Cairo University Faculty of Engineering

Electronics and Electrical Communications Department

Fourth Year Communications Report

Orthogonal Frequency Division Multiplexing

Name	Section	B.N.	I.D.
Khaled Waleed Samir Metwally	2	12	9202515
Mazen Wael Diaa	3	35	9210892

Table of Contents

List of Figures

Problem 1: Execution time of DFT and FFT

a) MATLAB function to implement DFT

In this part we implement the DFT function according to the following equation:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-\frac{j2\pi nk}{N}} \quad 0 \leq k \leq N-1$$

```
function X_k = DFT(x_n)
    N = length(x_n);
    X_k = zeros(1,N);
    n = 0:N-1;
    for k = 0:N-1
        X_k(k+1) = sum(x_n.*exp(-2j * pi * k * n / N));
    end
end
```

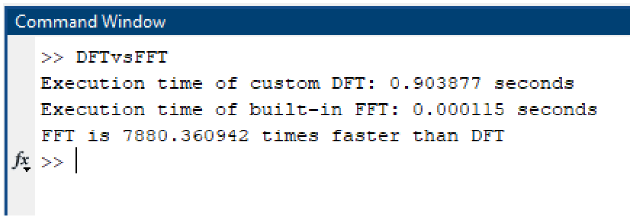
b) Testing the implemented DFT function and the FFT built-in function

```
% Generate a random signal of length 8192
N = 8192;
x_n = randn(1, N);

% Measure the execution time of the DFT
tic;            % Start timing
X_k = DFT(x_n); % Compute the DFT using the custom function
time_DFT = toc; % End timing and get elapsed time
fprintf('Execution time of custom DFT: %.6f seconds\n', time_DFT);

% Measure the execution time of the FFT
tic;                        % Start timing
X_k_builtin = fft(x_n); % Compute the FFT using the built-in function
time_builtin = toc;     % End timing and get elapsed time
fprintf('Execution time of built-in FFT: %.6f seconds\n', time_builtin);
fprintf('FFT is %.6f times faster than DFT \n',time_DFT/time_builtin);
```

c) Comparison between DFT and FFT in terms of execution time



```
>> DFTvsFFT
Execution time of custom DFT: 0.903877 seconds
Execution time of built-in FFT: 0.000115 seconds
FFT is 7880.360942 times faster than DFT
>>
```

Figure 1: *Execution time comparison between implemented DFT and built-in FFT.*

From the output in **Figure 1**, the built-in FFT has superior performance in terms of execution time vs the implemented DFT function. This significant speedup is due to the efficiency of the Fast Fourier Transform (FFT) algorithm, which reduces the computational complexity from $O(N^2)$ in the case of the straightforward DFT to $O(N\ LogN)$ for the FFT.

Problem 2: Bit-error rate performance for BPSK, QPSK and 16-QAM over Rayleigh flat fading channel

In this section it is required to simulate the Bit-Error Rate (BER) performance of BPSK, QPSK and 16-QAM when transmitting over Rayleigh flat fading channel. This is done by generating a random binary data stream then mapping the binary bits to symbols based on the modulation scheme used then sending over a Rayleigh flat fading channel. The channel's impulse response is simulated as:

$$h(t) = (h_r + jh_i)\delta(t)$$

where h_r is the real part of the channel impulse response and is a Gaussian random variable with zero mean and variance = 1/2. And similarly, h_i is the imaginary part of the channel impulse response and it is a Gaussian random variable with zero mean and variance = 1/2.

Additive White Gaussian Noise (AWGN) is also added to the signal and is simulated as:

$$n(t) = n_c + jn_s$$

So, the overall received symbol is:

$$y_k = (h_r + h_i)_k x_k + (n_c + jn_s)_k \; k = 0, 1, 2, \ldots$$

Then compensation for the channel gain is done at receiver is done assuming the channel is known at the receiver.

Finally, the received symbols are de-mapped to the binary bitstream and the BER is calculated for different SNR values. These steps are repeated but using a 1/3 rate repetition code by transmitting every "1" as three "1" s and every "0" as three "0" s then mapping the output coded bits to the selected modulation scheme. When using the repetition code, it is applied with the same energy per transmitted bit.

The BPSK system model is show in **Figure 2**.

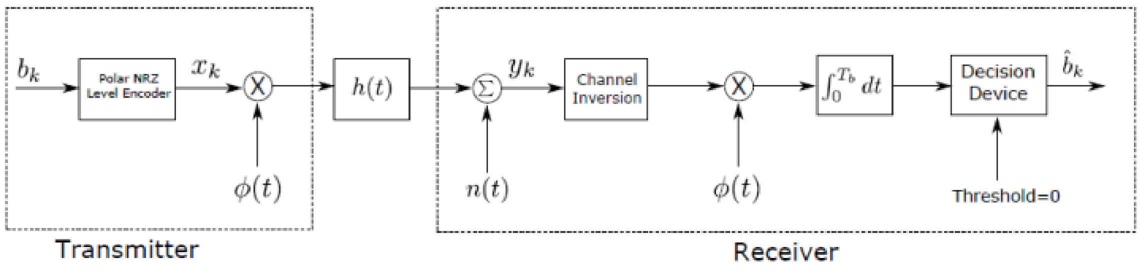


Figure 2: *BPSK System model.*

a) BER for BPSK over Rayleigh flat fading channel

```
numBits = 500000;                         % Number of bits to transmit
Eb = 1;                                   % Bit Energy
binary_data = randi([0 1], numBits, 1);   % Generate random binary data bitstream
EbN0_dB = -4:1:14;                        % Eb/N0 range in dB (SNR)
BER_BPSK = zeros(size(EbN0_dB,2),1);      % Array to store BER at different SNR
values
BER_BPSK_Coded = zeros(size(EbN0_dB,2),1); % Array to store BER at different SNR
values

% Applying rate 1/3 repetition code
binary_data_coded = repelem(binary_data, 3);

% BPSK Mapper
BPSK_data = sqrt(Eb)*(2*binary_data-1);
BPSK_data_coded = sqrt(Eb)*(2*binary_data_coded-1);

for i = 1 : size(EbN0_dB,2)
    % Generate the real part of the channel impulse response
    h_r = 0 + sqrt(1/2) * randn(length(BPSK_data), 1);
    h_r_coded = 0 + sqrt(1/2) * randn(length(BPSK_data_coded), 1);
    % Generate the imaginary part of the channel impulse response
    h_i = 0 + sqrt(1/2) * randn(length(BPSK_data), 1);
    h_i_coded = 0 + sqrt(1/2) * randn(length(BPSK_data_coded), 1);
    % Combine the real and imaginary parts to form the complex channel impulse
response
    h = h_r + 1j * h_i;
    h_coded = h_r_coded + 1j * h_i_coded;
    SNR_linear = 10^(EbN0_dB(i)/10);
    N0 = Eb/SNR_linear;    % noise power
    variance = N0/2; % variance of the Gaussian noise
    % Generate the real part of the noise vector
    n_c = 0 + sqrt(variance) * randn(length(BPSK_data), 1);
    n_c_coded = 0 + sqrt(variance) * randn(length(BPSK_data_coded), 1);
    % Generate the imaginary part of the noise vector
    n_s = 0 + sqrt(variance) * randn(length(BPSK_data), 1);
    n_s_coded = 0 + sqrt(variance) * randn(length(BPSK_data_coded), 1);
    % Combine the real and imaginary parts to form the complex noise vector
    n = n_c + 1j * n_s;
    n_coded = n_c_coded + 1j * n_s_coded;
    % Received data
    datareceived = h.*BPSK_data + n;
    datareceived_coded = h_coded.*BPSK_data_coded + n_coded;
    % Compensate for the channel gain at the receiver
    datareceived_equalized = datareceived./h;
    datareceived_equalized_coded = datareceived_coded./h_coded;
    % Demapping bitstream
    binary_data_demapped = real(datareceived_equalized) > 0;
    binary_data_demapped_coded = real(datareceived_equalized_coded) > 0;
    % Hard decision decoding of the rate 1/3 repetition code
    hard_decoding_reshape = reshape(binary_data_demapped_coded, 3, []).';
    hard_decoding_sum = sum(hard_decoding_reshape, 2);
    hard_decoding_binary_data = hard_decoding_sum >= 2;
    % Calculating BER
    num_error_bits = sum(binary_data ~= binary_data_demapped);
    num_error_bits_coded = sum(binary_data ~= hard_decoding_binary_data);
    BER_BPSK(i) = num_error_bits/numBits;
    BER_BPSK_Coded(i) = num_error_bits_coded/numBits;
end
% Plotting BER
figure('Name' , 'BER of BPSK over Rayleigh flat fading channel');
semilogy(EbN0_dB , BER_BPSK, 'r', 'linewidth', 1);
hold on;
semilogy(EbN0_dB , BER_BPSK_Coded, 'b', 'linewidth', 1);
hold off;
title('BER of BPSK over Rayleigh flat fading channel');
xlabel('EB/No(dB)');  ylabel('BER');  grid on;
legend('BPSK no coding','BPSK with rate 1/3 repetition code');
% Limit y-axis to 10^-3
ylim([1e-3, 1]);
```

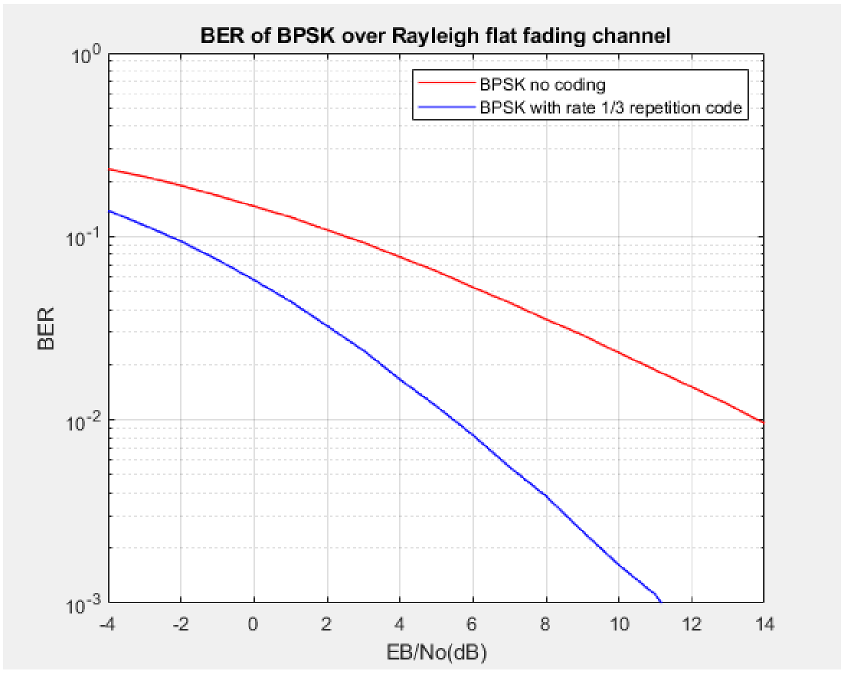


Figure 3: *BER of BPSK over Rayleigh flat fading channel with and without repetition code.*

Comment: Coding improves BER as it has the same energy per transmitted bit.

b) BER for QPSK over Rayleigh flat fading channel

In this part the QPSK mapping is performed by the following function:

```
% QPSK Gray Mapper
% Maps binary data to QPSK symbols using Gray code mapping.
% Inputs:
%   binary_data: Binary data to be mapped to QPSK symbols.
% Outputs:
%   QPSK_Data_Mapped: QPSK symbols mapped from binary data using Gray code.
function QPSK_Data_Mapped = QPSK_Gray_Mapper(binary_data)
% Reshape the binary data to have each 2 consecutive bits as 1 symbol
% (Each Row is a Symbol)
QPSK_Table = [-1-1i, -1+1i, 1-1i, 1+1i];
QPSK_Data = reshape(binary_data, 2, []).';
numRows = size(QPSK_Data, 1);
QPSK_Data_Mapped = zeros(numRows, 1);
for i = 1:numRows
    % Convert binary data to decimal
    decimalValue = bi2de(QPSK_Data(i, :), 'left-msb');

    % Use decimal value as index to access corresponding symbol from QPSK table
    % Add 1 because MATLAB indices start from 1
    QPSK_Data_Mapped(i) = QPSK_Table(decimalValue + 1);
end
end
```

And the QPSK de-mapping is performed by the following function:

```
% QPSK Gray Demapper
% Demaps received QPSK symbols to binary data using Gray code demapping.
% Inputs:
%   QPSK_Data: Received QPSK symbols to be demapped to binary data.
% Outputs:
%   QPSK_Data_Demapped: Demapped binary data from received QPSK symbols.
function QPSK_Data_Demapped = QPSK_Gray_Demapper(QPSK_Data)
numBits = size(QPSK_Data,1)*2;
QPSK_Data_Demapped = zeros(size(QPSK_Data, 1), 2);
QPSK_Table = [-1-1i, -1+1i, 1-1i, 1+1i];
% Iterate over each received symbol
for i = 1:size(QPSK_Data, 1)
    % Calculate distance to each QPSK constellation point
    distances = abs(QPSK_Data(i) - QPSK_Table);

    % Find index of closest constellation point
    [~, index] = min(distances);

    % Convert the index to binary representation
    binary_QPSK = de2bi(index - 1, 2, 'left-msb');

    % Store the binary representation in the demapped array
    QPSK_Data_Demapped(i, :) = binary_QPSK;
end
QPSK_Data_Demapped = reshape(QPSK_Data_Demapped.',1,numBits)';
end
```

The code using the QPSK mapping and de-mapping functions:

```
numBits = 500000;                         % Number of bits to transmit
Eb = 1;                                   % Bit Energy
binary_data = randi([0 1], numBits, 1);   % Generate random binary data bitstream
EbN0_dB = -4:1:14;                        % Eb/N0 range in dB (SNR)
BER_QPSK = zeros(size(EbN0_dB,2),1);      % Array to store BER at different SNR
values
BER_QPSK_Coded = zeros(size(EbN0_dB,2),1); % Array to store BER at different SNR
values

% Applying rate 1/3 repetition code
binary_data_coded = repelem(binary_data, 3);

% QPSK Mapper
QPSK_data = QPSK_Gray_Mapper(binary_data);
QPSK_data_coded = QPSK_Gray_Mapper(binary_data_coded);

for i = 1 : size(EbN0_dB,2)
    % Generate the real part of the channel impulse response
    h_r = 0 + sqrt(1/2) * randn(length(QPSK_data), 1);
    h_r_coded = 0 + sqrt(1/2) * randn(length(QPSK_data_coded), 1);
    % Generate the imaginary part of the channel impulse response
    h_i = 0 + sqrt(1/2) * randn(length(QPSK_data), 1);
    h_i_coded = 0 + sqrt(1/2) * randn(length(QPSK_data_coded), 1);
    % Combine the real and imaginary parts to form the complex channel impulse
response
    h = h_r + 1j * h_i;
    h_coded = h_r_coded + 1j * h_i_coded;
    SNR_linear = 10^(EbN0_dB(i)/10);
    N0 = Eb/SNR_linear;    % noise power
    variance = N0/2; % variance of the Gaussian noise
    % Generate the real part of the noise vector
    n_c = 0 + sqrt(variance) * randn(length(QPSK_data), 1);
    n_c_coded = 0 + sqrt(variance) * randn(length(QPSK_data_coded), 1);
    % Generate the imaginary part of the noise vector
    n_s = 0 + sqrt(variance) * randn(length(QPSK_data), 1);
    n_s_coded = 0 + sqrt(variance) * randn(length(QPSK_data_coded), 1);
    % Combine the real and imaginary parts to form the complex noise vector
    n = n_c + 1j * n_s;
    n_coded = n_c_coded + 1j * n_s_coded;
    % Received data
    datareceived = h.*QPSK_data + n;
    datareceived_coded = h_coded.*QPSK_data_coded + n_coded;
    % Compensate for the channel gain at the receiver
    datareceived_equalized = datareceived./h;
    datareceived_equalized_coded = datareceived_coded./h_coded;
    % Demapping bitstream
    binary_data_demapped = QPSK_Gray_Demapper(datareceived_equalized);
    binary_data_demapped_coded = QPSK_Gray_Demapper(datareceived_equalized_coded);
    % Hard decision decoding of the rate 1/3 repetition code
    hard_decoding_reshape = reshape(binary_data_demapped_coded, 3, []).';
    hard_decoding_sum = sum(hard_decoding_reshape, 2);
    hard_decoding_binary_data = hard_decoding_sum >= 2;
```

```
  % Calculating BER
    num_error_bits = sum(binary_data ~= binary_data_demapped);
    num_error_bits_coded = sum(binary_data ~= hard_decoding_binary_data);
    BER_QPSK(i) = num_error_bits/numBits;
    BER_QPSK_Coded(i) = num_error_bits_coded/numBits;
end
% Plotting BER
figure('Name' , 'BER of QPSK over Rayleigh flat fading channel');
semilogy(EbN0_dB , BER_QPSK, 'r', 'linewidth', 1);
hold on;
semilogy(EbN0_dB , BER_QPSK_Coded, 'b', 'linewidth', 1);
hold off;
title('BER of QPSK over Rayleigh flat fading channel');
xlabel('EB/No(dB)');  ylabel('BER');  grid on;
legend('QPSK no coding','QPSK with rate 1/3 repetition code');
% Limit y-axis to 10^-3
ylim([1e-3, 1]);
```

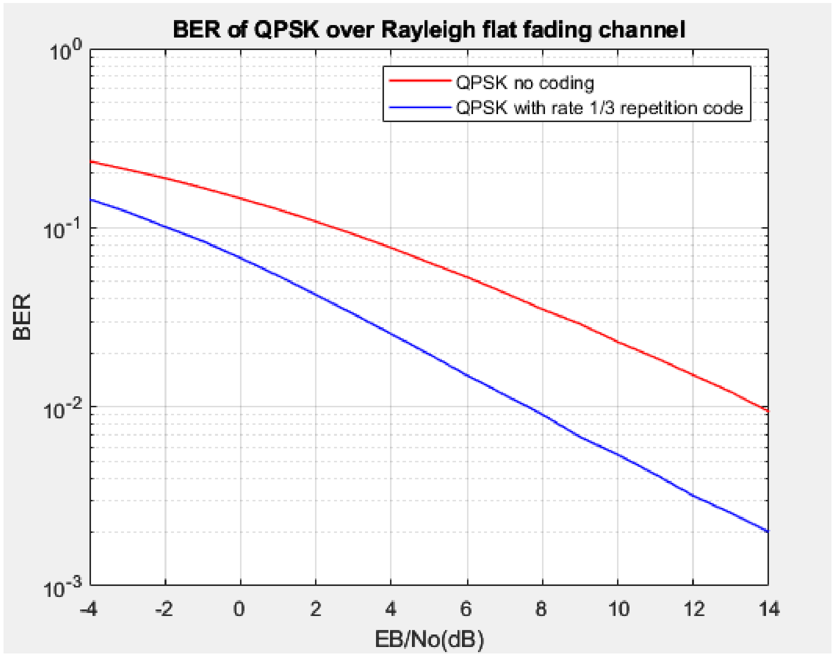


Figure 4: *BER of QPSK over Rayleigh flat fading channel with and without repetition code.*

Comment: QPSK has a similar performance to BPSK without coding but with coding QPSK has a worse performance than the coded BPSK.

c) BER for 16-QAM over Rayleigh flat fading channel

In this part the 16-QAM mapping is performed by the following function:

```
% 16-QAM Mapper
% QAM16_Mapper - Maps binary data to 16-QAM symbols
% Syntax: QAM16_Data_Mapped = QAM16_Mapper(binary_data)
% Input: binary_data: A binary vector containing the input data bits.
% Output: QAM16_Data_Mapped: A column vector containing the mapped 16-QAM symbols.
function QAM16_Data_Mapped = QAM16_Mapper(binary_data)
% Reshape the binary data to have each 4 consecutive bits as 1 symbol
% (Each Row is a Symbol)
QAM_16_Table = [-3-3i, -3-1i, -3+3i, -3+1i, -1-3i, -1-1i, -1+3i, -1+1i, 3-3i, 3-1i,
3+3i, 3+1i, 1-3i, 1-1i, 1+3i, 1+1i];
QAM_16_Data = reshape(binary_data, 4, []).';
% Map each row in QAM_16_Data to a value from the table
numRows = size(QAM_16_Data, 1);
QAM16_Data_Mapped = zeros(numRows, 1);
for i = 1:numRows
    % Convert binary data to decimal
    decimalValue = bi2de(QAM_16_Data(i, :), 'left-msb');
    % Use decimal value as index to access corresponding symbol from QAM table
    % Add 1 because MATLAB indices start from 1
    QAM16_Data_Mapped(i) = QAM_16_Table(decimalValue + 1);
end
end
```

And the 16-QAM de-mapping is performed by the following function:

```
% 16-QAM Demapper
% QAM16_Demapper - Demaps received 16-QAM symbols to binary data
% Syntax:
%   QAM16_Data_Demapped = QAM16_Demapper(QAM16_data)
% Input:
%   QAM16_data: A column vector containing the received 16-QAM symbols.
% Output:
%   QAM16_Data_Demapped: A column vector containing the demapped binary data.
function QAM16_Data_Demapped = QAM16_Demapper(QAM16_data)
numBits = size(QAM16_data,1)*4;
QAM16_Data_Demapped = zeros(size(QAM16_data, 1), 4);
QAM_16_Table = [-3-3i, -3-1i, -3+3i, -3+1i, -1-3i, -1-1i, -1+3i, -1+1i, 3-3i, 3-1i,
3+3i, 3+1i, 1-3i, 1-1i, 1+3i, 1+1i];
% Iterate over each received symbol
for i = 1:size(QAM16_data, 1)
    % Calculate distance to each 16-QAM constellation points
    distances = abs(QAM16_data(i) - QAM_16_Table);
    % Find index of closest constellation point
    [~, index] = min(distances);
    % Convert the index to binary representation
    binary_16QAM = de2bi(index - 1, 4, 'left-msb');
    % Store the binary representation in the demapped array
    QAM16_Data_Demapped(i, :) = binary_16QAM;
end
QAM16_Data_Demapped = reshape(QAM16_Data_Demapped.',1,numBits)';
end
```

The code using the 16-QAM mapping and de-mapping functions:

```
numBits = 500000;                           % Number of bits to transmit
Eb = 1;                                     % Bit Energy
binary_data = randi([0 1], numBits, 1);     % Generate random binary data bitstream
EbN0_dB = -4:1:14;                          % Eb/N0 range in dB (SNR)
BER_16QAM = zeros(size(EbN0_dB,2),1);       % Array to store BER at different SNR
values
BER_16QAM_Coded = zeros(size(EbN0_dB,2),1); % Array to store BER at different SNR
values

% Applying rate 1/3 repetition code
binary_data_coded = repelem(binary_data, 3);

% 16QAM Mapper
QAM16_data = QAM16_Mapper(binary_data);
QAM16_data_coded = QAM16_Mapper(binary_data_coded);

for i = 1 : size(EbN0_dB,2)
    % Generate the real part of the channel impulse response
    h_r = 0 + sqrt(1/2) * randn(length(QAM16_data), 1);
    h_r_coded = 0 + sqrt(1/2) * randn(length(QAM16_data_coded), 1);
    % Generate the imaginary part of the channel impulse response
    h_i = 0 + sqrt(1/2) * randn(length(QAM16_data), 1);
    h_i_coded = 0 + sqrt(1/2) * randn(length(QAM16_data_coded), 1);
    % Combine the real and imaginary parts to form the complex channel impulse
response
    h = h_r + 1j * h_i;
    h_coded = h_r_coded + 1j * h_i_coded;
    SNR_linear = 10^(EbN0_dB(i)/10);
    N0 = Eb/SNR_linear;    % noise power
    variance = 2.5*N0/2; % variance of the Gaussian noise
    % Generate the real part of the noise vector
    n_c = 0 + sqrt(variance) * randn(length(QAM16_data), 1);
    n_c_coded = 0 + sqrt(variance) * randn(length(QAM16_data_coded), 1);
    % Generate the imaginary part of the noise vector
    n_s = 0 + sqrt(variance) * randn(length(QAM16_data), 1);
    n_s_coded = 0 + sqrt(variance) * randn(length(QAM16_data_coded), 1);
    % Combine the real and imaginary parts to form the complex noise vector
    n = n_c + 1j * n_s;
    n_coded = n_c_coded + 1j * n_s_coded;
    % Received data
    datareceived = h.*QAM16_data + n;
    datareceived_coded = h_coded.*QAM16_data_coded + n_coded;
    % Compensate for the channel gain at the receiver
    datareceived_equalized = datareceived./h;
    datareceived_equalized_coded = datareceived_coded./h_coded;
    % Demapping bitstream
    binary_data_demapped = QAM16_Demapper(datareceived_equalized);
    binary_data_demapped_coded = QAM16_Demapper(datareceived_equalized_coded);
    % Hard decision decoding of the rate 1/3 repetition code
    hard_decoding_reshape = reshape(binary_data_demapped_coded, 3, []).';
    hard_decoding_sum = sum(hard_decoding_reshape, 2);
    hard_decoding_binary_data = hard_decoding_sum >= 2;
    % Calculating BER
    num_error_bits = sum(binary_data ~= binary_data_demapped);
    num_error_bits_coded = sum(binary_data ~= hard_decoding_binary_data);
    BER_16QAM(i) = num_error_bits/numBits;
    BER_16QAM_Coded(i) = num_error_bits_coded/numBits;
end
% Plotting BER
figure('Name' , 'BER of 16-QAM over Rayleigh flat fading channel');
semilogy(EbN0_dB , BER_16QAM, 'r', 'linewidth', 1);
hold on;
semilogy(EbN0_dB , BER_16QAM_Coded, 'b', 'linewidth', 1);
hold off;
title('BER of 16-QAM over Rayleigh flat fading channel');
xlabel('EB/No(dB)');  ylabel('BER');  grid on;
legend('BPSK no coding','16-QAM with rate 1/3 repetition code');
% Limit y-axis to 10^-3
ylim([1e-3, 1]);
```

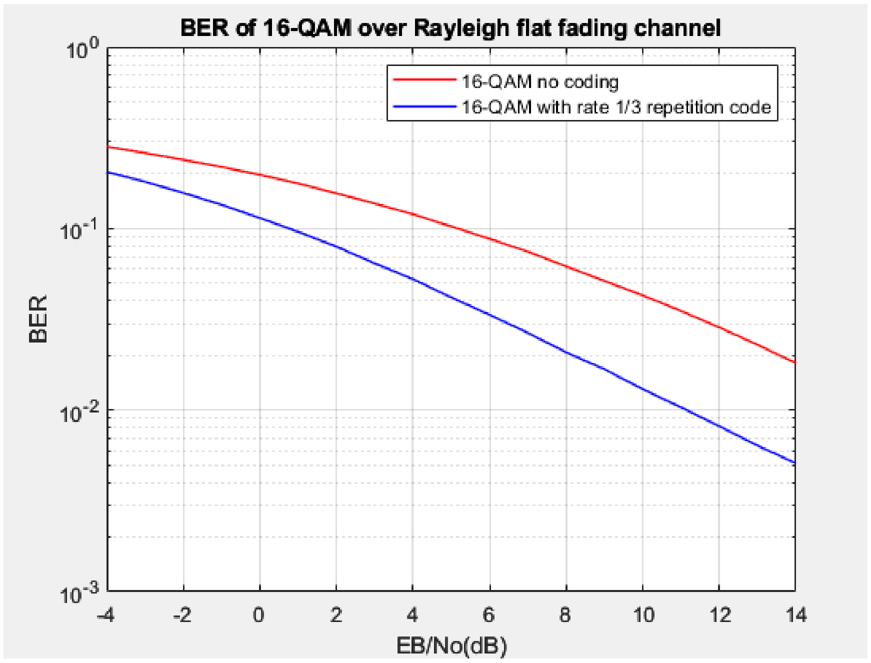


Figure 5: *BER of 16-QAM over Rayleigh flat fading channel with and without repetition code.*

Comment: Similarly coding improves the BER, also shown in **Figure 6** all the different modulation schemes with and without coding over a flat fading channel, BPSK with coding has the best BER then QPSK with coding then the 16-QAM with coding then the no coding BPSK and QPSK and they have similar performance and finally the 16-QAM with no coding has the most BER.

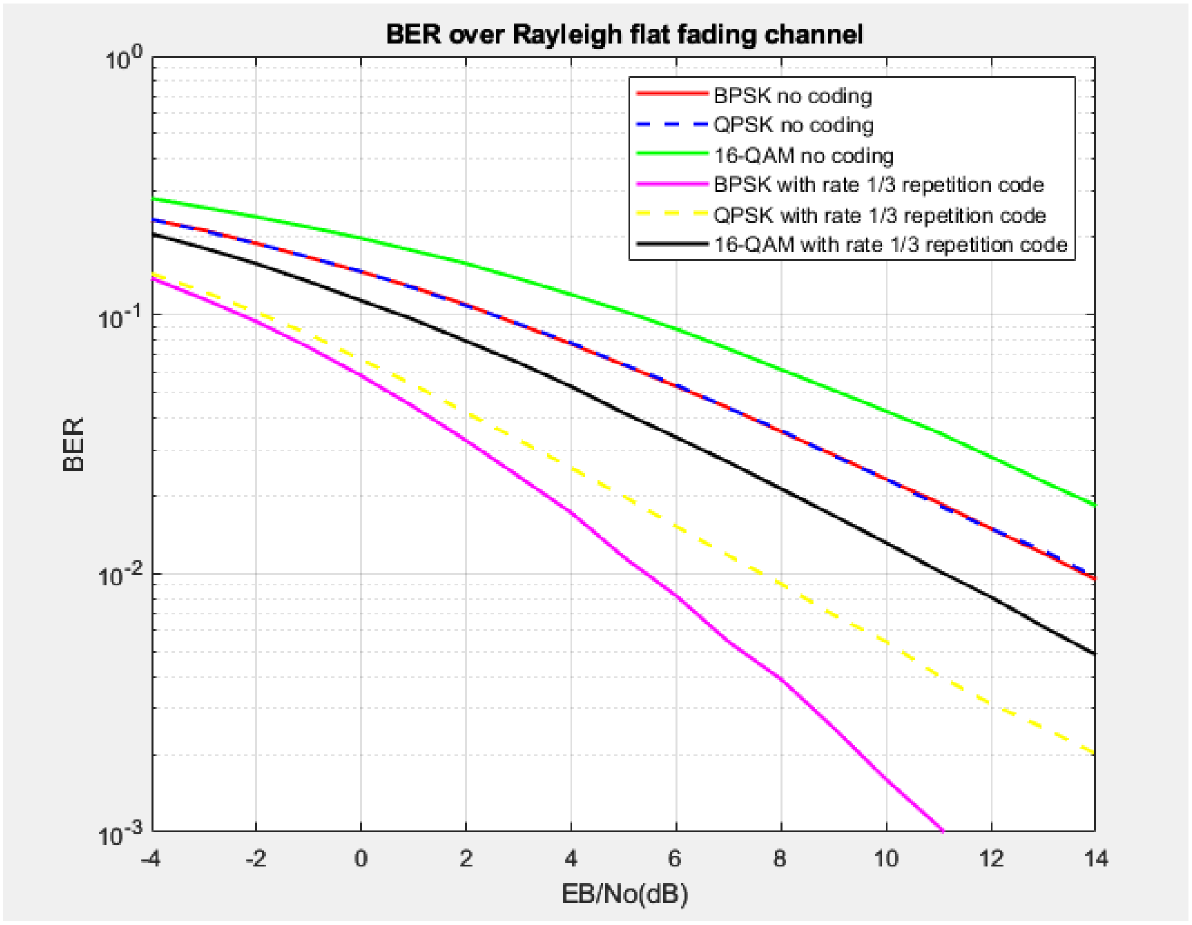


Figure 6: *BER of different modulation schemes with and without coding over a Rayleigh flat fading channel.*

Problem 3: OFDM system simulation

In this part we will be simulating the OFDM system shown in **Figure 7**.

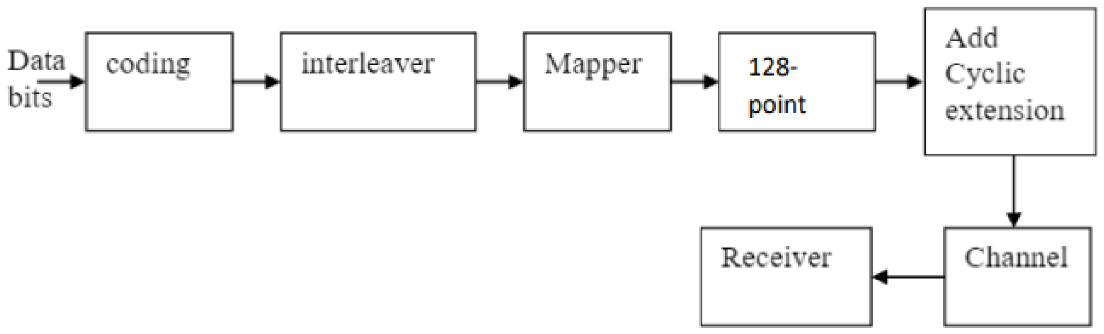


***Figure** 7: OFDM system block diagram.*

In this section there are multiple variations of the simulation, the variables are the modulation scheme, channel type and coding used so we will divide them into the following categories:

- QPSK over Rayleigh flat fading channel without coding and with 1/3 repetition code.
- QPSK over Frequency selective Fading channel without coding and with 1/3 repetition code.
- 16-QAM over Rayleigh flat fading channel without coding and with 1/3 repetition code.
- 16-QAM over Frequency selective Fading channel without coding and with 1/3 repetition code.

Shown below the code of the "`Interleaver`" function used:

```
function Interleaved_Data = Interleaver(binary_data, interleaver_size)
numInterleaverBits = prod(interleaver_size);
% Perform interleaving
numBlocks = length(binary_data) / numInterleaverBits;
Interleaved_Data = zeros(length(binary_data), 1);
for i = 1:numBlocks
    startIndex = (i-1)*numInterleaverBits + 1;
    endIndex = i*numInterleaverBits;
    dataBlock = binary_data(startIndex:endIndex);
    Inter = reshape(dataBlock, interleaver_size);
    Interleaved_Block = reshape(Inter.', 1, []);
    Interleaved_Data(startIndex:endIndex) = Interleaved_Block;
end
end
```

And for the QPSK and the 16-QAM the same mapping and de-mapping functions used in the previous problem will be used in this problem.

a) QPSK over Rayleigh flat fading channel without coding and with 1/3 repetition code

```
%.................................................%
%                                                 %
%  No Coding QPSK over Rayleigh flat fading Channel  %
%                                                 %
%.................................................%

%% ************************* Parameters ************************* %%
numBits = 1048576;                 % Number of bits
Eb = 1;                           % Bit energy
bits_per_symbol = 2;              % bits per symbol for QPSK
interleaver_size = [16, 16];      % Interleaver size
IFFT_size = 128;                  % Number of points for FFT and IFFT
cp_length = IFFT_size/4;          % Cyclic prefix length 25% FFT length
EbN0_dB = -5:1:20;                % Eb/N0 range in dB (SNR)
BER = zeros(size(EbN0_dB,2),1);   % Array to store BER at different SNR values
% Generate random binary data
binary_data = randi([0 1], numBits, 1);
%% ************************* TRANSMITTER ************************* %%
% Initalize the ofdm_tx_signal
ofdm_tx_signal = [];
for j = 1 : 256 : numBits
    % Interleaver
    Interleaved_Data = Interleaver(binary_data(j:j+255),interleaver_size);
    % Mapper
    QPSK = QPSK_Gray_Mapper(Interleaved_Data)';
    % Perform the 128-point IFFT
    IFFT = ifft(QPSK, IFFT_size);
    % Add cyclic extension
    data_with_cp = [IFFT(end - cp_length + 1:end), IFFT];
    ofdm_tx_signal = [ofdm_tx_signal data_with_cp];
end

for i = 1 : size(EbN0_dB,2)
    %% *******  Rayleigh flat fading Channel ********* %%
    binary_data_received = [];
    % Generate the real part of the channel impulse response
    h_r = 0 + sqrt(1/2) * randn(1,length(ofdm_tx_signal));
    % Generate the imaginary part of the channel impulse response
    h_i = 0 + sqrt(1/2) * randn(1,length(ofdm_tx_signal));
    % Combine the real and imaginary parts to form the complex channel impulse
response
    h = h_r + 1j * h_i;
    SNR_linear = 10^(EbN0_dB(i)/10);
    N0 = Eb/SNR_linear;    % noise power
    variance = N0/2; % variance of the Gaussian noise
    % Generate the real part of the noise vector
    n_c = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal));
    % Generate the imaginary part of the noise vector
    n_s = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal));
    % Combine the real and imaginary parts to form the complex noise vector
    n = n_c + 1j * n_s;
    ofdm_channel_signal = h.*ofdm_tx_signal + n;
    %% ************************  RECEIVER ************************ %%
    % Compensate for the channel gain at the receiver
    ofdm_rx_signal = ofdm_channel_signal./h;
    % Remove Cyclic extension
    for j = 1:numel(data_with_cp):numel(ofdm_channel_signal)
        data_without_cp = ofdm_rx_signal(cp_length +j : j+cp_length +IFFT_size-1);
        % Perform the 128-point FFT
        FFT = fft(data_without_cp , IFFT_size);
        % De-mapper
        QPSK_Recievedsignal = QPSK_Gray_Demapper(FFT');
        % De-interleaver
        QPSK_Recievedsignal = QPSK_Recievedsignal';
        DeInterleaved_Data1 = reshape(QPSK_Recievedsignal,interleaver_size);
        DeInterleaved_Data = reshape (DeInterleaved_Data1.',1,[]);
        binary_data_received = [binary_data_received DeInterleaved_Data];
    end
    % Calculating BER
    num_error_bits = sum(binary_data ~= binary_data_received');
    BER(i) = num_error_bits/numBits;
end
% Plotting BER
figure('Name' , 'BER of QPSK over Rayleigh flat fading channel');
semilogy(EbN0_dB , BER, 'r', 'linewidth', 1);
hold on;
title('OFDM System: BER of QPSK over Rayleigh flat fading channel');
xlabel('EB/No(dB)');  ylabel('BER');  grid on;
% Limit y-axis to 10^-3
ylim([1e-3, 1]);

%.................................................................%
%                                                                 %
% Rate 1/3 repetition code QPSK over Rayleigh flat fading Channel %
%                                                                 %
%.................................................................%

%% ************************ Parameters ************************ %%
numBits_coded = 5000*252;               % Number of bits
BER_coded = zeros(size(EbN0_dB,2),1);  % Array to store BER at different SNR values
% Generate random binary data
binary_data_2 = randi([0 1], numBits_coded, 1);
% Repeatition Coding
repetition_factor = 3;
binary_data_coded = repelem (binary_data_2, repetition_factor);

%% ************************ TRANSMITTER ************************ %%
% Initalize the ofdm_tx_signal
ofdm_tx_signal_coded = []
for j = 1 : 252 : numBits_coded*repetition_factor
    % padding the coded data with zeros
    pad = zeros(4,1);
    coded_data = [binary_data_coded(j:j+251); pad];
    % Interleaver
    Interleaved_Data_coded = Interleaver(coded_data, interleaver_size);
    % Mapper (QPSK)
    QPSK_coded = QPSK_Gray_Mapper(Interleaved_Data_coded)';
    % Perform the 128-point IFFT
    IFFT_coded = ifft(QPSK_coded, IFFT_size);
    % Add cyclic prefix
    coded_data_with_cp = [IFFT_coded(end - cp_length + 1:end), IFFT_coded];
    ofdm_tx_signal_coded = [ofdm_tx_signal_coded coded_data_with_cp];
end

for i = 1 : size(EbN0_dB,2)
    %% *********  Rayleigh flat fading Channel ******* %%
    binary_data_coded_received = [];
    decoded_data = [];
    % Generate the real part of the channel impulse response
    h_r = 0 + sqrt(1/2) * randn(1,length(ofdm_tx_signal_coded));
    % Generate the imaginary part of the channel impulse response
    h_i = 0 + sqrt(1/2) * randn(1,length(ofdm_tx_signal_coded));
    % Combine the real and imaginary parts to form the complex channel impulse
response
    h = h_r + 1j * h_i;
    SNR_linear = 10^(EbN0_dB(i)/10);
    N0 = Eb/SNR_linear;    % noise power
    variance = N0/2; % variance of the Gaussian noise
    % Generate the real part of the noise vector
    n_c = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal_coded));
    % Generate the imaginary part of the noise vector
    n_s = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal_coded));
    % Combine the real and imaginary parts to form the complex noise vector
    n = n_c + 1j * n_s;
    ofdm_channel_signal_coded = h.*ofdm_tx_signal_coded + n;

    %% ************************  RECEIVER ************************ %%
    % Compensate for the channel gain at the receiver
    ofdm_rx_signal_coded = ofdm_channel_signal_coded./h;

    % Remove Cyclic extension
    for j = 1:numel(coded_data_with_cp):numel(ofdm_channel_signal_coded)
        coded_data_without_cp = ofdm_rx_signal_coded(cp_length +j : j+cp_length
+IFFT_size-1);
        % Perform the 128-point FFT
        FFT_coded = fft(coded_data_without_cp , IFFT_size);
        % De-mapper
        QPSK_Recievedsignal_coded = QPSK_Gray_Demapper(FFT_coded');
        % De-interleaver
        QPSK_Recievedsignal_coded = QPSK_Recievedsignal_coded';
        DeInterleaved_Data1_coded =
reshape(QPSK_Recievedsignal_coded,interleaver_size);
        DeInterleaved_Data_coded = reshape (DeInterleaved_Data1_coded.',1,[]);
        binary_data_coded_received = [binary_data_coded_received
DeInterleaved_Data_coded];
        % Remove padding
        Decoded_Data1 = reshape(DeInterleaved_Data_coded(1 :
numel(DeInterleaved_Data_coded)-4), repetition_factor, []);
        % Hard decision decoding
        Decoded_Data2 = sum(Decoded_Data1, 1) >= 2;
        decoded_data = [decoded_data Decoded_Data2];
end
```

```
    % Calculating BER
    num_error_bits_coded = sum(binary_data_2 ~= decoded_data');
    BER_coded(i) = num_error_bits_coded/numBits_coded;
end
% Plotting BER
semilogy(EbN0_dB , BER_coded, 'b', 'linewidth', 1);
legend('QPSK no coding', 'QPSK with rate 1/3 repetition code');
```

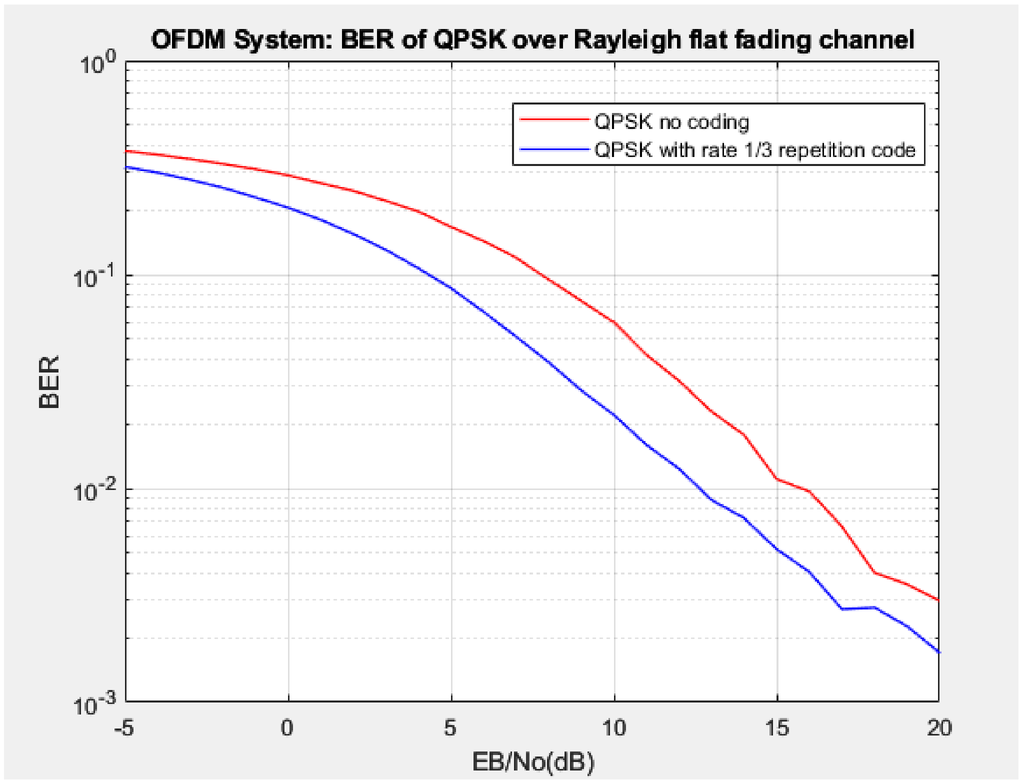


***Figure 8**: BER of QPSK using OFDM over Rayleigh flat fading channel with and without repetition code.*

Comment: When calculating the variance of the noise we divide by $\frac{1}{\sqrt{128}}$ (128 is the size of the used "`fft`" and "`ifft`") this is done to ensure that the noise power is correctly normalized per subcarrier in the OFDM system, maintaining the overall noise power as expected for the given SNR.

b) QPSK over Frequency selective Fading channel without coding and with 1/3 repetition code

```
%.........................................................%
%                                                         %
%  No Coding QPSK over Frequency selective Fading channel  %
%                                                         %
%.........................................................%
%% ************************* Parameters ************************* %%
numBits = 262144;               % Number of bits
Eb = 1;                          % Bit energy
bits_per_symbol = 2;             % bits per symbol for QPSK
interleaver_size = [16, 16];     % Interleaver size
IFFT_size = 128;                 % Number of points for FFT and IFFT
cp_length = IFFT_size/4;         % Cyclic prefix length 25% FFT length
EbN0_dB = -5:1:20;               % Eb/N0 range in dB (SNR)
BER = zeros(size(EbN0_dB,2),1);  % Array to store BER at different SNR values
% Generate random binary data
binary_data = randi([0 1], numBits, 1);
%% ************************* TRANSMITTER ************************* %%
% Initalize the ofdm_tx_signal
ofdm_tx_signal = [];
for j = 1 : 256 : numBits
    % Interleaver
    Interleaved_Data = Interleaver(binary_data(j:j+255),interleaver_size);
    % Mapper
    QPSK = QPSK_Gray_Mapper(Interleaved_Data)';
    % Perform the 128-point IFFT
    IFFT = ifft(QPSK, IFFT_size);
    % Add cyclic extension
    data_with_cp = [IFFT(end - cp_length + 1:end), IFFT];
    ofdm_tx_signal = [ofdm_tx_signal data_with_cp];
end
% Number of subchannels
num_subchannels = 128;
% Length of each subchannel
subchannel_length = length(ofdm_tx_signal) / num_subchannels;
% Initialize a cell array to store subchannels
ofdm_tx_subchannels = cell(1, num_subchannels);
ofdm_channel_subchannels = cell(1, num_subchannels);
% Split the OFDM transmitted signal into subchannels using a loop
for k = 1:num_subchannels
    start_idx = (k-1) * subchannel_length + 1;
    end_idx = k * subchannel_length;
    ofdm_tx_subchannels{k} = ofdm_tx_signal(start_idx:end_idx);
end
for i = 1 : size(EbN0_dB,2)
    %% ********* Frequency selective fading Channel ********* %%
    binary_data_received = [];
    % Generate the real and imaginary parts of the channel impulse response
    h_real = sqrt(1/2) * randn(num_subchannels, subchannel_length);
    h_imag = sqrt(1/2) * randn(num_subchannels, subchannel_length);
    SNR_linear = 10^(EbN0_dB(i)/10);
    N0 = Eb/SNR_linear;    % noise power
    variance = N0/2; % variance of the Gaussian noise
```

```
% Generate the real part of the noise vector
    n_c = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal));
    % Generate the imaginary part of the noise vector
    n_s = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal));
    % Combine the real and imaginary parts to form the complex noise vector
    n = n_c + 1j * n_s;
    % Split the OFDM transmitted signal into subchannels and apply channel effect
with noise
    for k = 1:num_subchannels
    start_idx = (k-1) * subchannel_length + 1;
    end_idx = k * subchannel_length;
    % Split the transmitted signal into subchannels
    ofdm_tx_subchannels{k} = ofdm_tx_signal(start_idx:end_idx);
    % Form the complex channel impulse response for each subchannel
    h = h_real(k, :) + 1j * h_imag(k, :);
    % Apply the channel effect and add noise to each subchannel
    ofdm_channel_subchannels{k} = ifft(h .* fft(ofdm_tx_subchannels{k})) +
n(start_idx:end_idx);
    end
    % Combine the received signals from all subchannels
    ofdm_channel_signal_combined = [ofdm_channel_subchannels{:}];

    %% ************************  RECEIVER ************************ %%
    % Initialize cell arrays to store the received subchannels
    ofdm_rx_subchannels = cell(1, num_subchannels);
    %Loop through each subchannel to compensate for the channel gain
    for k = 1:num_subchannels
        start_idx = (k-1) * subchannel_length + 1;
        end_idx = k * subchannel_length;
        % Split the combined received signal into subchannels
        ofdm_channel_subchannel = ofdm_channel_signal_combined(start_idx:end_idx);
        % Form the complex channel impulse response for each subchannel
        h = h_real(k, :) + 1j * h_imag(k, :);
        % Compensate for the channel gain at the receiver
        ofdm_rx_subchannels{k} = ifft(fft(ofdm_channel_subchannel) ./ h);
    end
    % Combine the compensated signals from all subchannels
    ofdm_rx_signal = [ofdm_rx_subchannels{:}];
    % Remove Cyclic extension
    for j = 1:numel(data_with_cp):numel(ofdm_channel_signal_combined)
        data_without_cp = ofdm_rx_signal(cp_length +j : j+cp_length +IFFT_size-1);
        % Perform the 128-point FFT
        FFT = fft(data_without_cp , IFFT_size);
        % De-mapper
        QPSK_Recievedsignal = QPSK_Gray_Demapper(FFT');
        % De-interleaver
        QPSK_Recievedsignal = QPSK_Recievedsignal';
        DeInterleaved_Data1 = reshape(QPSK_Recievedsignal,interleaver_size);
        DeInterleaved_Data = reshape (DeInterleaved_Data1.',1,[]);
        binary_data_received = [binary_data_received DeInterleaved_Data];
    end
    % Calculating BER
    num_error_bits = sum(binary_data ~= binary_data_received');
    BER(i) = num_error_bits/numBits;
end
```

```
% Plotting BER
figure('Name' , 'BER of QPSK over frequency selective fading channel');
semilogy(EbN0_dB , BER, 'r', 'linewidth', 1);
hold on;
title('OFDM System: BER of QPSK over frequency selective fading channel');
xlabel('EB/No(dB)');  ylabel('BER');  grid on;
% Limit y-axis to 10^-3
ylim([1e-3, 1]);

%.......................................................................%
%                                                                       %
% Rate 1/3 repetition code QPSK over frequency selective fading channel %
%                                                                       %
%.......................................................................%

%% ************************* Parameters ************************* %%
numBits_coded = 1000*252;              % Number of bits
BER_coded = zeros(size(EbN0_dB,2),1);  % Array to store BER at different SNR values
% Generate random binary data
binary_data_2 = randi([0 1], numBits_coded, 1);
% Repeatition Coding
repetition_factor = 3;
binary_data_coded = repelem (binary_data_2, repetition_factor);

%% ************************* TRANSMITTER ************************* %%
% Initalize the ofdm_tx_signal
ofdm_tx_signal_coded = []
for j = 1 : 252 : numBits_coded*repetition_factor
    % padding the coded data with zeros
    pad = zeros(4,1);
    coded_data = [binary_data_coded(j:j+251); pad];
    % Interleaver
    Interleaved_Data_coded = Interleaver(coded_data, interleaver_size);
    % Mapper (QPSK)
    QPSK_coded = QPSK_Gray_Mapper(Interleaved_Data_coded)';
    % Perform the 128-point IFFT
    IFFT_coded = ifft(QPSK_coded, IFFT_size);
    % Add cyclic prefix
    coded_data_with_cp = [IFFT_coded(end - cp_length + 1:end), IFFT_coded];
    ofdm_tx_signal_coded = [ofdm_tx_signal_coded coded_data_with_cp];
end
% Number of subchannels
num_subchannels = 128;
% Length of each subchannel
subchannel_length = length(ofdm_tx_signal_coded) / num_subchannels;
% Initialize a cell array to store subchannels
ofdm_tx_subchannels = cell(1, num_subchannels);
ofdm_channel_subchannels = cell(1, num_subchannels);
% Split the OFDM transmitted signal into subchannels using a loop
for k = 1:num_subchannels
    start_idx = (k-1) * subchannel_length + 1;
    end_idx = k * subchannel_length;
    ofdm_tx_subchannels{k} = ofdm_tx_signal_coded(start_idx:end_idx);
end
```

```
for i = 1 : size(EbN0_dB,2)
    %% ********* Frequency selective fading Channel ********* %%
    binary_data_coded_received = [];
    decoded_data = [];
    % Generate the real and imaginary parts of the channel impulse response
    h_real = sqrt(1/2) * randn(num_subchannels, subchannel_length);
    h_imag = sqrt(1/2) * randn(num_subchannels, subchannel_length);
    SNR_linear = 10^(EbN0_dB(i)/10);
    N0 = Eb/SNR_linear;    % noise power
    variance = N0/2; % variance of the Gaussian noise
    % Generate the real part of the noise vector
    n_c = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal_coded));
    % Generate the imaginary part of the noise vector
    n_s = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal_coded));
    % Combine the real and imaginary parts to form the complex noise vector
    n = n_c + 1j * n_s;
    % Split the OFDM transmitted signal into subchannels and apply channel effect
with noise
    for k = 1:num_subchannels
    start_idx = (k-1) * subchannel_length + 1;
    end_idx = k * subchannel_length;
    % Split the transmitted signal into subchannels
    ofdm_tx_subchannels{k} = ofdm_tx_signal_coded(start_idx:end_idx);
    % Form the complex channel impulse response for each subchannel
    h = h_real(k, :) + 1j * h_imag(k, :);
    % Apply the channel effect and add noise to each subchannel
    ofdm_channel_subchannels{k} = ifft(h .* fft(ofdm_tx_subchannels{k})) +
n(start_idx:end_idx);
    end
    % Combine the received signals from all subchannels
    ofdm_channel_signal_combined_coded = [ofdm_channel_subchannels{:}];

    %% *************************  RECEIVER ************************* %%
    % Initialize cell arrays to store the received subchannels
    ofdm_rx_subchannels = cell(1, num_subchannels);
    % Loop through each subchannel to compensate for the channel gain
    for k = 1:num_subchannels
        start_idx = (k-1) * subchannel_length + 1;
        end_idx = k * subchannel_length;
        % Split the combined received signal into subchannels
        ofdm_channel_subchannel =
ofdm_channel_signal_combined_coded(start_idx:end_idx);
        % Form the complex channel impulse response for each subchannel
        h = h_real(k, :) + 1j * h_imag(k, :);
        % Compensate for the channel gain at the receiver
        ofdm_rx_subchannels{k} = ifft(fft(ofdm_channel_subchannel) ./ h);
    end
    % Combine the compensated signals from all subchannels
    ofdm_rx_signal_coded = [ofdm_rx_subchannels{:}];

    % Remove Cyclic extension
    for j = 1:numel(coded_data_with_cp):numel(ofdm_channel_signal_combined_coded)
        coded_data_without_cp = ofdm_rx_signal_coded(cp_length +j : j+cp_length
+IFFT_size-1);
```

```
% Perform the 128-point FFT
        FFT_coded = fft(coded_data_without_cp , IFFT_size);
        % De-mapper
        QPSK_Recievedsignal_coded = QPSK_Gray_Demapper(FFT_coded');
        % De-interleaver
        QPSK_Recievedsignal_coded = QPSK_Recievedsignal_coded';
        DeInterleaved_Data1_coded =
reshape(QPSK_Recievedsignal_coded,interleaver_size);
        DeInterleaved_Data_coded = reshape (DeInterleaved_Data1_coded.',1,[]);
        binary_data_coded_received = [binary_data_coded_received
DeInterleaved_Data_coded];
        % Remove padding
        Decoded_Data1 = reshape(DeInterleaved_Data_coded(1 :
numel(DeInterleaved_Data_coded)-4), repetition_factor, []);
        % Hard decision decoding
        Decoded_Data2 = sum(Decoded_Data1, 1) >= 2;
        decoded_data = [decoded_data Decoded_Data2];
    end
    % Calculating BER
    num_error_bits_coded = sum(binary_data_2 ~= decoded_data');
    BER_coded(i) = num_error_bits_coded/numBits_coded;
end
% Plotting BER
semilogy(EbN0_dB , BER_coded, 'b', 'linewidth', 1);
legend('QPSK no coding', 'QPSK with rate 1/3 repetition code');
```

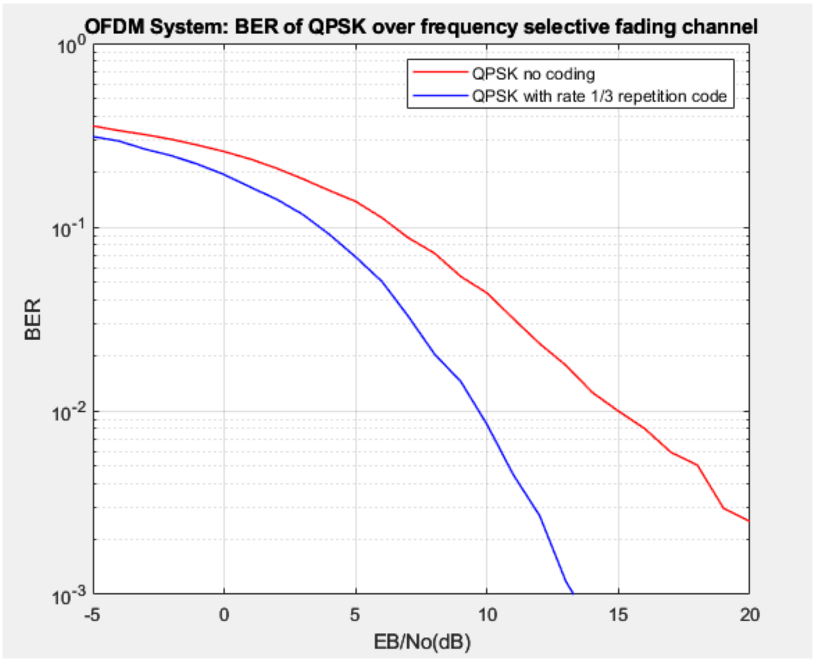


Figure 9: *BER of QPSK using OFDM over frequency selective fading channel with and without repetition code.*

Comments: when simulating the frequency selective fading channel, it is simulated in the frequency domain hence and we have the number of channels equal 128, we use “`fft`” command to go to frequency domain multiply by h and then “`ifft`” to return to the time domain and add the noise.

c) 16-QAM over Rayleigh flat fading channel without coding and with 1/3 repetition code

```
%.....................................................%
%                                                     %
%  No Coding 16-QAM over Rayleigh flat fading Channel %
%                                                     %
%.....................................................%

%% ************************* Parameters ************************* %%
numBits = 1048576;                % Number of bits
Eb = 1;                           % Bit energy
bits_per_symbol = 4;              % bits per symbol for 16 QAM
interleaver_size = [32, 16];      % Interleaver size
IFFT_size = 128;                  % Number of points for FFT and IFFT
cp_length = IFFT_size/4;          % Cyclic prefix length 25% FFT length
EbN0_dB = -5:1:20;                % Eb/N0 range in dB (SNR)
BER = zeros(size(EbN0_dB,2),1);  % Array to store BER at different SNR values
% Generate random binary data
binary_data = randi([0 1], numBits, 1);

%% ************************* TRANSMITTER ************************* %%
% Initalize the ofdm_tx_signal
ofdm_tx_signal = [];
for j = 1 : 512 : numBits
    % Interleaver
    Interleaved_Data = Interleaver(binary_data(j:j+511),interleaver_size);
    % Mapper
    QAM16 = QAM16_Mapper(Interleaved_Data).';
    % Perform the 128-point IFFT
    IFFT = ifft(QAM16, IFFT_size);
    % Add cyclic extension
    data_with_cp = [IFFT(end - cp_length + 1:end), IFFT];
    ofdm_tx_signal = [ofdm_tx_signal data_with_cp];
end
for i = 1 : size(EbN0_dB,2)
    %% ***********  Rayleigh flat fading Channel ***********  %%
    binary_data_received = [];
    % Generate the real part of the channel impulse response
    h_r = 0 + sqrt(1/2) * randn(1,length(ofdm_tx_signal));
    % Generate the imaginary part of the channel impulse response
    h_i = 0 + sqrt(1/2) * randn(1,length(ofdm_tx_signal));
    % Combine the real and imaginary parts to form the complex channel impulse
response
    h = h_r + 1j * h_i;
    SNR_linear = 10^(EbN0_dB(i)/10);
    N0 = Eb/SNR_linear;    % noise power
    variance = 2.5*N0/2; % variance of the noise
    % Generate the real part of the noise vector
    n_c = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal));
    % Generate the imaginary part of the noise vector
    n_s = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal));
    % Combine the real and imaginary parts to form the complex noise vector
    n = n_c + 1j * n_s;
    ofdm_channel_signal = h.*ofdm_tx_signal + n;
    %% ************************  RECEIVER ************************ %%
    % Compensate for the channel gain at the receiver
    ofdm_rx_signal = ofdm_channel_signal./h;
    % Remove Cyclic extension
    for j = 1:numel(data_with_cp):numel(ofdm_channel_signal)
        data_without_cp = ofdm_rx_signal(cp_length +j : j+cp_length +IFFT_size-1);
        % Perform the 128-point FFT
        FFT = fft(data_without_cp , IFFT_size);
        % De-mapper
        QAM16_Recievedsignal = QAM16_Demapper(FFT.');
        % De-interleaver
        QAM16_Recievedsignal = QAM16_Recievedsignal';
        % Reverse the interleaver size when de-interleaving
        DeInterleaved_Data1 =
reshape(QAM16_Recievedsignal,interleaver_size(2),interleaver_size(1));
        DeInterleaved_Data = reshape (DeInterleaved_Data1.',1,[]);
        binary_data_received = [binary_data_received DeInterleaved_Data];
    end
    binary_data_received = binary_data_received';
    % Calculating BER
    num_error_bits = sum(binary_data ~= binary_data_received);
    BER(i) = num_error_bits/numBits;
end
% Plotting BER
figure('Name' , 'BER of 16-QAM over Rayleigh flat fading channel');
semilogy(EbN0_dB , BER, 'r', 'linewidth', 1);
hold on;
title('OFDM System: BER of 16-QAM over Rayleigh flat fading channel');
xlabel('EB/No(dB)');  ylabel('BER');  grid on;
% Limit y-axis to 10^-3
ylim([1e-3, 1]);

%..................................................................%
%                                                                  %
% Rate 1/3 repetition code 16-QAM over Rayleigh flat fading Channel %
%                                                                  %
%..................................................................%

%% ************************ Parameters ************************ %%
numBits_coded = 1000*252;             % Number of bits
BER_coded = zeros(size(EbN0_dB,2),1);  % Array to store BER at different SNR values
% Generate random binary data
binary_data_2 = randi([0 1], numBits_coded, 1);
% Repeatition Coding
repetition_factor = 3;
binary_data_coded = repelem (binary_data_2, repetition_factor);

%% ************************ TRANSMITTER ************************ %%
% Initalize the ofdm_tx_signal
ofdm_tx_signal_coded = [];
for j = 1 : 504 : numBits_coded*repetition_factor
    % padding the coded data with zeros
    pad = zeros(8,1);
    coded_data = [binary_data_coded(j:j+503); pad];
    % Interleaver
    Interleaved_Data_coded = Interleaver(coded_data, interleaver_size);
    % Mapper (QPSK)
    QAM16_coded = QAM16_Mapper(Interleaved_Data_coded).';
    % Perform the 128-point IFFT
    IFFT_coded = ifft(QAM16_coded, IFFT_size);
    % Add cyclic prefix
    coded_data_with_cp = [IFFT_coded(end - cp_length + 1:end), IFFT_coded];
    ofdm_tx_signal_coded = [ofdm_tx_signal_coded coded_data_with_cp];
end

for i = 1 : size(EbN0_dB,2)
    %% *********  Rayleigh flat fading Channel *********  %%
    binary_data_coded_received = [];
    decoded_data = [];
    % Generate the real part of the channel impulse response
    h_r = 0 + sqrt(1/2) * randn(1,length(ofdm_tx_signal_coded));
    % Generate the imaginary part of the channel impulse response
    h_i = 0 + sqrt(1/2) * randn(1,length(ofdm_tx_signal_coded));
    % Combine the real and imaginary parts to form the complex channel impulse
response
    h = h_r + 1j * h_i;
    SNR_linear = 10^(EbN0_dB(i)/10);
    N0 = Eb/SNR_linear;    % noise power
    variance = 2.5*N0/2; % variance of the noise
    % Generate the real part of the noise vector
    n_c = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal_coded));
    % Generate the imaginary part of the noise vector
    n_s = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal_coded));
    % Combine the real and imaginary parts to form the complex noise vector
    n = n_c + 1j * n_s;
    ofdm_channel_signal_coded = h.*ofdm_tx_signal_coded + n;

    %% *************************  RECEIVER ************************* %%
    % Compensate for the channel gain at the receiver
    ofdm_rx_signal_coded = ofdm_channel_signal_coded./h;

    % Remove Cyclic extension
    for j = 1:numel(coded_data_with_cp):numel(ofdm_channel_signal_coded)
        coded_data_without_cp = ofdm_rx_signal_coded(cp_length +j : j+cp_length
+IFFT_size-1);
        % Perform the 128-point FFT
        FFT_coded = fft(coded_data_without_cp , IFFT_size);
        % De-mapper
        QAM16_Recievedsignal_coded = QAM16_Demapper(FFT_coded.');
        % De-interleaver
        QAM16_Recievedsignal_coded = QAM16_Recievedsignal_coded';
        % Reverse the interleaver size when de-interleaving
        DeInterleaved_Data1_coded =
reshape(QAM16_Recievedsignal_coded,interleaver_size(2),interleaver_size(1));
        DeInterleaved_Data_coded = reshape (DeInterleaved_Data1_coded.',1,[]);
        binary_data_coded_received = [binary_data_coded_received
DeInterleaved_Data_coded];
```

```
        % Remove padding
        Decoded_Data1 = reshape(DeInterleaved_Data_coded(1 :
numel(DeInterleaved_Data_coded)-8), repetition_factor, []);
        % Hard decision decoding
        Decoded_Data2 = sum(Decoded_Data1, 1) >= 2;
        decoded_data = [decoded_data Decoded_Data2];
    end
    % Calculating BER
    num_error_bits_coded = sum(binary_data_2 ~= decoded_data');
    BER_coded(i) = num_error_bits_coded/numBits_coded;
end
% Plotting BER
semilogy(EbN0_dB , BER_coded, 'b', 'linewidth', 1);
legend('16-QAM no coding', '16-QAM with rate 1/3 repetition code');
```

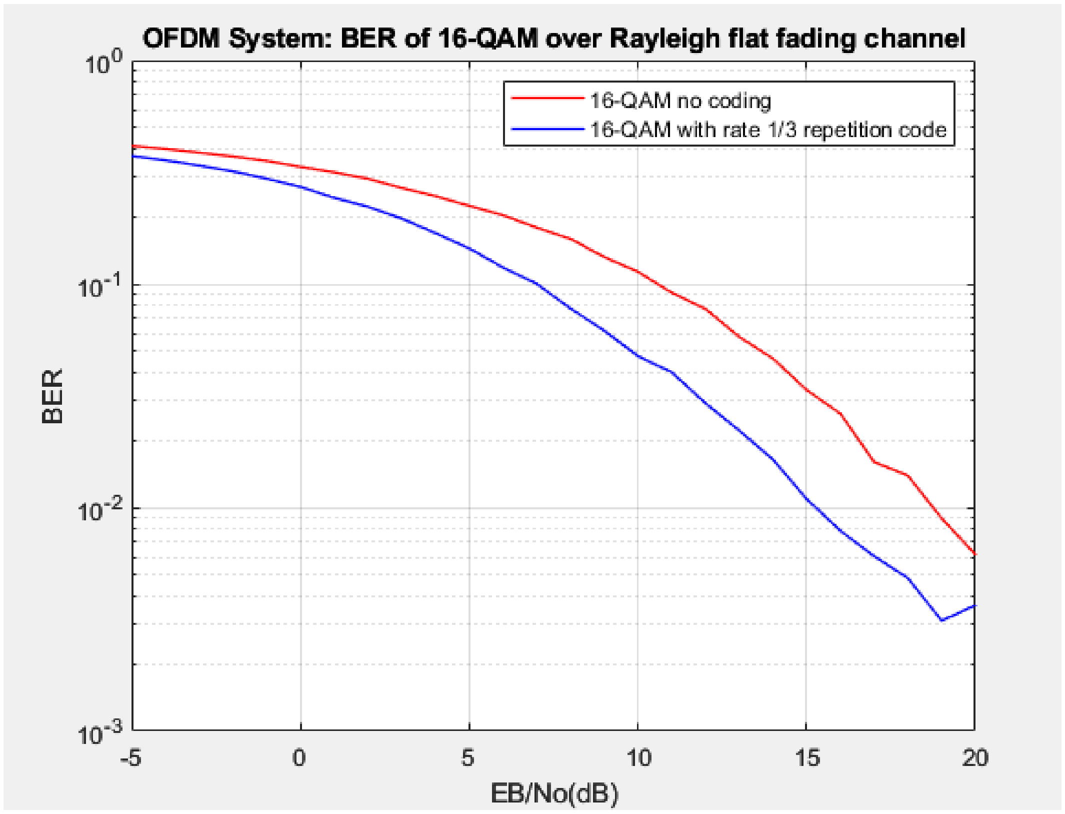


Figure 10: *BER of 16-QAM using OFDM over Rayleigh flat fading channel with and without repetition code.*

d) 16-QAM over Frequency selective Fading channel without coding and with 1/3 repetition code

```
%.........................................................%
%                                                         %
%  No Coding 16-QAM over frequency selective fading Channel %
%                                                         %
%.........................................................%
%% ************************ Parameters ************************ %%
numBits = 262144;              % Number of bits
Eb = 1;                         % Bit energy
bits_per_symbol = 4;            % bits per symbol for 16 QAM
interleaver_size = [32, 16];    % Interleaver size
IFFT_size = 128;                % Number of points for FFT and IFFT
cp_length = IFFT_size/4;        % Cyclic prefix length 25% FFT length
EbN0_dB = -5:1:20;              % Eb/N0 range in dB (SNR)
BER = zeros(size(EbN0_dB,2),1);  % Array to store BER at different SNR values
% Generate random binary data
binary_data = randi([0 1], numBits, 1);
%% ************************ TRANSMITTER ************************ %%
% Initalize the ofdm_tx_signal
ofdm_tx_signal = [];
for j = 1 : 512 : numBits
    % Interleaver
    Interleaved_Data = Interleaver(binary_data(j:j+511),interleaver_size);
    % Mapper
    QAM16 = QAM16_Mapper(Interleaved_Data).';
    % Perform the 128-point IFFT
    IFFT = ifft(QAM16, IFFT_size);
    % Add cyclic extension
    data_with_cp = [IFFT(end - cp_length + 1:end), IFFT];
    ofdm_tx_signal = [ofdm_tx_signal data_with_cp];
end
% Number of subchannels
num_subchannels = 128;
% Length of each subchannel
subchannel_length = length(ofdm_tx_signal) / num_subchannels;
% Initialize a cell array to store subchannels
ofdm_tx_subchannels = cell(1, num_subchannels);
ofdm_channel_subchannels = cell(1, num_subchannels);
% Split the OFDM transmitted signal into subchannels using a loop
for k = 1:num_subchannels
    start_idx = (k-1) * subchannel_length + 1;
    end_idx = k * subchannel_length;
    ofdm_tx_subchannels{k} = ofdm_tx_signal(start_idx:end_idx);
end
for i = 1 : size(EbN0_dB,2)
    %% ********** Frequency selective fading Channel ********** %%
    binary_data_received = [];
    % Generate the real and imaginary parts of the channel impulse response
    h_real = sqrt(1/2) * randn(num_subchannels, subchannel_length);
    h_imag = sqrt(1/2) * randn(num_subchannels, subchannel_length);
    SNR_linear = 10^(EbN0_dB(i)/10);
    N0 = Eb/SNR_linear;    % noise power
    variance = 2.5*N0/2; % variance of the noise
```

```
% Generate the real part of the noise vector
    n_c = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal));
    % Generate the imaginary part of the noise vector
    n_s = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal));
    % Combine the real and imaginary parts to form the complex noise vector
    n = n_c + 1j * n_s;
    % Split the OFDM transmitted signal into subchannels and apply channel effect
with noise
    for k = 1:num_subchannels
    start_idx = (k-1) * subchannel_length + 1;
    end_idx = k * subchannel_length;
    % Split the transmitted signal into subchannels
    ofdm_tx_subchannels{k} = ofdm_tx_signal(start_idx:end_idx);
    % Form the complex channel impulse response for each subchannel
    h = h_real(k, :) + 1j * h_imag(k, :);
    % Apply the channel effect and add noise to each subchannel
    ofdm_channel_subchannels{k} = ifft(h .* fft(ofdm_tx_subchannels{k})) +
n(start_idx:end_idx);
    end
    % Combine the received signals from all subchannels
    ofdm_channel_signal_combined = [ofdm_channel_subchannels{:}];

    %% ************************  RECEIVER ************************ %%
    % Initialize cell arrays to store the received subchannels
    ofdm_rx_subchannels = cell(1, num_subchannels);
    %Loop through each subchannel to compensate for the channel gain
    for k = 1:num_subchannels
        start_idx = (k-1) * subchannel_length + 1;
        end_idx = k * subchannel_length;
        % Split the combined received signal into subchannels
        ofdm_channel_subchannel = ofdm_channel_signal_combined(start_idx:end_idx);
        % Form the complex channel impulse response for each subchannel
        h = h_real(k, :) + 1j * h_imag(k, :);
        % Compensate for the channel gain at the receiver
        ofdm_rx_subchannels{k} = ifft(fft(ofdm_channel_subchannel) ./ h);
    end
    % Combine the compensated signals from all subchannels
    ofdm_rx_signal = [ofdm_rx_subchannels{:}];
    % Remove Cyclic extension
    for j = 1:numel(data_with_cp):numel(ofdm_channel_signal_combined)
        data_without_cp = ofdm_rx_signal(cp_length +j : j+cp_length +IFFT_size-1);
        % Perform the 128-point FFT
        FFT = fft(data_without_cp , IFFT_size);
        % De-mapper
        QAM16_Recievedsignal = QAM16_Demapper(FFT.');
        % De-interleaver
        QAM16_Recievedsignal = QAM16_Recievedsignal';
        % Reverse the interleaver size when de-interleaving
        DeInterleaved_Data1 =
reshape(QAM16_Recievedsignal,interleaver_size(2),interleaver_size(1));
        DeInterleaved_Data = reshape (DeInterleaved_Data1.',1,[]);
        binary_data_received = [binary_data_received DeInterleaved_Data];
    end
    binary_data_received = binary_data_received';
```

```
% Calculating BER
    num_error_bits = sum(binary_data ~= binary_data_received);
    BER(i) = num_error_bits/numBits;
end
% Plotting BER
figure('Name' , 'BER of 16-QAM over frequency selective fading channel');
semilogy(EbN0_dB , BER, 'r', 'linewidth', 1);
hold on;
title('OFDM System: BER of 16-QAM over frequency selective fading channel');
xlabel('EB/No(dB)');  ylabel('BER');  grid on;
% Limit y-axis to 10^-3
ylim([1e-3, 1]);

%.........................................................................%
%                                                                         %
% Rate 1/3 repetition code 16-QAM over frequency selective fading Channel %
%                                                                         %
%.........................................................................%

%% ************************ Parameters ************************ %%
numBits_coded = 1000*252;             % Number of bits
BER_coded = zeros(size(EbN0_dB,2),1);  % Array to store BER at different SNR values
% Generate random binary data
binary_data_2 = randi([0 1], numBits_coded, 1);
% Repeatition Coding
repetition_factor = 3;
binary_data_coded = repelem (binary_data_2, repetition_factor);

%% ************************ TRANSMITTER ************************ %%
% Initalize the ofdm_tx_signal
ofdm_tx_signal_coded = [];
for j = 1 : 504 : numBits_coded*repetition_factor
    % padding the coded data with zeros
    pad = zeros(8,1);
    coded_data = [binary_data_coded(j:j+503); pad];
    % Interleaver
    Interleaved_Data_coded = Interleaver(coded_data, interleaver_size);
    % Mapper (QPSK)
    QAM16_coded = QAM16_Mapper(Interleaved_Data_coded).';
    % Perform the 128-point IFFT
    IFFT_coded = ifft(QAM16_coded, IFFT_size);
    % Add cyclic prefix
    coded_data_with_cp = [IFFT_coded(end - cp_length + 1:end), IFFT_coded];
    ofdm_tx_signal_coded = [ofdm_tx_signal_coded coded_data_with_cp];
end
% Number of subchannels
num_subchannels = 128;
% Length of each subchannel
subchannel_length = length(ofdm_tx_signal_coded) / num_subchannels;
% Initialize a cell array to store subchannels
ofdm_tx_subchannels = cell(1, num_subchannels);
ofdm_channel_subchannels = cell(1, num_subchannels);
```

```
% Split the OFDM transmitted signal into subchannels using a loop
for k = 1:num_subchannels
    start_idx = (k-1) * subchannel_length + 1;
    end_idx = k * subchannel_length;
    ofdm_tx_subchannels{k} = ofdm_tx_signal_coded(start_idx:end_idx);
end
for i = 1 : size(EbN0_dB,2)
     %% ********** Frequency selective fading Channel ********** %%
    binary_data_coded_received = [];
    decoded_data = [];
    % Generate the real and imaginary parts of the channel impulse response
    h_real = sqrt(1/2) * randn(num_subchannels, subchannel_length);
    h_imag = sqrt(1/2) * randn(num_subchannels, subchannel_length);
    SNR_linear = 10^(EbN0_dB(i)/10);
    N0 = Eb/SNR_linear;    % noise power
    variance = 2.5*N0/2; % variance of the noise
    % Generate the real part of the noise vector
    n_c = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal_coded));
    % Generate the imaginary part of the noise vector
    n_s = 0 + (1/sqrt(128))*sqrt(variance) * randn(1,length(ofdm_tx_signal_coded));
    % Combine the real and imaginary parts to form the complex noise vector
    n = n_c + 1j * n_s;
    % Split the OFDM transmitted signal into subchannels and apply channel effect
with noise
    for k = 1:num_subchannels
    start_idx = (k-1) * subchannel_length + 1;
    end_idx = k * subchannel_length;
    % Split the transmitted signal into subchannels
    ofdm_tx_subchannels{k} = ofdm_tx_signal_coded(start_idx:end_idx);
    % Form the complex channel impulse response for each subchannel
    h = h_real(k, :) + 1j * h_imag(k, :);
    % Apply the channel effect and add noise to each subchannel
    ofdm_channel_subchannels{k} = ifft(h .* fft(ofdm_tx_subchannels{k})) +
n(start_idx:end_idx);
    end
    % Combine the received signals from all subchannels
    ofdm_channel_signal_combined_coded = [ofdm_channel_subchannels{:}];

    %% *************************  RECEIVER ************************* %%
    ofdm_rx_subchannels = cell(1, num_subchannels);
    %Loop through each subchannel to compensate for the channel gain
    for k = 1:num_subchannels
        start_idx = (k-1) * subchannel_length + 1;
        end_idx = k * subchannel_length;
        % Split the combined received signal into subchannels
        ofdm_channel_subchannel =
ofdm_channel_signal_combined_coded(start_idx:end_idx);
        % Form the complex channel impulse response for each subchannel
        h = h_real(k, :) + 1j * h_imag(k, :);
        % Compensate for the channel gain at the receiver
        ofdm_rx_subchannels{k} = ifft(fft(ofdm_channel_subchannel) ./ h);
    end
    % Combine the compensated signals from all subchannels
    ofdm_rx_signal_coded = [ofdm_rx_subchannels{:}];
```

```
% Remove Cyclic extension
    for j = 1:numel(coded_data_with_cp):numel(ofdm_channel_signal_combined_coded)
        coded_data_without_cp = ofdm_rx_signal_coded(cp_length +j : j+cp_length
+IFFT_size-1);
        % Perform the 128-point FFT
        FFT_coded = fft(coded_data_without_cp , IFFT_size);
        % De-mapper
        QAM16_Recievedsignal_coded = QAM16_Demapper(FFT_coded.');
        % De-interleaver
        QAM16_Recievedsignal_coded = QAM16_Recievedsignal_coded';
        % Reverse the interleaver size when de-interleaving
        DeInterleaved_Data1_coded =
reshape(QAM16_Recievedsignal_coded,interleaver_size(2),interleaver_size(1));
        DeInterleaved_Data_coded = reshape (DeInterleaved_Data1_coded.',1,[]);
        binary_data_coded_received = [binary_data_coded_received
DeInterleaved_Data_coded];
        % Remove padding
        Decoded_Data1 = reshape(DeInterleaved_Data_coded(1 :
numel(DeInterleaved_Data_coded)-8), repetition_factor, []);
        % Hard decision decoding
        Decoded_Data2 = sum(Decoded_Data1, 1) >= 2;
        decoded_data = [decoded_data Decoded_Data2];
    end
    % Calculating BER
    num_error_bits_coded = sum(binary_data_2 ~= decoded_data');
    BER_coded(i) = num_error_bits_coded/numBits_coded;
end
% Plotting BER
semilogy(EbN0_dB , BER_coded, 'b', 'linewidth', 1);
legend('16-QAM no coding', '16-QAM with rate 1/3 repetition code');
```

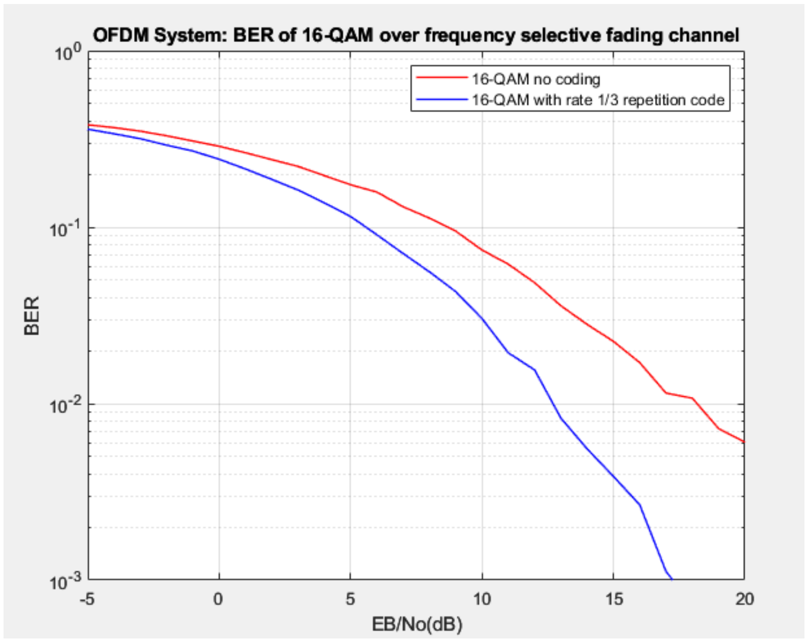


Figure 11: *BER of 16-QAM using OFDM over frequency selective fading channel with and without repetition code.*

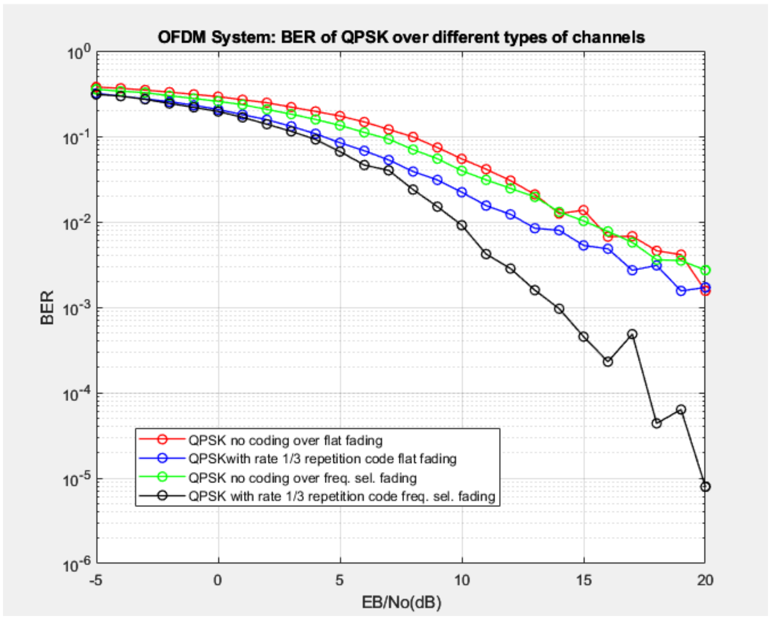


Figure 12: *BER of QPSK over different types of channels.*

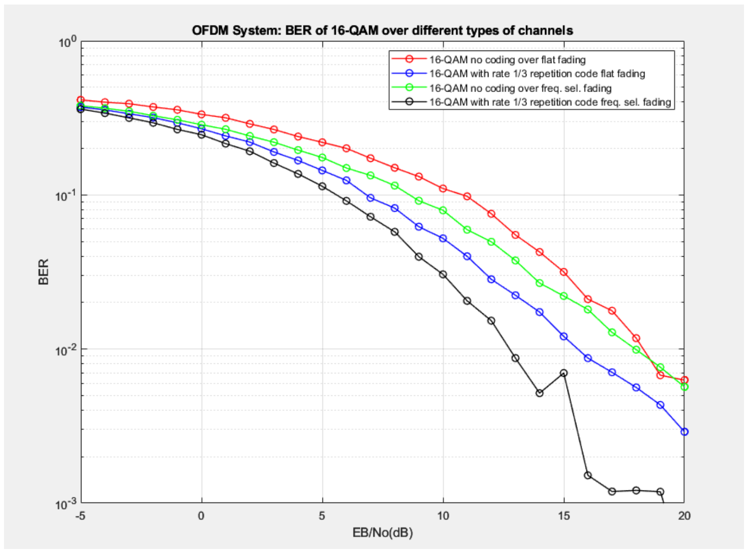


Figure 13: *BER of 16-QAM over different types of channels.*

Figure 12 & **Figure 13** show the effect of different types of channels and coding schemes used with QPSK and 16-QAM. The BER curve for a Rayleigh flat fading channel is worse than that of a frequency-selective channel. This is because flat fading affects all frequency components of the OFDM signal uniformly, leading to deep fades that can significantly degrade the entire signal. However, a frequency-selective channel introduces different gains and delays across various subcarriers. This means that while some subcarriers may experience deep fades, others may still maintain good signal quality. This variation allows for better overall performance when using appropriate techniques like OFDM.