L + m)**, repeat **Step 2**. Otherwise stop.

Notice that although we can use the Tree graph for the above decoding, the number of nodes at any level of the Trellis does not continue to grow as the number of incoming message bits increases, instead it remains a constant at $\mathbf{2^m}$.

There are $\mathbf{2^k}$ survivors from time unit '**m**' up to time unit **L**, one for each of the $\mathbf{2^k}$states. After **L** time units there are fewer survivors, since there are fewer states while the encoder is returning to the all-zero state. Finally, at time unit (**L + m**) there is only one state, the all-zero state and hence only one survivor and the algorithm terminates.

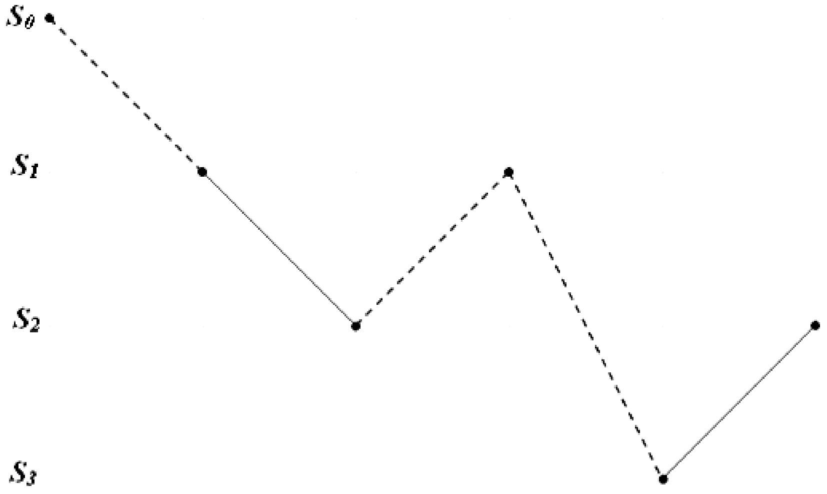


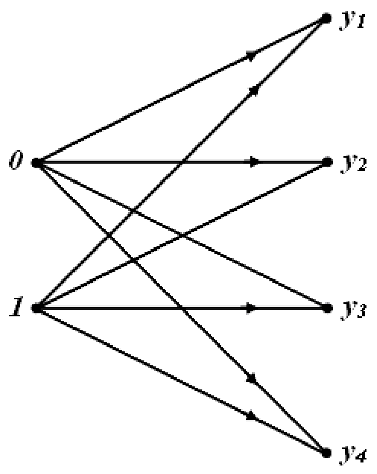
Fig 5.22 Survivor after time unit 'j'

Suppose that the maximum likely hood path is eliminated by the algorithm at time unit **j** as shown in Fig 8.22. This implies that the partial path metric of the survivor exceeds that of the maximum likely hood path at this point. Now, if the remaining portion of the maximum likely hood path is appended onto the survivor at time unit **j**, then the total metric of this path will exceed the total metric of the maximum likely hood path. But this contradicts the definition of the 'maximum likely hood path' as the 'path with largest metric'. Hence the maximum likely hood path cannot be eliminated by the algorithm and it must be the final survivor and it follows

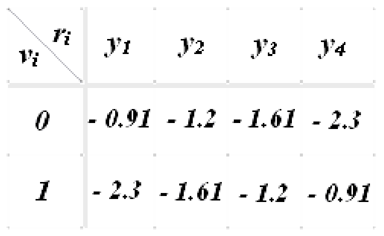
$M(r \mid \hat{v}) \geq M(r \mid v), \; \forall\, v \neq \hat{v}$.Thus it is clear that the Viterbi algorithm is optimum in the sense that it always finds the maximum likely hood path through the Trellis. From an implementation point of view, however, it would be very inconvenient to deal with fractional numbers. Accordingly, the bit metric **M ($r_i|v_i$) = ln P ($r_i|v_i$)** can be replaced by "**C_2 [ln P ($r_i|v_i$) + C_1]**", **C_1** is any real number and **C_2** is any positive real number so that the metric can be expressed as an integer. Notice that a path **v** which maximizes $M(r \mid v) = \sum_{i=1}^{N} M(r_i \mid v_i) = \sum_{i=1}^{N} ln\,P(r_i \mid v_i)$ also maximizes $\sum_{i=1}^{N} C_2\big[ln\,P(r_i \mid v_i) + C_1\big]$. Therefore, it is clear that the modified metrics can be used without affecting the performance of the Viterbi algorithm. Observe that we can always choose **C_1** to make the smallest metric as zero and then **C_2** can be chosen so that all other metrics can be approximated by nearest integers. Accordingly, there can be many sets of integer metrics possible for a given **DMC** depending on the choice of **C_2**. The performance of the Viterbi algorithm now becomes slightly sub-optimal due to the use of modified metrics, approximated by nearest integers. However the degradation in performance is typically very low.

Example 5.13:

As an illustration let us consider a binary input-quaternary output **DMC** shown in Fig 5.23(a). The bit metrics **ln P (r_i| v_i)** are shown in Fig 5.23(b). Choosing **C_1**= − 2.3 and **C_2** = 7.195 yields the "integer metric table" shown in Fig 5.23(c).

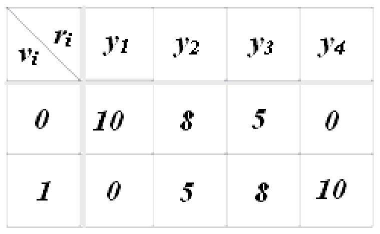


(a) Binary input – Quaternary output DMC



v_i \ r_i	y_1	y_2	y_3	y_4
0	- 0.91	- 1.2	- 1.61	- 2.3
1	- 2.3	- 1.61	- 1.2	- 0.91

(b) Metric Table



v_i \ r_i	y_1	y_2	y_3	y_4
0	10	8	5	0
1	0	5	8	10

(c) Integer Metric Table

Fig 5.23 Diagran for example 5.13

Now suppose that a code word from the (**2,1,2**) encoder of Fig 5.15, whose Trellis diagram is shown in Fig 5.21, is transmitted over the **DMC** of Fig 8.26 and the quaternary received sequence is:

r = {$y_3\, y_4,\; y_3\, y_1,\; y_3\, y_2,\; y_3\, y_4,\; y_3\, y_4,\; y_2\, y_4,\; y_1\, y_3$}

Let us apply Viterbi algorithm to determine the transmitted sequence.

In the first time unit (**j = 1**) there are two branches originating from the state **S_0** with output vectors (**00**) terminating at **S_0** and (**11**) terminating at **S_1**. The received sequence in this time unit is (**$y_3\, y_4$**) and using the integer metric table of Fig 8.23(c) we have:

$M[r_1|v_1^{(1)}] = M(y_3\,y_4|00) = M(y_3|0) + M(Y_4|0) = 5 + 0 = 5$, and

$M[r_1|v_1^{(2)}] = M(y_3\,y_4|11) = M(y_3|1) + M(Y_4|1) = 8 + 10 = 18$

These computations are indicated in Fig 8.24(a). The path discarded is shown by a cross. Note that the branch metrics are also indicated along the branches with in brackets and the state metrics are indicated at the nodes.

For **j = 2** there are single branches entering each state and the received sequence in this time unit is (**$y_3\,y_1$**). The four branch metrics are computed as below.

$M_1 = M(y_3\,y_1|00) = M(y_3|0) + M(y_1|0) = 5 + 10 = 15$

$M_2 = M(y_3\,y_1|11) = M(y_3|1) + M(y_1|1) = 8 + 0 = 8$

$M_3 = M(y_3\,y_1|10) = M(y_3|1) + M(y_1|0) = 8 + 10 = 18$

$M_4 = M(y_3\,y_1|01) = M(y_3|0) + M(y_1|1) = 5 + 0 = 5$

The metrics at the four states are obtained by adding the branch metrics to the metrics of the previous states (survivors) and are shown in Fig 8.24(b).

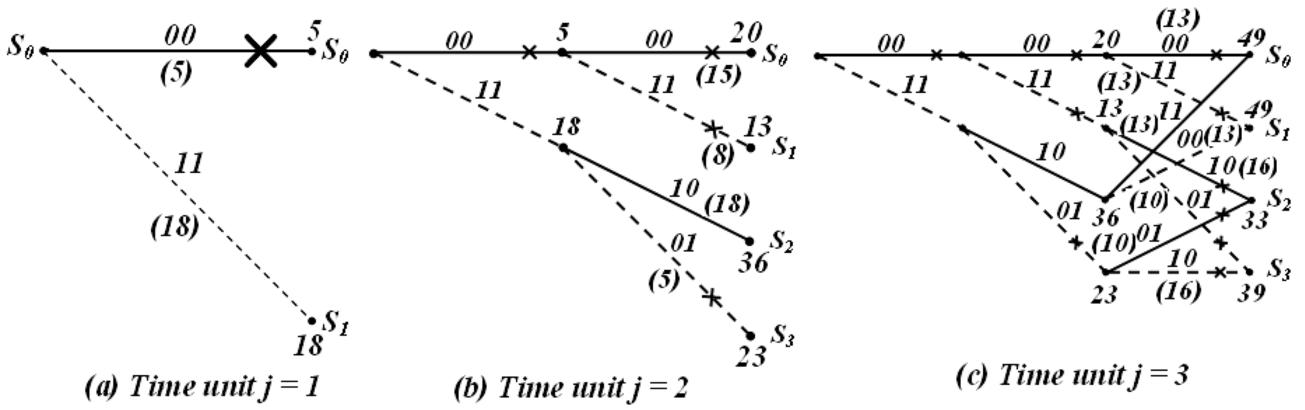


Fig 5.24 Computation for time units j=1, j=2 and j=3

Next for **j = 3**, notice that there are two branches entering each state as shown in Fig 8.24(c). The received sequence in this time unit is (**y_3, y_2**) and the branch metrics are computed as below:

$M_1 = M(y_3\,y_2|00) = M(y_3|0) + M(y_2|0) = 5 + 8 = 13$

$M_2 = M(y_3\,y_2|11) = M(y_3|1) + M(y_2|1) = 8 + 5 = 13$

$M_3 = M(y_3\,y_2|10) = M(y_3|1) + M(y_2|0) = 8 + 8 = 16$

$M_4 = M(y_3\,y_2|01) = M(y_3|0) + M(y_2|1) = 5 + 5 = 10$

Following the above steps, we arrive at the following diagram.

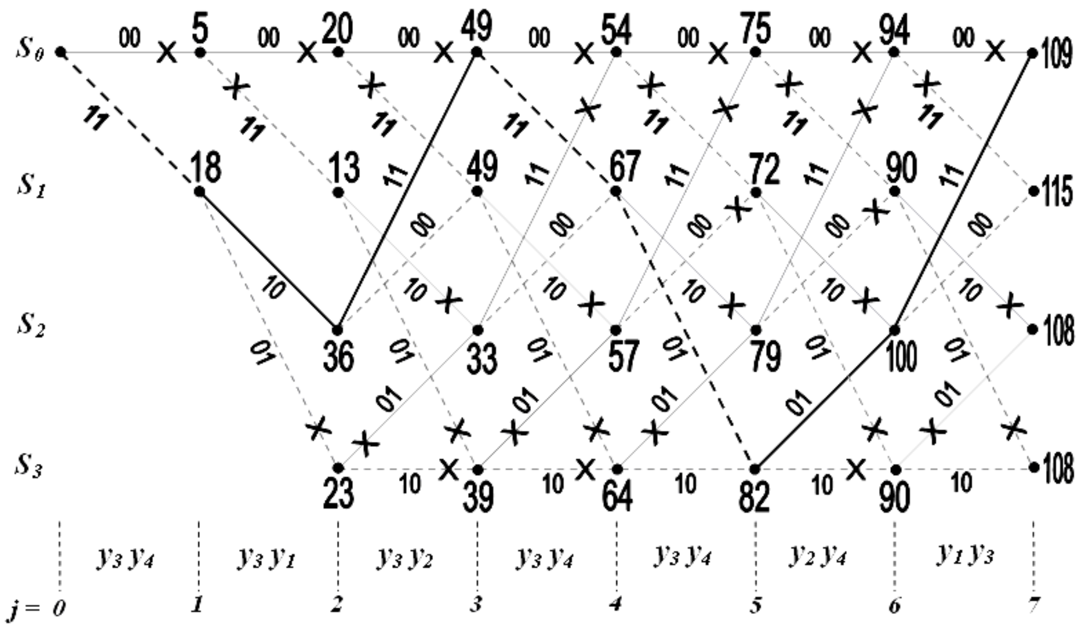


Fig 5.25 Application of Viterbi algorithm

Notice that, in the last step we have ignored the highest metric computed! Indeed, if the sequence had continued we should take this into account. However, **in the last m-time units remember that the path must remerge with S_0.**

From the path that has survived, we observe that the transmitted sequence is:

$\hat{v}$ = **(11, 10, 11, 11, 01, 01, 11)**

and the information sequence at the encoder input is: $\hat{u}$ **= (1 0 0 1 1)**

Notice that **"the final m-branches in any trellis path always corresponds to '0' inputs and hence not considered part of the information sequence"**.

As already mentioned, the **MLD** reduces to a **'minimum distance decoder'** for a **BSC** (see Eq 8.40). Hence the distances can be reckoned as metrics and the algorithm must now find the path through the trellis with the smallest metric (i.e. the path closest to **r** in Hamming distance). The details of the algorithm are exactly the same, except that the Hamming distance replaces the log likely hood function as the metric and the survivor at each state is the path with the smallest metric. The following example illustrates the concept.

Example 5.14:

Suppose the rode word **r = (01, 10, 10, 11, 01, 01, 11)**, from the encoder of Fig 5.15 is received through a **BSC**. The path traced is shown in Fig 5.25 as dark lines.

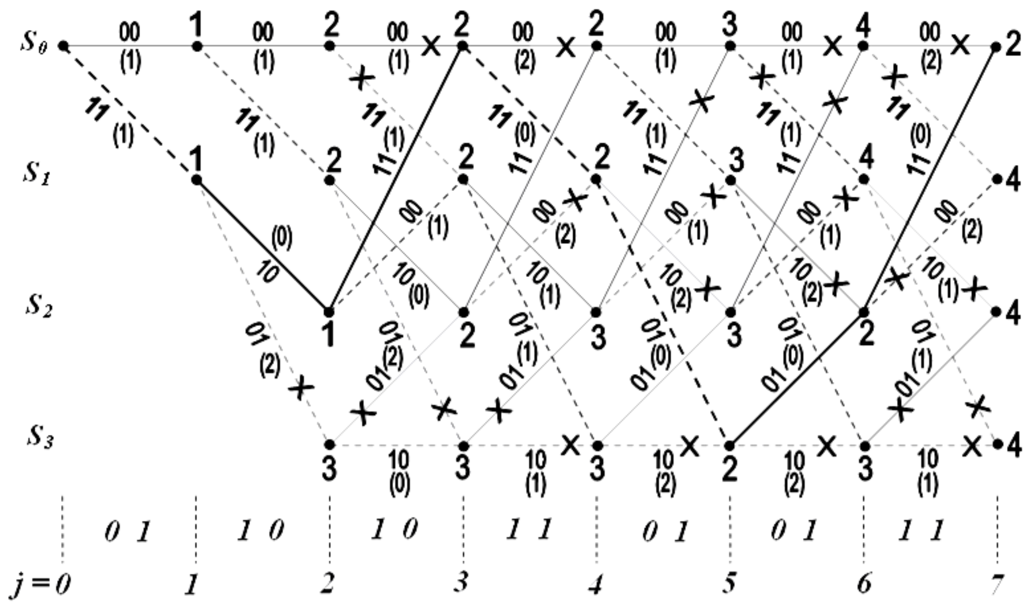


Fig 5.26 Viterbi algorithm for a BSC

The estimate of the transmitted code word is

$$\hat{v} = \mathbf{(11, 10, 11, 11, 01, 01, 11)}$$

and the corresponding information sequence is: $\hat{u}$ **= (1 0 0 1 1)**

Notice that the distances of the code words of each branch with respect to the corresponding received words are indicated in brackets. Note also that at some states neither path is crossed out indicating a tie in the metric values of the two paths entering that state. If the final survivor goes through any of these states there is more than one maximum likely hood path (i.e. there may be more than one path whose distance from **r** is a minimum). From an implementation point of view whenever a tie in metric values occur, one path is arbitrarily selected as survivor, because of the non-practicability of storing a variable number of paths. However, this arbitrary resolution of ties has no effect on the decoding error probability. Finally, the Viterbi algorithm cannot give fruitful results when more errors in the transmitted code word than permissible by the $\mathbf{d_{free}}$ of the code occur. For the example illustrated, the reader can verify that the algorithm fails if there are three errors. Discussion and details about the performance bounds, convolutional code construction, implementation of the Viterbi algorithm etc are beyond the scope of this book.

RECOMMENDATION QUESTIONS

1. Fig P 8.1 shows a convolutional encoder. i) What is the constraint length and rate efficiency? ii) Find the encoder output produced by the sequence **101101……..** iii) Is the code systematic?

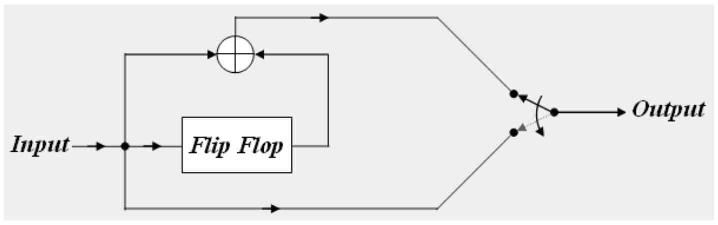


Fig P 8.1

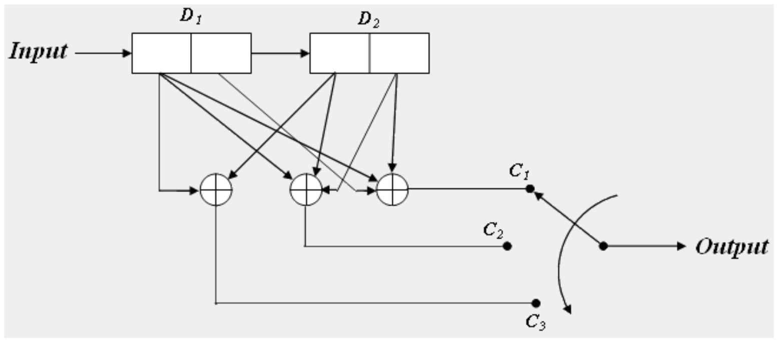


Fig P 8.2

2. Consider the convolutional encoder shown in Fig P 8.2. The message bits are shifted into the encoder two bits at a time. Repeat the questions asked in problem P 8.1.

3. A convolution encoder has a single shift register with two stages (i.e. **m=2**), three Modulo-2 adders and Output multiplexer. The generator sequences of the encoder are as follows:

$$\mathbf{g^{(1)}=(1,0,1);\ g^{(2)}=(1,1,0)\ ;\ g^{(3)}=(1,1,1)}$$

Draw the block diagram of the encoder.

4. Fig P8.4 shows the encoder of a rate **1/2**, constraint length =**4** convolutional encoder. Determine the encoder out-put produced by the information sequence (**10111…**) using the following two approaches: i) Time domain approach, based on convolution. ii) Transform domain approach.

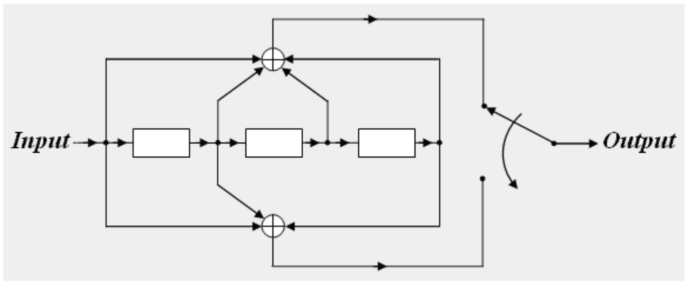


Fig P 8.4

5. For the encoder of problem 3,
   a) Find the generator matrix **G**
   b) Find the code word corresponding to the information sequence (**11101…**)

6. For the encoder of problem 3,
   a) Find the transfer function matrix **G(X)**
   b) Find the set of output sequences **V(X)** and the code word **v(X)** corresponding to the information sequence **$u(X)=1=X^2+X^3+X^4$**.

7. Determine which of the following rate **1/2** convolutional codes are 'catastrophic':
   (a) $\mathbf{g^{(1)}(X) = X^2,\ g^{(2)}(X) = 1 + X + X^3}$
   (b) $\mathbf{g^{(1)}(X) =1 + X^2 + X^4,\ g^{(2)}(X) = 1 + X + X^3+X^4}$
   (c) $\mathbf{g^{(1)}(X) =1 + X + X^2 + X^4,\ g^{(2)}(X) = 1 + X^3+X^4}$
   (d) $\mathbf{g^{(1)}(X) =1 + X^4 + X^5 + X^6,\ g^{(2)}(X) = 1 + X + X^3+X^5}$

8. Consider the encoder of problem 3,
   a) Draw the state diagram of the encoder
   b) Draw the modified state diagram
   c) Find the generating function **T(X)**
   d) Draw the augmented state diagram
   e) Find the Complete Path Enumerator function **T (D, L, I)**

8. Consider the (**3, 1, 5**) systematic code with $\mathbf{g^{(2)} = (101101)}$, $\mathbf{g^{(3)} = (110011)}$
   a) Find the generator matrix
   b) Find the parity sequences corresponding to the information sequence **u= (1101…)**

9. Consider the (**3, 2, 3**) systematic code with

$$\mathbf{g_1^{(3)}(X) = 1+X^2+X^3} \text{ and } \mathbf{g_2^{(3)}(X) = 1+X+X^3}$$

   a) Draw the straight forward realization of the encoder
   b) Draw a simple encoder realization which requires only three shift register stages.

10. Consider the (**2, 1, 2**) code with $\mathbf{G(X) = [1+X^2, 1+X+X^2]}$
   a) Find the **GCD** of its generator polynomials
   b) Find the transfer function matrix $\mathbf{G^{-1}(X)}$ of its minimum delay feed forward inverse.

11. Consider a (**2, 1, 3**) code with $\mathbf{G(X) = [1+X^2, 1+X+X^2+X^3]}$
   a) Find the **GCD** of its generator polynomials
   b) Draw the encoder state diagram.
   c) Find the infinite-weight information sequence that generates a code word of finite weight
   d) Is this code catastrophic?

12. Construct the code tree for the convolutional encoder of Fig P 8.1. Trace the path through the tree that corresponds to the information sequence (**101101…**) and compare the output with that determined in problem 21.

2. The code tree for the encoder of Fig 8.15, assuming that the incoming message sequence has **L = 2** is shown in Fig P 8.13.Validate this tree.

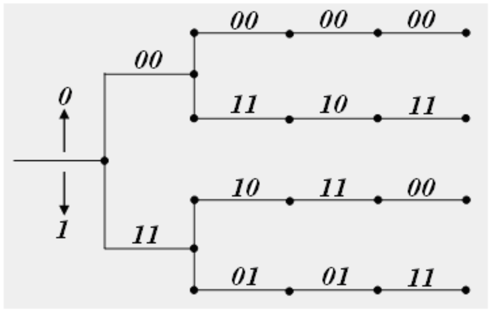


Fig P 8.13

14. Construct the code tree for the encoder of Fig P 8.4. Trace the path through the tree that corresponds to the information sequence (**10111…**). Compare the resulting encoder output with that found in problem 4.

15. Draw the state diagram and augmented state diagram for the encoder of Fig P.8.4.
    i) Show that generating function is

$$\mathbf{T(X)} = \frac{X^6 + X^7 - X^8}{1 - 2X - X^8}$$

    ii) What is the free distance of this code? How many errors it can correct?
    iii) Find the path enumerator function **T (D, L, I)**

16. Construct the Trellis diagram for the encoder of Fig P.8.4, assuming a message sequence of length **5**. Trace the path through the Trellis diagram corresponding to the message sequence (**10111…**). Compare the resulting encoder output with that found in problem 4.

17. Consider the encoder of Fig P 8.17.

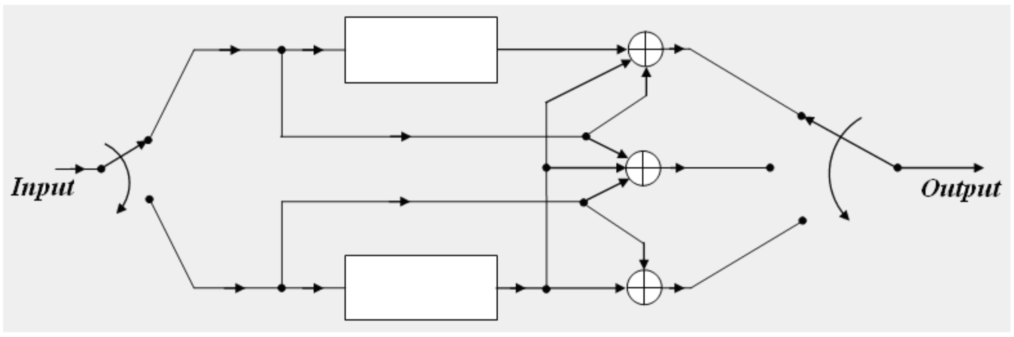


Fig P 8.17

i) What is its rate efficiency and constraint length?
ii) Using the transform domain approach, determine the code sequence produced by the information sequence (**10111…**).
iii) Construct the state diagram.
iv) Starting from the all zero state, trace the path that corresponds to the information sequence (**10111…**). Compare your answer with that in part (ii).

18. For a (**3, 1, 2**) code the transfer function matrix is: $\mathbf{G(X) = [1+X, 1+X^2, 1+X+X^2]}$

a) Draw the Trellis diagram for an information sequence of length **L = 5.**
b) Trace the path and write the code word corresponding to information sequence (**11101**).

19. Find the integer metric table for the **DMC** of Fig 8.26(a) with $\mathbf{C_1 = -2.3}$ and $\mathbf{C_2 = 3.0}$.
Decode the receiver sequence for the encoder of problem 18:
$\mathbf{r = (y_3y_4y_1,\ y_4y_4y_2,\ y_4y_4y_1,\ y_4y_4y_4,\ y_1y_3y_1,\ y_3y_2y_4,\ y_3y_1y_4)}$ using the Viterbi algorithm
Also decode the same sequence using the integer metric table of Fig 8.26(c). Compare the results.

20. Consider a binary input **8-ary** output **DMC** with transition probabilities $\mathbf{P\ (r_i|v_i)}$ given by the following table.

$v_i \backslash r_i$	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
0	*0.434*	*0.197*	*0.167*	*0.111*	*0.058*	*0.023*	*0.008*	*0.002*
1	*0.002*	*0.008*	*0.023*	*0.058*	*0.111*	*0.167*	*0.197*	*0.434*

Find the metric table and integer table for this channel.

21. Consider the (**2, 1, 3**) code with $\mathbf{G(X) = [1+X^2+X^3, 1+X+X^2+X^3]}$
    a) Draw the trellis diagram for an information sequence of length **L = 4.**
    b) Assume code vector is transmitted over the DMC of problem 20.
    Decode the received sequence: $\mathbf{r = [y_7y_8, y_7y_1, y_3y_1, y_1y_6, y_7y_2, y_3y_8, y_3y_2]}$

22. The **DMC** of problem 20 is converted to **BSC** by combining the soft decision outputs $\mathbf{y_1}$, $\mathbf{y_2}$, $\mathbf{y_3}$ and $\mathbf{y_4}$ into a single hard decision output '**0**' and combining the soft decision outputs $\mathbf{y_5}$, $\mathbf{y_6}$, $\mathbf{y_7}$ and $\mathbf{y_8}$ into a single hard decision output '**1**'. A code word from code of problem 21 is transmitted over this channel. The hard decision version of the received sequence is:

    r = [11, 10, 00, 01, 10, 01, 00]

    Decode this sequence and compare with the result of problem 21.

23. For a rate **1/3** systematic code: **g(1,1) = (1,0,1)** and **g(1,2) = (1,1,1)**
    a) Draw the tree graph, Trellis and state diagram.
    b) Find $\mathbf{d_{free}}$ and 't' for the code.
    c) For the information sequence **u = (00110100….)** find **v.**
    d) If the received vector is
    r = {001, 110, 110, 010, 100, 001, 011, 000}
    Find the transmitted **u** using Viterbi algorithm.

24. Repeat the problem 23. For a systematic code if **g (1, 1) = (1, 0, 1, 1)** and **g (1, 2) = (1, 1, 0, 1)**

25. For a non systematic rate **1/2** code given by: **g(1,1) = (1, 1,1)** and **g(1,2) = (1, 0,1,)** Repeat parts (a) and (b) of problem 23 for this code. Show, by an example, that the code corrects 2 errors in six channel bits.

26. A rate **1/3** non systematic code is given by the sub generators
    g(1,1) = (1,1,0 ,1), g(1,2) = (1,0 ,0,1) and g(3,1) = (1,1,1,0)
    a) Construct the coder
    b) Draw the tree graph, Trellis and state diagrams
    c) Find $\mathbf{d_{free}}$ and **t.**
    d) Tabulate the survivor paths and their Hamming distances for a given error vector **e = (001, 010, 101, 000)**

27. Consider the (**2, 1, 3**) code of problem 21.
    a) Draw the code tree for an information sequence of length **L= 4**.
    b) Find the code word corresponding to an information sequence **u = (1001).**

28. Consider the (**2, 1, 3**) code of problem 21.

a) For a BSC with **p =0.045**, find an integer metric table for the Fano metric.

b) Decode the received sequence: **r = (11, 00, 11, 00, 01, 10, 11)** using the stack algorithm. Compare the number of decoding steps with the number required by the Viterbi algorithm.

c) Repeat part (b) for the received sequence **r = (11, 10, 00,01,10,01,00)**

29. Consider again the (**2, 1, 3**) code of problem 21.

a) For the binary input-8-ary-output **DMC** of problem 20, find an integer metric table for the Fano metric.(Hint: Use appropriate scale factor for each metric and round to the nearest integer).

b) Decode the received sequence: **r = [**y_7y_8, y_7y_1, y_3y_1, y_1y_6, y_6y_2, y_3y_8, y_3y_2**]** using the Stack algorithm. Compare the final decoded path with the result of problem 21(b) where the same received sequence is decoded using Viterbi algorithm.

30. Repeat problem 29 using Fano algorithm with threshold increments of $\Delta = 5$ and $\Delta = 9$.Compare the final decoded path and the number of computations with the Stack algorithm.

31. Refer to the code tree of Fig P 8.13. The branch metrics are specified as below.

$$M(r_j\, r_i \mid v_j\, v_i) = \begin{cases} 1 & \text{if } \; r_j = v_j \;\; \text{and } r_i = v_i \\ -4 & \text{if } \; r_j = v_j \;\; \text{and } r_i \neq v_i \text{ or } r_j \neq v_j \text{ and } \; r_i = v_i \\ -9 & \text{if } \; r_j \neq v_j \text{ and } \; r_i \neq v_i \end{cases}$$

Calculate the path metrics at the nodes of the tree for an all zero transmitted sequence, and the received sequence (**10, 00, 01, 01**) with three transmissions errors. Then apply Fano algorithm to decode the received sequence under the assumptions:

a) The decoder chooses the lower branch in the case of a tie in path metrics.

b) The decoder chooses the upper branch in the case of a tie in path metrics.

OUTCOMES

- To know the encoding of convolutional codes.
- How to decode the convolutional codes using algorithm.

REFERENCE

- www.youtube.com/watch?v=AnyVu5eDhAQ
- **nptel**.ac.in/courses/117106031/
- elearning.**vtu**.ac.in/P4/EC63/S11.pdf