Digital Communication Lab

Laboratory report submitted for the partial fulfillment
of the requirements for the degree of

Bachelor of Technology
in
Electronics and Communication Engineering

by

Mohit Akhouri - 19ucc023

Course Coordinator
Dr. Nikhil Sharma

Department of Electronics and Communication Engineering
The LNM Institute of Information Technology, Jaipur

September 2021

Copyright © The LNMIIT 2021
All Rights Reserved

Contents

Chapter *8*

Experiment - 8

8.1 Name of the Experiment

Performance analysis of Convolutional Encoding and Viterbi Decoding

8.2 Software Used

- MATLAB

8.3 Theory

8.3.1 About Convolutional code :

In telecommunication, a **convolutional code** is a type of error-correcting code that generates parity symbols via the sliding application of a boolean polynomial function to a data stream. The sliding application represents the 'convolution' of the encoder over the data, which gives rise to the term 'convolutional coding'. The sliding nature of the convolutional codes facilitates trellis decoding using a time-invariant trellis. **Time invariant trellis decoding** allows convolutional codes to be **maximum-likelihood soft-decision decoded** with **reasonable complexity**.

Convolutional codes are often described as **continuous**. However, it may also be said that convolutional codes have arbitrary block length, rather than being continuous, since most real-world convolutional encoding is performed on blocks of data. Convolutionally encoded block codes typically employ termination. The arbitrary block length of convolutional codes can also be contrasted to classic block codes, which generally have fixed block lengths that are determined by **algebraic properties**.

The code rate of a convolutional code is commonly modified via **symbol puncturing**. The performance of a punctured convolutional code generally scales well with the amount of parity transmitted. The ability to perform economical soft decision decoding on convolutional codes, as well as the block length and code rate flexibility of convolutional codes, makes them very popular for **digital communications**.

8.3.2 About Convolutional Encoding :

To convolutionally encode data, start with **k memory registers**, each holding one input bit. Unless otherwise specified, all memory registers start with a value of 0. The encoder has n modulo-2 adders (a modulo 2 adder can be implemented with a single Boolean **XOR gate**, where the logic is: 0+0 = 0, 0+1 = 1, 1+0 = 1, 1+1 = 0), and n generator polynomials - one for each adder. An input bit m_1 is fed into the leftmost register. Using the **generator polynomials** and the existing values in the remaining registers, the encoder outputs n symbols. These symbols may be transmitted or punctured depending on the desired code rate. Now bit shift all register values to the right (m_1 moves to m_0, m_0 moves to m_{-1}) and wait for the next input bit. If there are no remaining input bits, the encoder continues shifting until all registers have returned to the zero state (**flush bit termination**).

8.3.2.1 1/2 Convolutional Encoder :

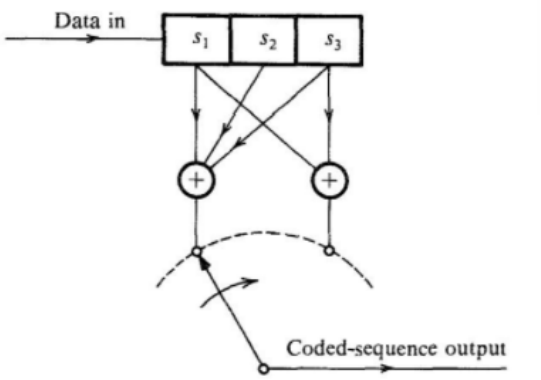

Figure 8.1 1/2 Convolutional Encoder

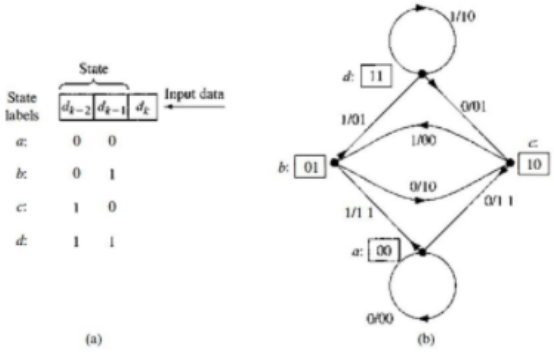

Figure 8.2 State and State Transition Diagram of 1/2 Convolutional Encoder

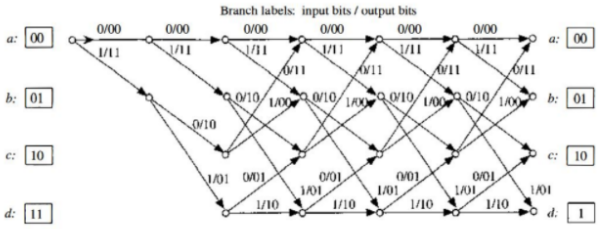

Figure 3: Trellis diagram for the encoder in Fig. 1

Figure 8.3 Trellis Diagram for 1/2 Convolutional Encoder

8.3.2.2 1/3 Convolutional Encoder :

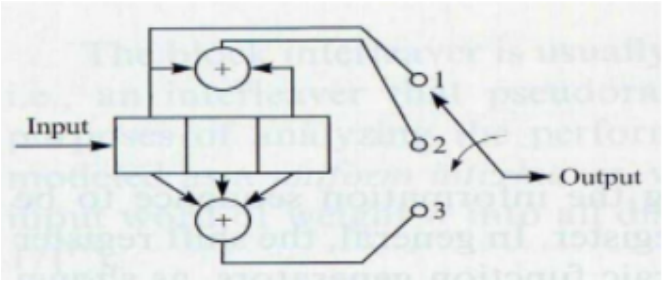

Figure 8.4 1/3 Convolutional Encoder

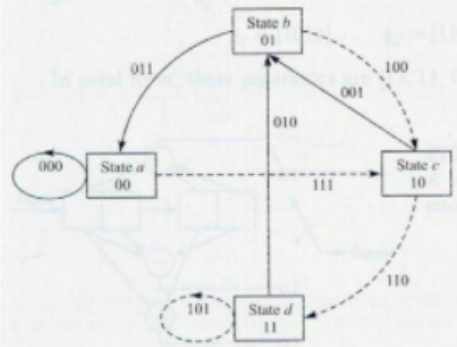

Figure 8.5 State Transition Diagram of 1/3 Convolutional Encoder

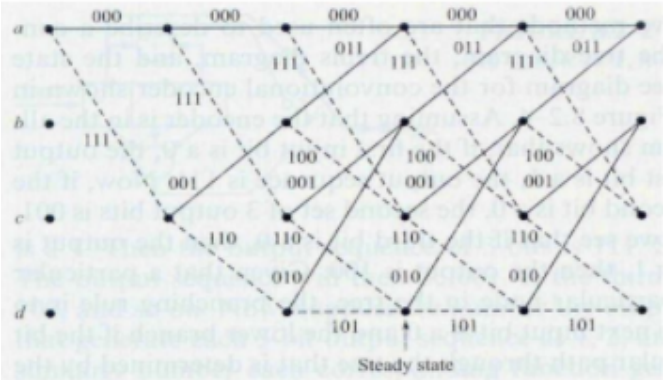

Figure 8.6 Trellis Diagram for 1/3 Convolutional Encoder

8.3.3 About Convolutional Decoding :

Several algorithms exist for decoding convolutional codes. For relatively small values of k, the **Viterbi algorithm** is universally used as it provides maximum likelihood performance and is **highly parallelizable**. Viterbi decoders are thus easy to implement in **VLSI hardware** and in software on CPUs with **SIMD instruction sets**.

Longer constraint length codes are more practically decoded with any of several sequential decoding algorithms, of which the **Fano algorithm** is the best known. Such codes were used in the Pioneer program of the early 1970s to Jupiter and Saturn, but gave way to shorter, Viterbi-decoded codes, usually concatenated with large **Reed–Solomon error correction codes** that steepen the overall bit-error-rate curve and produce extremely **low residual undetected error rates**.

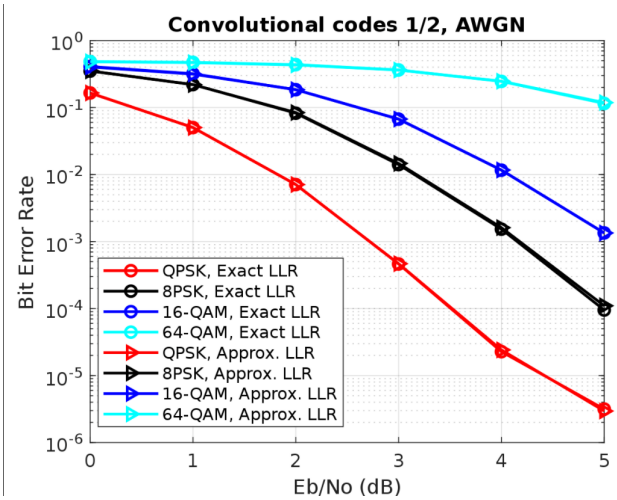

Figure 8.7 Bit error ratio curves for convolutional codes with different options of digital modulations

8.4 Code and Results

8.4.1 Generating Convolutional Encoded symbols through 2 methods :

```
% 19ucc023
% Mohit Akhouri
% Observation 1 - Generating Convolutional Encoded symbols ( via both
 INBUILT
% FUNCTION and MATLAB CODING )

% In this code, we will generate convolutionally encoded codewords
 using
% inbuilt function poly2trellis and also via MATLAB coding with help
 of
% arrays and loops.

clc;
clear all;
close all;

n = 5; % Number of random numbers to be generated
data = randi([0,1],1,n); % Generates a random sequence of 0's and 1's
 of length 1xn

% Display of random sequence of binary digits 0's and 1's generated
disp('The Random data generated is : ');
disp(data);

% Using Inbuilt function poly2trellis to generate convolutional codes
codes_trellis = poly2trellis(3,[7 5]); % Generating Trellis structure
 for convolutional code
codeword_inbuilt = convenc(data,codes_trellis); % Generating
 convolutional codeword with the help of trellis structure

% Display of convolutional codeword generated via INBUILT FUNCTION
% poly2trellis and convenc
disp('The Convolutional Codeword generated via INBUILT FUNCTION is :
 ');
disp(codeword_inbuilt);

% Using MATLAB coding to generate convolutional codes
codeword_matlab_coding = zeros(1,2*n); % To store the codeword
 generated
curr_data = zeros(1,3); % Temporary array to store the binary digits 0
 and 1

% Main Loop algorithm for the calculation of convolutional codewords
for i = 1:n
    curr_data(2:3) = curr_data(1:2); % Replacing bits 2 and 3 with
 bits 1 and 2
    curr_data(1) = data(i); % Starting point of codeword

    % Calculation of remaining convolutional codewords with the help
 of
    % 'mod' function
```

Figure 8.8 Part 1 of the Code for Generation of Convolutional Encoded symbols

```
    codeword_matlab_coding(2*i-1) = mod(curr_data(1) + curr_data(2) +
 curr_data(3) ,2);
    codeword_matlab_coding(2*i) = mod(curr_data(1) + curr_data(3) ,2);
end

% Display of convolutional codewords generated via MATLAB coding
disp('The Convolutional Codeword generated via MATLAB CODING is : ');
disp(codeword_matlab_coding);
```

Published with MATLAB® R2020b

Figure 8.9 Part 2 of the Code for Generation of Convolutional Encoded symbols

```
Command Window
The Random data generated is :
     0     1     0     1     0

The Convolutional Codeword generated via INBUILT FUNCTION is :
     0     0     1     1     1     0     0     0     1     0

The Convolutional Codeword generated via MATLAB CODING is :
     0     0     1     1     1     0     0     0     1     0
```

Figure 8.10 Display of **(a)** Random Code Sequence **(b)** Encoded Codeword using INBUILT function **(c)** Encoded Codeword using MATLAB coding

8.4.2 Performance analysis of convolutional encoded systems for BPSK Modulation :

```
% 19ucc023
% Mohit Akhouri
% Observation 2 - Performance analysis of convolutional encoded system

% In this code , we will do the performance analysis of convolutional
% encoded system and we plot the BER vs. SNR(dB)

clc;
clear all;
close all;

n = 1000000; % Size of random sequence of 0's and 1's

rate_1 = 1/2; % Rate for 1/2-Convolutional Encoder
rate_2 = 1/3; % Rate for 1/3-Convolutional Encoder

data = randi([0,1],1,n); % Generating Random Sequence of 0's and 1's

trellis_1_by_2 = poly2trellis(3,[7 5]);  % Trellis Structure of 1/2
 Convolutional Encoder
trellis_1_by_3 = poly2trellis(3,[4 5 7]); % Trellis Structure of 1/3
 Convolutional Encoder

codeword_1_by_2 = convenc(data,trellis_1_by_2); % Codeword generated
 for 1/2 Convolutional Encoder
codeword_1_by_3 = convenc(data,trellis_1_by_3); % Codeword generated
 for 1/3 Convolutional Encoder

max_SNR = 10; % Range for Maximum SNR value
BER_coded_using_1_by_2 = zeros(1,max_SNR); % BER in case of 1/2
 Convolutional Encoder
BER_coded_using_1_by_3 = zeros(1,max_SNR); % BER in case of 1/3
 Convolutional Encoder

BER_uncoded = zeros(1,max_SNR); % BER in case of uncoded system
SNR_dB = zeros(1,max_SNR); % Array to store the SNR values ( in dB )

% Main loop algorithm for calculation of BPSK Modulated waveform and
 Bit
% error rates for two types of convolutional encoders
for i = 1:max_SNR
    SNR_dB(i) = i; % SNR value (without any units)
    SNR = 10^(i/10); % SNR value (in dB)

    % Calculation of sigma factor for calculation of Noise
    Sigma_1 = sqrt(1/(2*rate_1*SNR));
    Sigma_2 = sqrt(1/(2*rate_2*SNR));
    % Calculation of Noise in case of BPSK Modulation
    Noise_1 = Sigma_1*randn(1,n/rate_1);
    Noise_2 = Sigma_2*randn(1,n/rate_2);

    % Computing Transmitted and Received codewords in case of BPSK
```

Figure 8.11 Part 1 of the Code for Performance analysis of convolutional encoded systems

```
    % Modulation for two types of encoder - 1/2 convolutional encoder
 and
    % 1/3 convolutional encoder
    transmitted_1 = 2*codeword_1_by_2 -1;
    received_1 = transmitted_1 + Noise_1;

    transmitted_2 = 2*codeword_1_by_3 -1;
    received_2 = transmitted_2 + Noise_2;

    % Detection of received codeword via HARD DECISION DECODER

    recovered_codeword_1_by_2 = (received_1>0); % Received codeword
 from 1/2 convolutional encoder
    recovered_codeword_1_by_3 = (received_2>0); % Received codeword
 from 1/3 convolutional encoder

    % Using 'vitdec' function to convolutionally decode binary data
    recovered_data_1_by_2 =
 vitdec(recovered_codeword_1_by_2,trellis_1_by_2,20,'trunc','hard');
    recovered_data_1_by_3 =
 vitdec(recovered_codeword_1_by_3,trellis_1_by_3,20,'trunc','hard');

    % Calculation of Net error rate in case of two types of encoders
    NER_hard_1_by_2 = sum(data~=recovered_data_1_by_2);
    NER_hard_1_by_3 = sum(data~=recovered_data_1_by_3);

    % Calculation of Bit error rates ( BER ) for both the encoders
    BER_coded_using_1_by_2(i) = NER_hard_1_by_2/n;
    BER_coded_using_1_by_3(i) = NER_hard_1_by_3/n;

    % Calculation of Bit error rate ( BER ) in case of uncoded system
    BER_uncoded(i) = qfunc(sqrt(SNR));
end

% Plots of BER vs. SNR (dB) for both types of encoders
figure;
semilogy(SNR_dB,BER_coded_using_1_by_2);
hold on;
semilogy(SNR_dB,BER_coded_using_1_by_3);
hold on;
semilogy(SNR_dB,BER_uncoded);
xlabel('SNR (dB) ->');
ylabel('BER ->');
title('19ucc023 - Mohit Akhouri','Plots of the BER vs. SNR(dB) for 1/2
 Convolutional Encoder and 1/3 Convolutional Encoder');
legend('BER in case of 1/2 Convolutional Encoder','BER in case of 1/3
 Convolutional Encoder','BER in case of uncoded system');
grid on;
hold off;
```

Published with MATLAB® R2020b

Figure 8.12 Part 2 of the Code for Performance analysis of convolutional encoded systems

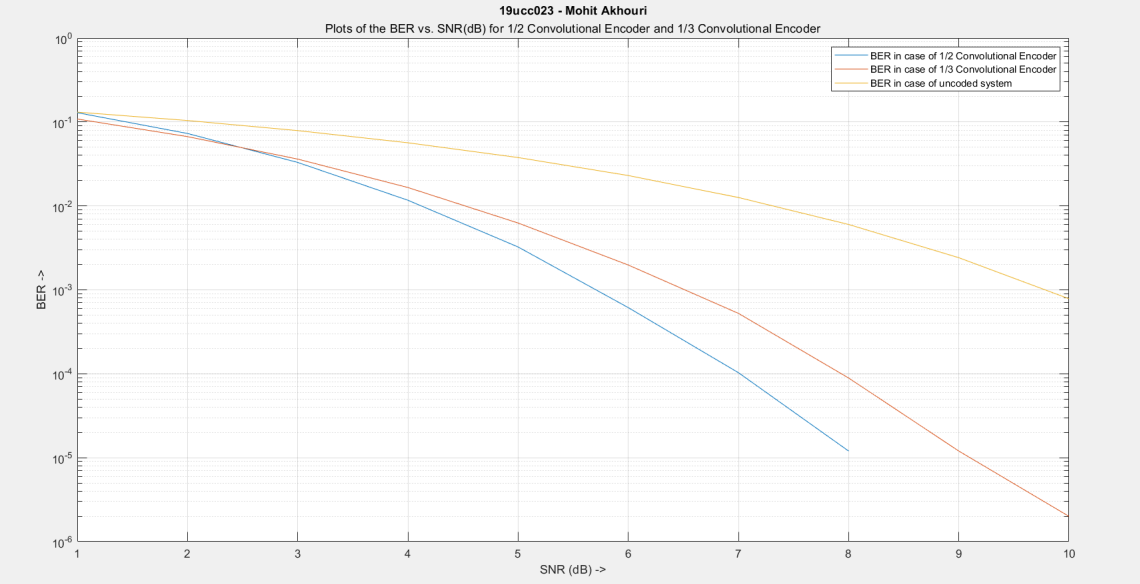

Figure 8.13 Plots of BER vs. SNR(dB) for Convolutionally encoded systems and uncoded system

8.5 Conclusion

In this experiment , We learnt about the **Convolutional Encoder** and **Convolutional Decoder**. We learnt about various concepts like **Trellis Diagram** and **Viterbi Decoding**. We implemented two types of convolutional encoder - for rates 1/2 and 1/3. We used the MATLAB inbuilt function - **poly2trellis** and designed MATLAB code for the construction of convolutional encoder. We performed the BPSK modulation and decoded the codewords using MATLAB function **vitdec**. We calculated values of BER for different values of SNR (dB) and plotted the graph between BER and SNR. We observed the graph for three types of cases - uncoded system , encoded system for 1/2 convolutional encoder and encoded system for 1/3 convolutional encoder. We learnt about many new MATLAB functions like - **poly2trellis**, **vitdec** and **mod**.