CT-216 PROJECT

Convolutional Coding

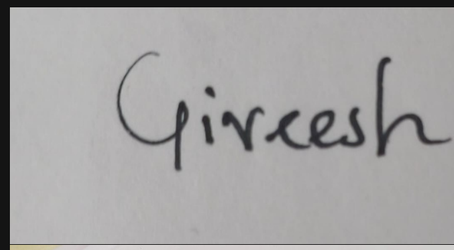
Prof. Yash Vasavada

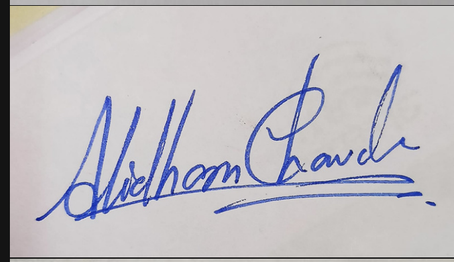
Lab Group 2
Project Group 2
Mentor TA : Naisheel Bhai

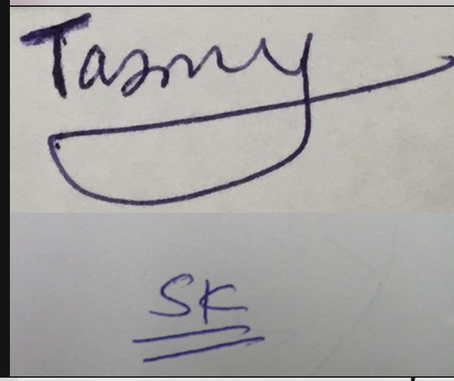
Honor Code

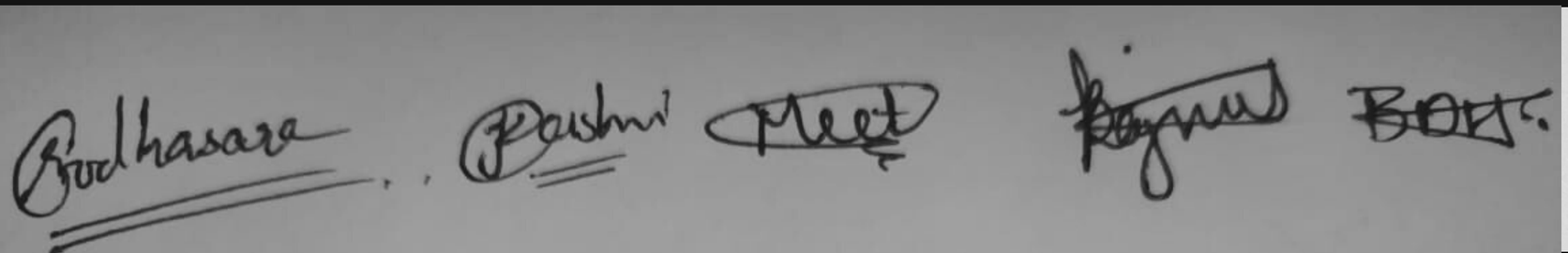
We declare that:

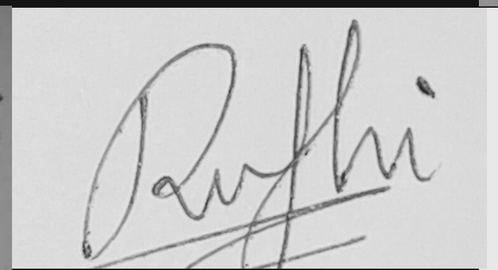
1. The work that we are presenting is our own work.
2. We have not copied the work (the code, the results etc.) that someone else has done.
3. Concepts, understanding and insights we will be describing are our own.
4. We make this pledge truthfully. We know that violation of this solemn pledge can carry grave consequences.

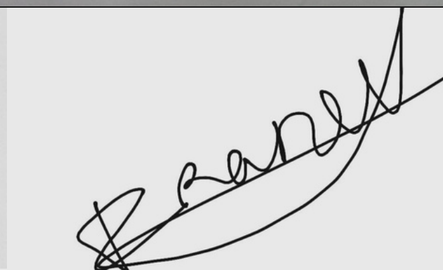








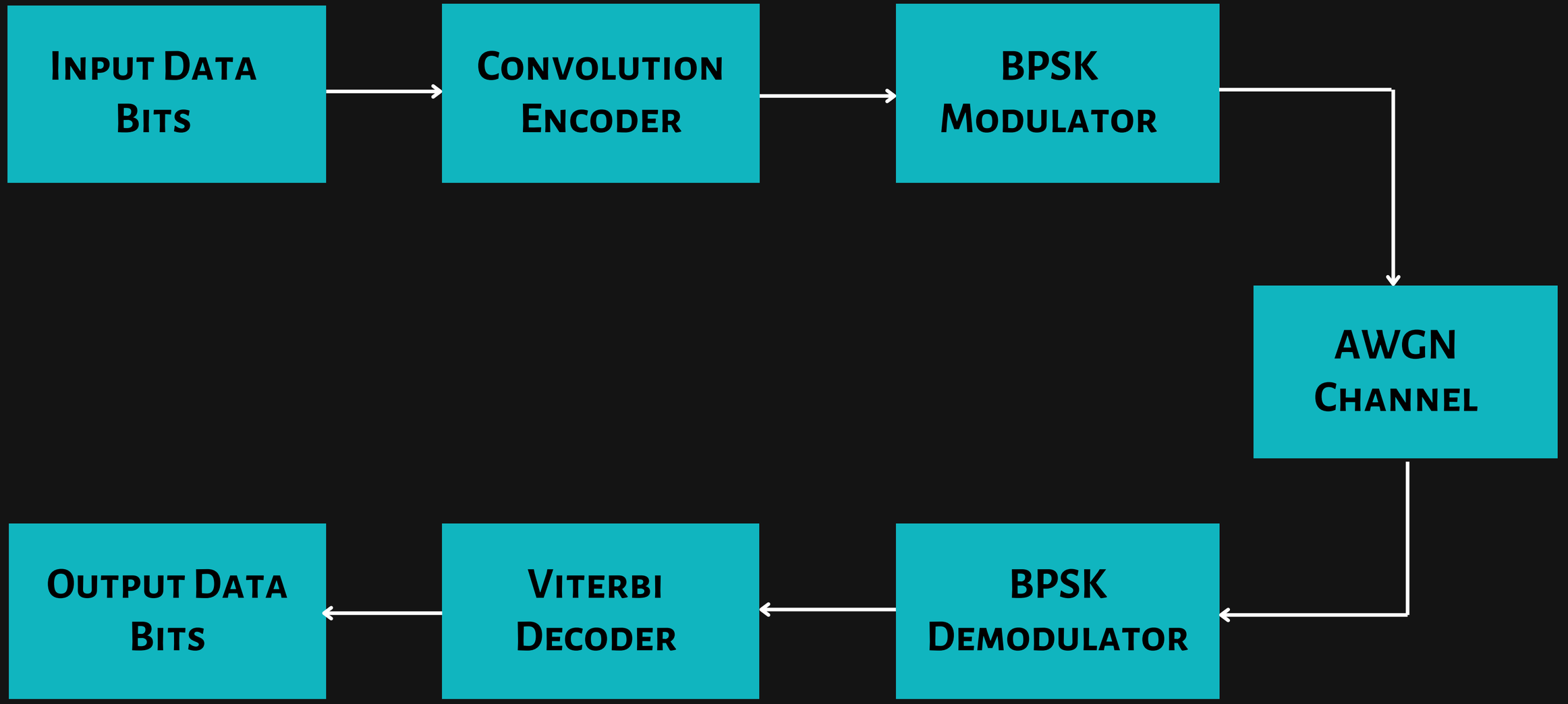




Introduction

- Convolution coding is a popular error-correcting coding method used to improve the reliability of communication system.
- Block codes process data in fixed-size blocks. Convolutional codes, on the other hand, offer continuous encoding and decoding.
- Convolutional encoders typically have lower latency compared to polar codes and LDPC codes.
- Easy to implement from hardware perspective.
- Because of this benefits, we choose convolutional coding over other coding techniques.

FLOW DIAGRAM



Convolutional Encoder

- A Convolutional Encoder has a bit level encoding technique. they are used in applications that require good performance with low implementation cost.

- Using convolutional codes, a continuous sequence of information bits is mapped into continuous sequence of encoder output bits.

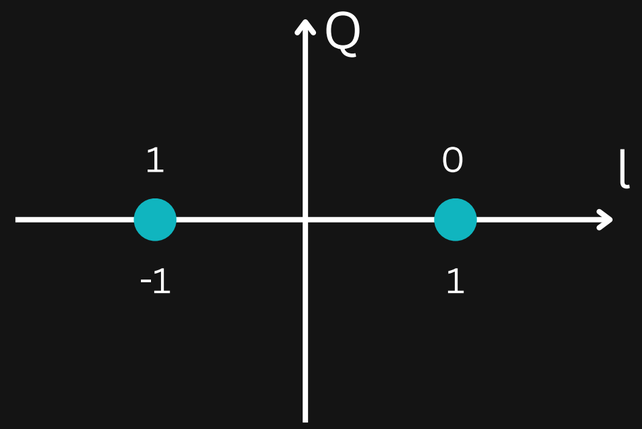
- The encoded bits not only depends on current inputs but also on past inputs and the storing of past input bits can be done using simple registers.

BPSK Modulator

- Binary Phase Shift Keying (BPSK) is a modulation technique employed in communication systems to transmit information via a communication channel

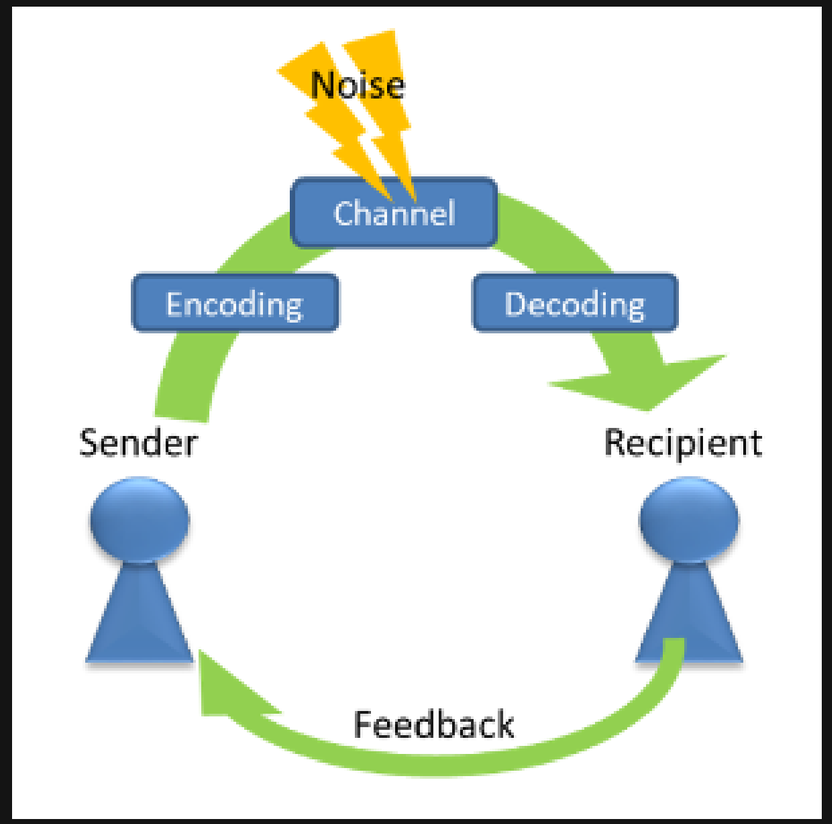
- BPSK simplifies digital communication by encoding binary data into phase shifts of a carrier signal.

- Binary data is mapped to symbols.



AWGN Channel

- Additive White Gaussian Noise (AWGN) poses a significant challenge to reliable communication.
- When a convolutional-coded signal is transmitted over a channel affected by AWGN, the noise distorts the signal.



BPSK Demodulator

- At the receiver, the received signal undergoes demodulation to recover the original binary data. This process typically involves:

- Coherent Detection: Multiplying the received signal with a local oscillator at the carrier frequency to extract the phase information.

- Decision Making: Comparing the phase of the received signal with a threshold value to determine the transmitted symbol. If the phase is closer to one phase state, it's interpreted as '0'; if it's closer to the other phase state, it's interpreted as '1'.

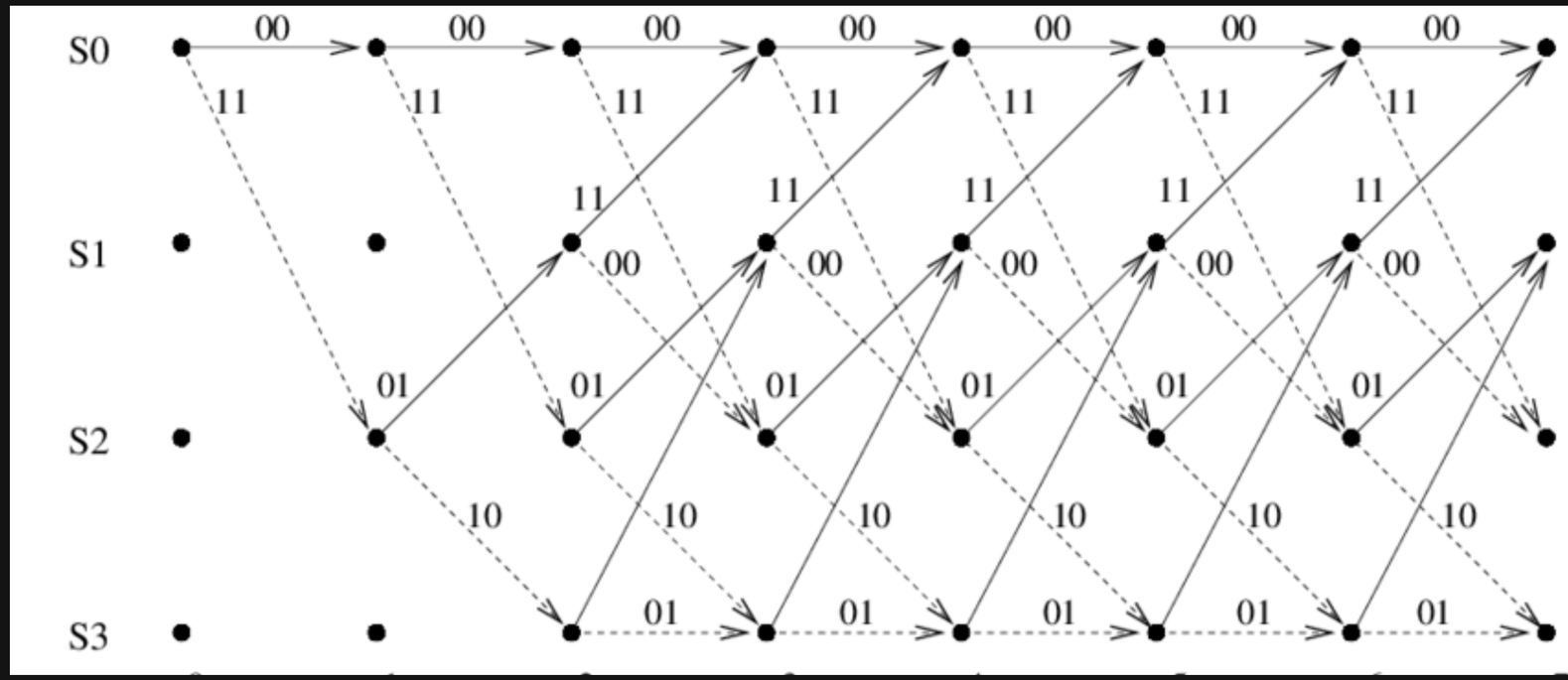
For soft decision decoding,

- If the convolutional code produces p parity bits, and the p corresponding analog samples are v = v1, v2,…,vp.

- These values are analog in nature as we do not perform demodulation of the received signal we directly pass it to the decoder.

Trellis Representation

- Convolutional codes are often represented graphically using a trellis diagram. Each node in the trellis represents a possible state of the encoder, and the edges between nodes represent possible state transitions corresponding to transmitted bits. The trellis diagram provides a systematic way to visualize the possible paths taken by the encoder.



Viterbi Decoder

- The Viterbi decoding algorithm uses two metrics: the branch metric (BM) and the path metric (PM) and two other blocks add compare and select unit (ACS) and trace back unit (TBU)

$$BM_{\text{Hard}}\ [u, v] = \sum_{i=1}^{p} (u_i \oplus v_i)$$

- The branch metric is a measure of the **“Hamming distance”** between what was transmitted and what was received. (For Hard Decision Decoding)

$$\text{BM}_{\text{soft}}[u, v] = \sum_{i=1}^{p} (u_i - v_i)^2$$

- In Case of Soft decision, we have to calculate **“Euclidean Distance”** instead of Hamming Distance

Computing path and branch metric

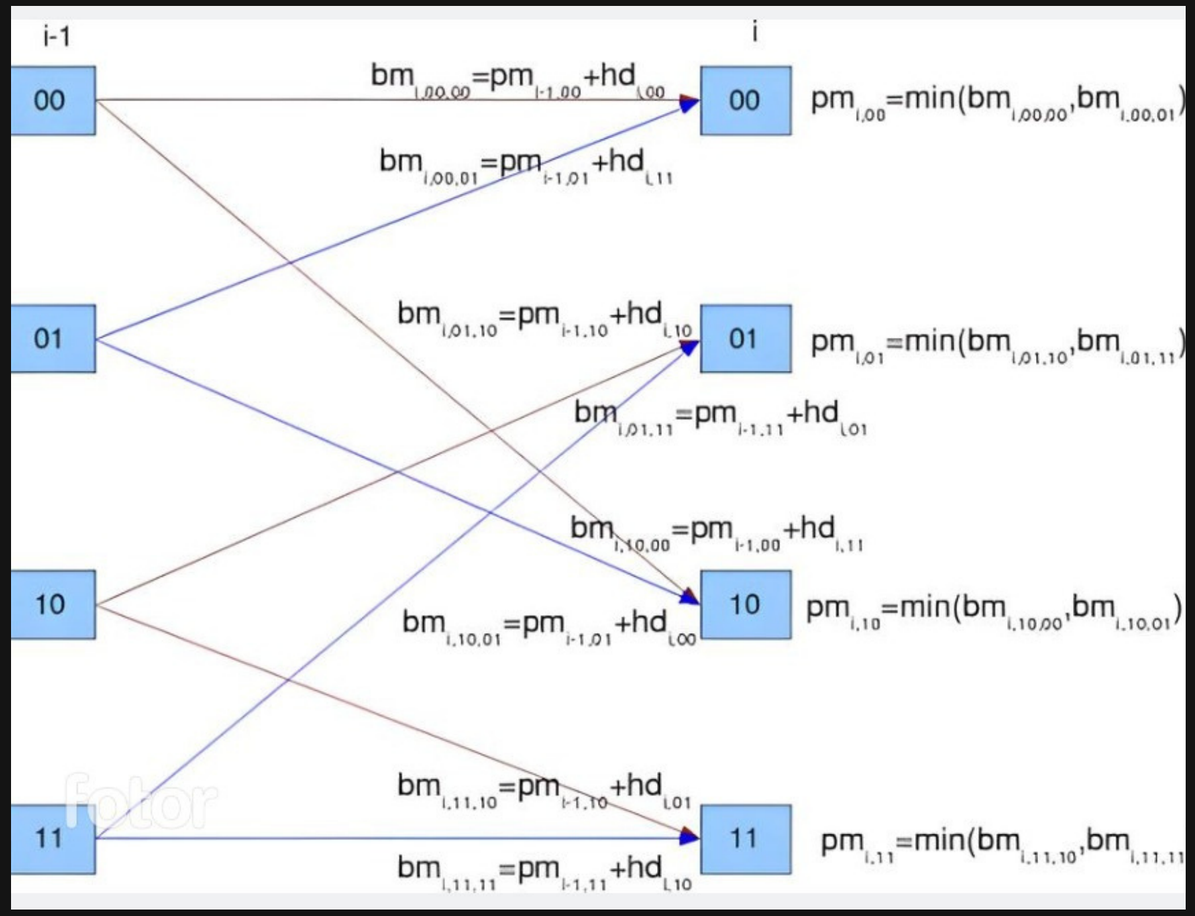
- The path metric represents the distance along the most likely path from the initial state to the current state in the trellis.

- The algorithm computes path metrics incrementally using branch metrics and previously computed path metrics. This approach finds the path with the smallest distance, minimizing bit errors and providing a reliable estimate of the transmitted data, especially when the bit error rate is low.

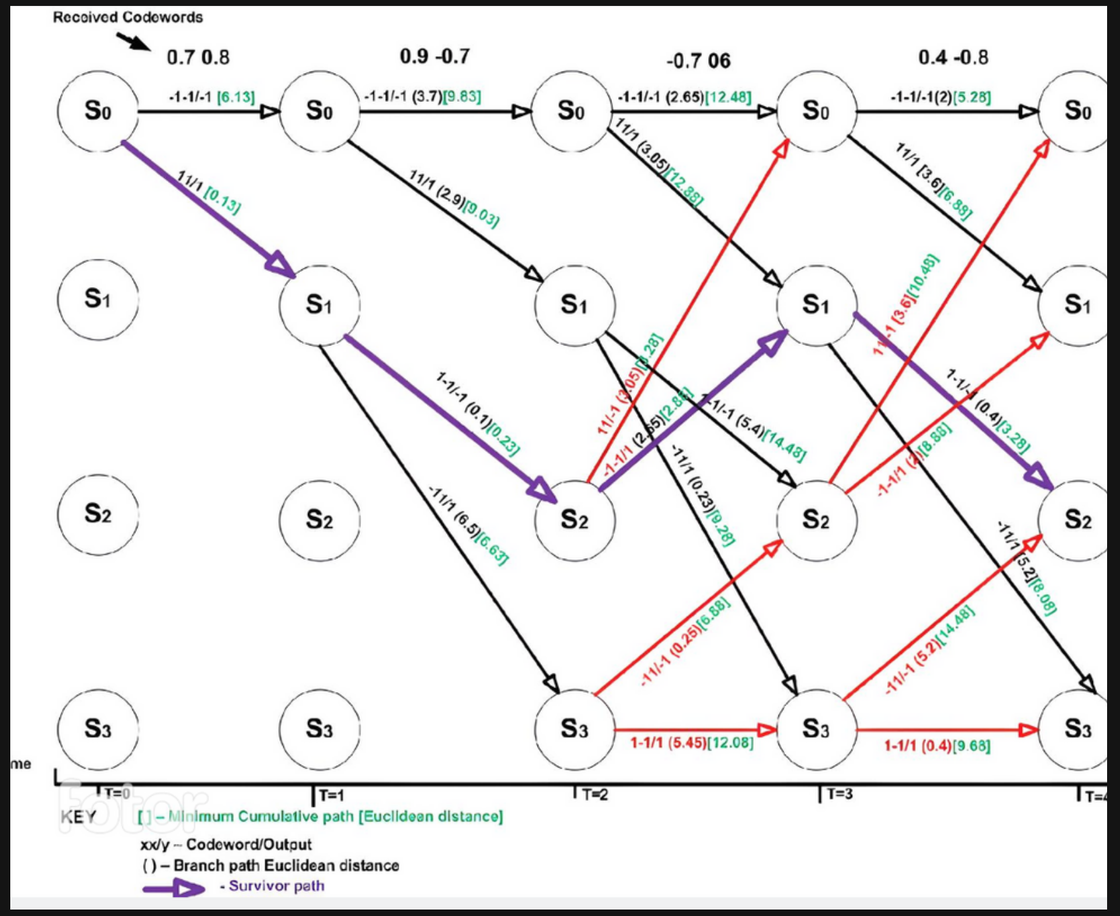
$$\mathrm{PM}[s, i+1] = \min(\mathrm{PM}[\alpha, i] + \mathrm{BM}[\alpha \to s], \mathrm{PM}[\beta, i] + \mathrm{BM}[\beta \to s])$$

α and β are two predecessor states

Hard Decison Decoding



Soft Decison Decoding



Source

Transfer Function

Transfer Function is an algebraic representation of all paths that starts and end at all zero state.

$$T(D, N, J) = \sum_{d=d_{free}}^{\infty} a_d D^d N^{f(d)} J^{g(d)}.$$

D – exponent 'd' indicates the number of ones in output code word
N – exponent '$f(d)$' indicates the number of ones in input block k –bits at a time
J – exponent '$g(d)$' indicates the number of branches spanned by the path

- Split zero states in two, one for starting state and other one terminating state
- All other states are in Transition (in - between) states
- The transfer function = (Output at end zero state / Input at zero start state)
- Provides the properties of all the paths.
- Minimum non-zero exponent of D gives the minimum hamming distance.

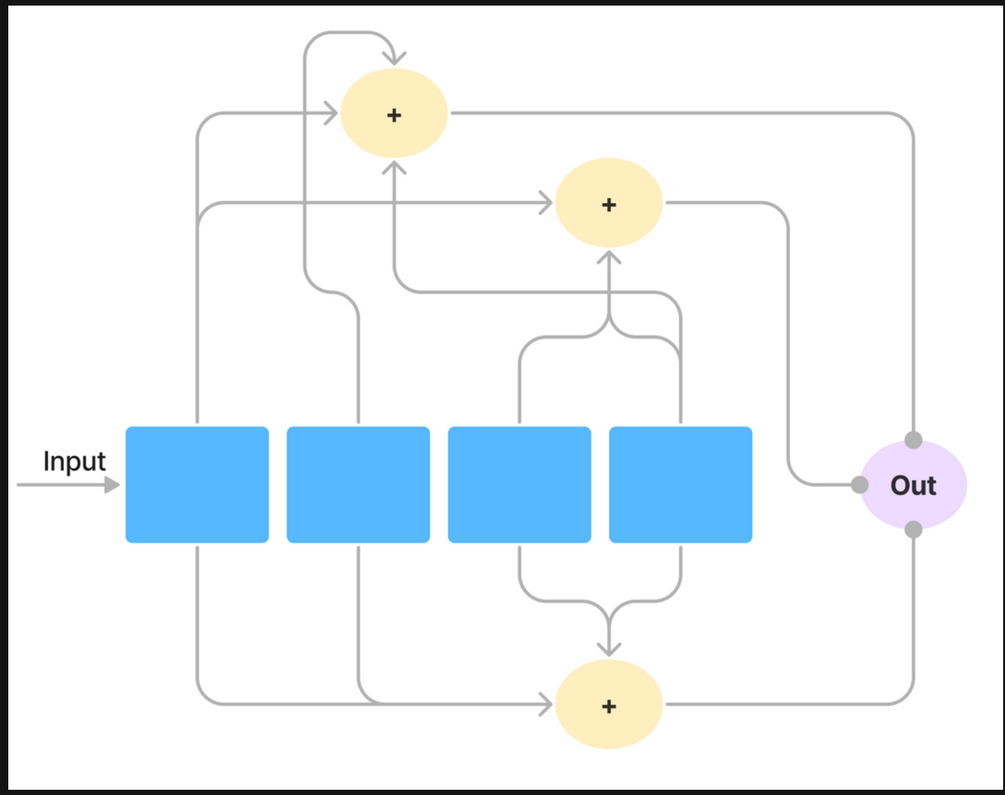
Some Definitions

- Rate : Ratio of the number of input bits to the number of output bits. generally, rate=k/n where k is information bits and n is encoded bits.

- Constraint length (Kc): The number of delay elements in the convolutional coding.

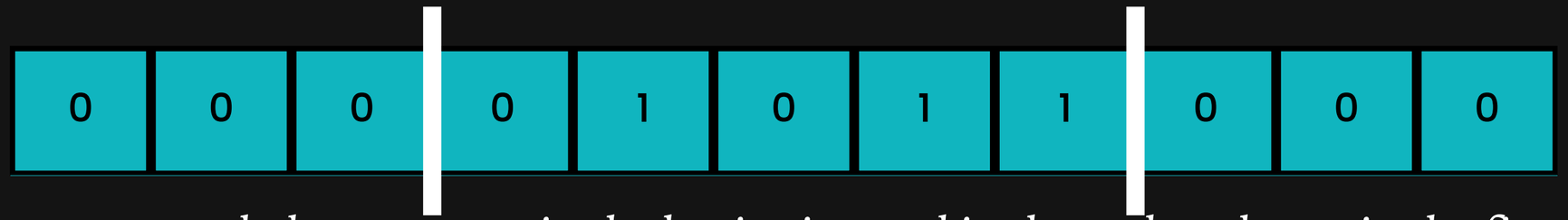
- Generator polynomial : Wiring of the input sequence with the delay elements to form the output.

Let's take an example:
Rate=1/3
Constraint Length=4
Generators are given g1=[1 0 1 1],
g2=[1 1 0 1] and g3=[1 1 1 1]



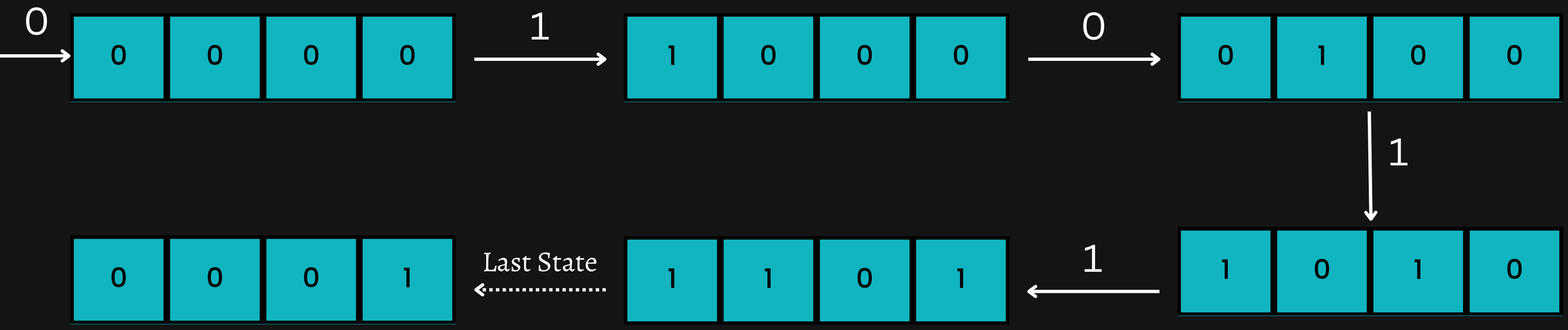
Solved Example

Input = [0 1 0 1 0 1 0 1]



Kc-1 zeros are appended to message in the beginning and in the end as shown in the figure, to get terminating state to all zero

We can use left/right shift and pass the input sequence continously. Initial state will be all zeros



According to the generator matrix mentioned in slide 13,
Xor Operation will be performed between input and states of sequence

Enc. seq = 0 0 0 1 1 1 0 1 1 0 1 0 0 1 1 1 1 0 0 1 0 1 1 1

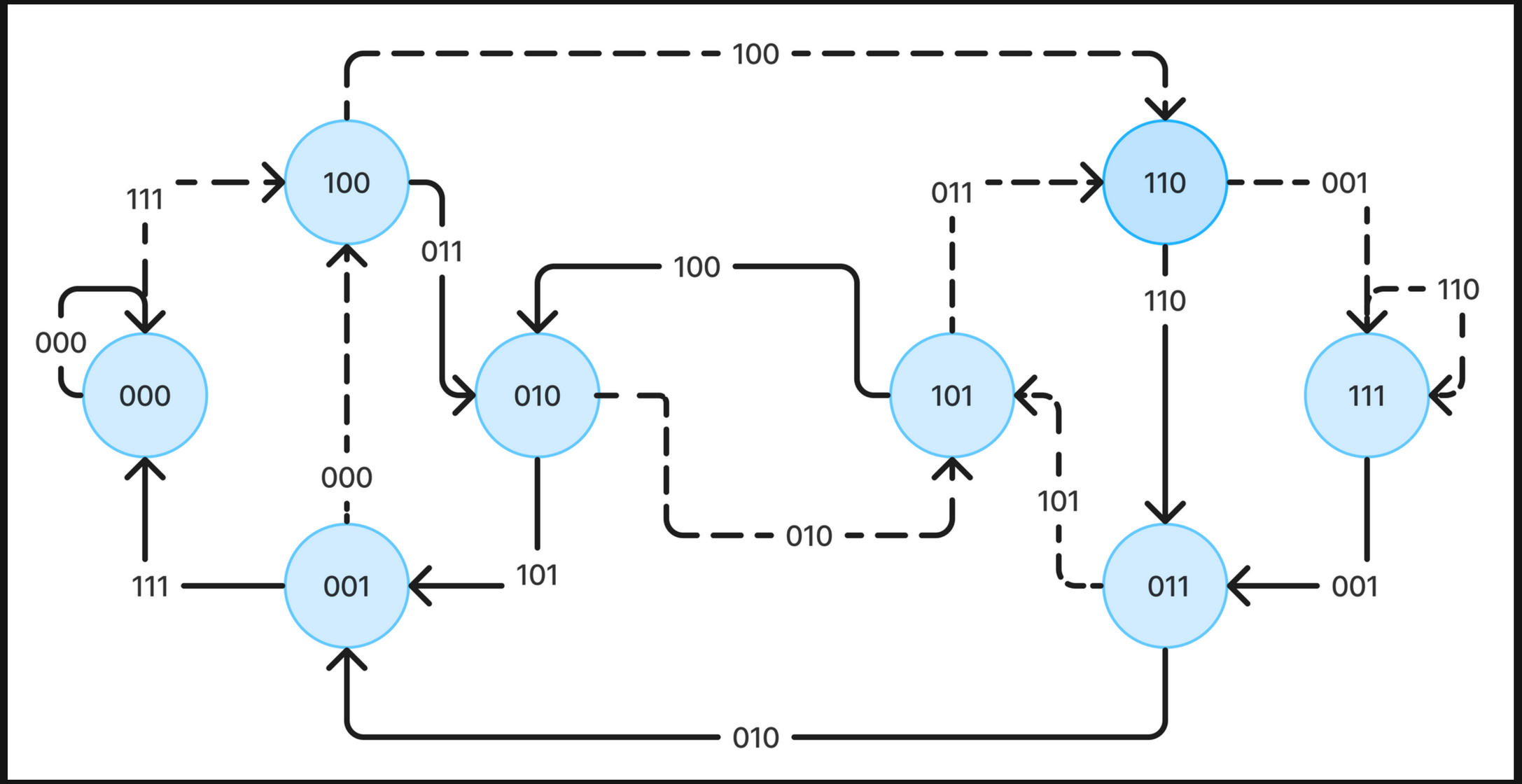
After Performing Modulation, procedure given in slide 6 /
mapping to (2b-1),

Mod. Seq = -1 -1 -1 1 1 1 -1 1 1 -1 1 -1 -1 1 1 1 1 -1 -1 1 -1 1 1 1

Pseudo code for encoding is given in slide 18 and matlab code is given in report.
Matlab code works for arbitary length of Kc, k, n and G.

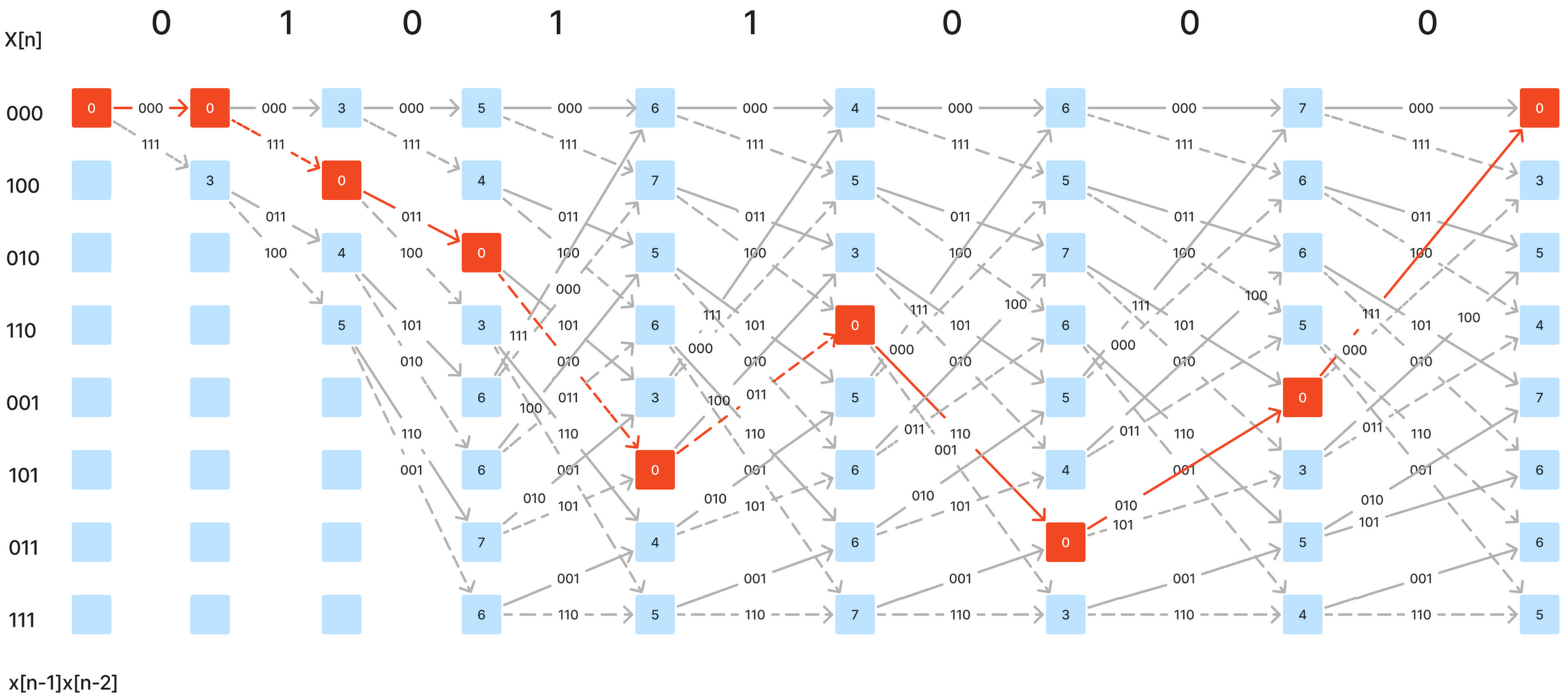
State Diagram

Input 0 - solid line
Input 1 - dotted line



Trellis Diagram

Input 0 - solid line
Input 1 - dotted line



Encoding : Pseudocode

```
encoding(G, Kc, input_seq)

    encoded_msg = []

    arr = [input_seq[1]]                    // Initialize shift register with the first element of the input sequence

    for i = 1 to Kc-1:                      // Fill the shift register with zeros to match the constraint length.
        arr = concatenate(arr, 0)

    for i = 1 to length(input_seq):

        arr1 = transpose(arr)
        arr2 = matrix_multiplication(G, arr1)   // Get the convolutional code by multiplying G with the shift register
        arr2 = modulo(arr2, 2)

        append_to_list(encoded_msg, arr2)       // Append the encoder output to the encoded message.

        for j = Kc to 2:                    // Shift the shift register to the right by one position.
            arr[j] = arr[j-1]

        if i != length(input_seq):          // Update first element of the shift register with the next input symbol.
            arr[1] = input_seq[i+1]

return encoded_msg
```

State Diagram : Pseudocode

```
state_diag(G, Kc, n)

    no_of_states = 2^(Kc-1)                          // Calculate the number of states

    arr = zeros(no_of_states, Kc-1)                  // Initialize an array to store state transitions(2D Array)

    for i = 0 to no_of_states-1                      // Generate all possible states

        x = int_to_binary(i, Kc-1)
        x1 = transpose(x)
        arr[i+1, :] = x1


    state_diagram = zeros(no_of_states, 4)           // Initialize an array to store the state diagram

     // In the first and the second columns, we are storing the output for the input bit 0 and 1
     // For the third and fourth columns we are storing the next states for input bit 0 and 1

  for i = 1 to no_of_states:

        arr0 = arr[i, :]
        arr0 = concatenate(0, arr0)
        op0 = mod(matrix_multiplication(G, arr0), 2)     // Compute the output for transition on input 0
        state_diagram[i, 1] = binary_to_int(op0, n)
```

```
    next_state0 = []

    for j = 1 to Kc-1:
        next_state0 = concatenate(next_state0, arr0[j])          // Determine next state for transition on input 0
        next_state_index0 = binary_to_int(next_state0, Kc-1)


        state_diagram[i, 3] = next_state_index0

         arr1 = arr[i, :]
         arr1 = concatenate(1, arr1)
         op1 = mod(matrix_multiplication(G, arr1), 2)            // Compute the output for transition on input 1
         state_diagram[i, 2] = binary_to_int(op1, n)


    next_state1 = []

    for j = 1 to Kc-1
        next_state1 = concatenate(next_state1, arr1[j])          // Determine next state for transition on input 1
        next_state_index1 = binary_to_int(next_state1, Kc-1)


        state_diagram[i, 4] = next_state_index1


return state_diagram
```

Viterbi Decoding(Hard)

```
decoding_hard(s, Kc, n, demod_seq, inp_len)

    rows = 2^(Kc-1)                                  // Calculate the number of rows and columns in the trellis
    cols = inp_len + 1

    val_arr = initialize_array(rows, cols, 1000)     // Initialize 2-D arrays to store values, previous states, and previous values

    prev_state = initialize_array(rows, cols, -1)

    prev_inp = initialize_array(rows, cols, -1)

    val_arr[1][1] = 0                                // Set initial value and state for the starting point
    prev_state[1][1] = -1

    for j from 1 to cols - 1                         // Iterate over each column of the trellis

        x = extract_bits(demod_seq, n, j)            // Extract bits from the demodulated sequence corresponding to the current column


        for i from 1 to rows                         // Iterate over each state in the trellis

            if val_arr[i][j] != 1000                 // Check if the current state has a valid value
                op0 = s[i][1]
                ns0 = s[i][3] + 1

                op0_bin = int_to_binary(op0, n)
                op0_bin = op0_bin'
```

```
hd0 = 0                                                    // Compute Hamming distance for transition 0
for k from 1 to length(x)
    if x[k] != op0_bin[k]
        hd0 = hd0 + 1

if hd0 + val_arr[i][j] < val_arr[ns0][j + 1]               // Update values if the transition improves the metric
    val_arr[ns0][j + 1] = hd0 + val_arr[i][j]
    prev_state[ns0][j + 1] = i
    prev_inp[ns0][j + 1] = 0

op1 = s[i][2]                                              // Compute Hamming distance for transition 1
ns1 = s[i][4] + 1
op1_bin = int_to_binary(op1, n)
op1_bin = op1_bin'
hd1 = 0

for k from 1 to length(x):
    if x[k] != op1_bin[k]:
        hd1 = hd1 + 1

if hd1 + val_arr[i][j] < val_arr[ns1][j + 1]               // Update values if the transition improves the metric
    val_arr[ns1][j + 1] = hd1 + val_arr[i][j]
    prev_state[ns1][j + 1] = i
    prev_inp[ns1][j + 1] = 1
```

```
 i = 1

decoded_seq = [ ]

for j from cols down to 2                                   // Backtrack through the trellis to find the most likely sequence

    decoded_seq = [decoded_seq prev_inp[i][j]]
    i = prev_state[i][j]


outputArg = reverse(decoded_seq)                            // Return the decoded sequence
```

Viterbi Decoding(Soft)

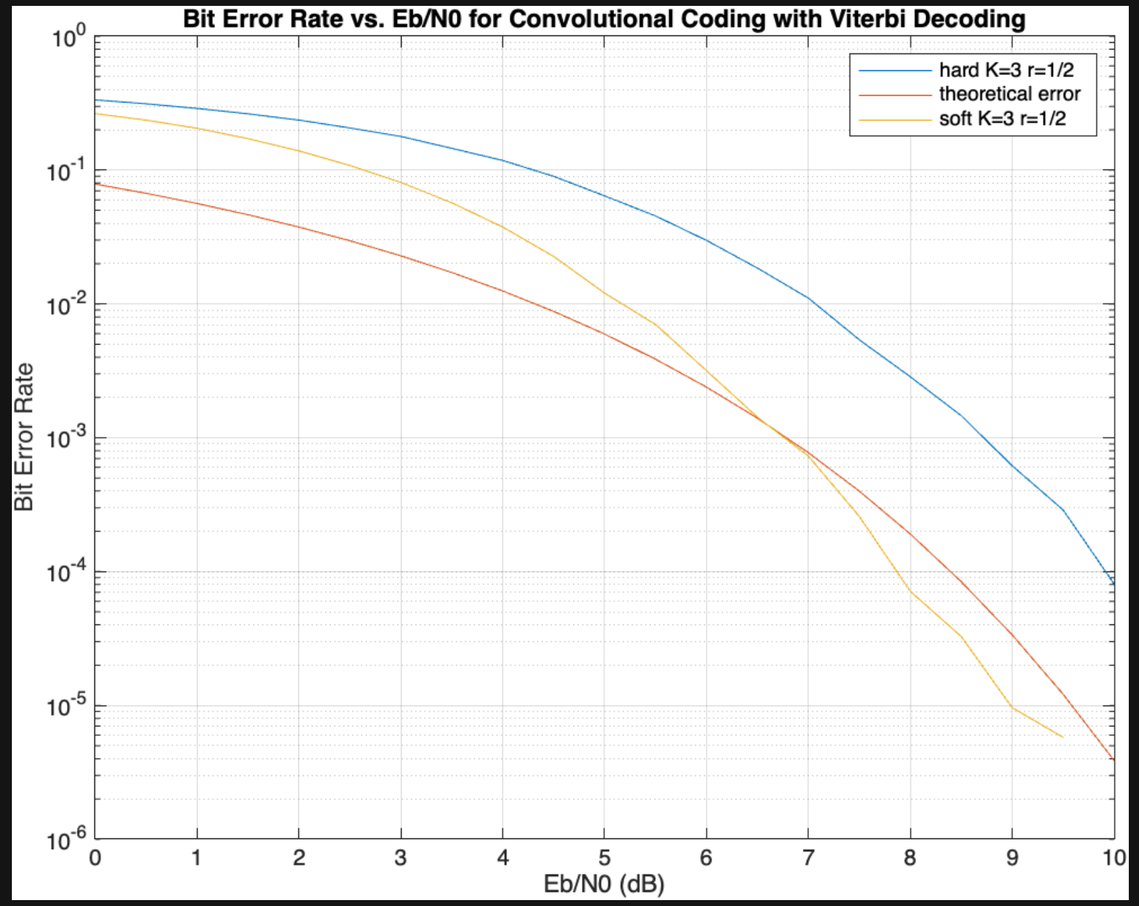
```
decoding_soft(s, Kc, n, demod_seq, inp_len)

  rows = 2^(Kc-1)                                  // Calculate the number of rows and columns in the trellis
  cols = inp_len + 1

  val_arr = 1000 * ones(rows, cols)                // Initialize arrays to store values, previous states,
  prev_state = -1 * ones(rows, cols)
  prev_inp = -1 * ones(rows, cols)                 // Initialize arrays to store previous inputs

  val_arr(1, 1) = 0                                // Set initial value and state for the starting point
  prev_state(1, 1) = -1

  for j from 1 to cols - 1                         // Iterate over each column of the trellis

    x = []                                         // Extract bits from the demodulated sequence for j

    for i from n*j - (n - 1) to n*j
      x = [x, demod_seq(i)]
```

```
for i from 1 to rows                                                    // Iterate over each state in the trellis

    if val_arr(i, j) != 1000                                            // Check if the current state has a valid value

        op0 = s(i, 1)                                                   // Compute path metric for transition 0
        ns0 = s(i, 3) + 1
        op0_bin = int_to_binary(op0, n)
        op0_bin = op0_bin'
        op0_bin = 1 - 2 * op0_bin
        path_metric0 = sum((x - op0_bin).^2)

        if path_metric0 + val_arr(i, j) < val_arr(ns0, j + 1)           // Update values if the transition improves the metric
            val_arr(ns0, j + 1) = path_metric0 + val_arr(i, j)
            prev_state(ns0, j + 1) = i
            prev_inp(ns0, j + 1) = 0

        op1 = s(i, 2)                                                   // Compute path metric for transition 1
        ns1 = s(i, 4) + 1
        op1_bin = int_to_binary(op1, n)
        op1_bin = op1_bin'
        op1_bin = 1 - 2 * op1_bin
        path_metric1 = sum((x - op1_bin).^2)

        if path_metric1 + val_arr(i, j) < val_arr(ns1, j + 1)           // Update values if the transition improves the metric
            val_arr(ns1, j + 1) = path_metric1 + val_arr(i, j)
            prev_state(ns1, j + 1) = i
            prev_inp(ns1, j + 1) = 1
```

```
i = 1

decoded_seq = [ ]

for j from cols down to 2                                   // Backtrack through the trellis to find the most likely sequence

    decoded_seq = [decoded_seq, prev_inp(i, j) ]
    i = prev_state(i, j)


outputArg = reverse(decoded_seq)                            // Return the decoded sequence
```

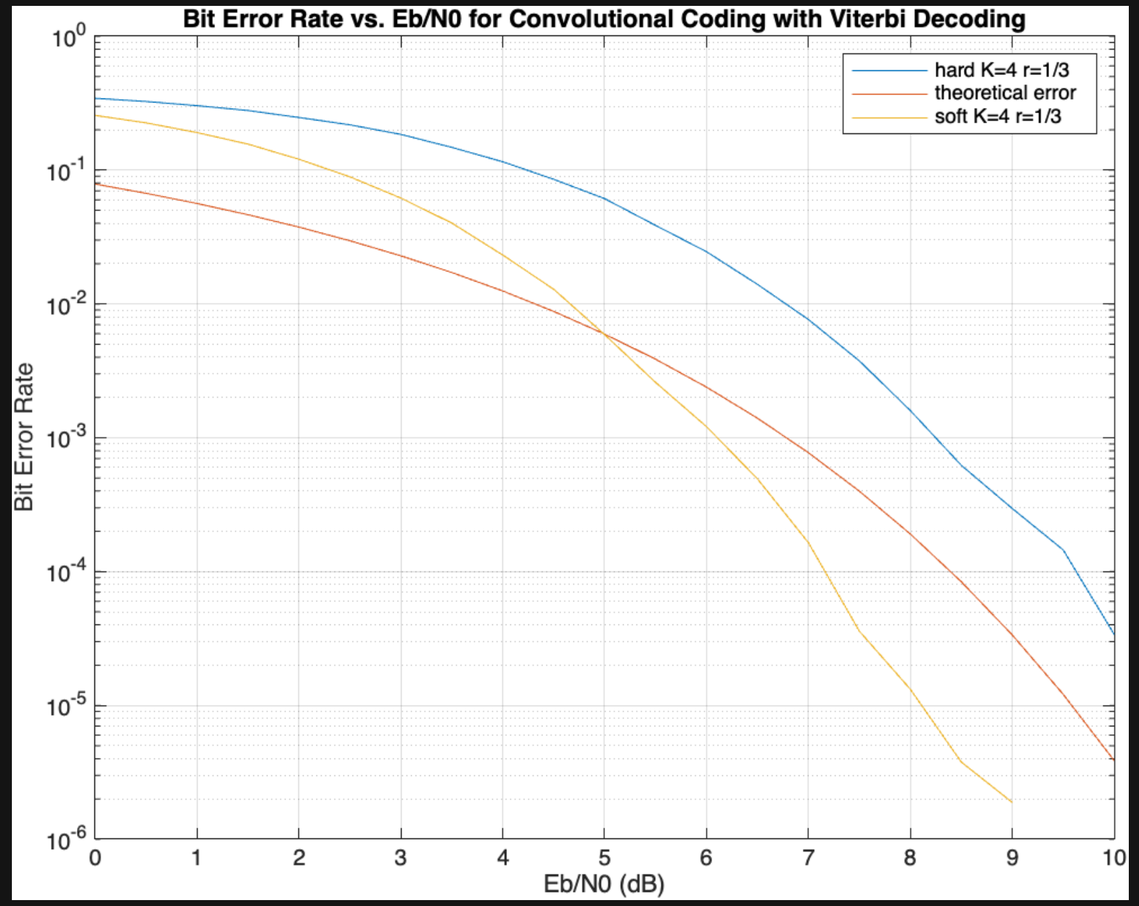
Simulation result: r=1/2, Kc=3

Input Length = 50
Number of Simulations = 10000



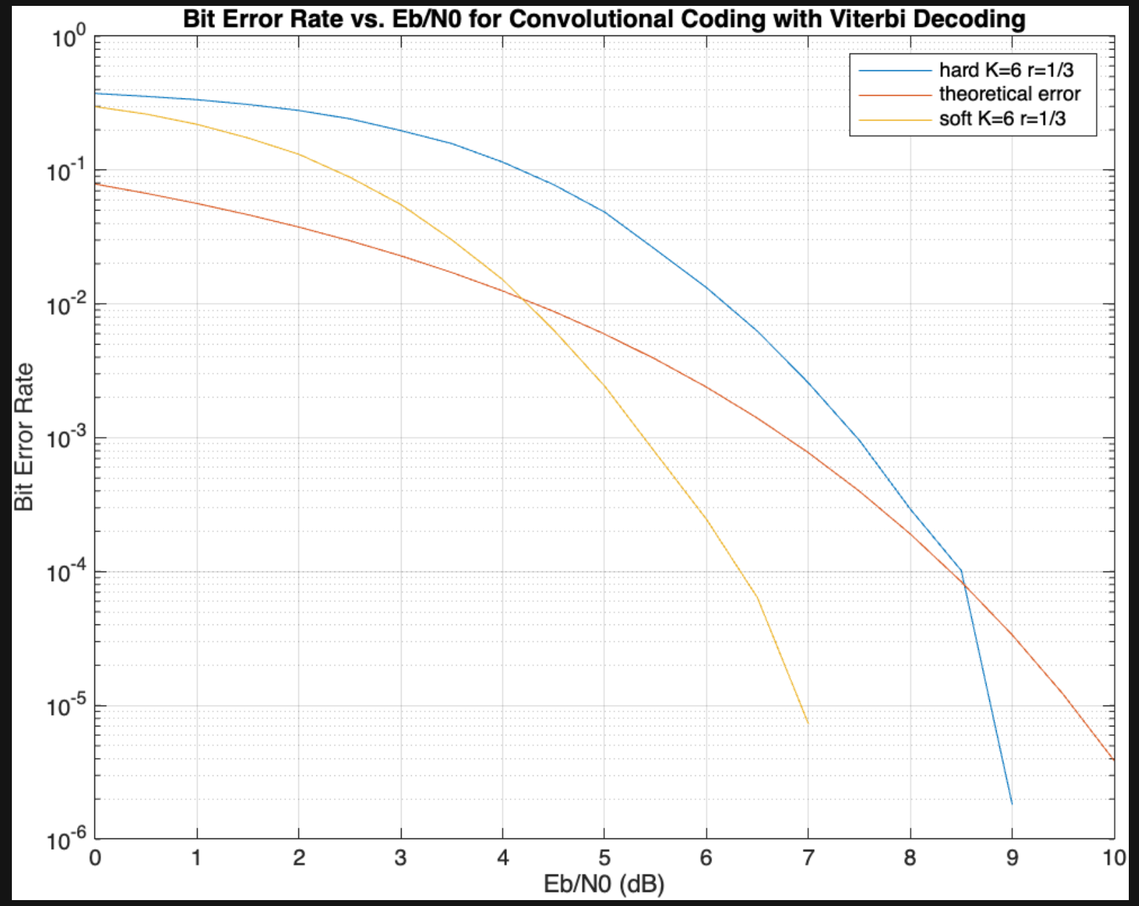
Simulation result: r=1/3, Kc=4

Input Length = 50
Number of Simulations = 10000



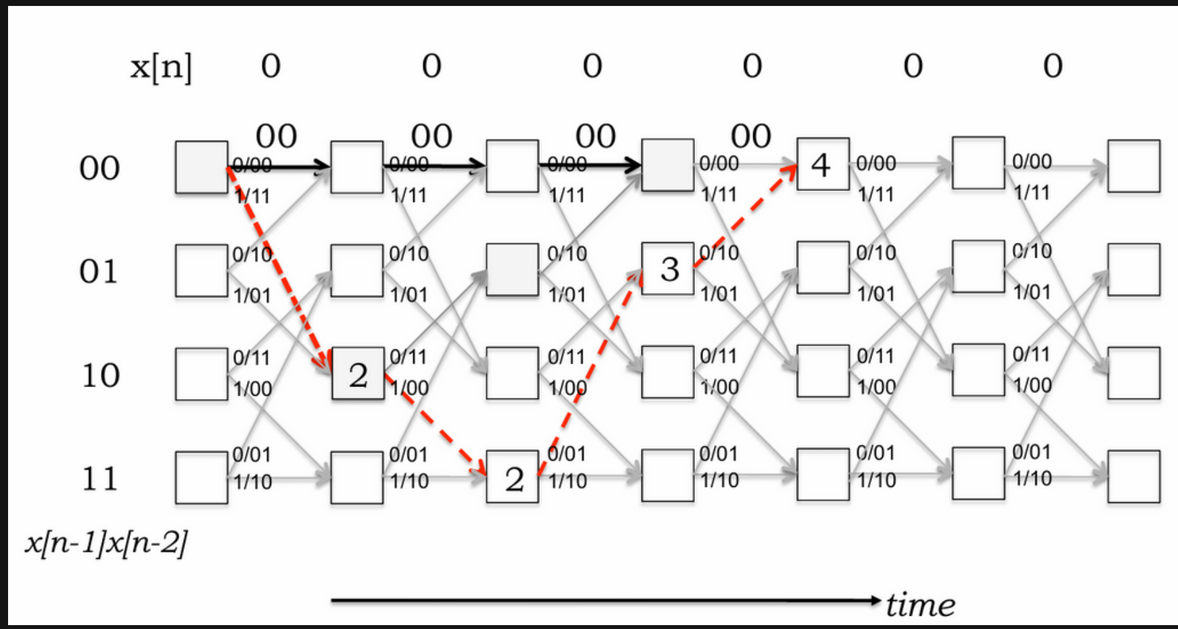
Simulation result: R=1/3, Kc=6

Input Length = 50
Number of Simulations = 10000



Free Distance Instead of Hamming Distance

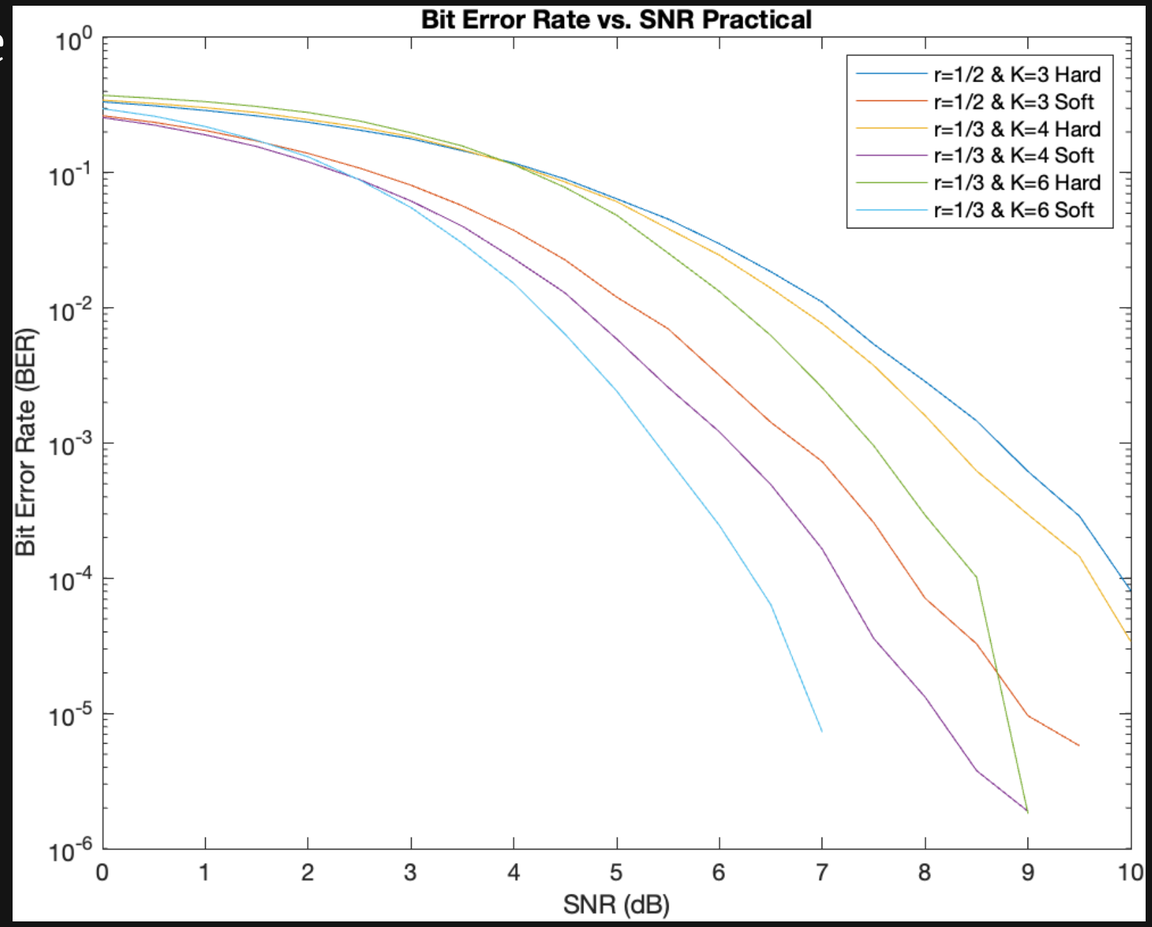
- In context of convolutional codes, the smallest hamming distance between two valid codewords is called free distance
- It is the difference in path metrics between all zero output and path with the smallest non zero path metric going from initial 00 state to some future 00 state
- In this example shown the free distance is 4 and as tc (error correction) is defined as floor((f-1)/2) where f is free distance
- More is the free distance of a convolutional coding scheme more it is error correction capability



Comparing bit error rate (BER) with SNR

- As we can see from the graph the bit error rate is lowest for K=6, then K=4, then K=3, this is because the free distance for subsequent values of k is more as compared to lower ones so their error correction capability is more and as a result of high error correction capability bit error rate is low.

- At low SNR values, the BER is relatively high due to high impact of noise on the received symbols, hence as SNR increases BER also decreases rapidly.

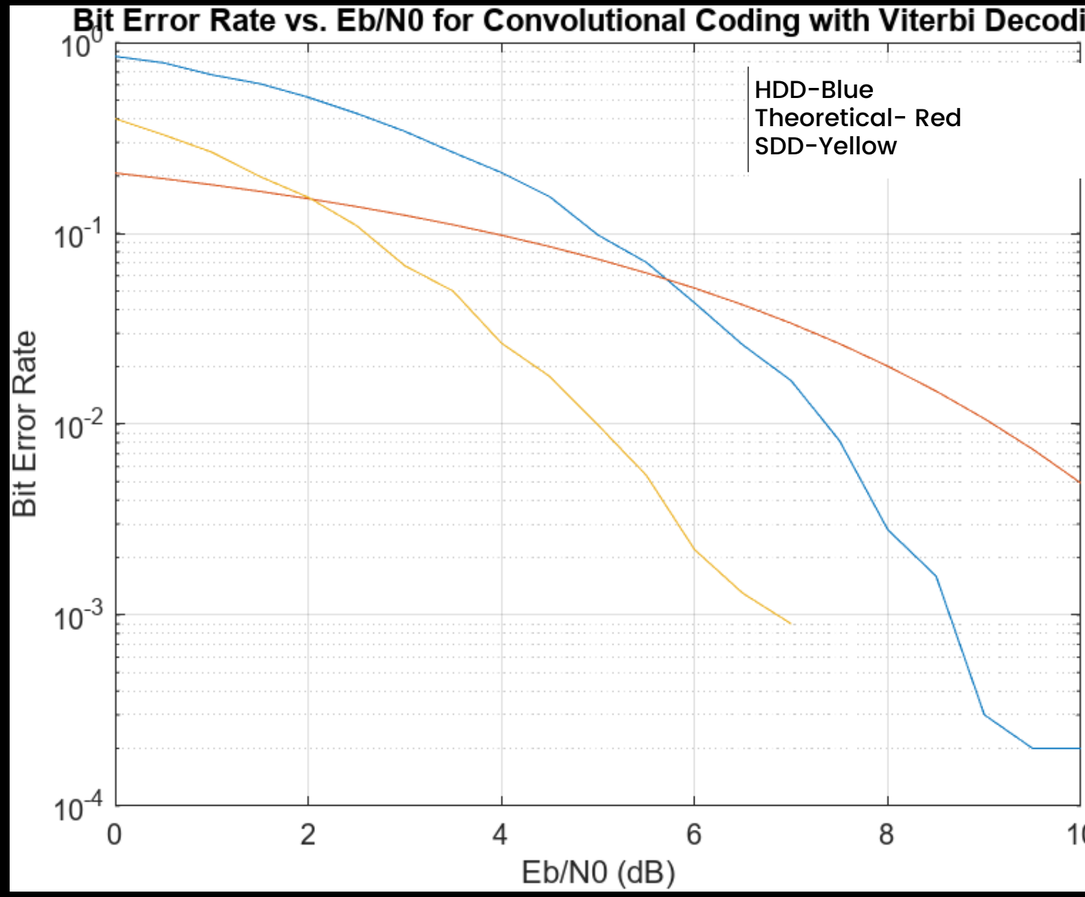


Hard v/s Soft Viterbi Decoding

- As we can see from the graph soft decoding has lower BER as compared to hard decoding, this is because HDD tends to make binary decisions solely on the received signal which can lead to error propagation in noisy environments whereas SDD considers the likelihood of various transmitted sequences which can help in mitigating effect of noise and reducing error propagation

- At higher Signal-to-Noise Ratio (SNR) values, Hard Decision Decoding (HDD) exhibits an asymptotic behavior in the Bit Error Rate (BER) curve, where further increases in SNR do not significantly reduce the BER. This phenomenon stems from the accumulation of errors due to incorrect decisions made in earlier stages of decoding.

- Besides these SDD also provides higher gain(in dB) as compared to HDD and also have low sensitivity to signal distortion as compared to HDD



References

[1] Lecture Slides of Channel Coding by Prof. Yash Vasavada

[2] J. G. Proakis. Digital Communications. McGraw-Hill, 4th edition, 1995.

[3] Matthew Valenti and Bibin Baby John. Introduction to Communication Systems
A Lecture on Convolution Coding and Viterbi Decoding. DA-IICT, Winter 2024.

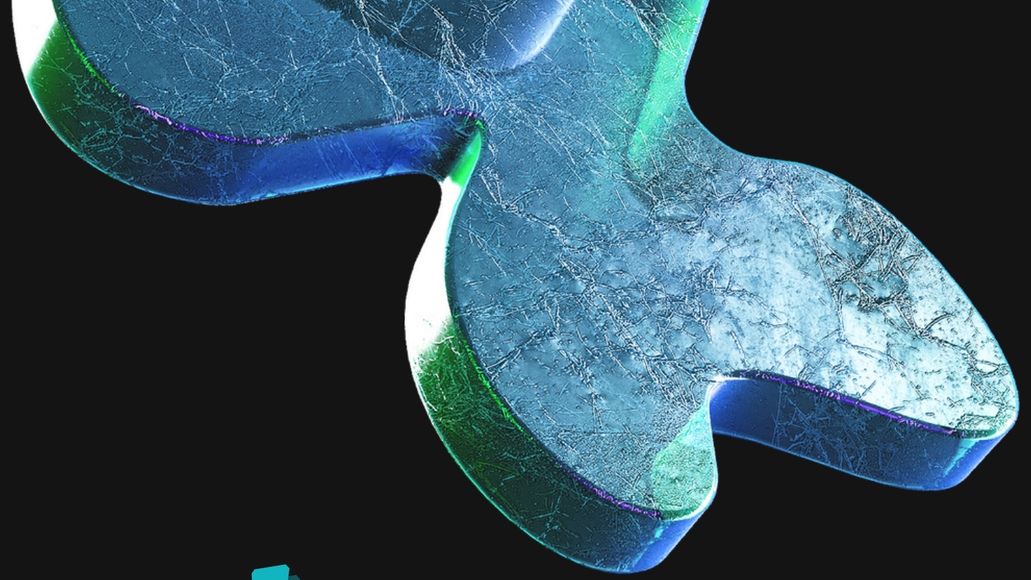
[4] MIT Lectures: https://web.mit.edu/6.02/www/f2010/handouts/lectures/L8.pdf

[5] Krishna Sankar Article on Channel Encoding and Viterbi decoding:
https://dsplog.com/2009/01/04/convolutional-code/

[6] Subhramanyam K N Video Lectures on Youtube: https://shorturl.at/hZ136

Group Members

1. Parjanya Rajput 202201115
2. Rushi Makadiya 202201117
3. Gireesh Reddy 202201122
4. Tasmay Patel 202201129
5. Praneel Vania 202201131
6. Vidhan Chavda 202201133
7. Dip Baldha 202201142
8. Meet Andharia 202201145
9. Kashvi Bhanderi 202201149
10. Rishi Godhasara 202201154
11. Siddhant Kotak 202201410



Thank You!