Module 6

Channel Coding

Lesson 35

Convolutional Codes

After reading this lesson, you will learn about

- ***Basic concepts of Convolutional Codes;***
- ***State Diagram Representation;***
- ***Tree Diagram Representation;***
- ***Trellis Diagram Representation;***
- ***Catastrophic Convolutional Code;***
- ***Hard - Decision Viterbi Algorithm;***
- ***Soft - Decision Viterbi Algorithm;***

Convolutional codes are commonly described using two parameters: the code rate and the constraint length. The code rate, k/n, is expressed as a ratio of the number of bits into the convolutional encoder (k) to the number of channel symbols output by the convolutional encoder (n) in a given encoder cycle. The constraint length parameter, K, denotes the "length" of the convolutional encoder, i.e. how many k-bit stages are available to feed the combinatorial logic that produces the output symbols. Closely related to K is the parameter m, which indicates how many encoder cycles an input bit is retained and used for encoding after it first appears at the input to the convolutional encoder. The m parameter can be thought of as the memory length of the encoder.

Convolutional codes are widely used as channel codes in practical communication systems for error correction. The encoded bits depend on the current k input bits and a few past input bits. The main decoding strategy for convolutional codes is based on the widely used Viterbi algorithm. As a result of the wide acceptance of convolutional codes, there have been several approaches to modify and extend this basic coding scheme. Trellis coded modulation (TCM) and turbo codes are two such examples. In TCM, redundancy is added by combining coding and modulation into a single operation. This is achieved without any reduction in data rate or expansion in bandwidth as required by only error correcting coding schemes.

A simple convolutional encoder is shown in **Fig. 6.35.1**. The information bits are fed in small groups of k-bits at a time to a shift register. The output encoded bits are obtained by modulo-2 addition (EXCLUSIVE-OR operation) of the input information bits and the contents of the shift registers which are a few previous information bits.

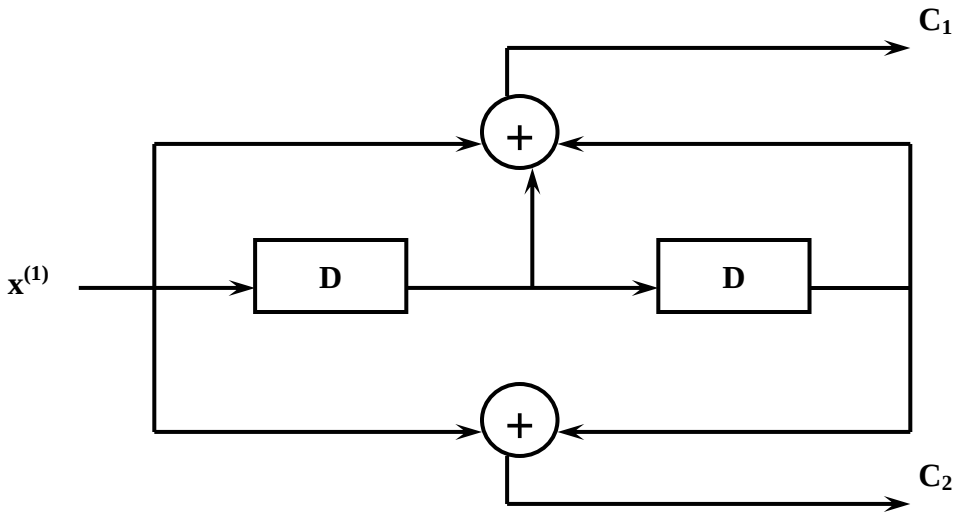

Fig. 6.35.1 *A convolutional encoder with k=1, n=2 and r=1/2*

If the encoder generates a group of 'n' encoded bits per group of 'k' information bits, the code rate R is commonly defined as R = k/n. In **Fig. 6.35.1**, k = 1 and n = 2. The number, K of elements in the shift register which decides for how many codewords one information bit will affect the encoder output, is known as the constraint length of the code. For the present example, K = 3.

The shift register of the encoder is initialized to all-zero-state before encoding operation starts. It is easy to verify that encoded sequence is 00 11 10 00 01 ….for an input message sequence of 01011….

The operation of a convolutional encoder can be explained in several but equivalent ways such as, by a) state diagram representation, b) tree diagram representation and c) trellis diagram representation.

a) State Diagram Representation

A convolutional encoder may be defined as a finite state machine. Contents of the rightmost (K-1) shift register stages define the states of the encoder. So, the encoder in **Fig. 6.35.1** has four states. The transition of an encoder from one state to another, as caused by input bits, is depicted in the state diagram. **Fig. 6.35.2** shows the state diagram of the encoder in **Fig. 6.35.1**. A new input bit causes a transition from one state to another. The path information between the states, denoted as b/c_1c_2, represents input information bit 'b' and the corresponding output bits (c_1c_2). Again, it is not difficult to verify from the state diagram that an input information sequence **b** = (1011) generates an encoded sequence **c** = (11, 10, 00, 01).

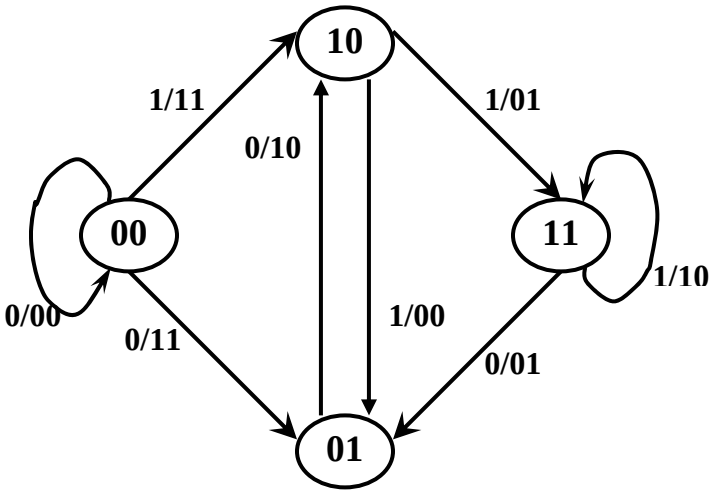

Fig.6.35.2 *State diagram representation for the encoder in Fig. 6.35.1*

b) Tree Diagram Representation

The tree diagram representation shows all possible information and encoded sequences for the convolutional encoder. **Fig. 6.35.3** shows the tree diagram for the encoder in **Fig. 6.35.1**. The encoded bits are labeled on the branches of the tree. Given an input sequence, the encoded sequence can be directly read from the tree. As an example, an input sequence (1011) results in the encoded sequence (11, 10, 00, 01).

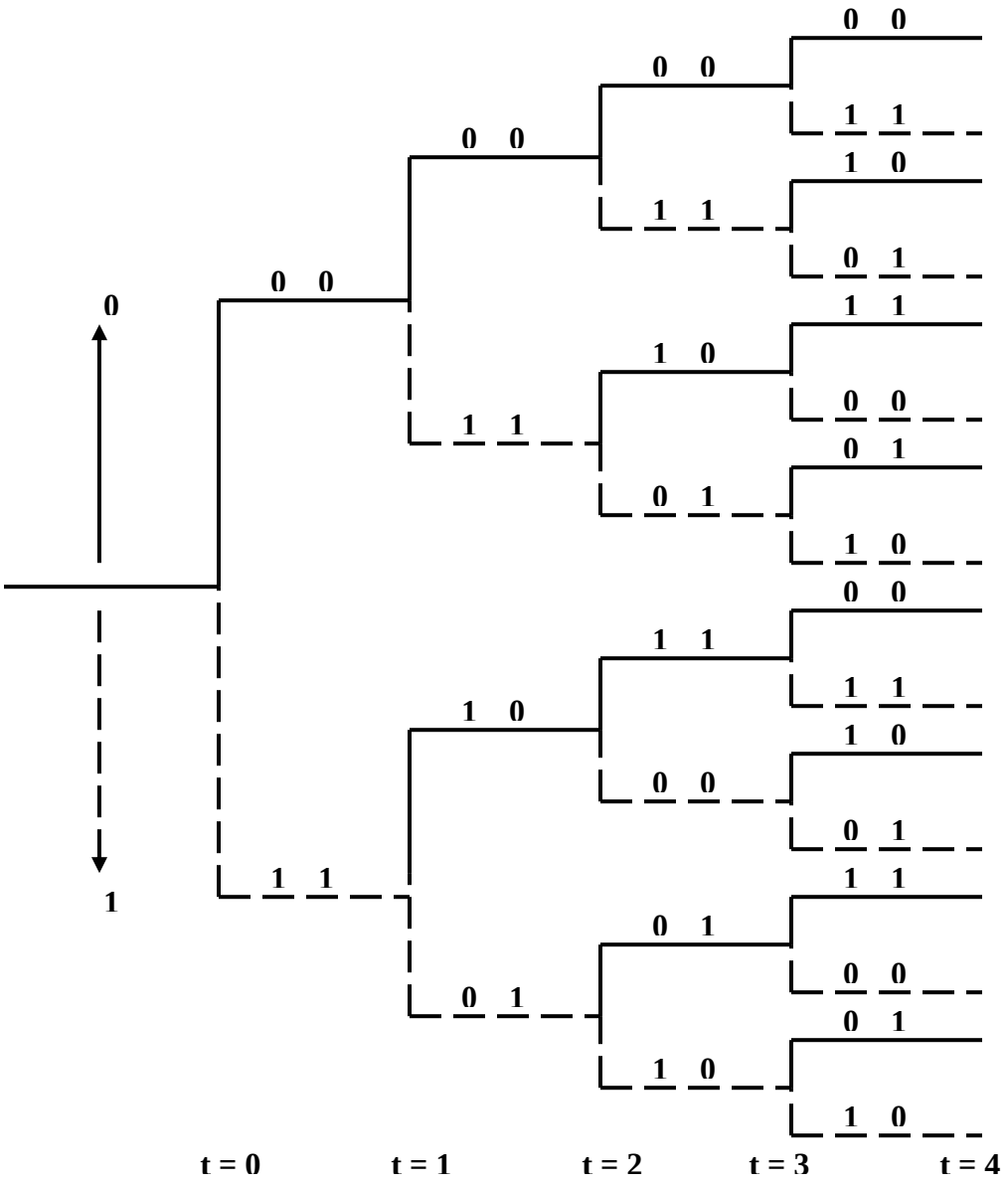

Fig. 6.35.3 *A tree diagram for the encoder in Fig. 6.35.1*

c) Trellis Diagram Representation

The trellis diagram of a convolutional code is obtained from its state diagram. All state transitions at each time step are explicitly shown in the diagram to retain the time dimension, as is present in the corresponding tree diagram. Usually, supporting descriptions on state transitions, corresponding input and output bits etc. are labeled in the trellis diagram. It is interesting to note that the trellis diagram, which describes the operation of the encoder, is very convenient for describing the behavior of the

corresponding decoder, especially when the famous 'Viterbi Algorithm (VA)' is followed. **Figure 6.35.4** shows the trellis diagram for the encoder in **Figure 6.35.1**.

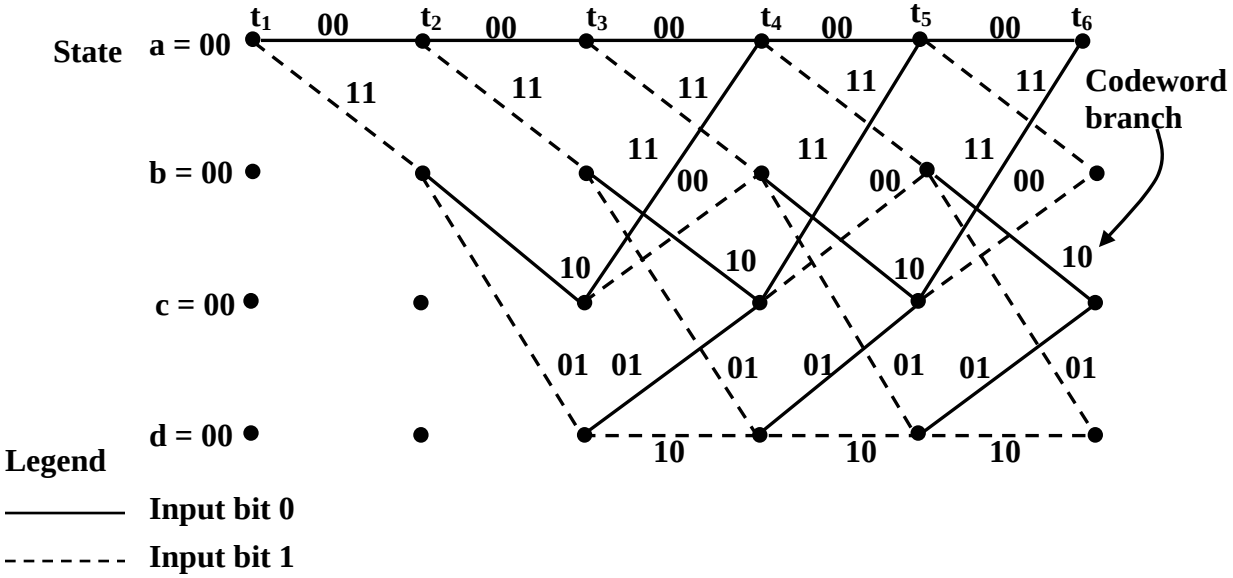

Fig.6.35.4(a) *Trellis diagram for the encoder in Fig. 6.35.1*

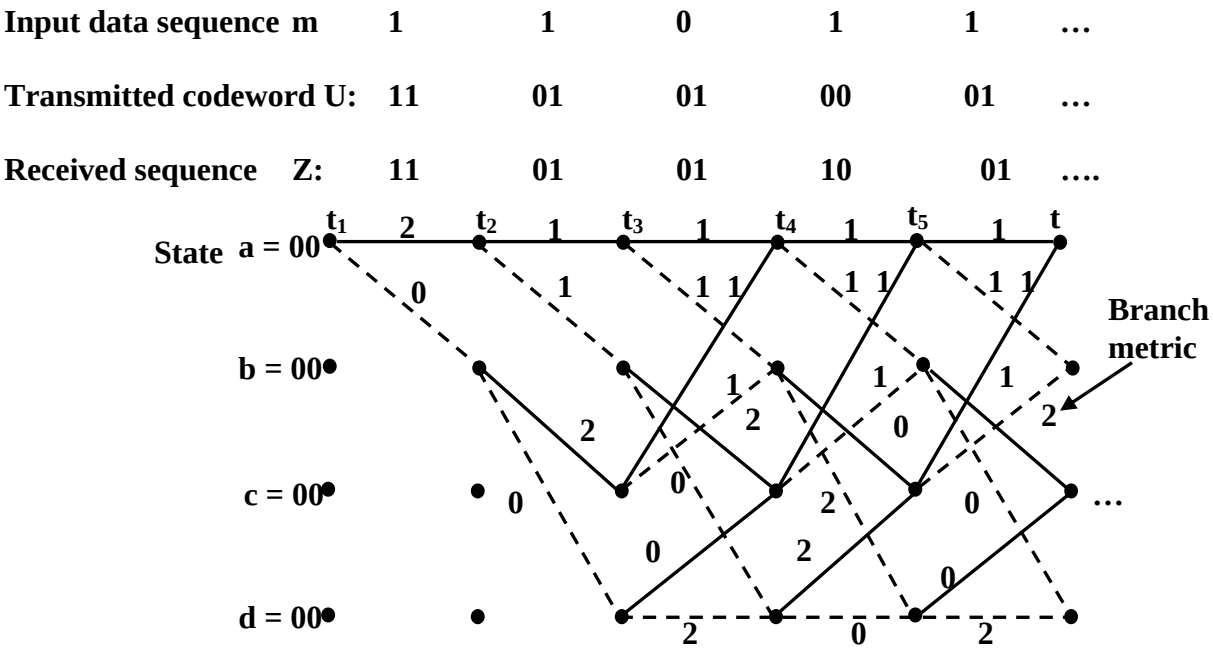

Fig.6.35.4(b) *Trellis diagram, used in the decoder corresponding to the encoder in Fig. 6.35.1*

Catastrophic Convolutional Code

The taps of shift registers used for a convolutional encoder are to be chosen carefully so that the code can effectively correct errors in received data stream. One measure of error correction capability of a convolutional code is its 'minimum free distance, d_{free}', which indicates the minimum weight (counted in number of '1'-s) of a path that branches out from the all-zero path of the code trellis and again merges with the all-zero path. For example, the code in **Fig.6.35.1** has d_{free}= 5. Most of the convolutional codes of present interest are good for correcting random errors rather than correcting error bursts. If we assume that sufficient number of bits in a received bit sequence are error free and then a few bits are erroneous randomly, the decoder is likely to correct these errors. It is expected that (following a hard decision decoding approach, explained later) the decoder will correct up to $(d_{free}-1)/2$ errors in case of such events. So, the taps of a convolutional code should be chosen to maximize d_{free}. There is no unique method of finding such convolutional codes of arbitrary rate and constraint length that ensures maximum d_{free}. However, comprehensive description of taps for 'good ' convolutional codes of practical interest have been prepared through extensive computer search techniques and otherwise.

While choosing a convolutional code, one should also avoid 'catastrophic convolutional code'. Such codes can be identified by the state diagram. The state diagram of a 'catastrophic convolutional code' includes at least one loop in which a nonzero information sequence corresponds to an all-zero output sequence. Tree diagram of a 'catastrophic convolutional code' is shown in **Fig.6.35.5**.

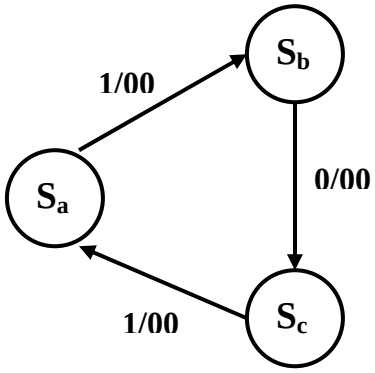

Fig.6.35.5 *Example of a catastrophic code*

Hard-Decision and Soft-Decision Decoding

Hard-decision and soft-decision decoding are based on the type of quantization used on the received bits. Hard-decision decoding uses 1-bit quantization on the received samples. Soft-decision decoding uses multi-bit quantization (e.g. 3 bits/sample) on the received sample values.

Hard-Decision Viterbi Algorithm

The Viterbi Algorithm (VA) finds a maximum likelihood (ML) estimate of a transmitted code sequence **c** from the corresponding received sequence **r** by maximizing the probability p(**r**|**c**) that sequence **r** is received conditioned on the estimated code sequence **c.** Sequence **c** must be a valid coded sequence.

The Viterbi algorithm utilizes the trellis diagram to compute the path metrics. The channel is assumed to be memory less, i.e. the noise sample affecting a received bit is independent from the noise sample affecting the other bits. The decoding operation starts from state '00', i.e. with the assumption that the initial state of the encoder is '00'. With receipt of one noisy codeword, the decoding operation progresses by one step deeper into the trellis diagram. The branches, associated with a state of the trellis tell us about the corresponding codewords that the encoder may generate starting from this state. Hence, upon receipt of a codeword, it is possible to note the 'branch metric' of each branch by determining the Hamming distance of the received codeword from the valid codeword associated with that branch. Path metric of all branches, associated with all the states are calculated similarly.

Now, at each depth of the trellis, each state also carries some 'accumulated path metric', which is the addition of metrics of all branches that construct the 'most likely path' to that state. As an example, the trellis diagram of the code shown in **Fig. 6.35.1**, has four states and each state has two incoming and two outgoing branches. At any depth of the trellis, each state can be reached through two paths from the previous stage and as per the VA, the path with lower accumulated path metric is chosen. In the process, the 'accumulated path metric' is updated by adding the metric of the incoming branch with the 'accumulated path metric' of the state from where the branch originated. No decision about a received codeword is taken from such operations and the decoding decision is deliberately delayed to reduce the possibility of erroneous decision.

The basic operations which are carried out as per the hard-decision Viterbi Algorithm after receiving one codeword are summarized below:

a) All the branch metrics of all the states are determined;

b) Accumulated metrics of all the paths (two in our example code) leading to a state are calculated taking into consideration the 'accumulated path metrics' of the states from where the most recent branches emerged;

c) Only one of the paths, entering into a state, which has minimum 'accumulated path metric' is chosen as the 'survivor path' for the state (or, equivalently 'node');

d) So, at the end of this process, each state has one 'survivor path'. The 'history' of a survivor path is also maintained by the node appropriately (e.g. by storing the codewords or the information bits which are associated with the branches making the path);

e) Steps a) to d) are repeated and decoding decision is delayed till sufficient number of codewords has been received. Typically, the delay in decision making = Lx k codewords where L is an integer, e.g. 5 or 6. For the code in **Fig. 6.35.1**, the decision delay of 5x3 = 15 codewords may be sufficient for most occasions. This means, we decide about the first received codeword after receiving the 16^{th} codeword. The decision strategy is simple. Upon receiving the 16^{th} codeword and carrying out steps a) to d), we compare the 'accumulated path metrics' of all the states (four in our example) and chose the state with minimum overall 'accumulated path metric' as the 'winning node' for the first codeword. Then we trace back the history of the path associated with this winning node to identify the codeword tagged to the first branch of the path and declare this codeword as the most likely transmitted first codeword.

The above procedure is repeated for each received codeword hereafter. Thus, the decision for a codeword is delayed but once the decision process starts, we decide once for every received codeword. For most practical applications, including delay-sensitive digital speech coding and transmission, a decision delay of Lx k codewords is acceptable.

Soft-Decision Viterbi Algorithm

In soft-decision decoding, the demodulator does not assign a '0' or a '1' to each received bit but uses multi-bit quantized values. The soft-decision Viterbi algorithm is very similar to its hard-decision algorithm except that squared Euclidean distance is used in the branch metrics instead of simpler Hamming distance. However, the performance of a soft-decision VA is much more impressive compared to its HDD (Hard Decision Decoding) counterpart [**Fig. 6.35.6 (a) and (b)**]. The computational requirement of a Viterbi decoder grows exponentially as a function of the constraint length and hence it is usually limited in practice to constraint lengths of K = 9.

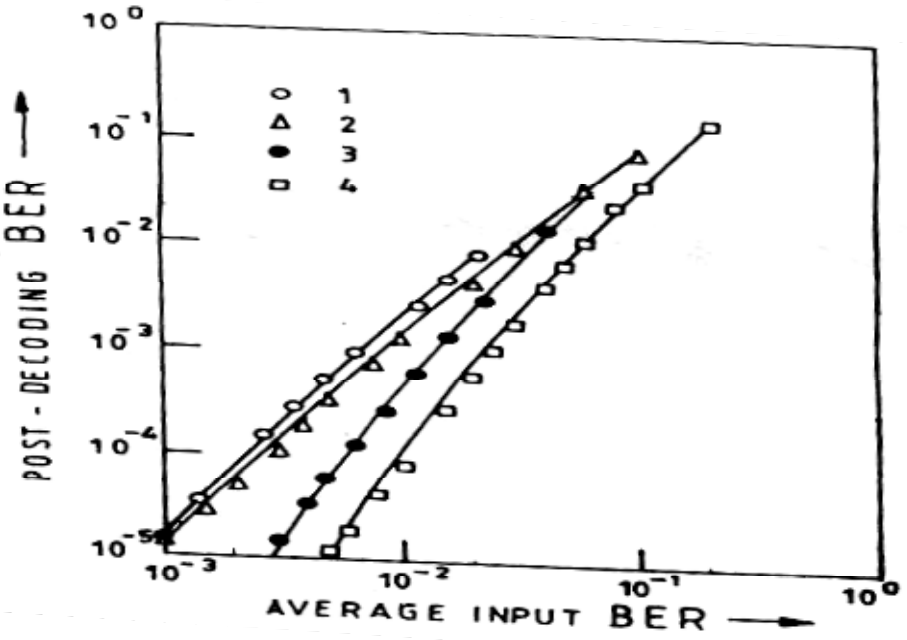

Fig. 6.35.6 (a) *Decoded BER vs input BER for the rate – half convolutional codes with Viterbi Algorithm ; 1) k = 3 (HDD),2) k = 5 (HDD),3) k = 3 (SDD), and 4) k= 5 (SDD). HDD: Hard Decision Decoding; SDD: Soft Decision Decoding.*

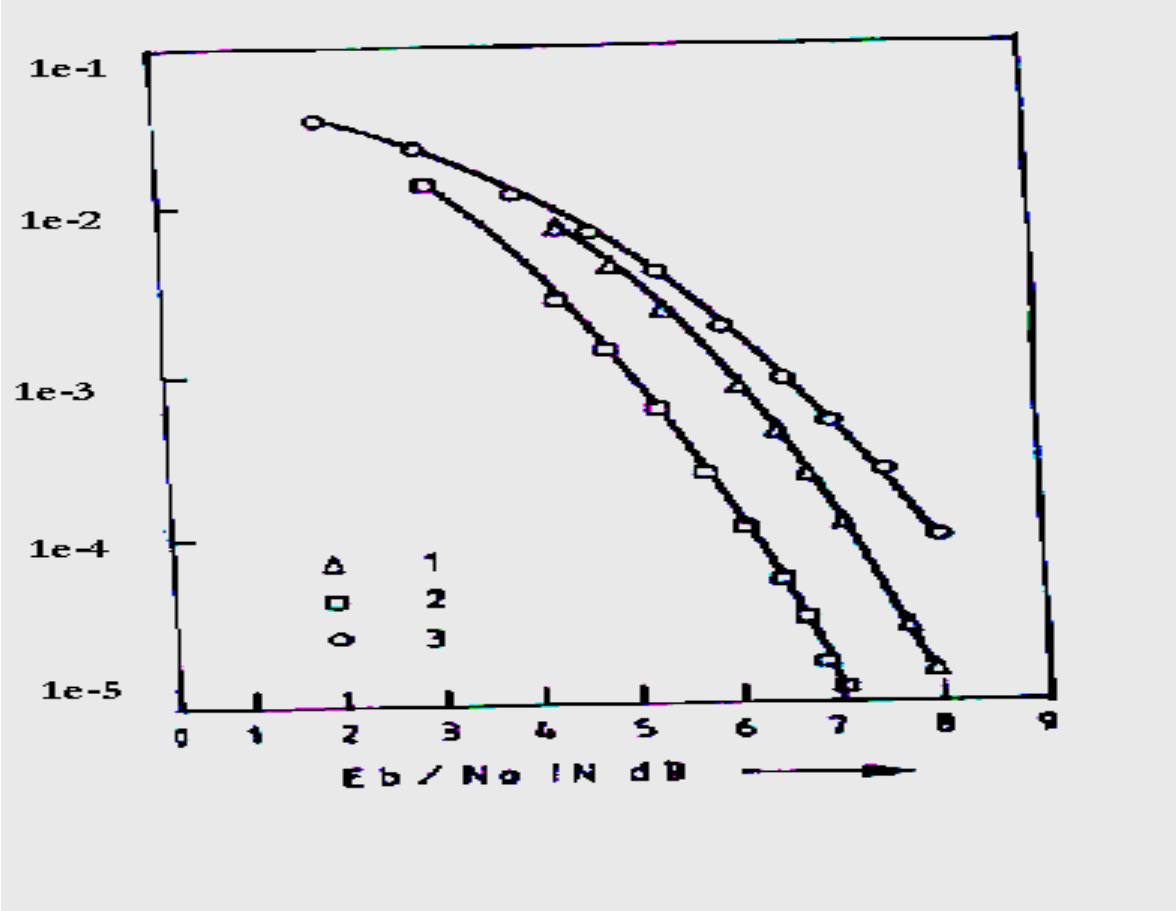

Fig. 6.35.6 (b) *Decoded BER vs E_b/N_o (in dB) for the rate – half convolutional codes with Viterbi Algorithm ; 1) Uncoded system; 2) with k = 3 (HDD) and 3) k = 3 (SDD). HDD: Hard Decision Decoding; SDD: Soft Decision Decoding.*