DESIGN OF DIGITAL COMMUNICATION SYSTEM

EE 5368 COURSE PROJECT - FALL 2010

INSTRUCTOR:

Dr. TRACY JING LIANG

PROJECT REPORT BY:

1) VASUDEVAN MURALI #1000723391
2) ARUNKUMAR KUPPUSAMY #1000728387
3) THAMIZH SELVAN ILAVARASU #1000 726055

INDEX

PROBLEM STATEMENT

In this project we were requested to perform a system-level design for a mobile satellite-based wireless user terminal. The specifications for its general packet data channel (GPDCH, a logic channel) are listed as follows

- Modulation scheme: QPSK with bits to symbol mapping
  $00 \rightarrow 1$, $01 \rightarrow j$, $11 \rightarrow -1$, $10 \rightarrow -j$
- Burst structure:

Guard Symbols (5 SYMBOLS)	Unique word (40 SYMBOLS)	Information symbols (Payload) (800 SYMBOLS)	Guard symbols (5 SYMBOLS))

Fig.1 burst structure

and transmitting such a burst needs 0.5ms.

- Since it's a satellite-based system, Rician flat fading is assumed.
- Sampling rate of 16 samples / symbol should be used based on hardware requirements.
- The roll-off factor for square root raised cosine filter is 0.35.

Design a simulation reference system for GPDCH subject to Rician flat fading using MATLAB or C, and design a demodulator for such a fading channel with QPSK modulation. And to provide a report describing our design with the
following performance plots:

1. The Rician flat fading channel is used with K = 7dB and Doppler shift (fading bandwidth) fd = 20Hz. Plot the raw bit error rate (BER) obtained in the simulation program at Eb/No = 1, 2, …, 6, 7 dB and compare with the theoretical raw BER at 1, 2, …, 6, 7 dB. The theoretical raw BER is given by: BER = [0.09 0.0699 0.0527 0.0387 0.0277 0.0193 0.0135].

2. The Rician flat fading channel is used with K = 12dB and fd = 100Hz. The raw bit error rate (BER) obtained in the simulation program at Eb/No = 1, 2, …,6, 7 dB will be plotted and compared with the theoretical raw BER at 1, 2, …,6, 7 dB. The theoretical BER is given by:BER = [0.06565 0.04664 0.03124 0.01958 0.011396 0.0061246 0.00302]

3. The Rician flat fading channel is used with K = 200dB and fd = 0Hz. The raw bit error rate (BER) obtained in the simulation program at Eb/No = 1, …, 6 ,7dB will be plotted and compared with the theoretical raw BER:) (0 *NEQ b*

4. If four-path diversity with maximal ratio combining (MRC) is used, repeat steps (1)(2)(3) and observe how many dB gains have achieved in each case.

5. If convolutional codes with coding rate ½ and with connections (in octal number) represented in binary [001 011 011] and [001 111 001] in the encoder and viterbi decoder are used in the design, repeat (1)(2)(3) and observe how many dB gains have achieved in each case.

6. Puncturing (b3 is punctured in every four bits b0-b3) and block interleaver (10×5) are used in the design. What are the performances for the above three channels? How many dB losses comparing to the same channel in (5) for each channel? What's the main advantage of this scheme?

7. From the above systematic studies, derive the conclusions.

INTRODUCTION

Wireless communication refers to the successful transmission and reception of information through air media. Though we transmit only information bits , the physical medium has several noise components that gets added to our information signal .As a result the signal degrades in quality and error gets introduced in the communication channel . Thus the channel is always associated with noise and we always get a faded channel at the receiver end. One such fading is called as rician fading and the corresponding channel is called as rician fading channel.

In this project we have simulated a wireless communication channel with rician fading and to overcome this fading effect we use Block Phase Sequence Estimation and Phase compensation techniques.

BLOCK DIAGRAM

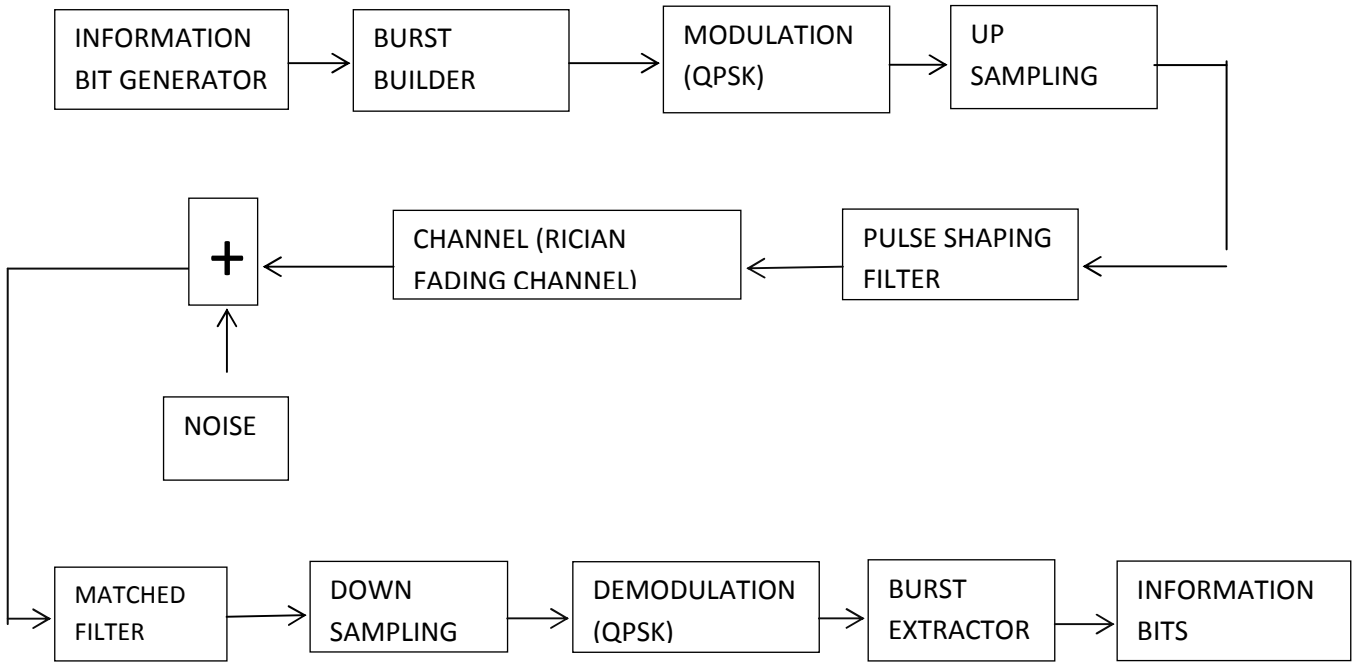


Fig 2 Block Diagram of the Basic System

TRANSMITTER

1) INFORMATION BIT GENERATOR
2) BURST BUILDER
3) MODULATION
4) UP SAMPLING
5) PULSE SHAPING FILTER

1) Information bit generator

This block generates information bits, Guard bits and unique word. As per the design payload of 1000bits, Guard band 12 bits and Unique word 80 bits are generated using the Randint function available in MATLAB®. The entire length of the burst is 6+80+1000+6 = 1092 bits.

2) Burst builder

This block combines the information bits, guard bits and unique word bits to form a burst structure comprising of 546 bits in total. Thus the information is sent in the form of bursts where guard bits (3) are added at the beginning and the end of the burst structure to ensure that the information and the unique word are not damaged during transmission.

3) Modulation

The modulation used in this communication system is QPSK (Quadrature Phase Shift Keying).
Here bits to symbols mapping takes place and the mapping is as follows,
00 → 1
01 → j
11 → -1
10 → -j

4) Up sampling

In digital communication the signal is either 0 or 1 and the signal is discrete , to obtain a smooth transition between two symbols we use up sampling . There are 576 bits (6+40+500+6) in bursts. These are all in symbols in symbol domain.Since the sampling rate we use is 16 samples/second we add 15 zeros or one's in between two symbols. But in our project we introduce zeroes inbetween the signals. Thus now our burst has 8736 bits (546*16). This reduces the effect of noise on the information symbols contained in the message.

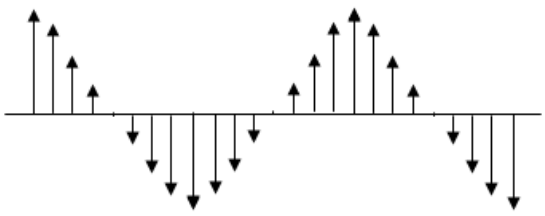


Fig.3 Up sampling

5) Pulse shaping filter or Raised cosine filter

Since we have added zeros in between two symbols we have to assign values to these newly appended bits. We have used the RCOSINE filter available in MATLAB®, to achieve the desired result. Shape of RCOSINE filter is shown below.

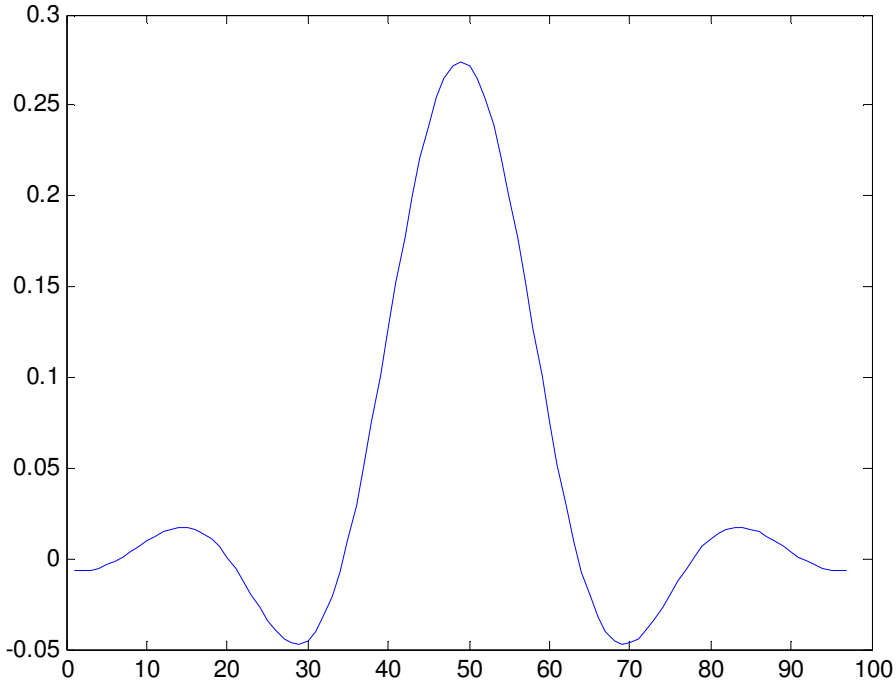


Fig.4: RCOSINE plot

We have a roll off factor of **0.35** for our project because in time domain we have infinite SINC function, but practically we cannot implement an infinite length function. Hence we roll it off using the roll off factor. Now, after the convolution of the up sampled signals with the pulse shaping filter we get the total length of the convolved signal as L+M-1 (convolution property).

Where,
M is the Length of burst and L is the Length of filter.

We should make the length of convolved signal same as input signal. So we will cut L-1 extra samples. Thus we remove (L-1)/2 samples at the start of the burst and (L-1)/2 samples at the end of the burst. As we are adding guard bits, cutting symbols at the start and end would not affect the information in anyway.

Channel

The channel assumed for the proposed satellite based system is Rician flat fading channel. Rician fading are given by the fading factor (k) and the Doppler frequency (fd). The simulation of the Rician channel is done by using sum of sinusoids using the Jake's model.

RECEIVER

1) Matched filter
2) Down Sampling
3) Demodulation
4) Burst extractor

1) Matched Filter

It is the reciprocal for the Pulse shaping filter used in the transmitter section. It removes the effects of Pulse shaping filter from the received symbols. It introduces M additional samples which are truncated at the front and end of the convoluted output to maintain it at 546*16 samples.

2) Down sampling

The output of matched filter is in sample domain as the up sampler in the transmitter samples the information signal at the rate of 16 samples / signal. Hence we have to convert it in into symbol domain again The down sampler extracts the symbols in multiples of 16 thus the zeros between two symbols are neglected and thus the original transmitted burst length of 546 symbols is obtained.

3) Demodulation

The demodulation for the proposed rician flat fading channel consists of the following steps

Block phase estimation
Channel compensation
Hard slicing and mapping

⇨ Block Phase Estimation

As the fading is prevalent in the communication channel it degrades the signal amplitude and phase. The Estimation gives the compensating angle, by which we have to calibrate the phase shift of the symbols to their intended phase.

The steps in block phase estimation is as follows,

- A window size of length *N*(in our case 70) is chosen with a step size of *M*(17 in our case)
- y is defined by the received signal r(*z*) raised to the power four.
  before demodulation and after down sampling .
- The phase angle is calculated over the window size using the formula below:

$$\theta_{ph} = \frac{1}{4}\tan -1\left(\frac{\sum realpart(y)}{\sum imagpart(y)}\right)$$

- The calculated phase angle has ambiguity as theta can have values $\theta_{ac} = \theta_{ph} + \frac{k\pi}{4}$
  where *k* varies from *-4 to +4.*
- The phase rotation is accurately found by calculating the phase of unique word in the burst structure over the length *L* of unique word:

$$\theta_{UW} = angle\left(\frac{1}{L}\sum UW_R * conjUW_T\right)$$

- From the angle calculated for unique word above, the actual value of theta is calculated which is nearest to the calculated angle of unique word and is termed *theta 1* which is taken as a reference in calculating *theta 2.*

- Now block window is incremented to [M, M+N-1] and similarly the phase candidate theta is obtained.
- The *theta 2* is calculated from the above theta in *step g* by choosing one closest to *theta 1*
- The above steps are repeated till the length of the burst structure .
- Linear interpolation is implemented to get the phase rotation for every symbol in the burst

⇨ Channel Compensation

After modulation the transmitted signal passes through the channel, and gets rotated by a phase angle. At the receiver end before demodulating the received signal using the hard slicer, it has to be rotated in the anticlockwise by the same phase to compensate the phase angle rotation. This is the principle behind Channel Compensation.

⇨ Detection using Hard Slicing:

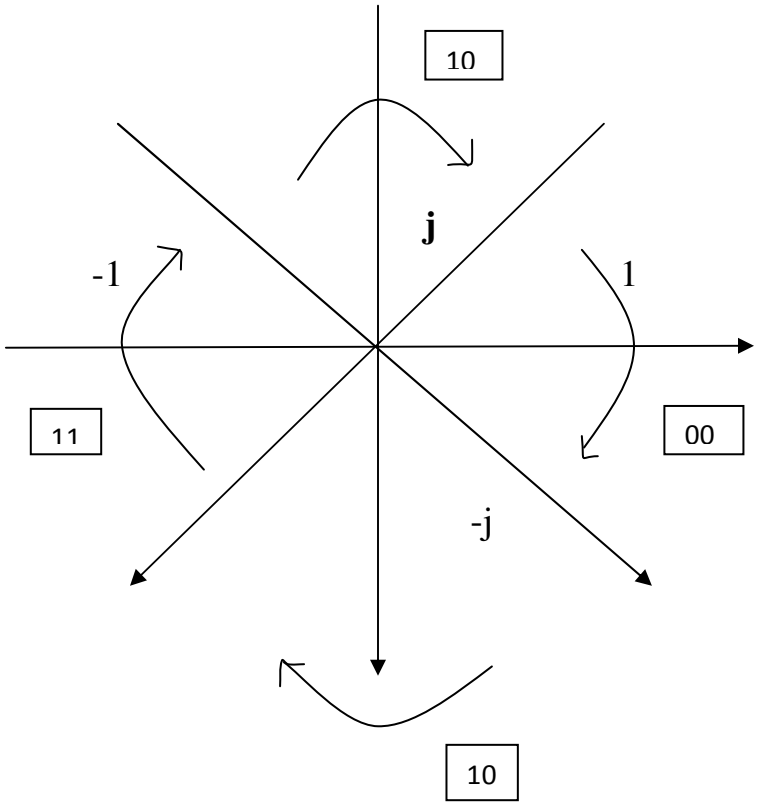


Fig.5 hard slicing

⇨ Mapping

The detected symbols corresponding to the angle in Hard slicer are Translated into bits. This process describes the symbol to bit mapping.

Thus 1 corresponds to 00 , j corresponds to 10 , -1 corresponds to 11 and –j corresponds to 10.

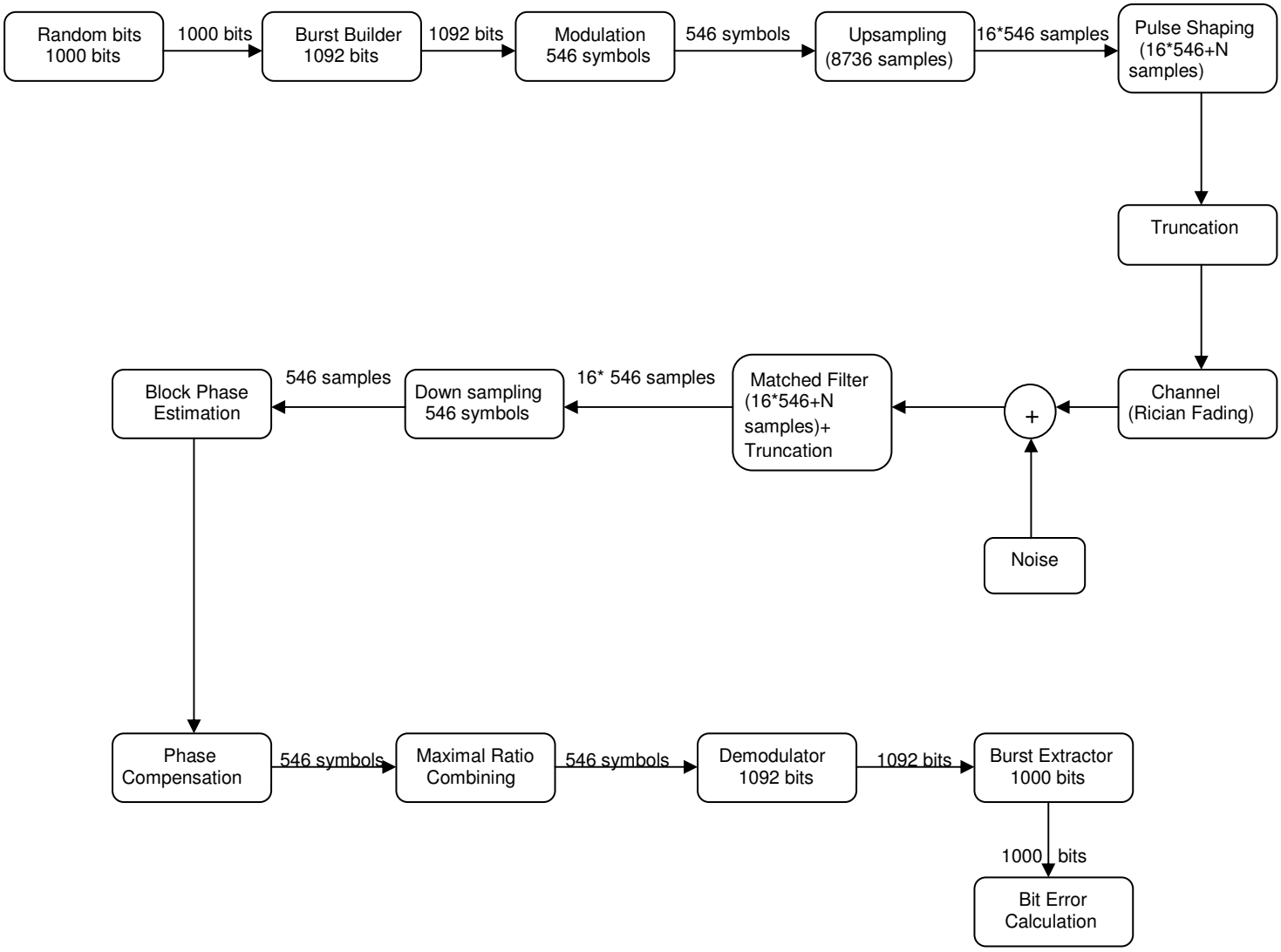
⇨ Burst extractor

This is the reverse process of burst building process in which the information bits (1000 bits) are extracted from the guard band and unique word.

⇨ Raw BER Calculation:

Raw ber is defined as the ratio of number of error bits in the received signal to the total number of transmitted bits , that is the length of the transmitted signal .

RICIAN FLAT FADING CHANNEL WITH MRC



⇨ Diversity Combining

Diversity combining is one of the methods to combat channel fading. We send multiple copies of the same signal over independent fading paths. The combination of the

independent paths of the received signal reduces the variance of SNR. Among the different methods of Diversity combining, we are implementing maximal – ratio combining (MRC) to improve the performance of the system.

⇨ Maximal Ratio Combining (MRC)

MRC is based on maximum likelihood principle to determine which signal was transmitted. In this method, the signals are added together and the gain of each channel is made directly proportional to the rms signal level and inversely proportional to the mean square noise level. MRC is also known as ratio squared combining and pre-detection combining. Based on derivations it can be proved if we have M copies of signal in the vector r and their corresponding gain in vector g, the sent signal will be the one having maximum real value after going the following process described in pictorial diagram below:

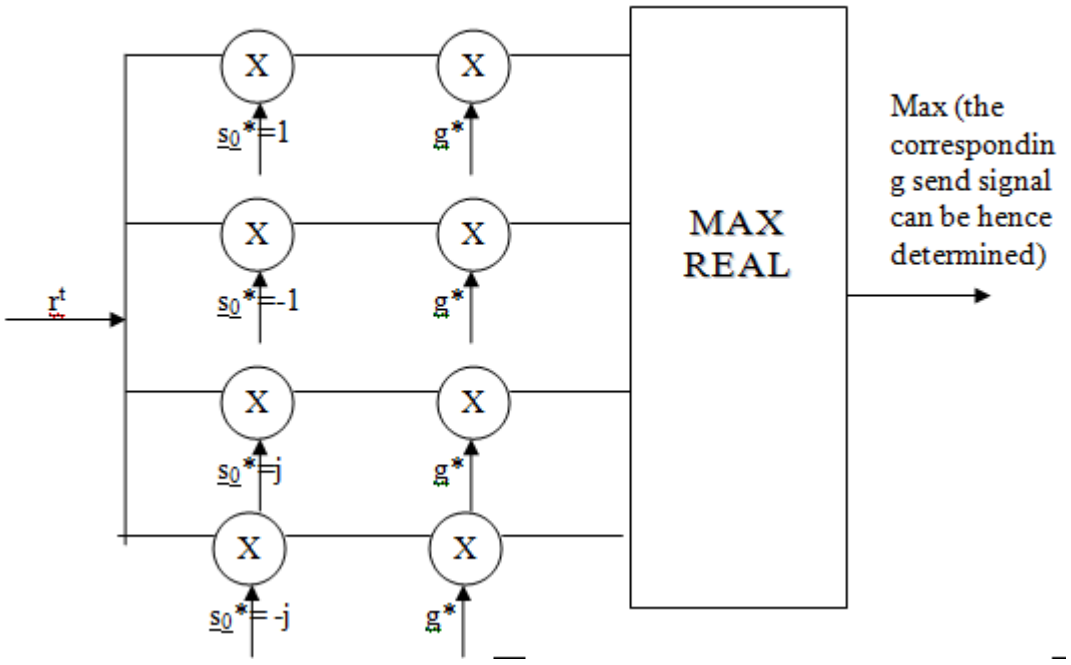
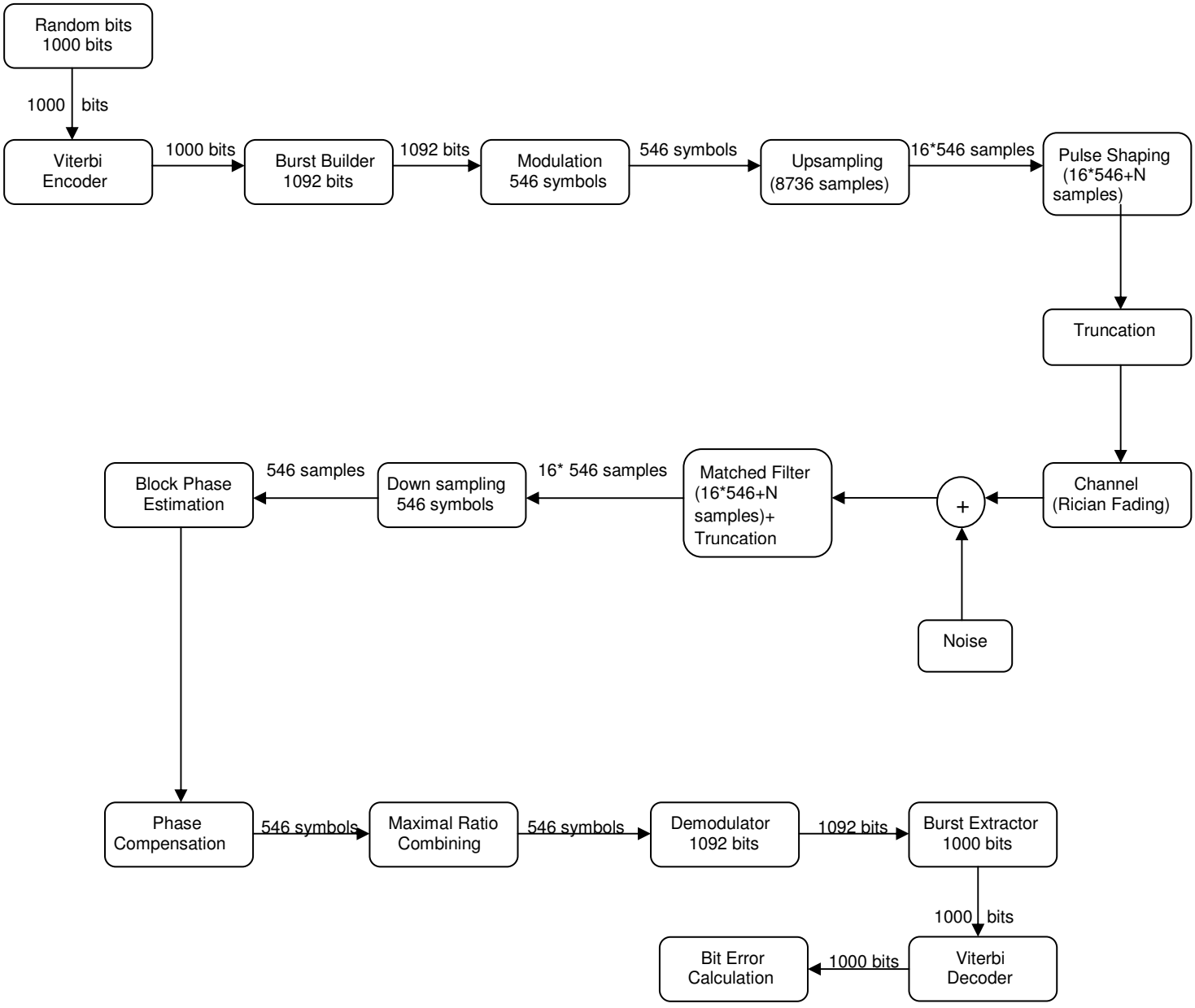


Fig: 6 Logic Diagram For Maximum Ratio Combining (MRC)

Basic system with Viterbi encoder and decoder



Convolution code is a type of error correcting code to achieve the reliable data transfer. Convolutional codes are used to improve the BER performance of the system. Transmitted signal is corrupted mainly by additive white gaussian noise (AWGN) present in the channel for which convolutional encoding with Viterbi decoding (FEC technique) will be the most effective .

Viterbi Encoder and Decoder

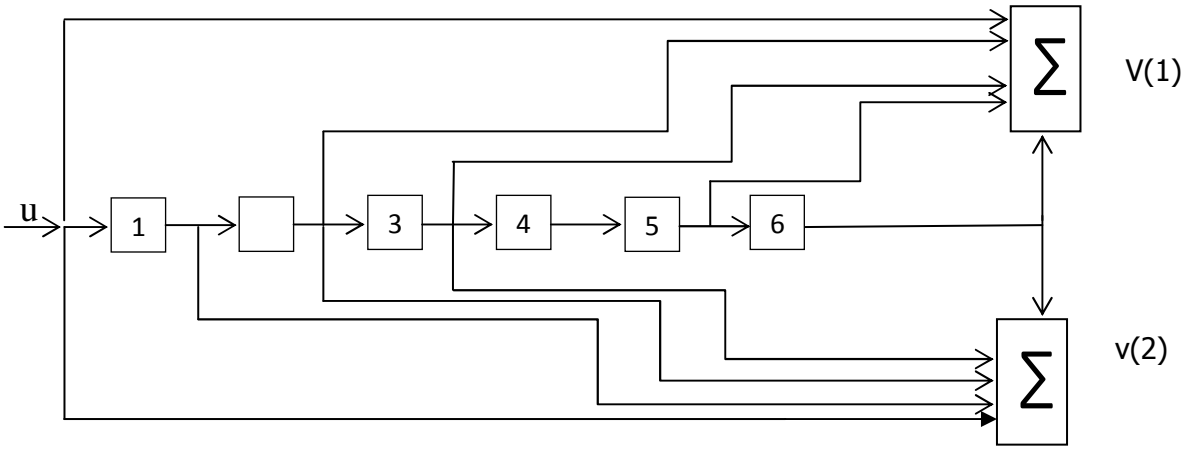
Convolutional codes (Encoder) :

Forward error correction is to increase the channel capacity of the system by adding some additional information in the transmitted data of the channel and this process of adding redundant bits is called channel coding. Convolutional codes are often use to improve the performance of the digital cellular systems by improving the BER performance of the system.

There are there three basic elements for convolutional codes , they are " n,k,m "
Where,

n = number of output bits;
k= number of input bits;
m= number of memory registers.
k/n= code rate

The structure for the convolutional encoder is shown below .

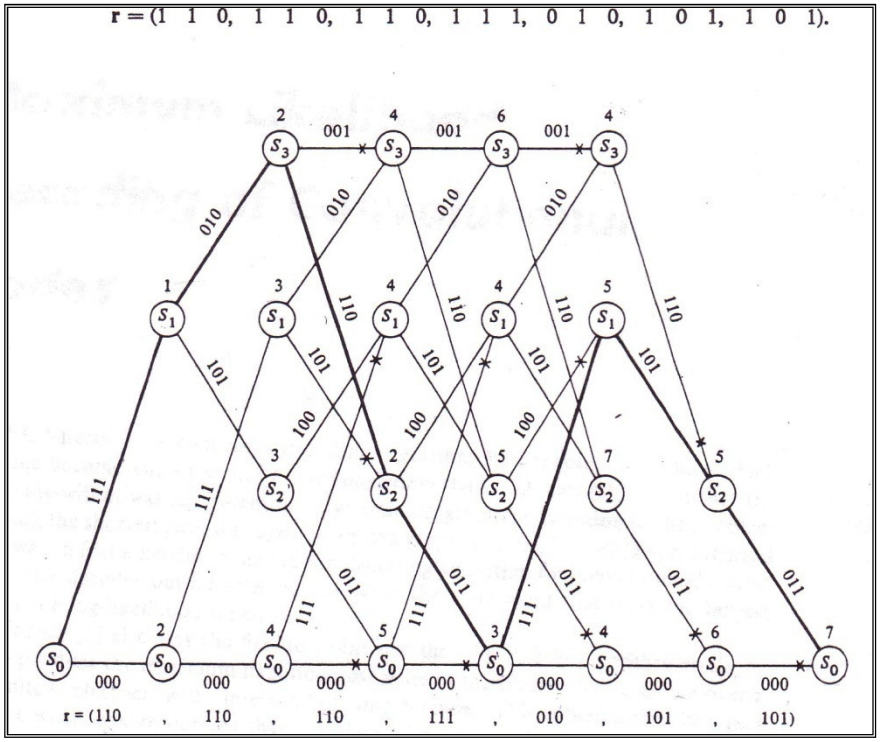


Thus the convolutional encoder with rate ½ and 6 memory registers. For an encoder with memory *m*, we can write the convolution operation using the following mathematical model stated below. Where $v^j{}_l$ signifies output l^{th} bit of j^{th} output, *u* corresponds to the input bit sequence and $g_i{}^j$ corresponds to the i^{th} bit of j^{th} generator sequence

$$v^j{}_l = \sum_{i=0}^{m} u_{l-i} g_i{}^j$$

Viterbi Decoder

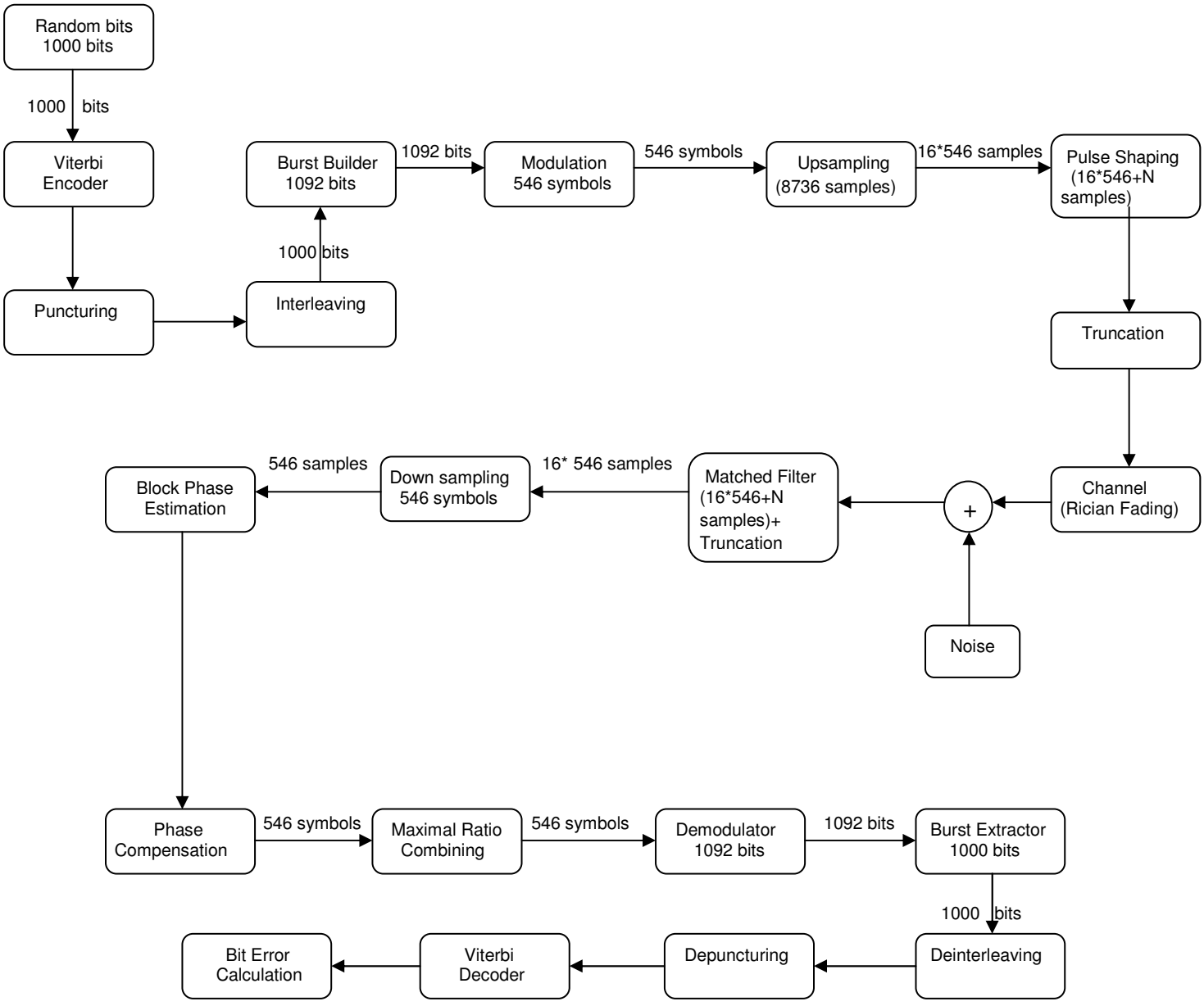
Viterbi algorithm is used for error correction in transmission of messages. It requires coding and decoding stages in the sender and receiver, respectively. These types of algorithms are very useful for the transmission of a type of message where some degree of error in the received message is acceptable, such as, voice and video. Viterbi algorithm is based on trellis diagram and the maximum likelihood principle. A diagrammatic representation of the viterbi decoder is shown below.



The upper branch in the trellis diagram corresponds to the input is 1 and the lower branch corresponds to the input is 0.From the above trellis diagram the decoded bits could be determined by the minimum hamming distance which is equal to difference in the number of received bits.

Basic system with interleaver and Puncturing

Block Diagram



- **Puncturing**

Puncturing is deployed to achieve variable data rates ranging from 4 to 51 Mbps. However it is not feasible to design encoder and decoder for all data rates. Puncturing means not to transmit some of the coded bits. The receiver inserts the dummy bits for the punctured bits to retrieve the coded bits. This increases the coding rate of the transmitter .The puncturing pattern is shown in the figure.

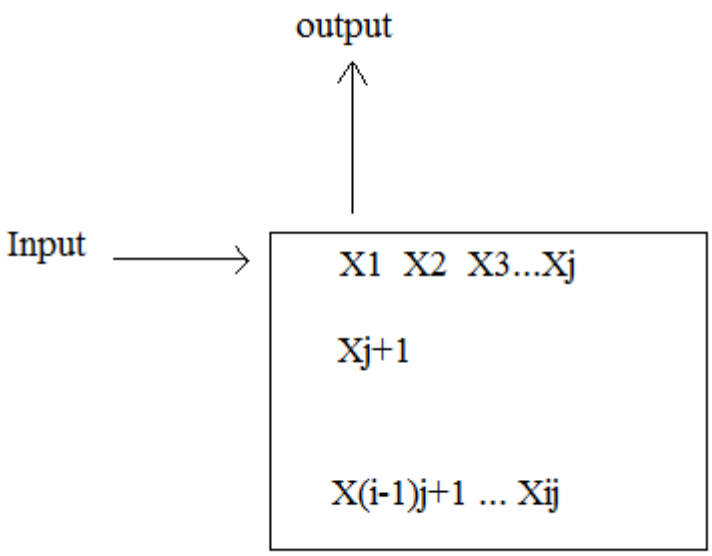
As shown above out of six coded bits only 3 bits are transmitted, the remaining 1 bit is punctured.

Mother code = ½

New Code rate =1/(2*(3/4)) = 2/3

- **Interleaving**

Interleaving is phenomenon in which the bits at the lower amplitude of the envelope are scattered so as to avoid the loss of sequence of the information. Interleaving is one of the effective ways to combat channel fading. The information bits are entered row wise in the matrix and taken out column wise from the matrix.



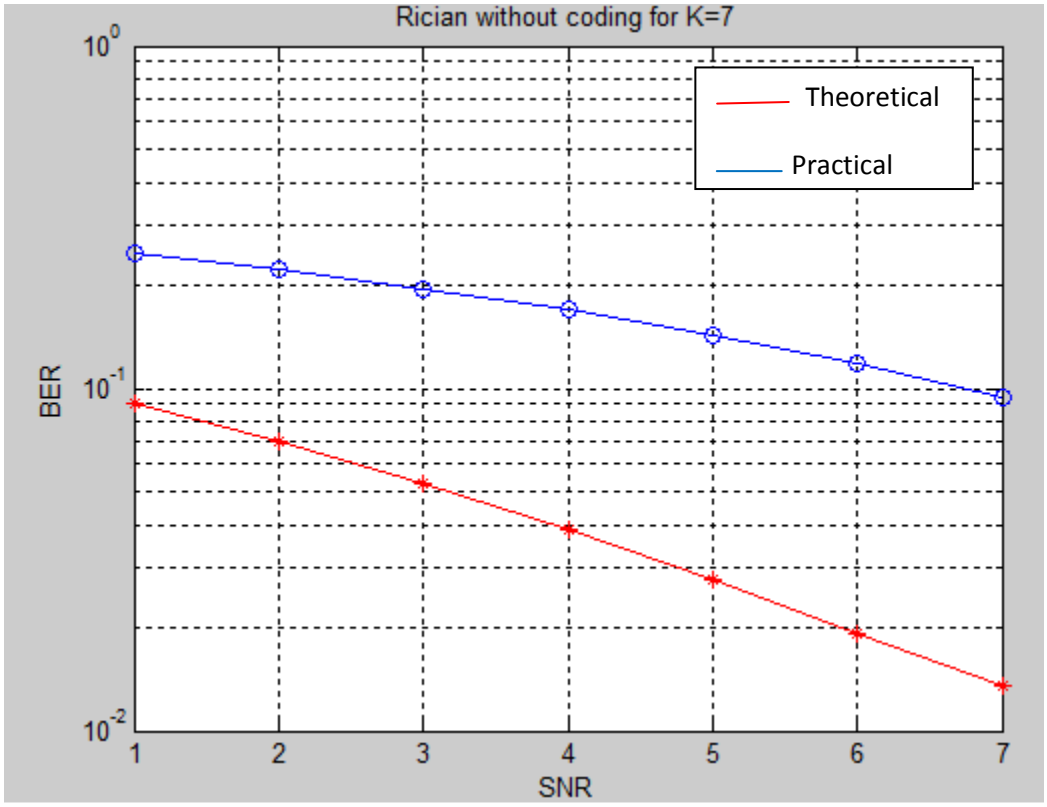
As shown in figure the bits are entered in the matrix as X1….Xj…….Xij and the sequence is taken out as X1,Xj+1,X(I-1)j+1…..X2, X2j+1……

The difference between two neighbors X1…X2 is **I** bits.

Simulation Outputs

1) Basic System
   Case 1:
   Fm=20 hz ,K=7db

Eb/NO	1	2	3	4	5	6	7
BER(Practical)	0.247109	0.22141	0.195631	0.169928	0.143490	0.11772	0.0943352
BER(Theory)	0.09	0.0699	0.0527	0.0387	0.0277	0.0193	0.0135

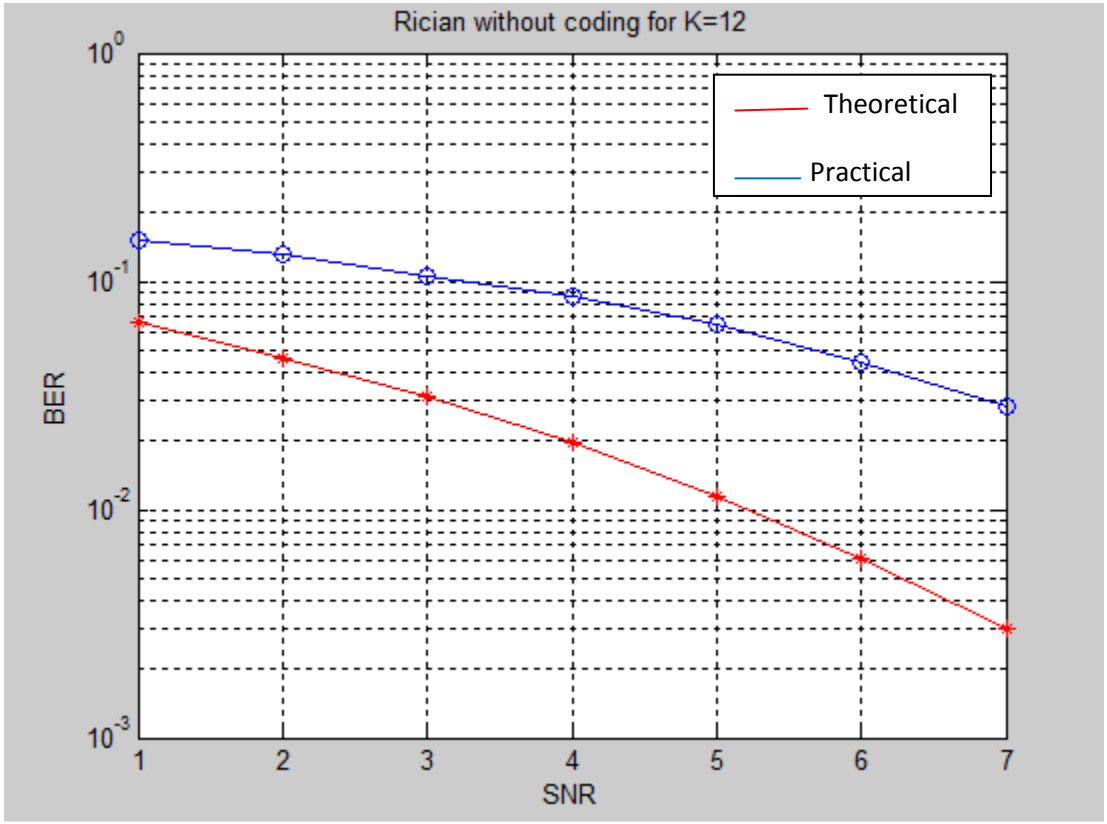


2) Basic System

Case 2:

Fm=100 hz, K=12db

Eb/NO	1	2	3	4	5	6	7
BER(Practical)	0.1527	0.1323	0.1043	0.0960	0.0866	0.0571	0.0342
BER(Theory)	0.06565	0.04664	0.03124	0.01958	0.011396	0.0061246	0.00302

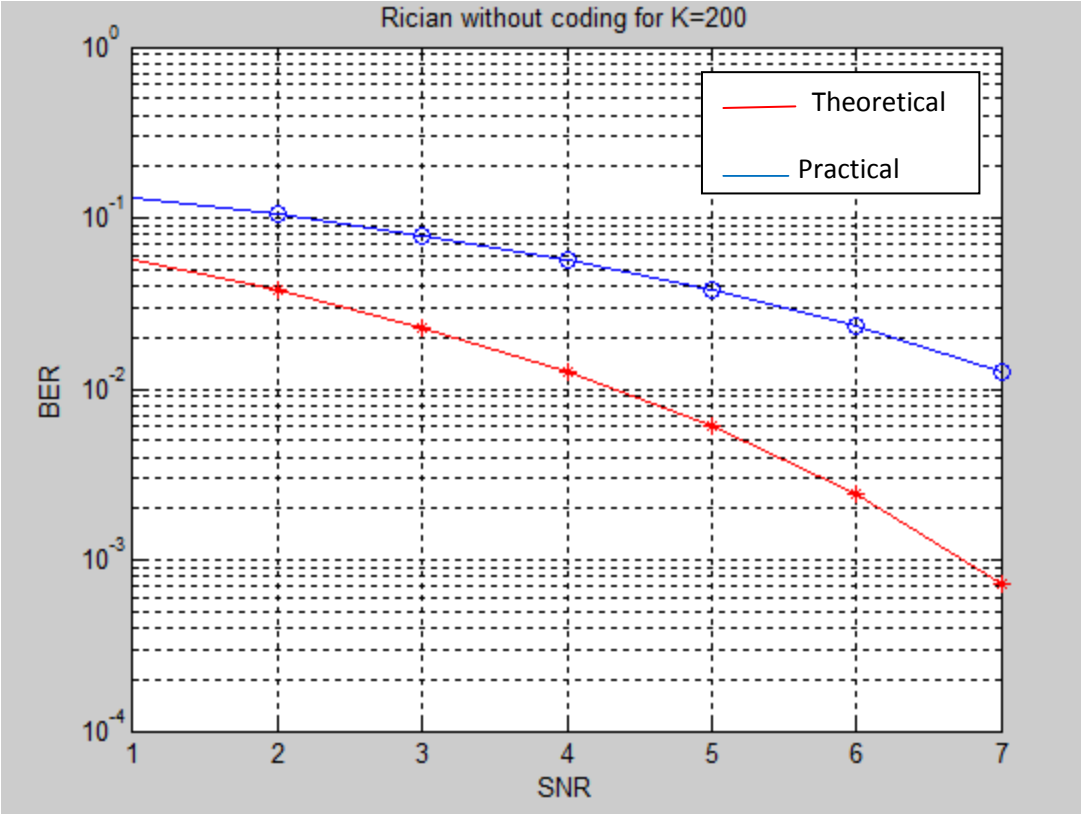


3) Basic System

Case 3:

Fm=0 hz ,K=200 db

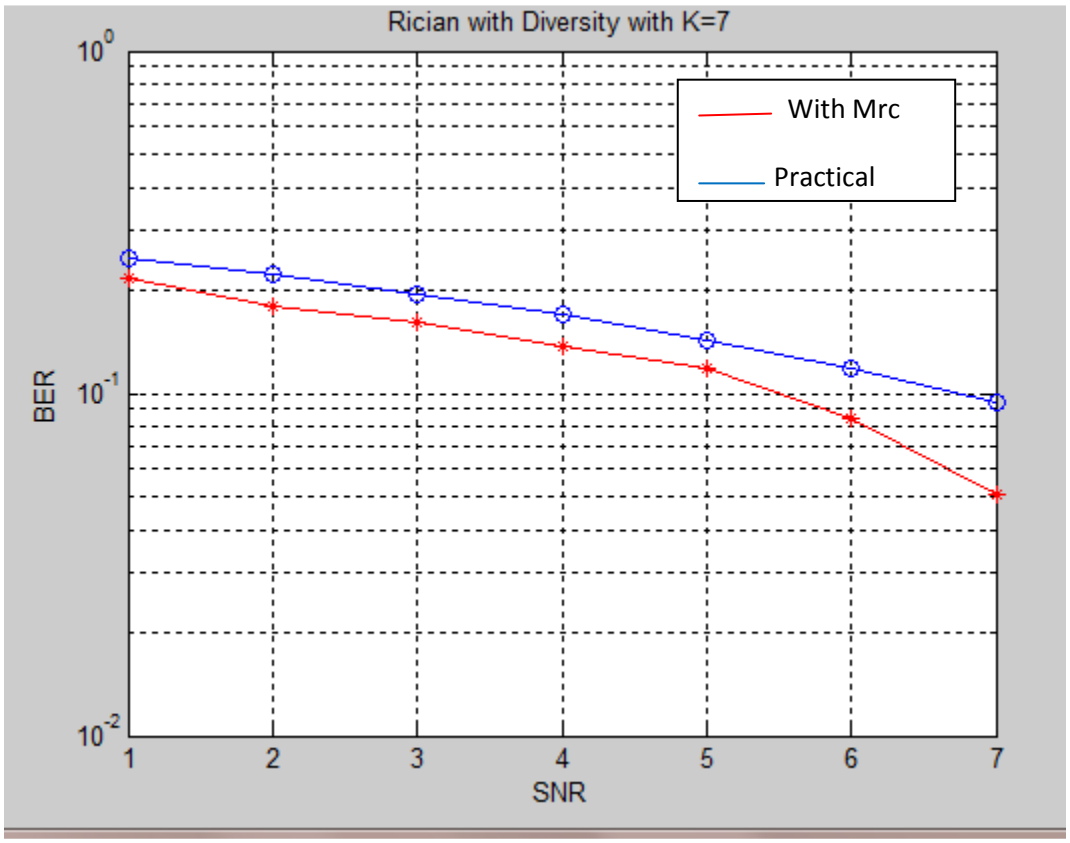
Eb/NO	1	2	3	4	5	6	7
BER(Practical)	0.1325	0.1075	0.0782	0.0569	0.0378	0.0233	0.0125
BER(Theory)	0.0562	0.0343	0.0229	0.0125	0.0060	0.0024	0.0008



4)MRC

a) Case 1: Fm=20 hz, K=7db

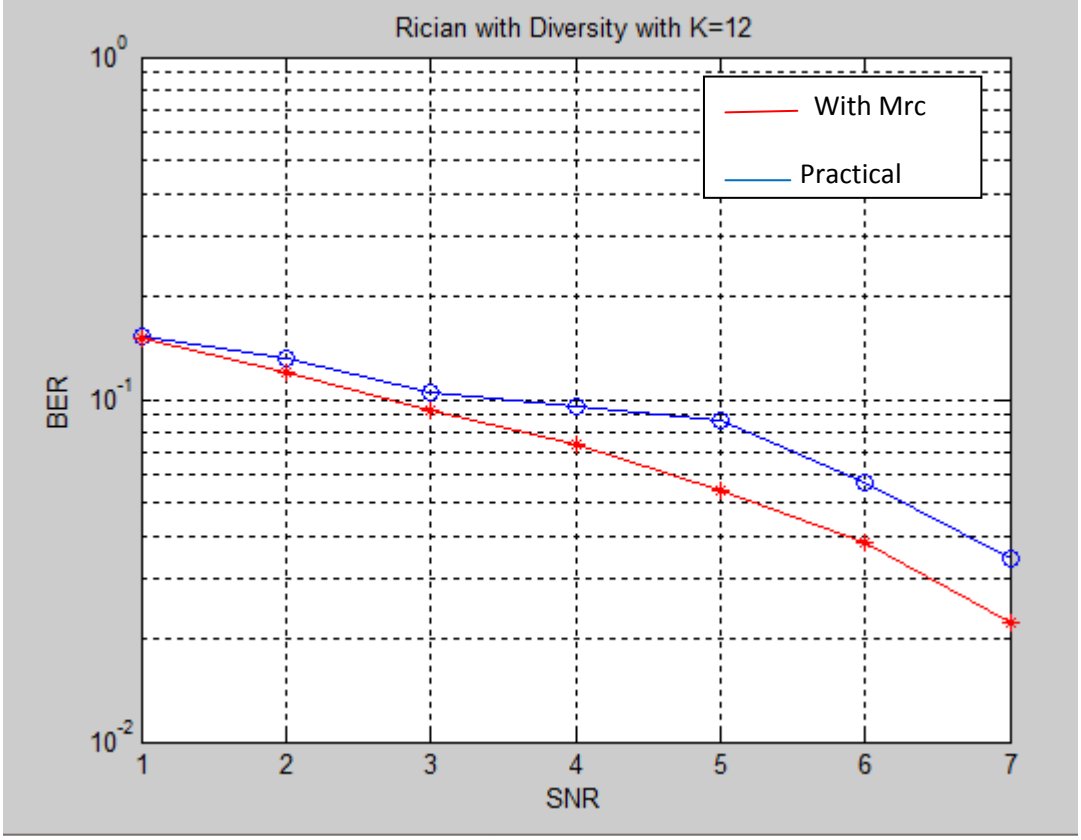
Eb/NO	1	2	3	4	5	6	7
BER(Practical)	0.247109	0.22141	0.195631	0.169928	0.143490	0.11772	0.0943352
BER(MRC)	0.2165	0.1801	0.1613	0.1365	0.1176	0.0843	0.0507



Gain due to MRC Code: **1.3 db**

b) Case 2:
Fm=100 hz, K=12 db

Eb/NO	1	2	3	4	5	6	7
BER(Practical)	0.1527	0.1323	0.1043	0.0960	0.0866	0.0571	0.0342
BER(MRC)	0.1500	0.1200	0.0932	0.0741	0.0539	0.0384	0.0223

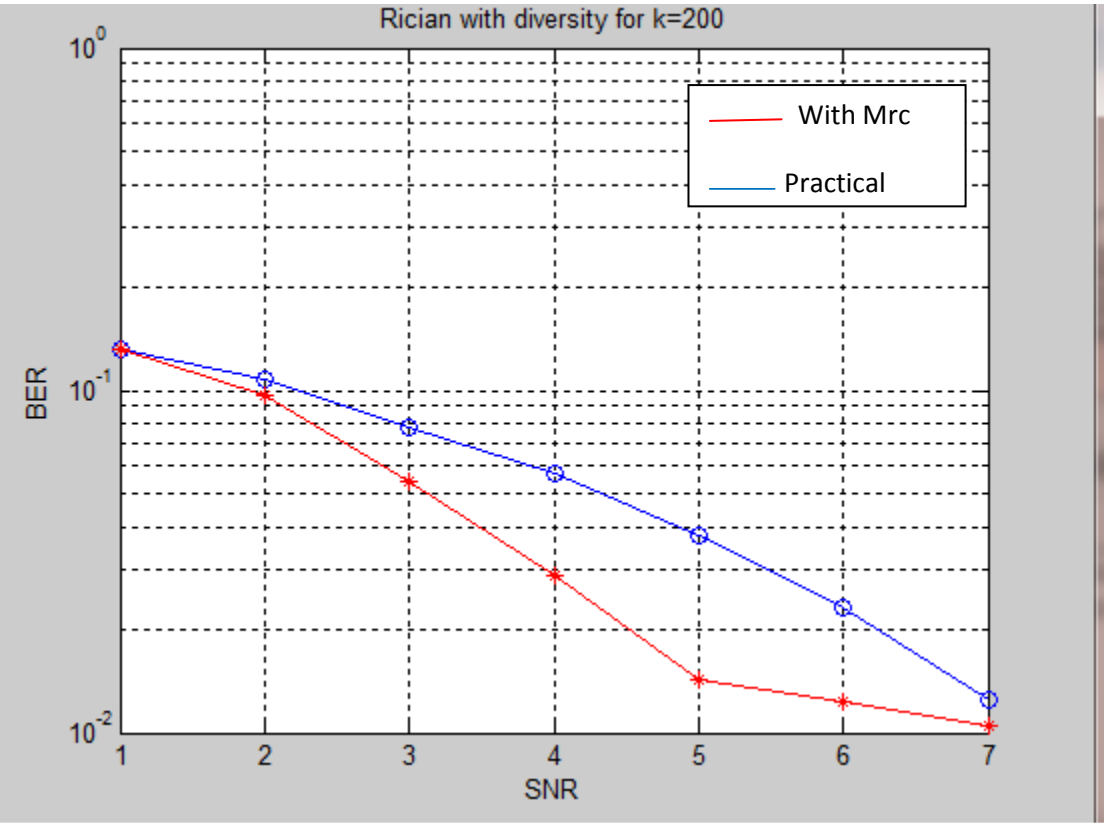


Gain due to MRC: 1.7db

c)Case 3:

Fm=0 hz K=200db

Eb/NO	1	2	3	4	5	6	7
BER(Practical)	0.0807	0.0489	0.0250	0.0141	0.0066	0.0028	0.0011
BER(MRC)	0.1320	0.0968	0.0543	0.0287	0.0143	0.0123	0.0105



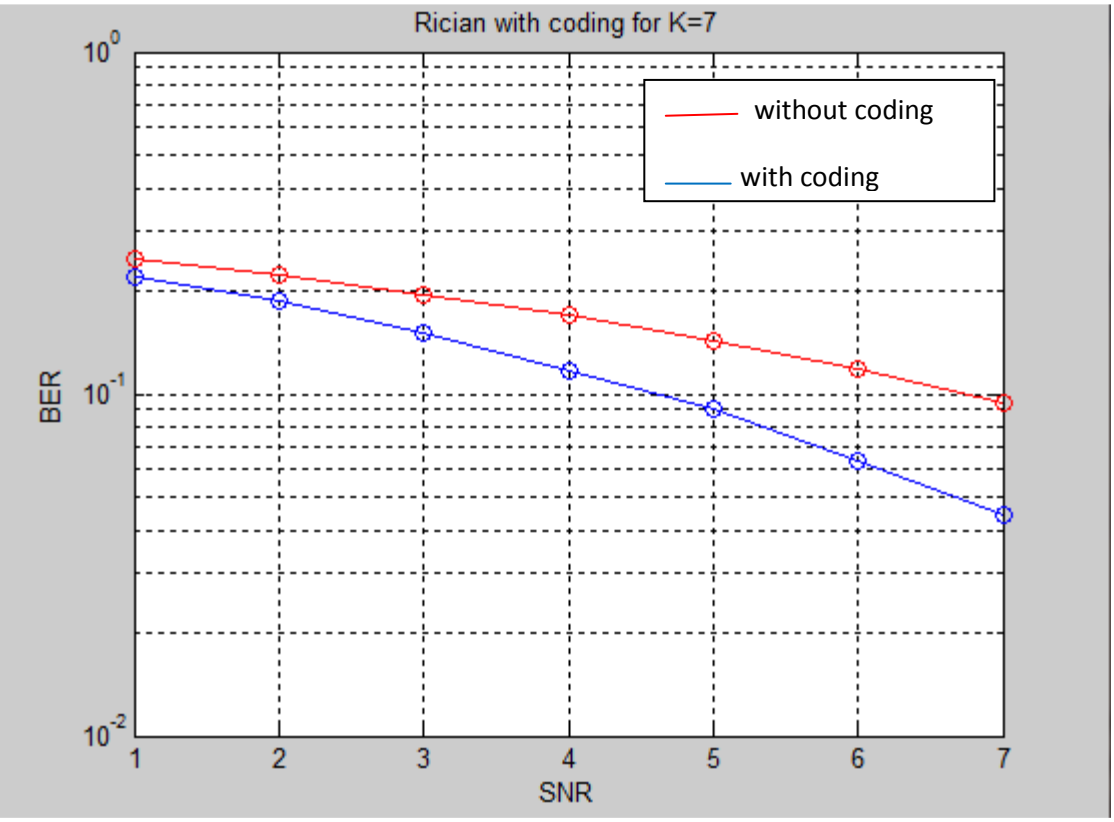
Gain due to MRC Code: **1.4 db**

5) Convolutional Coder:

a) Case 1:

Fm=20 hz, K=7db

Eb/NO	1	2	3	4	5	6	7
BER(Theory)	0.2471	0.22141	0.195631	0.169928	0.143490	0.11772	0.0943352
BER(Practical)	0.2268	0.1985	0.1678	0.1345	0.1135	0.09642	0.04360

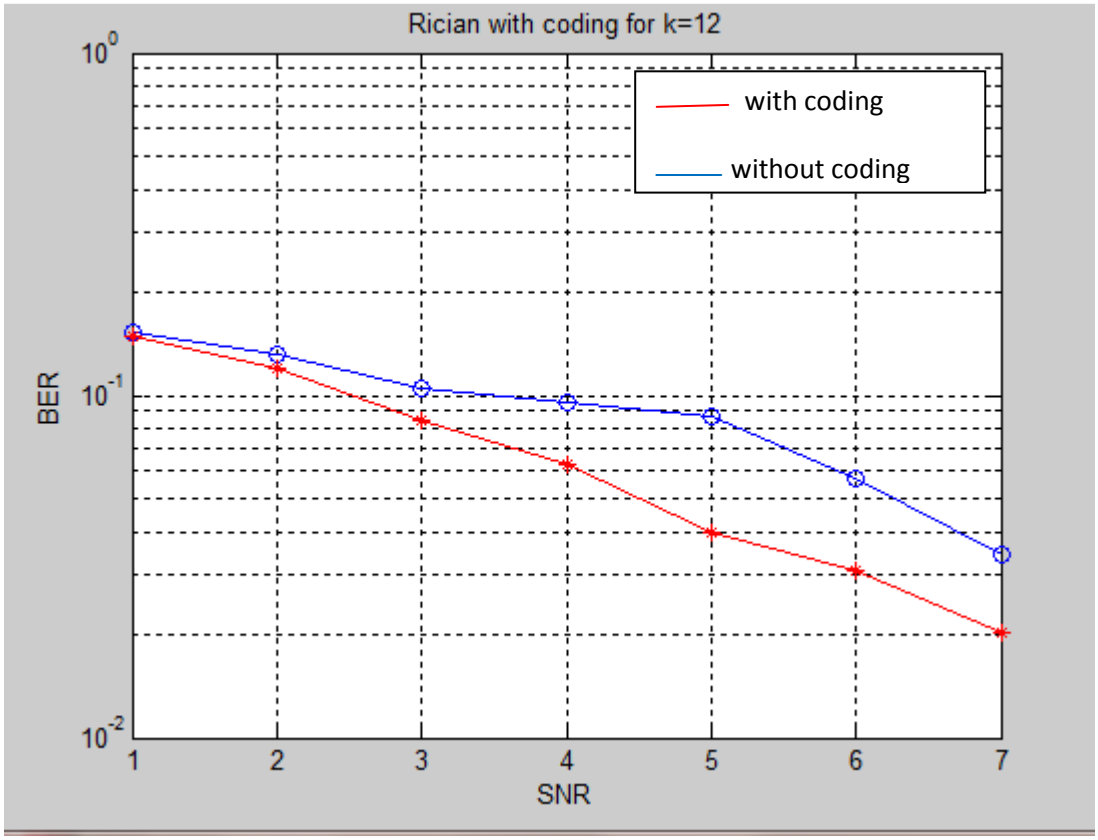


Gain due to convolutional coder: **2.0 db**

b) Case 2:

Fm=100hz, K=12 db

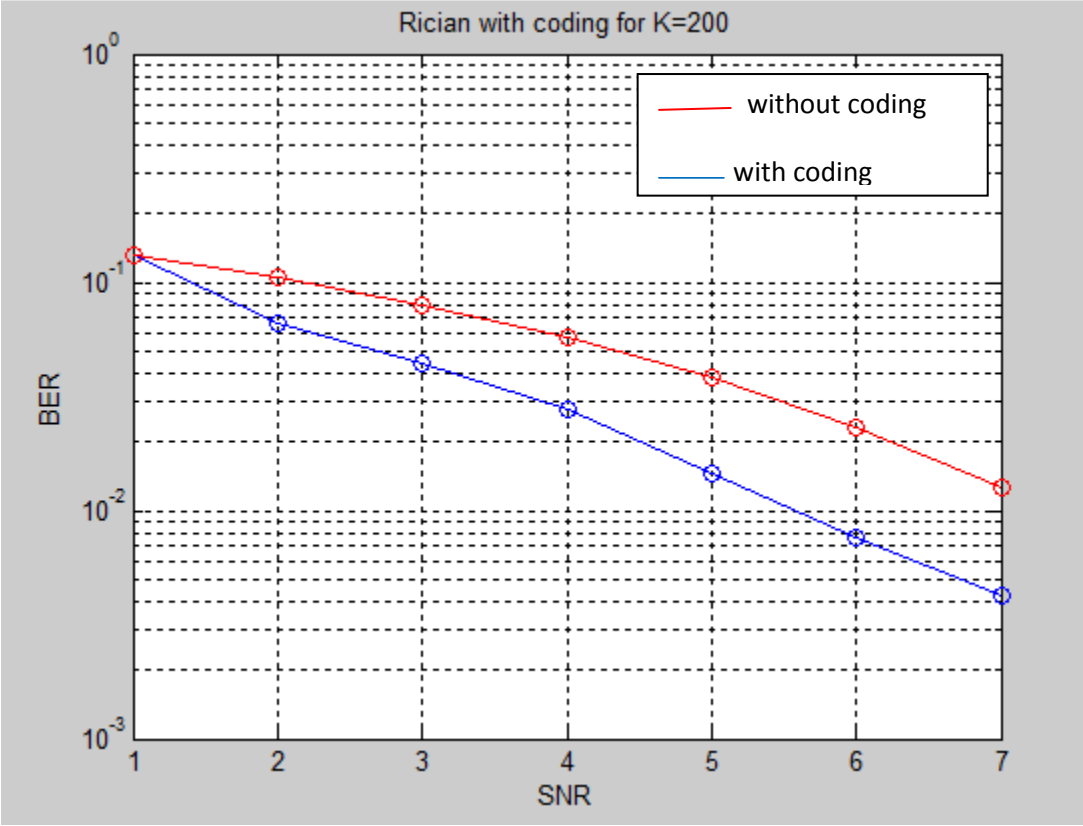
Eb/NO	1	2	3	4	5	6	7
BER(Theory)	0.1527	0,1323	0.1043	0.0960	0.0866	0.0571	0.0342
BER(Practical)	0.1489	0.1196	0.0844	0.0625	0.0397	0.0306	0.0203



Gain due to convolutional coder: **1.8 db**

c) Case 3:
Fm=0hz, k=200db

Eb/NO	1	2	3	4	5	6	7
BER(Theory)	0.1325	0.1075	0.0782	0.0569	0.0378	0.0233	0.0125
BER(Practical)	0.1288	0.0670	0.0215	0.0077	0.0049	0.0047	0.0044

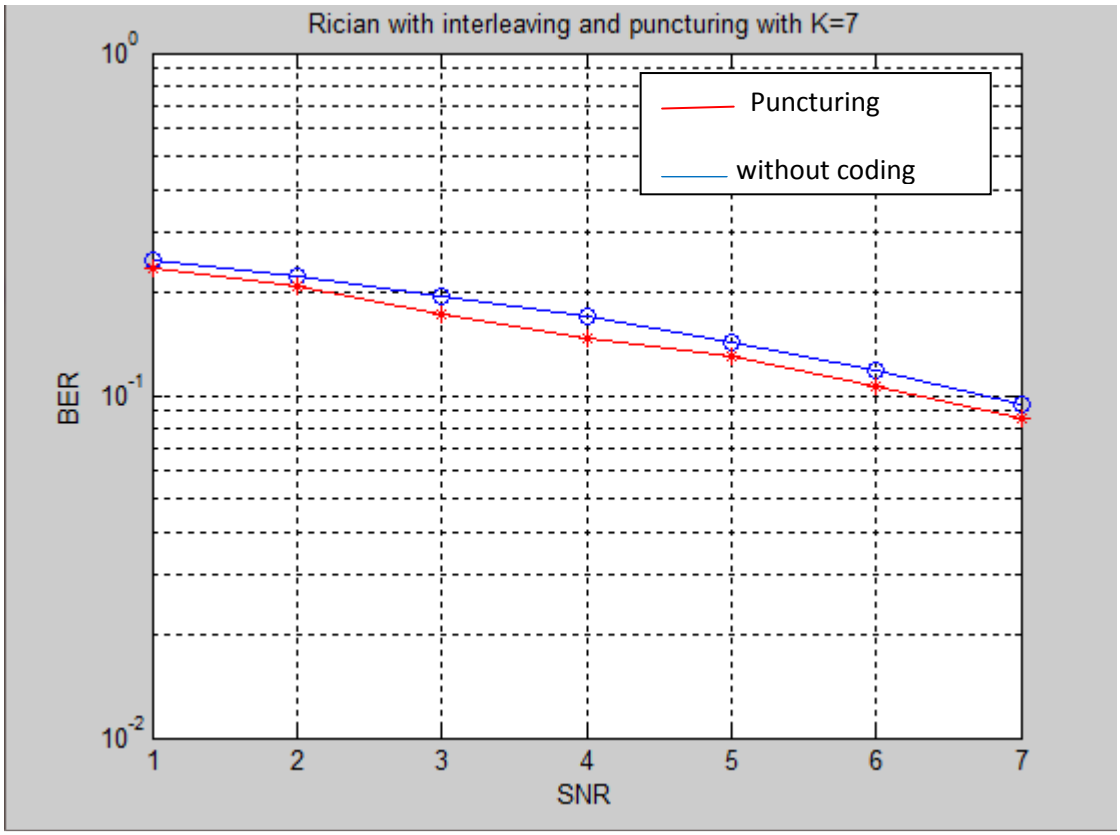


Gain due to convolutional coder: **1.4 db**

6) Puncturing and Interleaving:

a)Case 1:
Fm=20hz, K=7db

Eb/NO	1	2	3	4	5	6	7
BER(Theory)	0.2471	0.2214	0.1956	0.1699	0.1435	0.1177	0.0943
BER(Practical)	0.2265	0.2011	0.1713	0.1465	0.1276	0.1043	0.0732

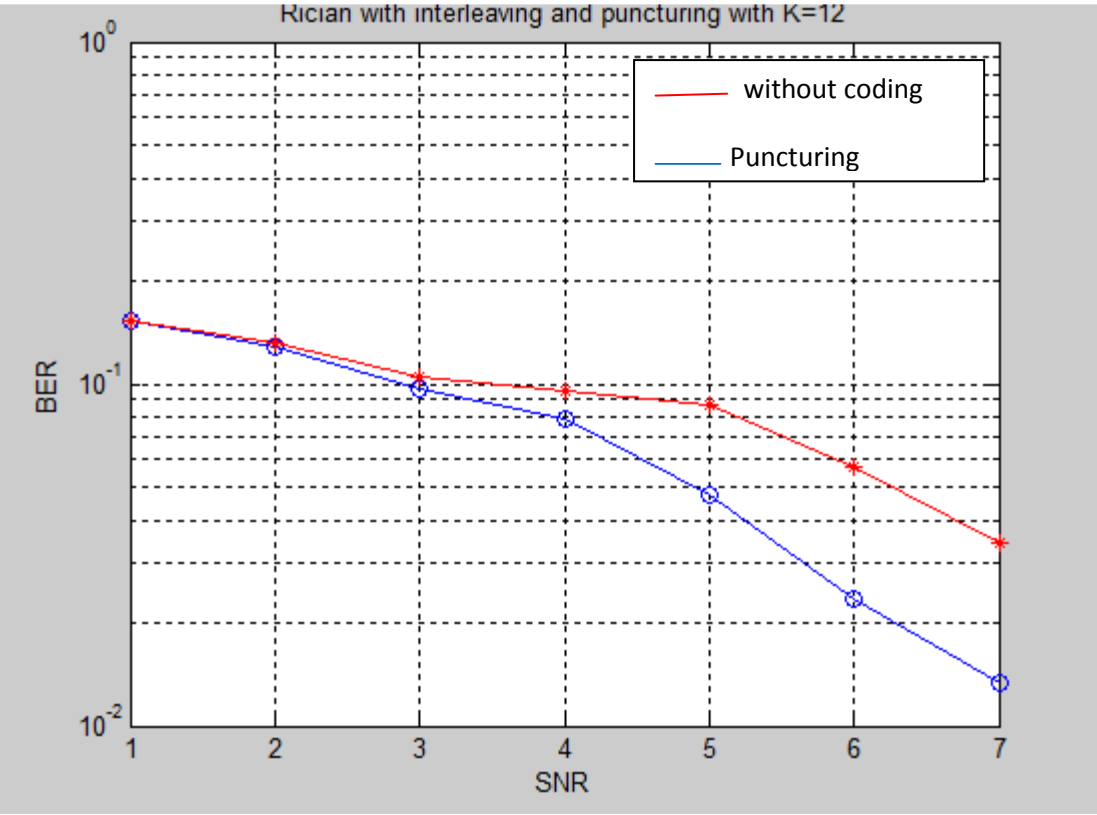


Gain due to Interleaving and puncturing: 1.0db

Loss compared to convolutional encoder: 1.0 db

Case 2: Fm=100hz, K=12 db

Eb/NO	1	2	3	4	5	6	7
BER(Theory)	0.0975	0.0717	0.0532	0.0235	0.0134	0.0077	0.0045
BER(Practical)	0.4968	0.4981	0.5011	0.4964	0.4958	0.5006	0.5008



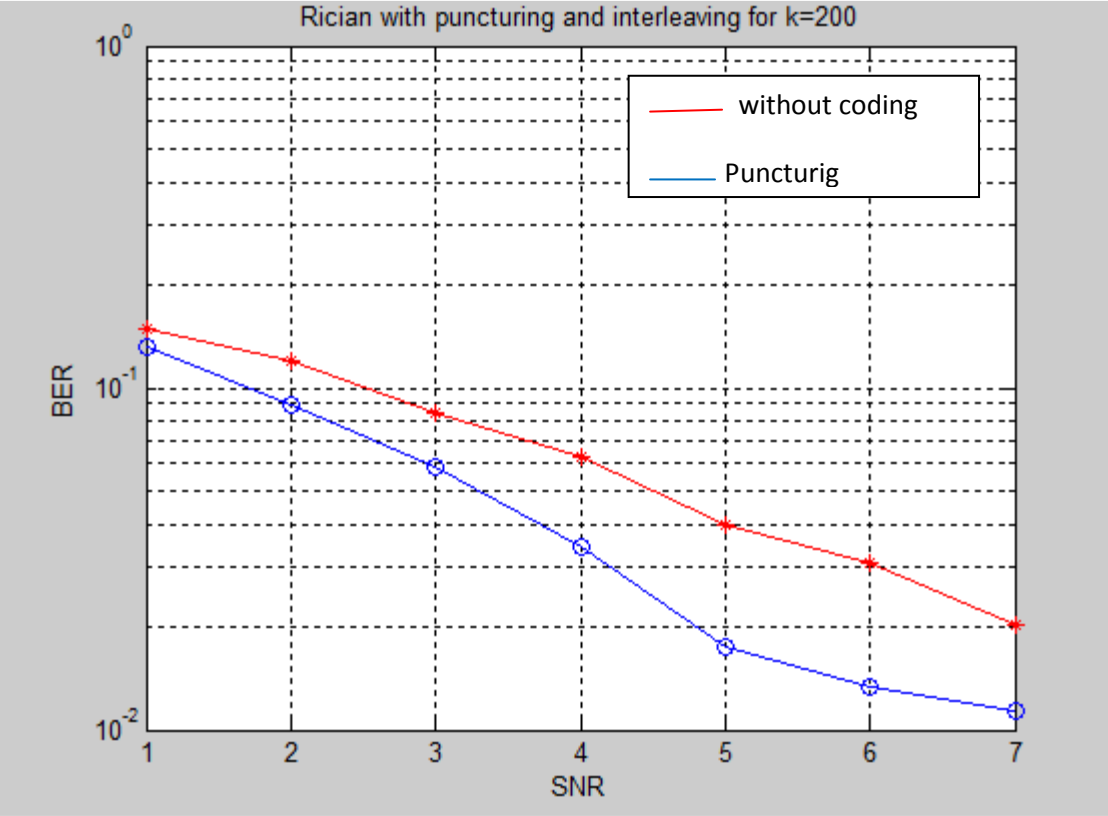
Gain due to Interleaving and Puncturing: 1.2db

Loss compared toConvolutional coding: 0.6db

c) Case 3:

Fm=0hz, K=200db

Eb/NO	1	2	3	4	5	6	7
BER(Theory)	0.1325	0.1075	0.0782	0.0569	0.0378	0.0233	0.0125
BER(Practical)	0.1324	0.0886	0.0584	0.0345	0.0175	0.0133	0.0113



Gain due to puncturing and interleaving: 1.3db

Loss compared to convolutional encoder: 0.3db

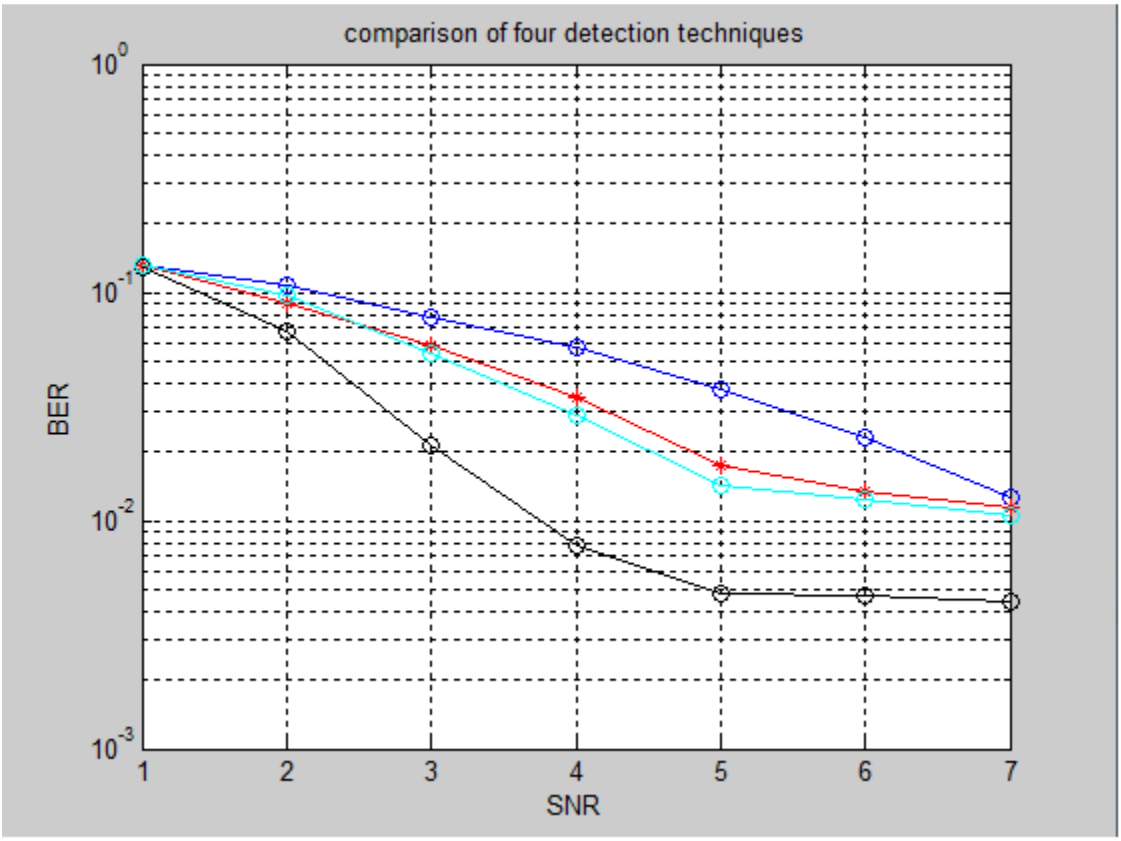
Advantages of Interleaving and Puncturing:

1) Considering variable data rate its impractical to design Encoder and decoder for all data rates. Puncturing overcomes this drawback.

2) High Coding rate is achieved on transmitter side.

3) Interleaving combats channel fading and avoids loss of sequence of information.

CONCLUSION:



Without coding
Puncturing and interleaving
Mrc coding
With coding

1) It can be said that BER is Inversly proportional to SNR for a particular value of Rician factor and Doppler shift. (As BER decreases SNR increases).

   The performance for MRC case is better than the first two as the Doppler shift is zero.

2) MRC improves the performance when embedded in the basic system. BER for diversity is less than basic Rician channel and MRC is based on Diversity. MRC reduces BER as Eb/No increases for a particular value of Rician factor and Doppler shift.

3) A significant reduction in BER is observed as compared to theoretical values on implementation of convolutional coding. Thus convolutional coding is the system with the best performance.

4) Puncturing produces good performance compared to the basic system. But it is not as good as convolutional coding system. But the main advantage of this system is the variable coding rate.

REFERENCE:

1) A. J. Viterbi and A. M. Viterbi, "Nonlinear estimation of PSK-modulated carrier phase with application to burst digital transmission", *IEEE Trans on Information Theory*, vol. 29, no. 4, July 1983, pp. 543-551.
2) Wireless Communication by Andrea Goldsmith
3) Class notes by Dr. Liang
4) Matlab help

Matlab Code:

Basic System Design code:

```
clear all
BERric = zeros(1,7);
for dbCount = 1:1:7
snr = dbCount;
TempBER = 0;
for burstCount=1:1:1000
%% transmission

%generate information bits

mInfoBitsTx = randint(1,1000);

%generate guard bits

mGuardBitsTx = zeros(1,6);

%generate unique word

a = [1,0,1,1,0,1,0,0];

mUniqueWordTx = [a,a,a,a,a,a,a,a,a,a];

% generate burst

mBurstTx = [mGuardBitsTx,mUniqueWordTx,mInfoBitsTx,mGuardBitsTx];

%modulation

mSymbolsTx = QpskModulation(mBurstTx,length(mBurstTx));

% upsampling
```

```
mUpSamples = upsample(mSymbolsTx,16);

% Generate a square root rcosine filter system

mPulseShaping = rcosine(1,16,'sqrt',0.35);

% Pulse shaping

mSignalTx = PulseShaping(mUpSamples,mPulseShaping);

%es(i)=var(mSignalTx);

%% generate rican fast fading channel
Tsample = 0.5 * 1 * 10 ^ (-3) / (546 * 16);

Fm = 20;

K = 7;

mChannelRician = Rician(Tsample,Fm,K);

% generate noise
% snr = 10 log10(signalpower / (k^2 * noisepower)
%snr = 7;

signalPow = 0.9980;

noisePow = 1.9908;

k = sqrt((signalPow) / ((10^(snr/10)) * (noisePow)));

mNoise = complex(k*randn(1,8736),k*randn(1,8736));

% mix the signal with the channel

mSignalRx = mSignalTx .* mChannelRician + mNoise;

%% reception
% Matched filter
```

```
mFilteredSignalRx = MatchedFilter(mSignalRx,mPulseShaping);
% downsampling
mSymbolsRx = downsample(mFilteredSignalRx,16);
% BPSE
mUwTx = mSymbolsTx(4:43);
mUwRx = mSymbolsRx(4:43);
mUwLen = length(mUwTx);
mSigma = 0;
for i=1:1:mUwLen
mSigma = mSigma + (mUwTx(i) .* conj(mUwRx(i)));
end
theta = angle(mSigma / mUwLen);
theta1 = theta;
Nbpe = 50;
Mbpe = 20;
mBusrtLen = length(mSymbolsRx);
mAngleArray = zeros(1,mBusrtLen);
for i = 1 : Mbpe : (mBusrtLen - Nbpe)
innerLen = i + Nbpe;
mImagSignal = 0;
```

```
mRealSignal = 0;

for j = i : 1 : innerLen

mImagSignal = mImagSignal + (imag(mSymbolsRx(j)) ^ 4);

mRealSignal = mRealSignal + (real(mSymbolsRx(j)) ^ 4);

end

mAngle = atan((mImagSignal/mRealSignal)) / 4;

mTempAngles = [mAngle, (mAngle + (pi / 4)), (mAngle - (pi / 4)), (mAngle + (pi
/ 2)), (mAngle - (pi / 2)),   (mAngle + (3 * pi / 4)), (mAngle - (3 *pi /4)),
(mAngle+pi)];

[min_difference, array_position] = min(abs(mTempAngles - theta1));

mAngleArray(i) = mTempAngles(array_position);

theta1 = mAngleArray(i);

end

hm = mfilt.linearinterp(Mbpe);

y = filter(hm,mAngleArray(1:Mbpe:(mBusrtLen - Nbpe)));
mAngleArray1 = zeros(1,mBusrtLen);
for i = 1:length(y)-Mbpe

mAngleArray1(i) = y(i+Mbpe-1);

end
for count = length(y)-Mbpe+1:1:length(mAngleArray1)

mAngleArray1(count) = y(length(y));

end
%stem(1:1:(mBusrtLen - Nbpe),mAngleArray(1:1:(mBusrtLen - Nbpe)));
%figure(2);
```

```
%plot(mAngleArray1);
%disp('end');

mSymbolsRx1 = zeros(1,length(mAngleArray1));

%phase componsation
for value = 1:1:length(mAngleArray1)
mSymbolsRx1(value) = mSymbolsRx(value) .*exp(-j*mAngleArray1(value));
end

%hard-slicer
mSymbolsRx2 = zeros(1,length(mAngleArray1));
for count= 1:1:length(mSymbolsRx1)
symbolAngle = angle(mSymbolsRx1(count));

if  symbolAngle > pi
symbolAngle = symbolAngle - 2*pi;
elseif symbolAngle < -pi
symbolAngle = symbolAngle + 2*pi;
end


if (symbolAngle >= 0 && symbolAngle < pi/4) || (symbolAngle >= -pi/4 &&
symbolAngle < 0)
mSymbolsRx2(count) = 1;
elseif (symbolAngle >= pi/4 && symbolAngle < 3*pi/4)
mSymbolsRx2(count) = complex(0,1);
elseif (symbolAngle >= 3*pi/4 && symbolAngle < pi) || (symbolAngle >= -pi &&
symbolAngle < -3*pi/4 )
mSymbolsRx2(count) = -1;
elseif (symbolAngle >= -3*pi/4 && symbolAngle < -pi/4)
mSymbolsRx2(count) = complex(0,-1);
else
mSymbolsRx2(count) = 0;
disp('a');
end
end
%       result = (mSymbolsTx==mSymbolsRx2);

%demodulation
```

```
mBurstRx = QpskDemodulation(mSymbolsRx2,length(mSymbolsRx));

%mGuardBitsRxFront = mBurstRx(1,1:6);

%mUniqueWordRx = mBurstRx (1,7:86);

mInfoBitsRx = mBurstRx(1,87:1086);

%mGuardBitsRxRear = mBurstRx(1,1087:1092);

TempBER = TempBER +
(biterr(mInfoBitsTx,mInfoBitsRx)/length(mInfoBitsRx));
end
snr
BERric(dbCount) = TempBER/burstCount;
end
%BERtheo = [0.09 0.0699 0.0527 0.0387 0.0277 0.0193 0.0135];
semilogy(1:1:7,BERric,'-bo');
hold on
%semilogy(1:1:7,BERtheo,'-r*');
hold off
grid on
xlabel('SNR (dB)')
ylabel('BER')
title('BER vs SNR')
disp('end');
```

Basic System using MRC(Matlab code):

```
clear all
BERric = zeros(1,7);
for dbCount = 1:1:7
snr = dbCount;
TempBER = 0;
Smreceived=zeros(1,546);
```

```
m=0;
for burstCount=1:1:500
%% transmission

%generate information bits

mInfoBitsTx = randint(1,1000);

%generate guard bits

mGuardBitsTx = zeros(1,6);

%generate unique word

a = [1,0,1,1,0,1,0,0];

mUniqueWordTx = [a,a,a,a,a,a,a,a,a,a];

% generate burst

mBurstTx = [mGuardBitsTx,mUniqueWordTx,mInfoBitsTx,mGuardBitsTx];

%modulation

mSymbolsTx = QpskModulation(mBurstTx,length(mBurstTx));

% upsampling

mUpSamples = upsample(mSymbolsTx,16);

% Generate a square root rcosine filter system

mPulseShaping = rcosine(1,16,'sqrt',0.35);

% Pulse shaping

mSignalTx = PulseShaping(mUpSamples,mPulseShaping);

%es(i)=var(mSignalTx);
```

```
%% generate rican fast fading channel
Tsample = 0.5 * 1 * 10 ^ (-3) / (546 * 16);

Fm = 20;

K = 7;

mChannelRician2 = Rician(Tsample,Fm,K);
mChannelRician3 = Rician(Tsample,Fm,K);
mChannelRician4 = Rician(Tsample,Fm,K);
mChannelRician5 = Rician(Tsample,Fm,K);

% generate noise
% snr = 10 log10(signalpower / 0(k^2 * noisepower)
%snr = 7;

signalPow = 0.9980;

noisePow = 1.9908;

k = sqrt((signalPow) / ((10^(snr/10)) * (noisePow)));

mNoise = complex(k*randn(1,8736),k*randn(1,8736));

% mix the signal with the channel

mSignalRx1 = mSignalTx .* mChannelRician2 + mNoise;
mSignalRx2 = mSignalTx .* mChannelRician3 + mNoise;
mSignalRx3 = mSignalTx .* mChannelRician4 + mNoise;
mSignalRx4 = mSignalTx .* mChannelRician5 + mNoise;

%% reception
% Matched filter

mFilteredSignalRx1 = MatchedFilter(mSignalRx1,mPulseShaping);
mFilteredSignalRx2 = MatchedFilter(mSignalRx2,mPulseShaping);
mFilteredSignalRx3 = MatchedFilter(mSignalRx3,mPulseShaping);
mFilteredSignalRx4 = MatchedFilter(mSignalRx4,mPulseShaping);

% downsampling
```

```
mSymbols1 = downsample(mFilteredSignalRx1,16);
mSymbols2 = downsample(mFilteredSignalRx2,16);
mSymbols3 = downsample(mFilteredSignalRx3,16);
mSymbols4 = downsample(mFilteredSignalRx4,16);

mSymbolsRx1 = BPSE(mSymbols1,mSymbolsTx);
mSymbolsRx2 = BPSE(mSymbols2,mSymbolsTx);
mSymbolsRx3 = BPSE(mSymbols3,mSymbolsTx);
mSymbolsRx4 = BPSE(mSymbols4,mSymbolsTx);


%Sm=[1 -1 j -j];
for i=1:1:length(mSymbolsTx)
m=(conj(mChannelRician2(i)).*mSymbolsRx1(i))+(conj(mChannelRician3(i)).*m
SymbolsRx2(i))+(conj(mChannelRician4(i)).*mSymbolsRx3(i))+(conj(mChannel
Rician5(i)).*mSymbolsRx4(i));
S1=real(m);
S2=real(-m);
S3=real(-j.*m);
S4=real(j.*m);
S=[S1 S2 S3 S4];
value=max(S);
if(value==S1)
Smreceived(i)=1;
elseif(value==S2)
Smreceived(i)=-1;
elseif(value==S3)
Smreceived(i)=j;
else
Smreceived(i)=-j;
end
end

result = (mSymbolsTx==mSymbolsRx2);

%demodulation

mBurstRx = QpskDemodulation(Smreceived,length(Smreceived));
```

```
%mGuardBitsRxFront = mBurstRx(1,1:6);

%mUniqueWordRx = mBurstRx (1,7:86);

mInfoBitsRx = mBurstRx(1,87:1086);

%mGuardBitsRxRear = mBurstRx(1,1087:1092);

TempBER = TempBER +
(biterr(mInfoBitsTx,mInfoBitsRx)/length(mInfoBitsRx));
end
snr
BERric(dbCount) = TempBER/burstCount;
end
%BERtheo = [0.09 0.0699 0.0527 0.0387 0.0277 0.0193 0.0135];
semilogy(1:1:7,BERric,'-bo');
hold on
%semilogy(1:1:7,BERtheo,'-r*');
hold off
grid on
xlabel('SNR (dB)')
ylabel('BER')
title('BER vs SNR')
disp('end');
```

Convolutional coder (Matlab Code):

```
clear all
BERric = zeros(1,7);
Decodedbits=zeros(1,500);

for dbCount = 1:1:7
snr = dbCount;
TempBER = 0;
for burstCount=1:1:1000
%% transmission

%generate information bits
```

```
mInfoBitsTx = randint(1,500);
trellis=poly2trellis(7,[133,171]);
c=convenc(mInfoBitsTx,trellis);

%generate guard bits

mGuardBitsTx = zeros(1,6);

%generate unique word

a = [1,0,1,1,0,1,0,0];

mUniqueWordTx = [a,a,a,a,a,a,a,a,a,a];

% generate burst

mBurstTx = [mGuardBitsTx,mUniqueWordTx,c,mGuardBitsTx];

%modulation

mSymbolsTx = QpskModulation(mBurstTx,length(mBurstTx));

% upsampling

mUpSamples = upsample(mSymbolsTx,16);

% Generate a square root rcosine filter system

mPulseShaping = rcosine(1,16,'sqrt',0.35);

% Pulse shaping

mSignalTx = PulseShaping(mUpSamples,mPulseShaping);

%es(i)=var(mSignalTx);

%% generate rican fast fading channel
Tsample = 0.5 * 1 * 10 ^ (-3) / (546 * 16);
```

```
Fm = 100;

K = 12;

mChannelRician = Rician(Tsample,Fm,K);

% generate noise
% snr = 10 log10(signalpower / (k^2 * noisepower)
%snr = 7;

signalPow = 0.9980;

noisePow = 1.9908;

k = sqrt((signalPow) / ((10^(snr/10)) * (noisePow)));

mNoise = complex(k*randn(1,8736),k*randn(1,8736));

% mix the signal with the channel

mSignalRx = mSignalTx .* mChannelRician + mNoise;

%% reception
% Matched filter

mFilteredSignalRx = MatchedFilter(mSignalRx,mPulseShaping);

% downsamplingK

mSymbolsRx = downsample(mFilteredSignalRx,16);

% BPSE

mUwTx = mSymbolsTx(4:43);

mUwRx = mSymbolsRx(4:43);

mUwLen = length(mUwTx);

mSigma = 0;
```

```
for i=1:1:mUwLen

mSigma = mSigma + (mUwTx(i) .* conj(mUwRx(i)));

end

theta = angle(mSigma / mUwLen);

theta1 = theta;

Nbpe = 50;

Mbpe = 20;

mBusrtLen = length(mSymbolsRx);

mAngleArray = zeros(1,mBusrtLen);

for i = 1 : Mbpe : (mBusrtLen - Nbpe)

innerLen = i + Nbpe;

mImagSignal = 0;

mRealSignal = 0;

for j = i : 1 : innerLen

mImagSignal = mImagSignal + (imag(mSymbolsRx(j)) ^ 4);

mRealSignal = mRealSignal + (real(mSymbolsRx(j)) ^ 4);

end

mAngle = atan(mImagSignal/mRealSignal) / 4;

mTempAngles = [mAngle, (mAngle + (pi / 4)), (mAngle - (pi / 4)), (mAngle + (pi
/ 2)), (mAngle - (pi / 2)),   (mAngle + (3 * pi / 4)), (mAngle - (3 *pi /4)), (mAngle -
pi)];
```

```
[min_difference, array_position] = min(abs(mTempAngles - theta1));

mAngleArray(i) = mTempAngles(array_position);

theta1 = mAngleArray(i);

end

hm = mfilt.linearinterp(Mbpe);

y = filter(hm,mAngleArray(1:Mbpe:(mBusrtLen - Nbpe)));
mAngleArray1 = zeros(1,mBusrtLen);
for i = 1:length(y)-Mbpe

mAngleArray1(i) = y(i+Mbpe-1);

end
for i = length(y)-Mbpe+1:1:length(mAngleArray1)

mAngleArray1(i) = y(length(y));

end
%stem(1:1:(mBusrtLen - Nbpe),mAngleArray(1:1:(mBusrtLen - Nbpe)));
%figure(2);
%plot(mAngleArray1);
%disp('end');

mSymbolsRx1 = zeros(1,length(mAngleArray1));

%phase componsation
for i = 1:1:length(mAngleArray1)
mSymbolsRx1(i) = mSymbolsRx(i) .*exp(-j*mAngleArray1(i));
end

%hard-slicer
mSymbolsRx2 = zeros(1,length(mAngleArray1));
for i = 1:1:length(mSymbolsRx1)
symbolAngle = angle(mSymbolsRx1(i));
```

```
if  symbolAngle > pi
symbolAngle = symbolAngle - 2*pi;
elseif symbolAngle < -pi
symbolAngle = symbolAngle + 2*pi;
end


if (symbolAngle >= 0 && symbolAngle < pi/4) || (symbolAngle >= -pi/4 &&
symbolAngle < 0)
mSymbolsRx2(i) = 1;
elseif (symbolAngle >= pi/4 && symbolAngle < 3*pi/4)
mSymbolsRx2(i) = complex(0,1);
elseif (symbolAngle >= 3*pi/4 && symbolAngle < pi) || (symbolAngle >= -pi &&
symbolAngle < -3*pi/4 )
mSymbolsRx2(i) = -1;
elseif (symbolAngle >= -3*pi/4 && symbolAngle < -pi/4)
mSymbolsRx2(i) = complex(0,-1);
else
mSymbolsRx2(i) = 0;
disp('a');
end
end
%       result = (mSymbolsTx==mSymbolsRx2);

%demodulation

mBurstRx = QpskDemodulation(mSymbolsRx2,length(mSymbolsRx));

%mGuardBitsRxFront = mBurstRx(1,1:6);

%mUniqueWordRx = mBurstRx (1,7:86);

mInfoBitsRx = mBurstRx(1,87:1086);
Decodedbits=vitdec(mInfoBitsRx,trellis,35,'term','hard');

%mGuardBitsRxRear = mBurstRx(1,1087:1092);

TempBER = TempBER +
(biterr(Decodedbits,mInfoBitsTx)/length(mInfoBitsRx));
end
```

```
snr
BERric(dbCount) = TempBER/burstCount;
end
BERtheo = [0.09 0.0699 0.0527 0.0387 0.0277 0.0193 0.0135];
semilogy(1:1:7,BERric,'-bo');
hold on
semilogy(1:1:7,BERtheo,'-r*');
hold off
grid on
xlabel('SNR (dB)')
ylabel('BER')
title('BER vs SNR')
disp('end');
```

Puncturing and Interleaving (Matlab code):

```
clear all
BERric = zeros(1,7);
Decodedbits=zeros(1,666);
puncturedbits=zeros(1,1000);
depuncturedbits=zeros(1,1332);
txbits=zeros(1,1000);
rxbits=zeros(1,1000);
c=zeros(1,1000);
d=zeros(1,50);
e=zeros(1,50);
intbits=zeros(1,50);
for dbCount = 1:1:7
snr = dbCount;
TempBER = 0;
for burstCount=1:1:800
%% transmission

%generate information bits

mInfoBitsTx = randint(1,666);
trellis=poly2trellis(7,[133,171]);
c=convenc(mInfoBitsTx,trellis,[1 1 1 0]);
```

```
c(1000)=0;
for j=1:50:1000
d=c(j:j+49);
intbits=matintrlv(d,10,5);
txbits(j:j+49)=intbits;
end

%generate guard bits

mGuardBitsTx = zeros(1,6);

%generate unique word

a = [1,0,1,1,0,1,0,0];

mUniqueWordTx = [a,a,a,a,a,a,a,a,a,a];

% generate burst

mBurstTx = [mGuardBitsTx,mUniqueWordTx,txbits,mGuardBitsTx];

%modulation

mSymbolsTx = QpskModulation(mBurstTx,length(mBurstTx));

% upsampling

mUpSamples = upsample(mSymbolsTx,16);

% Generate a square root rcosine filter system

mPulseShaping = rcosine(1,16,'sqrt',0.35);

% Pulse shaping

mSignalTx = PulseShaping(mUpSamples,mPulseShaping);

%es(i)=var(mSignalTx);

%% generate rican fast fading channel
```

```
Tsample = 0.5 * 1 * 10 ^ (-3) / (546 * 16);

Fm =0;

K = 200;

mChannelRician = Rician(Tsample,Fm,K);

% generate noise
% snr = 10 log10(signalpower / (k^2 * noisepower)
%snr = 7;

signalPow = 0.9980;

noisePow = 1.9908;

k = sqrt((signalPow) / ((10^(snr/10)) * (noisePow)));

mNoise = complex(k*randn(1,8736),k*randn(1,8736));

% mix the signal with the channel

mSignalRx = mSignalTx .* mChannelRician + mNoise;

%% reception
% Matched filter

mFilteredSignalRx = MatchedFilter(mSignalRx,mPulseShaping);

% downsampling

mSymbolsRx = downsample(mFilteredSignalRx,16);

% BPSE

mUwTx = mSymbolsTx(4:43);

mUwRx = mSymbolsRx(4:43);

mUwLen = length(mUwTx);
```

```
mSigma = 0;
for i=1:1:mUwLen
mSigma = mSigma + (mUwTx(i) .* conj(mUwRx(i)));
end
theta = angle(mSigma / mUwLen);
theta1 = theta;
Nbpe = 50;
Mbpe = 20;
mBusrtLen = length(mSymbolsRx);
mAngleArray = zeros(1,mBusrtLen);
for i = 1 : Mbpe : (mBusrtLen - Nbpe)
innerLen = i + Nbpe;
mImagSignal = 0;
mRealSignal = 0;
for j = i : 1 : innerLen
mImagSignal = mImagSignal + (imag(mSymbolsRx(j)) ^ 4);
mRealSignal = mRealSignal + (real(mSymbolsRx(j)) ^ 4);
end
mAngle = (atan(mImagSignal/mRealSignal)) / 4;
```

```
mTempAngles = [mAngle, (mAngle + (pi / 4)), (mAngle - (pi / 4)), (mAngle + (pi
/ 2)), (mAngle - (pi / 2)),   (mAngle + (3 * pi / 4)), (mAngle - (3 *pi /4)), (mAngle -
pi)];

[min_difference, array_position] = min(abs(mTempAngles - theta1));

mAngleArray(i) = mTempAngles(array_position);

theta1 = mAngleArray(i);

end

hm = mfilt.linearinterp(Mbpe);

y = filter(hm,mAngleArray(1:Mbpe:(mBusrtLen - Nbpe)));
mAngleArray1 = zeros(1,mBusrtLen);
for i = 1:length(y)-Mbpe

mAngleArray1(i) = y(i+Mbpe-1);

end
for i = length(y)-Mbpe+1:1:length(mAngleArray1)

mAngleArray1(i) = y(length(y));

end
%stem(1:1:(mBusrtLen - Nbpe),mAngleArray(1:1:(mBusrtLen - Nbpe)));
%figure(2);
%plot(mAngleArray1);
%disp('end');

mSymbolsRx1 = zeros(1,length(mAngleArray1));

%phase componsation
for i = 1:1:length(mAngleArray1)
mSymbolsRx1(i) = mSymbolsRx(i) .*exp(-i*mAngleArray1(i));
end

%hard-slicer
mSymbolsRx2 = zeros(1,length(mAngleArray1));
```

```
for i = 1:1:length(mSymbolsRx1)
symbolAngle = angle(mSymbolsRx1(i));

if  symbolAngle > pi
symbolAngle = symbolAngle - 2*pi;
elseif symbolAngle < -pi
symbolAngle = symbolAngle + 2*pi;
end


if (symbolAngle >= 0 && symbolAngle < pi/4) || (symbolAngle >= -pi/4 &&
symbolAngle < 0)
mSymbolsRx2(i) = 1;
elseif (symbolAngle >= pi/4 && symbolAngle < 3*pi/4)
mSymbolsRx2(i) =complex(0,1);
elseif (symbolAngle >= 3*pi/4 && symbolAngle < pi) || (symbolAngle >= -pi &&
symbolAngle < -3*pi/4 )
mSymbolsRx2(i) = -1;
elseif (symbolAngle >= -3*pi/4 && symbolAngle < -pi/4)
mSymbolsRx2(i) =complex(0,-1);
else
mSymbolsRx2(i) = 0;
disp('a');
end
end
%       result = (mSymbolsTx==mSymbolsRx2);

%demodulation

mBurstRx = QpskDemodulation(mSymbolsRx2,length(mSymbolsRx));

%mGuardBitsRxFront = mBurstRx(1,1:6);

%mUniqueWordRx = mBurstRx (1,7:86);

mInfoBitsRx = mBurstRx(1,87:1086);
for j=1:50:1000
e=mInfoBitsRx(j:j+49);
intbits=matdeintrlv(e,10,5);
rxbits(j:j+49)=intbits;
```

```
end


deinterleavedbits=rxbits(1:999);
Decodedbits=vitdec(deinterleavedbits,trellis,70,'trunc','hard',[1 1 1 0]);




%mGuardBitsRxRear = mBurstRx(1,1087:1092);

TempBER = TempBER +
(biterr(Decodedbits,mInfoBitsTx)/length(mInfoBitsRx));
end
snr
BERric(dbCount) = TempBER/burstCount;
end
BERtheo = [0.09 0.0699 0.0527 0.0387 0.0277 0.0193 0.0135];
semilogy(1:1:7,BERric,'-bo');
hold on
semilogy(1:1:7,BERtheo,'-r*');
hold off
grid on
xlabel('SNR (dB)')
ylabel('BER')
title('BER vs SNR')
disp('end');
```

1) QPSK Modulation User function:

```
function[mSymbolsTx] = QpskModulation(mBurstTx,mBurstLen)

mSymbolsTx = zeros(1,(mBurstLen/2));

mCount=0;

for mIndex = 1:mBurstLen/2

mCount=mCount+1;
```

```
if mBurstTx(2*mIndex-1) == 0

if mBurstTx(2*mIndex) == 0

mSymbolsTx(mCount) = 1;

elseif mBurstTx(2*mIndex) == 1

mSymbolsTx(mCount) = j;

end

elseif mBurstTx(2*mIndex-1) == 1

if mBurstTx(2*mIndex) == 1

mSymbolsTx(mCount) = -1;

elseif mBurstTx(2*mIndex) == 0

mSymbolsTx(mCount) = -j;

end
end
end
```

2) Rician User Function:

```
function[Rician] = Rician(Tsample, Fm, K)

%N - No of multipaths
N = 70;

M = 0.5 * ((N / 2) - 1);

Wm = 2 * pi * Fm;

a = 0;
```

```
t0 = rand;
G = zeros(1,8736);
Gi = zeros(1,8736);
Gq = zeros(1,8736);

for i = 1:1:8736
t = t0 +(i* Tsample);
for n = 1:1:M

Bn = (n * pi) / M;

Wn = Wm * cos((2 * pi * n )/ N);

Gi(i) = Gi(i) + (2 * cos(Bn) * cos(Wn * t));

Gq(i) = Gq(i) + (2 * sin(Bn) * cos(Wn * t));

end

Gi(i) = (Gi(i) + (sqrt(2) * cos(Wm * t) * cos(a))) / sqrt(M + 1);

Gq(i) = (Gq(i) + (sqrt(2) * cos(Wm * t) * sin(a))) / sqrt(M);

end

G = complex(Gi,Gq);

%Rayleigh

Aric = 10 ^ (-K / 20);

Bric = 1 / sqrt(1 + Aric ^ 2);

Rician = (1 + (Aric .* G)) .* Bric;

%m = 1:1:8736
%subplot(2,2,1), plot(m,real(Rician));
%subplot(2,2,2), plot(m,imag(Rician));
```

QPSK De-Modulation User Function:

```
function[mBurstRx]=QpskDemodulation(mSymbolsRx,mSymbolsRxLe)

mBurstRx = zeros(1,mSymbolsRxLen*2);
mCount = 1;

for mIndex = 1:mSymbolsRxLen

if mSymbolsRx(1,mIndex) == 1

mBurstRx(mCount) = 0;
mBurstRx(mCount+1) = 0;

elseif mSymbolsRx(1,mIndex) == j

mBurstRx(mCount) = 0;
mBurstRx(mCount+1) = 1;

elseif mSymbolsRx(1,mIndex) == -1

mBurstRx(mCount) = 1;
mBurstRx(mCount+1) = 1;

elseif mSymbolsRx(1,mIndex) == -j

mBurstRx(mCount) = 1;
mBurstRx(mCount+1) = 0;

end
mCount = mCount + 2;
end
```

BPSE user function:

```
function[mSymbolsRx1]=BPSE1(mSymbols1,mSymbolsTx)
```

```
% BPSE
mUwTx = mSymbolsTx(4:43);
mUwRx = mSymbols1(4:43);
mUwLen = length(mUwTx);
mSigma = 0;
for i=1:1:mUwLen
mSigma = mSigma + (mUwTx(i) .* conj(mUwRx(i)));
end
theta = angle(mSigma / mUwLen);
theta1 = theta;
Nbpe = 50;
Mbpe = 20;
mBusrtLen = length(mSymbols1);
mAngleArray = zeros(1,mBusrtLen);
for i = 1 : Mbpe : (mBusrtLen - Nbpe)
innerLen = i + Nbpe;
mImagSignal = 0;
mRealSignal = 0;
for j = i : 1 : innerLen
```

```
mImagSignal = mImagSignal + (imag(mSymbols1(j)) ^ 4);

mRealSignal = mRealSignal + (real(mSymbols1(j)) ^ 4);

end

mAngle = atan((mImagSignal/mRealSignal)) / 4;

mTempAngles = [mAngle, (mAngle + (pi / 4)), (mAngle - (pi / 4)), (mAngle + (pi
/ 2)), (mAngle - (pi / 2)),   (mAngle + (3 * pi / 4)), (mAngle - (3 *pi /4)),
(mAngle+pi)];

[min_difference, array_position] = min(abs(mTempAngles - theta1));

mAngleArray(i) = mTempAngles(array_position);

theta1 = mAngleArray(i);

end

hm = mfilt.linearinterp(Mbpe);

y = filter(hm,mAngleArray(1:Mbpe:(mBusrtLen - Nbpe)));
mAngleArray1 = zeros(1,mBusrtLen);
for i = 1:length(y)-Mbpe

mAngleArray1(i) = y(i+Mbpe-1);

end
for i = length(y)-Mbpe+1:1:length(mAngleArray1)

mAngleArray1(i) = y(length(y));

end
%stem(1:1:(mBusrtLen - Nbpe),mAngleArray(1:1:(mBusrtLen - Nbpe)));
%figure(2);
%plot(mAngleArray1);
%disp('end');

mSymbolsRx1 = zeros(1,length(mAngleArray1));
```

```
%phase componsation

for i = 1:1:length(mAngleArray1)
mSymbolsRx1(i) = mSymbols1(i) .*exp(-j*mAngleArray1(i));
end
```

Pulse shaping user function:

```
function[mSignalTx] = PulseShaping(mUpSamples,mPulseShaping)

mPulseShapingLen = length(mPulseShaping);

mSignalConvTx = conv(mUpSamples,mPulseShaping);

mUpSamplesLen = length(mSignalConvTx);

mSignalTx = mSignalConvTx(((mPulseShapingLen 1)/2):(mUpSamplesLen -
((mPulseShapingLen - 1)/2)));
```

Matched Filter user function:

```
function[mFilteredSignalRx] = MatchedFilter(mSignalRx,mPulseShaping)

mPulseShapingLen = length(mPulseShaping);

mSignalConvRx = conv(mSignalRx,mPulseShaping);

mSignalRxLen = length(mSignalConvRx);

mFilteredSignalRx = mSignalConvRx(((mPulseShapingLen +
1)/2):(mSignalRxLen - ((mPulseShapingLen - 1)/2)));
```