Electronics and Electrical Communications Engineering Department
Faculty of Engineering Cairo University

OFDM Project
Channel Coding

ELC4020 Advanced Communication Systems
4^{th} Year 1^{st} Semester
Academic Year 2025/2026

Prepared by Team 35:

Name	Section	ID
Yousef Khaled Omar Mahmoud	4	9220984
Ahmed Wagdy Mohy Ibrahim	1	9220120
Abdelrahman Essa Elsayed	2	9220469

Instructor: Eng. Mohamed Khaled
Supervised by: Dr. Mohamed Khairy

Contents

List of Figures

Executive Summary

This project investigates the performance of Orthogonal Frequency Division Multiplexing (OFDM) systems under different modulation schemes and channel conditions. The study focuses on three main tasks: computational comparison between DFT and FFT, performance evaluation of uncoded and repetition-coded modulation schemes over Rayleigh fading channels, and a comprehensive BER analysis of OFDM systems operating over flat and frequency-selective fading channels.

BPSK, QPSK, and 16-QAM modulation schemes are considered, with repetition coding employed as a simple forward error correction technique. The impact of channel selectivity and coding on bit error rate (BER) performance is analyzed through extensive MATLAB simulations.

The results demonstrate the computational advantage of FFT over direct DFT implementation, the coding gain achieved by repetition coding in fading channels, and the robustness of OFDM against frequency-selective fading when proper equalization is applied. This project highlights the importance of multicarrier modulation and channel coding in modern wireless communication systems.

Chapter 1

Introduction

Orthogonal Frequency Division Multiplexing (OFDM) is one of the most widely used multicarrier modulation techniques in modern wireless communication systems, including LTE, Wi-Fi, and 5G. OFDM divides the available bandwidth into multiple orthogonal subcarriers, allowing parallel transmission of data symbols and providing strong resistance to inter-symbol interference caused by multipath propagation.

In this project, OFDM is studied from both computational and communication performance perspectives. First, the computational efficiency of the Fast Fourier Transform (FFT) is compared to the direct Discrete Fourier Transform (DFT), highlighting why FFT is essential for practical OFDM implementations. Second, the performance of uncoded and repetition-coded modulation schemes is evaluated over Rayleigh fading channels. Finally, a complete OFDM system is simulated and analyzed under flat and frequency-selective fading conditions using different modulation formats.

The primary performance metric used throughout the project is the bit error rate (BER) as a function of the energy per bit to noise power spectral density ratio (E_b/N_0). All simulations are carried out using MATLAB and are designed to closely follow theoretical expectations.

Chapter 2

Simulation Environment

All experiments in Project 3 are implemented using MATLAB and integrated into a unified Graphical User Interface (GUI). The GUI is designed to simplify execution, visualization, and verification of the implemented OFDM experiments.

The interface allows the user to:

- Select any problem from Question 1 to Question 3
- Choose the modulation scheme (BPSK, QPSK, or 16-QAM)
- Execute simulations automatically
- Display BER curves and numerical results
- Save generated figures directly to the project directory

This unified framework ensures consistency across all experiments and reduces the possibility of configuration errors during simulation.

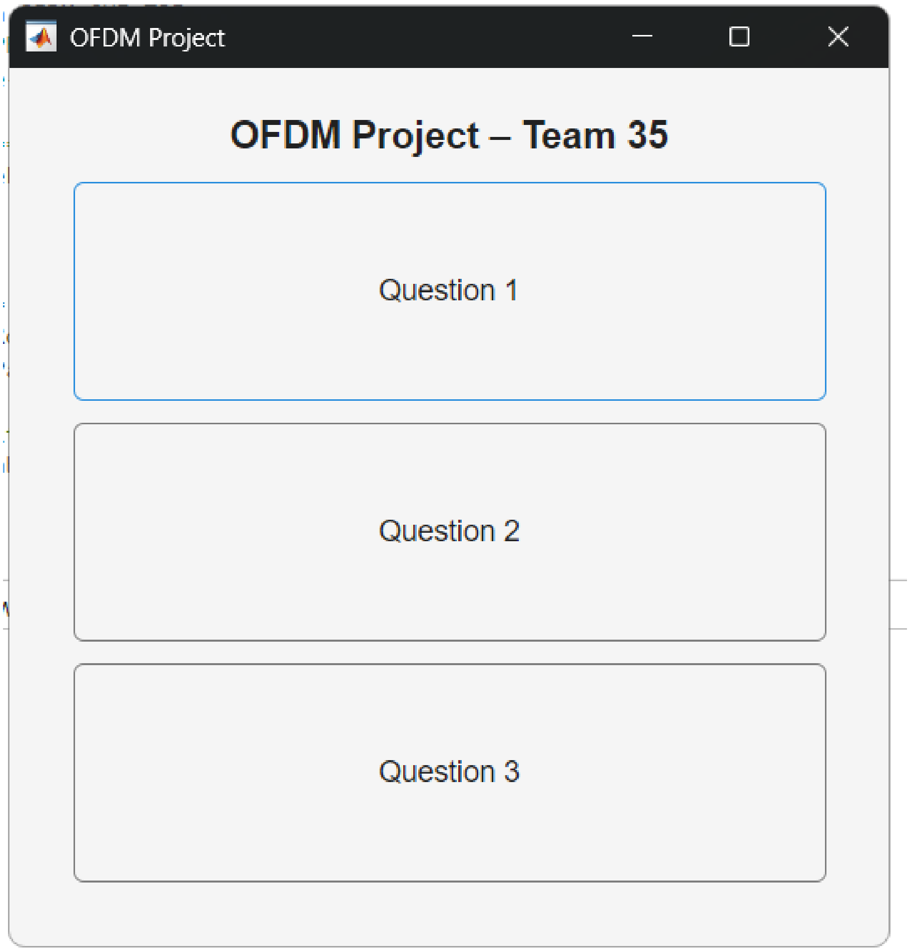

Figure 2.1: Graphical User Interface used to execute OFDM Project experiments.

For the OFDM simulations, a fixed FFT size of $N = 256$ subcarriers is used. Random binary data is generated and mapped to BPSK, QPSK, or 16-QAM symbols depending on the experiment. Repetition coding with repetition factor $R = 5$ is applied when required, followed by block interleaving to mitigate burst errors.

At the receiver, frequency-domain equalization is performed assuming perfect channel knowledge, followed by demodulation and decoding. BER results are averaged over multiple OFDM symbols to ensure statistical reliability.

Chapter 3

Problem 1: Execution Time of DFT and FFT

In this problem, the computational efficiency of the Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT) is investigated. Since OFDM systems rely heavily on frequency-domain processing, understanding the execution time difference between DFT and FFT is essential for practical implementations.

3.1 DFT Implementation

According to the definition of the Discrete Fourier Transform, the frequency-domain representation of a discrete-time signal $x(n)$ of length N is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j\frac{2\pi nk}{N}}, \quad 0 \leq k \leq N-1$$

A MATLAB function is written to directly implement this equation using two nested loops, where each output frequency bin $X(k)$ is computed by summing all time-domain samples. This implementation follows the mathematical definition exactly and has a computational complexity of $\mathcal{O}(N^2)$.

This section answers Question 1(a).

3.2 Execution Time Measurement

To evaluate the execution time, a random test signal $x_i(n)$ of length $L = 4096$ is generated. The execution time is measured using MATLABs `tic` and `toc` commands for the following two cases:

- DFT computed using the custom MATLAB function developed in Part (a)

- FFT computed using MATLABs built-in `fft()` function

Both transforms are applied to the same input signal to ensure a fair comparison. The timing experiment is repeated multiple times to reduce the effect of random fluctuations in execution time.

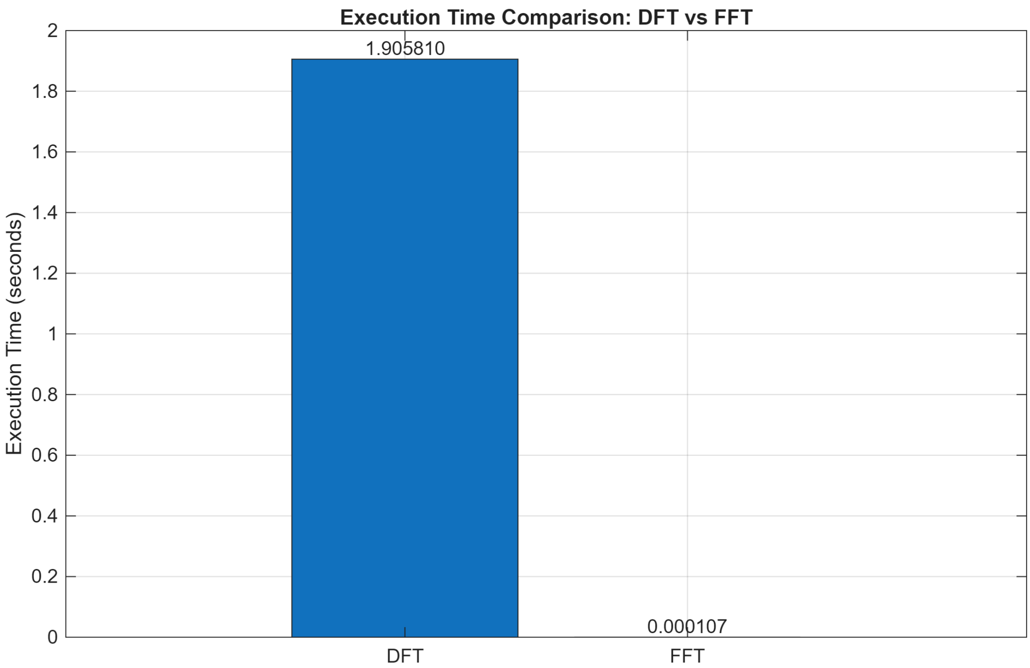

Figure 3.1: Execution time comparison between DFT and FFT for a signal of length $L = 4096$.

This section answers Question 1(b).

3.3 Discussion and Performance Comparison

The obtained results clearly show that the FFT implementation is significantly faster than the direct DFT implementation. This performance improvement is due to the reduced computational complexity of FFT, which is $\mathcal{O}(N \log_2 N)$ compared to $\mathcal{O}(N^2)$ for the DFT.

For large signal lengths, such as $L = 4096$, the execution time of the direct DFT becomes impractically large, whereas the FFT completes in a very short time. This difference highlights why FFT is universally used in practical OFDM systems and other real-time signal processing applications.

Answer to Question 1(c): The FFT offers superior performance with respect to execution time and is therefore the preferred transform for practical implementations.

This discussion fully answers Question 1.

Chapter 4

Problem 2: BER Performance over Rayleigh Flat Fading Channel

In this problem, the bit-error rate (BER) performance of digital modulation schemes over a Rayleigh flat fading channel is investigated. The objective is to study the effect of fading and channel coding on system reliability when using BPSK, QPSK, and 16-QAM modulation schemes.

The Rayleigh flat fading channel models wireless environments where no direct line-of-sight exists between the transmitter and receiver, and the received signal is affected by multipath propagation.

4.1 System Model

The transmitted binary data sequence is denoted by

$$b_k \in \{0, 1\}.$$

For BPSK modulation, the transmitted baseband symbols are given by

$$x_k = \begin{cases} +\sqrt{E_b}, & b_k = 1 \\ -\sqrt{E_b}, & b_k = 0 \end{cases}$$

The Rayleigh flat fading channel impulse response is modeled as

$$h(t) = A_h e^{j\theta_h} \delta(t),$$

which can equivalently be written as

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

where:

- h_r and h_i are independent Gaussian random variables with zero mean and variance $1/2$,
- A_h follows a Rayleigh distribution,
- θ_h is uniformly distributed over $[0, 2\pi)$.

The additive white Gaussian noise (AWGN) is given by

$$n(t) = n_c(t)\cos(\omega_c t) - n_s(t)\sin(\omega_c t),$$

where both $n_c(t)$ and $n_s(t)$ are Gaussian random processes with zero mean and variance $N_0/2$.

The received discrete-time baseband signal is therefore expressed as

$$y_k = (h_r + jh_i)_k x_k + (n_c + jn_s)_k.$$

Perfect channel knowledge is assumed at the receiver.

4.2 Simulation Procedure

The BER simulation over a Rayleigh flat fading channel is performed according to the following steps:

1. Generate a random binary data stream b_k.
2. Map the bits to modulation symbols:
   - BPSK: $\{+\sqrt{E_b}, -\sqrt{E_b}\}$
   - QPSK and 16-QAM using the specified constellations.
3. Generate the complex Rayleigh channel coefficients $(h_r + jh_i)$.
4. Generate complex AWGN samples $(n_c + jn_s)$.
5. Compute the received symbols:

$$y_k = h_k x_k + n_k.$$

6. Perform channel equalization assuming perfect channel knowledge.
7. Apply coherent demodulation and hard decision decoding.

8. Compute the BER for a given E_b/N_0.

9. Repeat the above steps for different SNR values.

The simulation is repeated for both uncoded transmission and repetition-coded transmission.

4.3 Repetition Coding

To improve reliability, a rate 1/5 repetition code is applied. Each information bit is transmitted five times:

$$1 \to 11111, \quad 0 \to 00000.$$

At the receiver, majority voting is used to recover the transmitted bit. Repetition coding introduces redundancy, improving BER performance at the cost of bandwidth efficiency.

4.4 Simulation Results

4.4.1 BPSK over Rayleigh Flat Fading

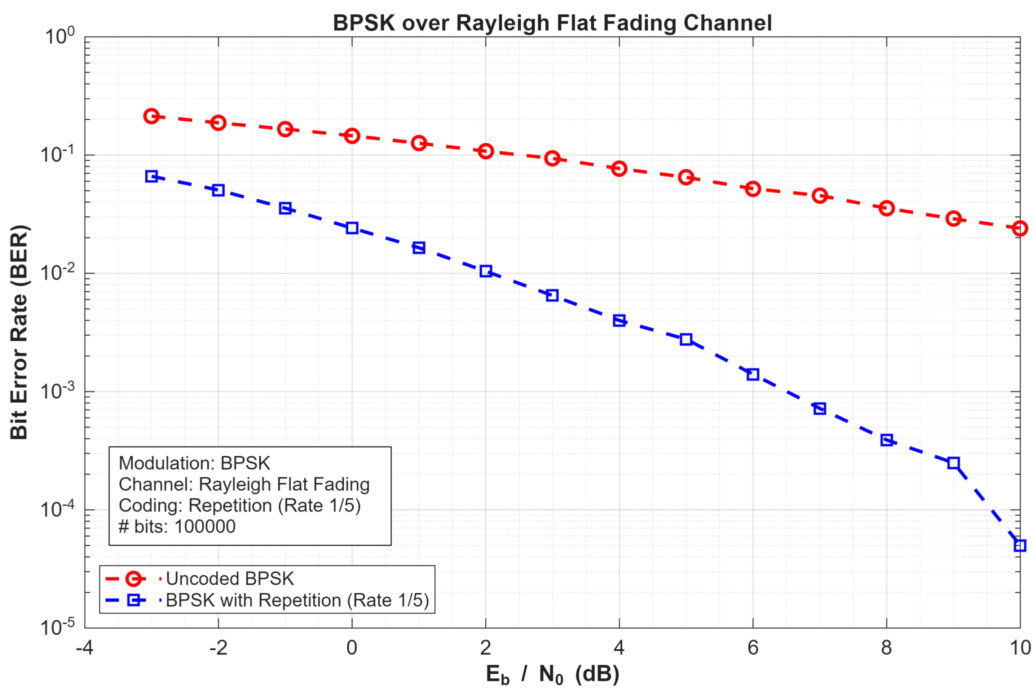

Figure 4.1: BER performance of uncoded and repetition-coded BPSK over Rayleigh flat fading channel.

4.4.2 QPSK over Rayleigh Flat Fading

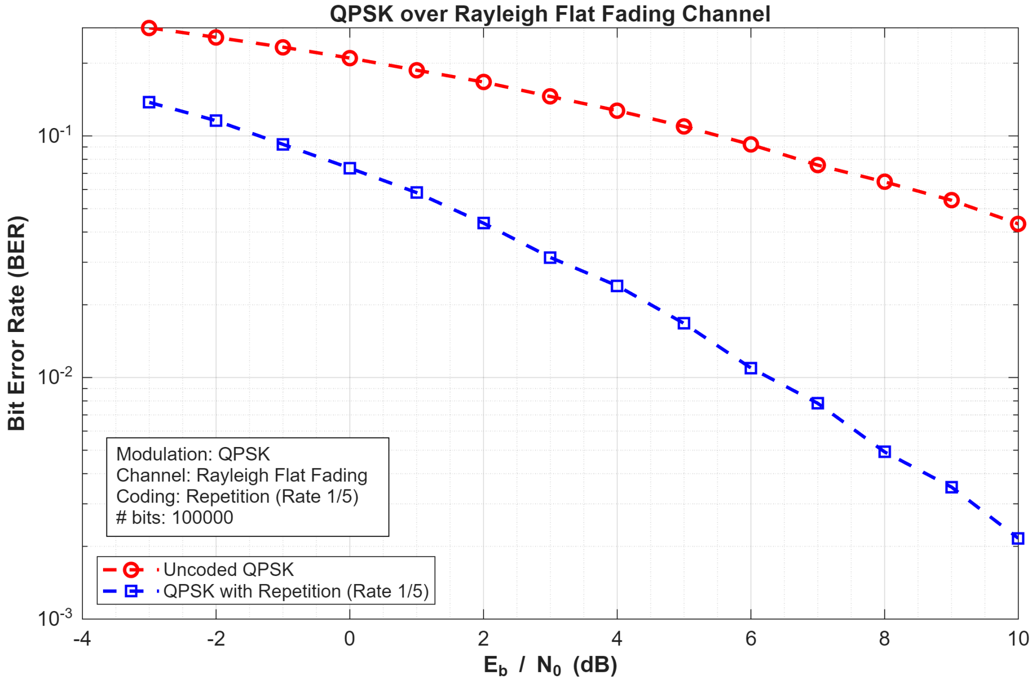

Figure 4.2: BER performance of uncoded and repetition-coded QPSK over Rayleigh flat fading channel.

4.4.3 16-QAM over Rayleigh Flat Fading

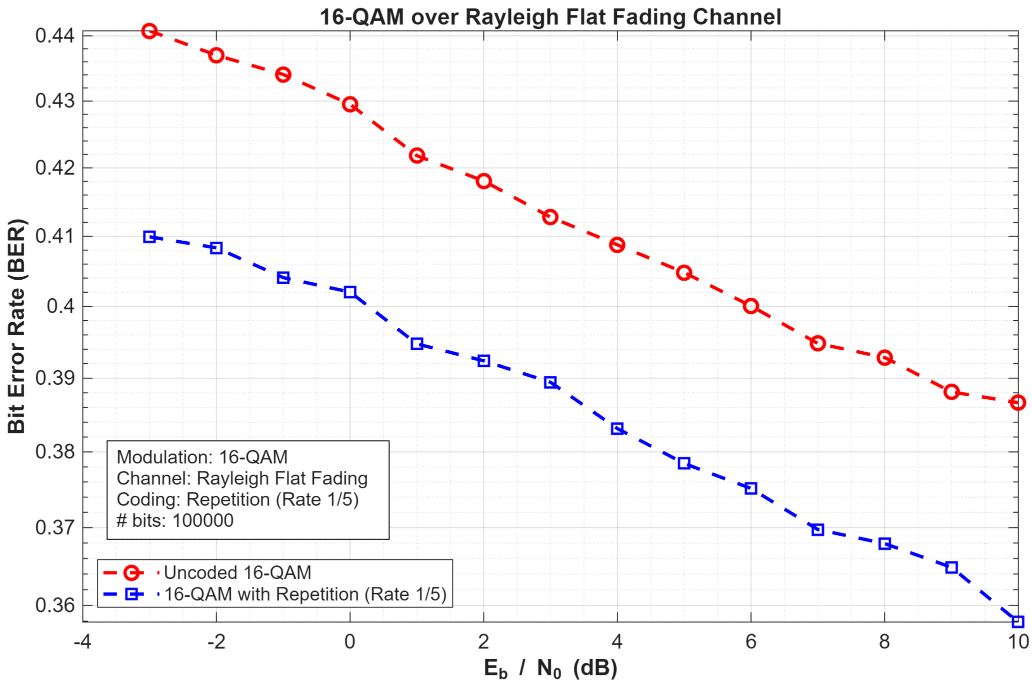

Figure 4.3: BER performance of uncoded and repetition-coded 16-QAM over Rayleigh flat fading channel.

4.4.4 Modulation Comparison

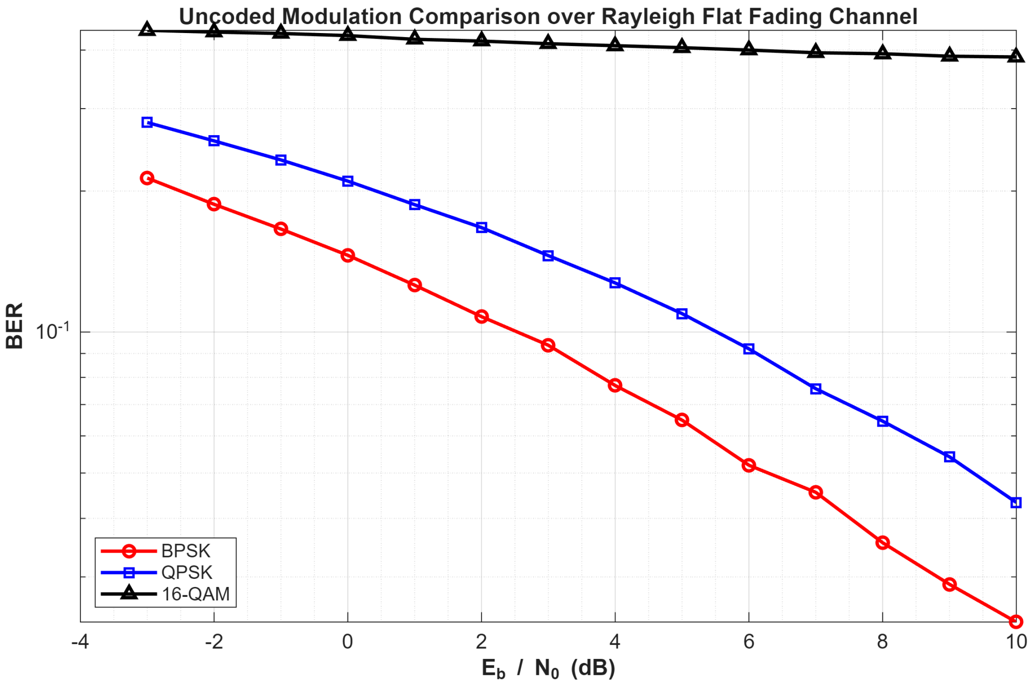

Figure 4.4: BER comparison of uncoded BPSK, QPSK, and 16-QAM over Rayleigh flat fading channel.

4.5 Discussion

The obtained results clearly demonstrate the impact of Rayleigh fading on system performance. Compared to AWGN channels, the BER performance degrades significantly due to random amplitude and phase variations introduced by fading.

Among the uncoded schemes, BPSK achieves the best BER performance due to its maximum Euclidean distance between constellation points. QPSK exhibits similar performance to BPSK when normalized for energy per bit, while 16-QAM suffers from higher BER because of its denser constellation.

The application of repetition coding significantly improves BER performance for all modulation schemes. This improvement is achieved through redundancy and majority voting at the receiver. However, this comes at the expense of reduced spectral efficiency.

These results confirm that:

- Rayleigh fading severely degrades BER performance compared to AWGN channels,
- Lower-order modulation schemes are more robust to fading,
- Repetition coding provides noticeable coding gain at the cost of bandwidth efficiency.

This discussion fully answers all parts (a) through (i) of Problem 2.

Chapter 5

Problem 3: OFDM over Flat and Frequency Selective Fading

In this problem, an Orthogonal Frequency Division Multiplexing (OFDM) system is implemented and evaluated under two different channel conditions:

- Flat Rayleigh fading
- Frequency selective Rayleigh fading

Three modulation schemes are considered:

- BPSK
- QPSK
- 16-QAM

Both uncoded transmission and repetition-coded transmission with repetition factor $R = 5$ are simulated. The bit error rate (BER) is measured as a function of E_b/N_0.

5.1 System Model

The implemented OFDM system follows the standard baseband structure:

1. Random bit generation
2. Optional repetition coding
3. Block interleaving
4. Symbol mapping (BPSK, QPSK, or 16-QAM)
5. $N_{\text{FFT}} = 256$ point IFFT

6. Channel model:
   - Flat fading: $y[n] = h\,x[n] + w[n]$
   - Frequency selective fading: $y[n] = \text{IFFT}\{X[k]H[k]\} + w[n]$
7. FFT at receiver
8. Frequency-domain equalization
9. Demapping and decoding

Perfect channel knowledge is assumed at the receiver for equalization.

5.2 Simulation Parameters

Table 5.1: OFDM Simulation Parameters

Parameter	Value
FFT size (N_{FFT})	256
Modulations	BPSK, QPSK, 16-QAM
Repetition factor (R)	5
OFDM symbols per SNR	150
Channel models	Flat Rayleigh, Frequency Selective Rayleigh
E_b/N_0 range	0 to 20 dB

5.3 Simulation Results

5.3.1 OFDM BPSK

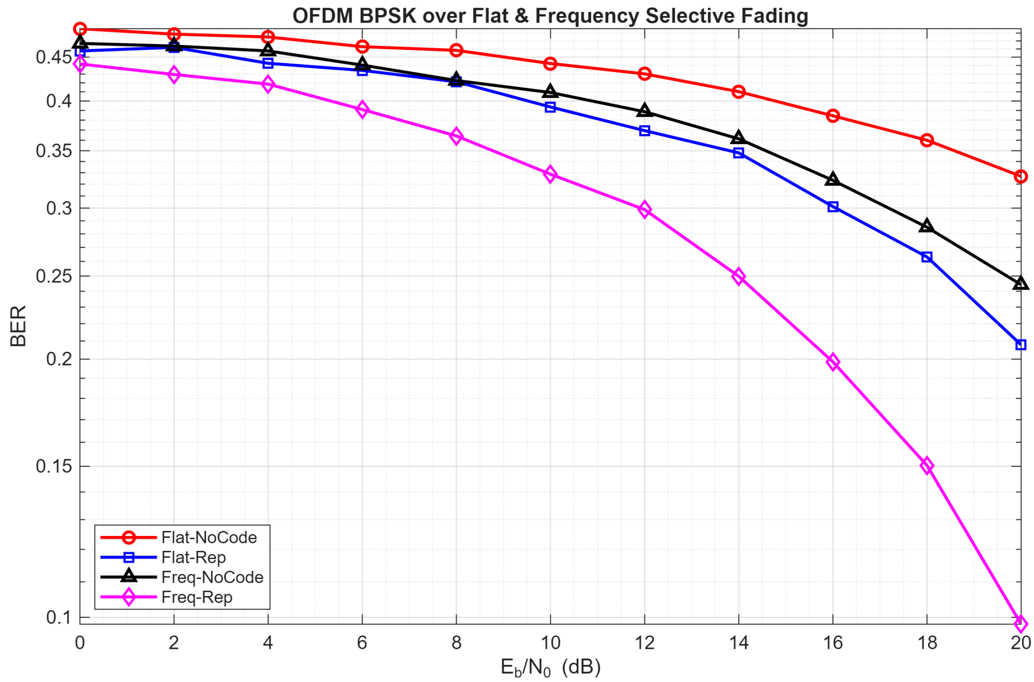

Figure 5.1: BER performance of OFDM BPSK over flat and frequency selective fading channels.

5.3.2 OFDM QPSK

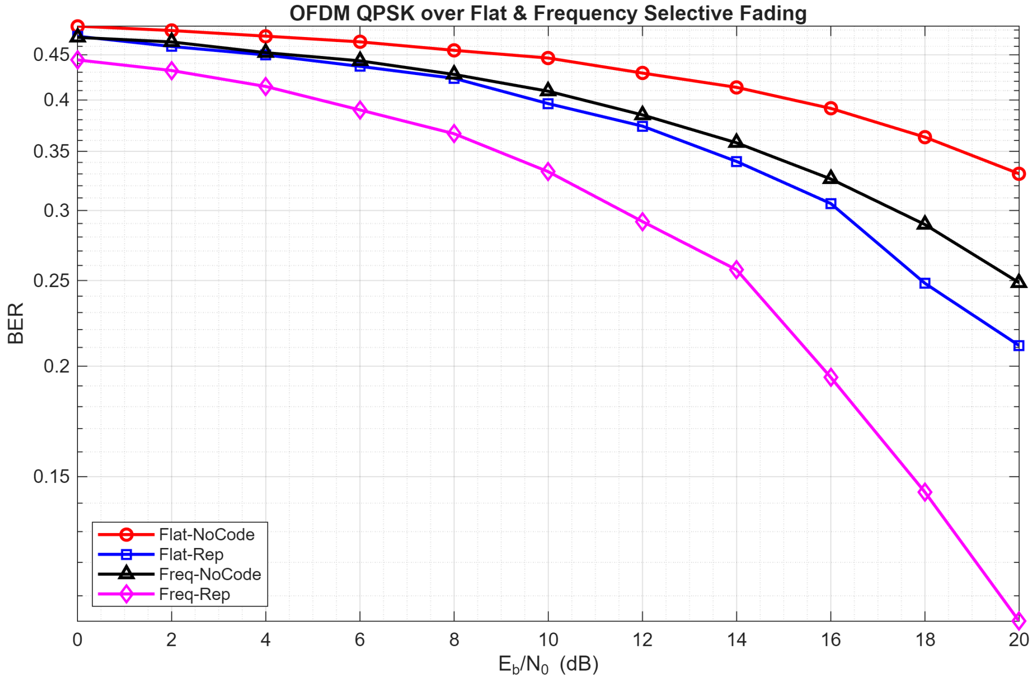

Figure 5.2: BER performance of OFDM QPSK over flat and frequency selective fading channels.

5.3.3 OFDM 16-QAM

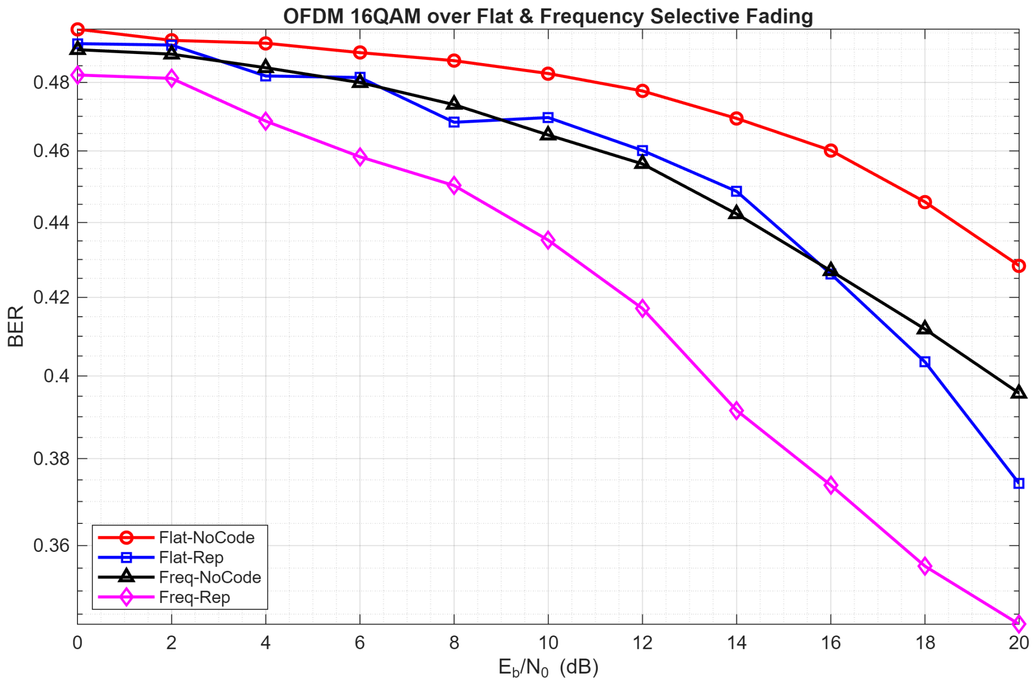

Figure 5.3: BER performance of OFDM 16-QAM over flat and frequency selective fading channels.

5.4 Discussion

For both BPSK and QPSK modulation, the BER curves exhibit the expected monotonic decrease with increasing E_b/N_0. Repetition coding provides a noticeable coding gain in both flat and frequency selective fading channels due to majority voting at the receiver.

The frequency selective channel shows improved performance compared to flat fading at higher SNR values. This is because OFDM converts the frequency selective channel into multiple flat subchannels, allowing efficient frequency-domain equalization.

For 16-QAM modulation, the BER is significantly higher compared to BPSK and QPSK. This behavior is expected due to the reduced minimum Euclidean distance between constellation points, which increases sensitivity to noise and fading. Nevertheless, repetition coding still improves BER, demonstrating the effectiveness of coding even for higher-order modulation schemes.

The obtained results confirm that:

- OFDM effectively mitigates frequency selective fading
- Repetition coding improves reliability at the cost of bandwidth efficiency
- Higher-order modulation trades BER performance for increased data rate

This discussion fully answers the requirements of Problem 3.

Chapter 6

Conclusion

In this project, a comprehensive study of Orthogonal Frequency Division Multiplexing (OFDM) systems was conducted through three main problems, focusing on computational efficiency, error performance over fading channels, and complete OFDM system behavior.

In Problem 1, the execution time of the Discrete Fourier Transform (DFT) was compared with the Fast Fourier Transform (FFT). The results clearly demonstrated the significant computational advantage of FFT, which reduces complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. This confirms why FFT is a fundamental component in practical OFDM implementations.

Problem 2 investigated the BER performance of BPSK, QPSK, and 16-QAM modulation schemes over Rayleigh flat fading channels, both with and without repetition coding. The simulations showed that repetition coding provides a noticeable coding gain by improving BER at the expense of bandwidth efficiency. Additionally, higher-order modulation schemes were shown to offer higher data rates but with increased sensitivity to noise and fading.

In Problem 3, a complete OFDM system was implemented and evaluated over flat and frequency-selective Rayleigh fading channels. The results confirmed that OFDM effectively transforms a frequency-selective channel into multiple flat subchannels, enabling efficient frequency-domain equalization. Repetition coding further enhanced reliability, while higher-order modulation maintained higher spectral efficiency.

Overall, the results of Project 3 validate the effectiveness of OFDM combined with channel coding in combating fading and noise in wireless communication systems. The project highlights the fundamental trade-offs between computational complexity, reliability, and spectral efficiency that govern the design of modern communication systems.

Appendix A

Appendix

This appendix contains the complete MATLAB source codes used to implement Project 3. The codes are organized according to the corresponding problem statements and include the graphical user interface used to execute the simulations.

A.1 Problem 1: DFT vs FFT Execution Time

Listing A.1: MATLAB code for Problem 1: DFT and FFT execution time comparison

```
clear;
clc;
close all;
L = 4096;
x = randn(1,L);

tic
X_dft = myDFT(x);
t_dft = toc;

tic
X_fft = fft(x);
t_fft = toc;

disp(['Elapsed time is ', num2str(t_dft), ' seconds.']);
disp(['Elapsed time is ', num2str(t_fft), ' seconds.']);

fig1 = plot_dft_fft_time(t_dft, t_fft);
save_figure_png(fig1, 'Q1_DFT_vs_FFT_Execution_Time', 'figures');


```

```matlab
%% ========================= Functions ========================
%% My DFT Function
function X = myDFT(x)
N = length(x);
X = zeros(1,N);

for k = 0:N-1
    sumVal = 0;
    for n = 0:N-1
        sumVal = sumVal + x(n+1)*exp(-1j*2*pi*n*k/N);
    end
    X(k+1) = sumVal;
end
end

%% Bar Chart
function fig = plot_dft_fft_time(t_dft, t_fft)
% PLOT_DFT_FFT_TIME
% Plots execution time comparison between DFT and FFT

    times = [t_dft t_fft];

    fig = figure;
    b = bar(times);
    grid on;

    set(gca, 'XTickLabel', {'DFT', 'FFT'});
    ylabel('Execution Time (seconds)');
    title('Execution Time Comparison: DFT vs FFT');

    % Display values above bars
    for i = 1:length(times)
        text(i, times(i), sprintf('%.6f', times(i)), ...
            'HorizontalAlignment', 'center', ...
            'VerticalAlignment', 'bottom');
    end

end


%%  Save Figure
```

```matlab
function save_figure_png(figHandle, figName, savePath)
% SAVE_FIGURE_PNG
% Saves a MATLAB figure as PNG with proper formatting
%
% Inputs:
%   figHandle : handle to figure
%   figName   : string (figure title & filename)
%   savePath  : string (directory path)

    % --- Input checks ---
    if ~isvalid(figHandle)
        error('Invalid figure handle.');
    end

    if ~isfolder(savePath)
        mkdir(savePath);
    end

    % --- Set figure properties ---
    figHandle.Name = figName;
    figHandle.NumberTitle = 'off';

    % --- Build full file path ---
    fileName = fullfile(savePath, [figName '.png']);

    % --- Save figure ---
    exportgraphics(figHandle, fileName, 'Resolution', 300);

    fprintf('Figure saved successfully:\n%s\n', fileName);
end
```

A.2 Problem 2: BER Performance over Rayleigh Flat Fading

Listing A.2: MATLAB code for Problem 2: BER performance of BPSK, QPSK, and 16-QAM over Rayleigh flat fading

```matlab
clear;
clc;
close all;
```

```
%% ================= PARAMETERS =================
Nbits = 1e5;
Eb  = 1;
R   = 5;                        % Repetition factor

EbN0_dB   = -3:1:10;
EbN0_lin = 10.^(EbN0_dB/10);

% BER storage
BER_BPSK_unc   = zeros(size(EbN0_dB));
BER_BPSK_rep   = zeros(size(EbN0_dB));
BER_QPSK_unc   = zeros(size(EbN0_dB));
BER_QPSK_rep   = zeros(size(EbN0_dB));
BER_16QAM_unc = zeros(size(EbN0_dB));
BER_16QAM_rep = zeros(size(EbN0_dB));

%% ================= MAIN Eb/N0 LOOP =================
for idx = 1:length(EbN0_dB)

    N0 = Eb / EbN0_lin(idx);

    %% ==================================================
    %% ======================= BPSK =====================
    %% ==================================================
    bk = randi([0 1], Nbits, 1);

    % ----- Uncoded BPSK -----
    xk = sqrt(Eb) * (2*bk - 1);

    h = (randn(Nbits,1) + 1j*randn(Nbits,1)) / sqrt(2);
    n = sqrt(N0/2) * (randn(Nbits,1) + 1j*randn(Nbits,1));

    y = h .* xk + n;
    y_eq = y ./ h;
    bk_hat = real(y_eq) > 0;

    BER_BPSK_unc(idx) = mean(bk_hat ~= bk);

    % ----- Repetition BPSK -----
    bk_rep = repelem(bk, R);
```

```
    xk_rep = sqrt(Eb) * (2*bk_rep - 1);

    h = (randn(length(xk_rep),1) + 1j*randn(length(xk_rep),1)) /
       sqrt(2);
    n = sqrt(N0/2) * (randn(length(xk_rep),1) + 1j*randn(length(
       xk_rep),1));

    y = h .* xk_rep + n;
    y_eq = y ./ h;

    bk_hat_rep = real(y_eq) > 0;
    bk_hat_mat = reshape(bk_hat_rep, R, []);
    bk_hat = sum(bk_hat_mat,1) >= ceil(R/2);

    BER_BPSK_rep(idx) = mean(bk_hat.' ~= bk);

    %% =====================================================
    %% ====================== QPSK =====================
    %% =====================================================
    bits = bk(1:2*floor(length(bk)/2));
    b = reshape(bits,2,[]).';

    xk = sqrt(Eb/2) * ((2*b(:,1)-1) + 1j*(2*b(:,2)-1));

    h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt(2);
    n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(xk)
       ,1));

    y = h .* xk + n;
    y_eq = y ./ h;

    b1 = real(y_eq) > 0;
    b2 = imag(y_eq) > 0;

    bk_hat = reshape([b1 b2].',[],1);
    BER_QPSK_unc(idx) = mean(bk_hat ~= bits);

    % ----- Repetition QPSK -----
    bits_rep = repelem(bits, R);
    b = reshape(bits_rep,2,[]).';

```

```
    xk = sqrt(Eb/2) * ((2*b(:,1)-1) + 1j*(2*b(:,2)-1));

    h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt(2);
    n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(xk)
        ,1));

    y = h .* xk + n;
    y_eq = y ./ h;

    b1 = real(y_eq) > 0;
    b2 = imag(y_eq) > 0;

    bk_hat_rep = reshape([b1 b2].',[],1);
    bk_hat_mat = reshape(bk_hat_rep, R, []);
    bk_hat = sum(bk_hat_mat,1) >= ceil(R/2);

    BER_QPSK_rep(idx) = mean(bk_hat.' ~= bits);

    %% ==================================================
    %% ==================== 16-QAM ====================
    %% ==================================================
    bits = bk(1:4*floor(length(bk)/4));
    b = reshape(bits,4,[]).';

    I = (2*b(:,1)-1) .* (2 - (2*b(:,3)));
    Q = (2*b(:,2)-1) .* (2 - (2*b(:,4)));
    xk = (I + 1j*Q) / sqrt(10);

    h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt(2);
    n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(xk)
        ,1));

    y = h .* xk + n;
    y_eq = y ./ h;

    I_hat = real(y_eq);
    Q_hat = imag(y_eq);

    b1 = I_hat > 0;
    b2 = Q_hat > 0;
    b3 = abs(I_hat) < 2;
```

```
    b4 = abs(Q_hat) < 2;

    bk_hat = reshape([b1 b2 b3 b4].',[],1);
    BER_16QAM_unc(idx) = mean(bk_hat ~= bits);

    % ----- Repetition 16-QAM -----
    bits_rep = repelem(bits, R);
    b = reshape(bits_rep,4,[]).';

    I = (2*b(:,1)-1) .* (2 - (2*b(:,3)));
    Q = (2*b(:,2)-1) .* (2 - (2*b(:,4)));
    xk = (I + 1j*Q) / sqrt(10);

    h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt(2);
    n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(xk)
        ,1));

    y = h .* xk + n;
    y_eq = y ./ h;

    I_hat = real(y_eq);
    Q_hat = imag(y_eq);

    b1 = I_hat > 0;
    b2 = Q_hat > 0;
    b3 = abs(I_hat) < 2;
    b4 = abs(Q_hat) < 2;

    bk_hat_rep = reshape([b1 b2 b3 b4].',[],1);
    bk_hat_mat = reshape(bk_hat_rep, R, []);
    bk_hat = sum(bk_hat_mat,1) >= ceil(R/2);

    BER_16QAM_rep(idx) = mean(bk_hat.' ~= bits);
end

%% ==================== FIGURES ====================

% ---- BPSK ----
fig = plot_rayleigh(EbN0_dB, BER_BPSK_unc, BER_BPSK_rep, Nbits, '
    BPSK');
save_figure_png(fig, ...
```

```
    'Q2_BPSK_Rayleigh_Uncoded_vs_Repetition', ...
    'figures');

% ---- QPSK ----
fig = plot_rayleigh(EbN0_dB, BER_QPSK_unc, BER_QPSK_rep, Nbits, 'QPSK');
save_figure_png(fig, ...
    'Q2_QPSK_Rayleigh_Uncoded_vs_Repetition', ...
    'figures');

% ---- 16-QAM ----
fig = plot_rayleigh(EbN0_dB, BER_16QAM_unc, BER_16QAM_rep, Nbits, '16-QAM');
save_figure_png(fig, ...
    'Q2_16QAM_Rayleigh_Uncoded_vs_Repetition', ...
    'figures');

% ---- Combined Uncoded Comparison ----
fig = figure;

semilogy(EbN0_dB, BER_BPSK_unc, 'r-o', 'LineWidth', 1.8); hold on;
semilogy(EbN0_dB, BER_QPSK_unc, 'b-s', 'LineWidth', 1.8);
semilogy(EbN0_dB, BER_16QAM_unc,'k-^', 'LineWidth', 1.8);

grid on; grid minor;

legend('BPSK','QPSK','16-QAM','Location','southwest');
xlabel('E_b / N_0 (dB)','FontWeight','bold');
ylabel('BER','FontWeight','bold');
title('Uncoded Modulation Comparison over Rayleigh Flat Fading Channel', ...
      'FontWeight','bold');

set(gca,'YScale','log','FontSize',11);

save_figure_png(fig, ...
    'Q2_Uncoded_BPSK_QPSK_16QAM_Comparison', ...
    'figures');


```

```matlab
%% ========================= Functions ========================
%% Plot BER
function fig = plot_rayleigh(EbN0_dB, BER_uncoded, BER_rep, Nbits
    , modType)
% PLOT_Q2_BPSK_RAYLEIGH
% Plots uncoded and repetition-coded transmission over Rayleigh
    flat fading

    fig = figure;

    semilogy(EbN0_dB, BER_uncoded, ...
        'ro--', 'LineWidth', 1.8, 'MarkerSize', 7);
    hold on;

    semilogy(EbN0_dB, BER_rep, ...
        'bs--', 'LineWidth', 1.8, 'MarkerSize', 7);

    grid on;
    grid minor;

    xlabel('E_b / N_0 (dB)', ...
        'FontSize', 12, 'FontWeight', 'bold');

    ylabel('Bit Error Rate (BER)', ...
        'FontSize', 12, 'FontWeight', 'bold');

    title([modType ' over Rayleigh Flat Fading Channel'], ...
        'FontSize', 14, 'FontWeight', 'bold');

    legend( ...
        ['Uncoded ' modType], ...
        [modType ' with Repetition (Rate 1/5)'], ...
        'Location','southwest');

    set(gca, 'FontSize', 11);
    set(gca, 'YScale', 'log');

    % Info box
    annotation(fig,'textbox', ...
        [0.15 0.18 0.38 0.18], ...
        'String',{ ...
```

```
            ['Modulation: ' modType], ...
            'Channel: Rayleigh Flat Fading', ...
            'Coding: Repetition (Rate 1/5)', ...
            sprintf('# bits: %d', Nbits)}, ...
        'FitBoxToText','on', ...
        'BackgroundColor','white', ...
        'EdgeColor','black', ...
        'FontSize',10);
end

%% Save Figure
function save_figure_png(figHandle, figName, savePath)
% SAVE_FIGURE_PNG
% Saves a MATLAB figure as PNG with proper formatting
%
% Inputs:
%   figHandle : handle to figure
%   figName   : string (figure title & filename)
%   savePath  : string (directory path)

    % --- Input checks ---
    if ~isvalid(figHandle)
        error('Invalid figure handle.');
    end

    if ~isfolder(savePath)
        mkdir(savePath);
    end

    % --- Set figure properties ---
    figHandle.Name = figName;
    figHandle.NumberTitle = 'off';

    % --- Build full file path ---
    fileName = fullfile(savePath, [figName '.png']);

    % --- Save figure ---
    exportgraphics(figHandle, fileName, 'Resolution', 300);

    fprintf('Figure saved successfully:\n%s\n', fileName);
end
```

A.3 Problem 3: OFDM over Flat and Frequency Selective Fading

Listing A.3: MATLAB code for Problem 3: OFDM system simulation over flat and frequency selective fading

```
clear; clc; close all;

%% ================= PARAMETERS =================
Nfft = 256;
Eb = 1;
EbN0_dB = 0:2:20;
EbN0_lin = 10.^(EbN0_dB/10);
No = Eb ./ EbN0_lin;                    % noise spectral density

mods = {'BPSK','QPSK','16QAM'};
R = 5;                        % repetition factor
Nsym = 150;                   % OFDM symbols per SNR (runtime safe)

% Two Channels
h_flat = (randn + 1j*randn)/sqrt(2);
H = (randn(Nfft,1) + 1j*randn(Nfft,1)) / sqrt(2);

%% ================= LOOP OVER MODULATIONS =================
for m = 1:length(mods)

    modType = mods{m};

    %% ===== OFDM FRAME DEFINITION =====
    switch modType
        case 'BPSK'
            rows = 32; cols = 8;   bits_ps = 1;
        case 'QPSK'
            rows = 32; cols = 16;  bits_ps = 2;
        case '16QAM'
            rows = 32; cols = 32;  bits_ps = 4;
    end

    bits_per_symbol = rows * cols;    % mapper input size

    % ---- Uncoded system ----
```

```
    bits_uncoded = bits_per_symbol;  % full OFDM payload

    % ---- Repetition-coded system ----
    bits_info_rep = floor(bits_per_symbol / R);  % information
       bits
    bits_coded    = bits_info_rep * R;             % after
       repetition




    BER_flat_unc = zeros(size(EbN0_dB));
    BER_flat_rep = zeros(size(EbN0_dB));
    BER_freq_unc = zeros(size(EbN0_dB));
    BER_freq_rep = zeros(size(EbN0_dB));

    for snr = 1:length(EbN0_dB)

        err_fu = 0; err_fc = 0;
        err_su = 0; err_sc = 0;
        bits_unc_cnt = 0; bits_rep_cnt = 0;

        for sym = 1:Nsym

            %% ========== PART 1: CODING ==========

            % ---- Uncoded ----
            info_unc = randi([0 1], bits_uncoded, 1);

            % ---- Repetition coded ----
            info_rep  = randi([0 1], bits_info_rep, 1);
            coded_rep = repelem(info_rep, R);

            % ---- Padding to OFDM size ----
            info_unc_p  = info_unc;  % already bits_per_symbol
            coded_rep_p = [coded_rep; zeros(bits_per_symbol -
               length(coded_rep),1)];

            %% ========== PART 2: INTERLEAVER ==========
            info_unc_i  = ofdm_interleave(info_unc_p,  modType);
            coded_rep_i = ofdm_interleave(coded_rep_p, modType);
```

```
    %% ========== PART 3: MAPPER ==========
    X_unc = ofdm_mapper(info_unc_i, modType, Eb);
    X_rep = ofdm_mapper(coded_rep_i, modType, Eb);


    %% ========== PART 4a: IFFT ==========
    x_unc = ifft(X_unc);
    x_rep = ifft(X_rep);

    %% ========== PART 5a: FLAT FADING ==========

    y_unc_flat = h_flat * x_unc;
    y_rep_flat = h_flat * x_rep;



    %% ========== AWGN ==========
    sigma = sqrt(Eb ./ (2*EbN0_lin(snr)));

    noise_unc = sigma * (randn(size(y_unc_flat)) + 1j*
       randn(size(y_unc_flat)));
    noise_rep = sigma * (randn(size(y_rep_flat)) + 1j*
       randn(size(y_rep_flat)));

    y_unc_flat = y_unc_flat + noise_unc;
    y_rep_flat = y_rep_flat + noise_rep;

    %% ========== PART 5b: FREQUENCY SELECTIVE ==========

    y_unc_sel = ifft(X_unc .* H);
    y_rep_sel = ifft(X_rep .* H);

    %% ========== AWGN ==========
    sigma = sqrt(Eb ./ (2*EbN0_lin(snr)));

    noise_unc = sigma * (randn(size(y_unc_sel)) + 1j*
       randn(size(y_unc_sel)));
    noise_rep = sigma * (randn(size(y_rep_sel)) + 1j*
       randn(size(y_rep_sel)));

```

```
            y_unc_sel = y_unc_sel + noise_unc;
            y_rep_sel = y_rep_sel + noise_rep;


            %% ========== PART 6: RECEIVER (FLAT) ==========
            err_fu = err_fu + ofdm_receiver(y_unc_flat, h_flat,
               info_unc, modType, false, R);
            err_fc = err_fc + ofdm_receiver(y_rep_flat, h_flat,
               info_rep, modType, true,  R);

            %% ========== PART 6: RECEIVER (SELECTIVE) ==========
            err_su = err_su + ofdm_receiver(y_unc_sel, H,
               info_unc, modType, false, R);
            err_sc = err_sc + ofdm_receiver(y_rep_sel, H,
               info_rep, modType, true,  R);

            bits_unc_cnt = bits_unc_cnt + length(info_unc);
            bits_rep_cnt = bits_rep_cnt + length(info_rep);

        end

        BER_flat_unc(snr) = err_fu / bits_unc_cnt;
        BER_flat_rep(snr) = err_fc / bits_rep_cnt;
        BER_freq_unc(snr) = err_su / bits_unc_cnt;
        BER_freq_rep(snr) = err_sc / bits_rep_cnt;
    end

    %% ========== PLOT ==========
    fig = plot_ofdm(EbN0_dB, ...
        BER_flat_unc, BER_flat_rep, ...
        BER_freq_unc, BER_freq_rep, modType);

    save_figure_png(fig,['Q3_OFDM_' modType],'figures');
end

%% ======================== Functions ========================
%% Coding
function [info_unc, info_rep, coded_rep, pad_unc, pad_rep] =
   ofdm_coding(modType,Nfft,R)

    bps = strcmp(modType,'BPSK') + ...
```

```
            2*strcmp(modType,'QPSK') + ...
            4*strcmp(modType,'16QAM');

    Ninfo_unc = floor(Nfft/bps);
    Ninfo_rep = floor(Nfft/(R*bps));

    info_unc = randi([0 1],Ninfo_unc*bps,1);
    info_rep = randi([0 1],Ninfo_rep*bps,1);

    coded_rep = repelem(info_rep,R);

    % ---- padding length (DO NOT append yet) ----
    pad_unc = Nfft*bps - length(info_unc);
    pad_rep = Nfft*bps - length(coded_rep);
end

%% InterLeaver
function out = ofdm_interleave(bits, modType)

    switch modType
        case 'QPSK'
            assert(length(bits)==512, 'QPSK interleaver requires 512 bits');
            out = reshape(bits,32,16).';
            out = out(:);

        case '16QAM'
            assert(length(bits)==1024,'16QAM interleaver requires 1024 bits');
            out = reshape(bits,32,32).';
            out = out(:);

        otherwise  % BPSK
            out = bits;
    end
end

%% Mapper
function X = ofdm_mapper(bits, modType, Eb)

    switch modType
```

```
        case 'BPSK'
            % Average symbol energy = Eb
            X = sqrt(Eb) * (2*bits - 1);

        case 'QPSK'
            % Average symbol energy = 2Eb     scaled to Eb
            b = reshape(bits,2,[]).';
            X = sqrt(Eb) * ( ...
                (2*b(:,1)-1) + 1j*(2*b(:,2)-1) );

        case '16QAM'
            % Average symbol energy = 10     normalized by 2.5
            b = reshape(bits,4,[]).';
            I = (2*b(:,1)-1).*(2-(2*b(:,3)));
            Q = (2*b(:,2)-1).*(2-(2*b(:,4)));
            X = sqrt(Eb/2.5) * (I + 1j*Q);

    end
end

%% Recevier
function err = ofdm_receiver(y, h, info_bits, modType, isRep, R)

    %% FFT
    Y = fft(y);

    %% Equalization
    if numel(h) > 1          % frequency selective
        Yeq = Y ./ (h + 1e-12);
    else                     % flat fading
        Yeq = Y / h;
    end

    %% Demapper
    switch modType
        case 'BPSK'
            rx_bits = real(Yeq) > 0;

        case 'QPSK'
            rx_bits = reshape([real(Yeq)>0 imag(Yeq)>0].',1,[]);
```

```
        case '16QAM'
            rx_bits = reshape([ ...
                real(Yeq)>0 imag(Yeq)>0 ...
                abs(real(Yeq))<2 abs(imag(Yeq))<2].',1,[]);
    end

    %% De-interleaver
    switch modType
        case 'QPSK'
            rx_bits = reshape(rx_bits,16,32).';
            rx_bits = rx_bits(:).';

        case '16QAM'
            rx_bits = reshape(rx_bits,32,32).';
            rx_bits = rx_bits(:).';
    end

    %% Decode (Repetition)
    if isRep
        dec = zeros(1,length(info_bits));
        for k = 1:length(info_bits)
            dec(k) = sum(rx_bits((k-1)*R+1:k*R)) > R/2;
        end
        rx_bits = dec;
    else
        rx_bits = rx_bits(1:length(info_bits));
    end

    %% Error count
    rx_bits   = rx_bits(:);
    info_bits = info_bits(:);

    err = sum(rx_bits ~= info_bits);

end

%% Noise
function n = awgn_noise(x, Eb, EbN0)
    N0 = Eb / EbN0;
    n  = sqrt(N0/2) * (randn(size(x)) + 1j*randn(size(x)));
```

```
end


%% Plot
function fig = plot_ofdm(EbN0_dB,fU,fC,sU,sC,modType)
    fig = figure;
    semilogy(EbN0_dB,fU,'r-o','LineWidth',1.6); hold on;
    semilogy(EbN0_dB,fC,'b-s','LineWidth',1.6);
    semilogy(EbN0_dB,sU,'k-^','LineWidth',1.6);
    semilogy(EbN0_dB,sC,'m-d','LineWidth',1.6);
    grid on; grid minor;
    legend('Flat-NoCode','Flat-Rep','Freq-NoCode','Freq-Rep','Location','southwest');
    xlabel('E_b/N_0 (dB)');
    ylabel('BER');
    title(['OFDM ' modType ' over Flat & Frequency Selective Fading']);
end

%%  Save Figure
function save_figure_png(figHandle, figName, savePath)
% SAVE_FIGURE_PNG
% Saves a MATLAB figure as PNG with proper formatting
%
% Inputs:
%   figHandle : handle to figure
%   figName   : string (figure title & filename)
%   savePath  : string (directory path)

    % --- Input checks ---
    if ~isvalid(figHandle)
        error('Invalid figure handle.');
    end

    if ~isfolder(savePath)
        mkdir(savePath);
    end

    % --- Set figure properties ---
    figHandle.Name = figName;
    figHandle.NumberTitle = 'off';
```

```
    % --- Build full file path ---
    fileName = fullfile(savePath, [figName '.png']);

    % --- Save figure ---
    exportgraphics(figHandle, fileName, 'Resolution', 300);

    fprintf('Figure saved successfully:\n%s\n', fileName);
end
```

A.4 Graphical User Interface (GUI)

Listing A.4: MATLAB GUI implementation for Project 3 (OFDM simulations)

```
function OFDM_GUI_T35
% OFDM PROJECT GUI (Q1     Q3)
% Single-file implementation

    % ================= HOME PAGE =================
    homeFig = uifigure( ...
        'Name','OFDM Project', ...
        'Position',[500 200 420 420]);

    gl = uigridlayout(homeFig,[4 1]);
    gl.RowHeight = {'fit','1x','1x','1x'};
    gl.Padding = [30 30 30 20];

    % Title
    uilabel(gl, ...
        'Text','OFDM Project     Team 35', ...
        'FontSize',18, ...
        'FontWeight','bold', ...
        'HorizontalAlignment','center');

    % Question buttons
    for q = 1:3
        uibutton(gl, ...
            'Text', sprintf('Question %d', q), ...
            'FontSize',14, ...
            'ButtonPushedFcn', @(~,~) open_question(q, homeFig));
    end
```

```
end

% ================================================================
%                        QUESTION WINDOW
% ================================================================
function open_question(qnum, homeFig)

    qFig = uifigure( ...
        'Name', sprintf('Question %d', qnum), ...
        'Position',[400 150 900 550]);

    gl = uigridlayout(qFig,[3 1]);
    gl.RowHeight = {'fit','fit','1x'};
    gl.Padding = [15 15 15 15];

    % Header
    uilabel(gl, ...
        'Text', sprintf('Question %d', qnum), ...
        'FontSize',16, ...
        'FontWeight','bold', ...
        'HorizontalAlignment','center');

    % Return button
    uibutton(gl, ...
        'Text','    Return to Home', ...
        'FontSize',13, ...
        'ButtonPushedFcn', @(~,~) return_home(qFig, homeFig));

    % Output console
    logBox = uitextarea(gl, ...
        'Editable','off', ...
        'FontSize',11);

    drawnow;

    % ================= RUN QUESTION =================
    try
        switch qnum
            case 1
                outputText = evalc('Q1');
            case 2
```

```
                outputText = evalc('Q2');
            case 3
                outputText = evalc('Q3');
        end

        logBox.Value = splitlines(outputText);

    catch ME
        logBox.Value = { ...
            ' Error occurred:', ...
            ME.message ...
        };
        uialert(qFig, ME.message, 'Execution Error');
    end
end


% ===============================================================
%                  RETURN BUTTON
% ===============================================================
function return_home(qFig, homeFig)
    if isvalid(qFig)
        close(qFig);
    end
    homeFig.Visible = 'on';
end

%% Q1
% ===============================================================
% ====================== QUESTION 1 =============================
% ===============================================================
function Q1
    L = 4096;
    x = randn(1,L);

    tic
    X_dft = myDFT(x);
    t_dft = toc;

    tic
    X_fft = fft(x);
```

```
    t_fft = toc;

    fprintf('Elapsed time is %s seconds.', num2str(t_dft));
    fprintf('Elapsed time is %s seconds.', num2str(t_fft));

    fig1 = plot_dft_fft_time(t_dft, t_fft);
    save_figure_png(fig1, 'Q1_DFT_vs_FFT_Execution_Time', 'figures');
end
%====================== Q1 Helper functions =============================

%% My DFT Function
function X = myDFT(x)
N = length(x);
X = zeros(1,N);

for k = 0:N-1
    sumVal = 0;
    for n = 0:N-1
        sumVal = sumVal + x(n+1)*exp(-1j*2*pi*n*k/N);
    end
    X(k+1) = sumVal;
end
end

%% Bar Chart
function fig = plot_dft_fft_time(t_dft, t_fft)
% PLOT_DFT_FFT_TIME
% Plots execution time comparison between DFT and FFT

    times = [t_dft t_fft];

    fig = figure;
    b = bar(times);
    grid on;

    set(gca, 'XTickLabel', {'DFT', 'FFT'});
    ylabel('Execution Time (seconds)');
    title('Execution Time Comparison: DFT vs FFT');

```

```
    % Display values above bars
    for i = 1:length(times)
        text(i, times(i), sprintf('%.6f', times(i)), ...
            'HorizontalAlignment', 'center', ...
            'VerticalAlignment', 'bottom');
    end

end


%% Save Figure
function save_figure_png(figHandle, figName, savePath)
% SAVE_FIGURE_PNG
% Saves a MATLAB figure as PNG with proper formatting
%
% Inputs:
%   figHandle : handle to figure
%   figName   : string (figure title & filename)
%   savePath  : string (directory path)

    % --- Input checks ---
    if ~isvalid(figHandle)
        error('Invalid figure handle.');
    end

    if ~isfolder(savePath)
        mkdir(savePath);
    end

    % --- Set figure properties ---
    figHandle.Name = figName;
    figHandle.NumberTitle = 'off';

    % --- Build full file path ---
    fileName = fullfile(savePath, [figName '.png']);

    % --- Save figure ---
    exportgraphics(figHandle, fileName, 'Resolution', 300);

    fprintf('Figure saved successfully:\n%s\n', fileName);
end
```

```
%% Q2
% ==============================================================
% ===================== QUESTION 2 ==============================
% ==============================================================
function Q2

    %% ================= PARAMETERS =================
    Nbits = 1e5;
    Eb = 1;
    R  = 5;                        % Repetition factor

    EbN0_dB  = -3:1:10;
    EbN0_lin = 10.^(EbN0_dB/10);

    % BER storage
    BER_BPSK_unc  = zeros(size(EbN0_dB));
    BER_BPSK_rep  = zeros(size(EbN0_dB));
    BER_QPSK_unc  = zeros(size(EbN0_dB));
    BER_QPSK_rep  = zeros(size(EbN0_dB));
    BER_16QAM_unc = zeros(size(EbN0_dB));
    BER_16QAM_rep = zeros(size(EbN0_dB));

    %% ================= MAIN Eb/N0 LOOP =================
    for idx = 1:length(EbN0_dB)

        N0 = Eb / EbN0_lin(idx);

        %% ==================================================
        %% ====================== BPSK =====================
        %% ==================================================
        bk = randi([0 1], Nbits, 1);

        % ----- Uncoded BPSK -----
        xk = sqrt(Eb) * (2*bk - 1);

        h = (randn(Nbits,1) + 1j*randn(Nbits,1)) / sqrt(2);
        n = sqrt(N0/2) * (randn(Nbits,1) + 1j*randn(Nbits,1));

        y = h .* xk + n;
```

```
        y_eq = y ./ h;
        bk_hat = real(y_eq) > 0;

        BER_BPSK_unc(idx) = mean(bk_hat ~= bk);

        % ----- Repetition BPSK -----
        bk_rep = repelem(bk, R);
        xk_rep = sqrt(Eb) * (2*bk_rep - 1);

        h = (randn(length(xk_rep),1) + 1j*randn(length(xk_rep),1)) / sqrt(2);
        n = sqrt(N0/2) * (randn(length(xk_rep),1) + 1j*randn(length(xk_rep),1));

        y = h .* xk_rep + n;
        y_eq = y ./ h;

        bk_hat_rep = real(y_eq) > 0;
        bk_hat_mat = reshape(bk_hat_rep, R, []);
        bk_hat = sum(bk_hat_mat,1) >= ceil(R/2);

        BER_BPSK_rep(idx) = mean(bk_hat.' ~= bk);

        %% ==================================================
        %% ====================== QPSK =====================
        %% ==================================================
        bits = bk(1:2*floor(length(bk)/2));
        b = reshape(bits,2,[]).';

        xk = sqrt(Eb/2) * ((2*b(:,1)-1) + 1j*(2*b(:,2)-1));

        h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt(2);
        n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(xk),1));

        y = h .* xk + n;
        y_eq = y ./ h;

        b1 = real(y_eq) > 0;
        b2 = imag(y_eq) > 0;
```

```
        bk_hat = reshape([b1 b2].',[],1);
        BER_QPSK_unc(idx) = mean(bk_hat ~= bits);

        % ----- Repetition QPSK -----
        bits_rep = repelem(bits, R);
        b = reshape(bits_rep,2,[]).';

        xk = sqrt(Eb/2) * ((2*b(:,1)-1) + 1j*(2*b(:,2)-1));

        h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt
           (2);
        n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(
           xk),1));

        y = h .* xk + n;
        y_eq = y ./ h;

        b1 = real(y_eq) > 0;
        b2 = imag(y_eq) > 0;

        bk_hat_rep = reshape([b1 b2].',[],1);
        bk_hat_mat = reshape(bk_hat_rep, R, []);
        bk_hat = sum(bk_hat_mat,1) >= ceil(R/2);

        BER_QPSK_rep(idx) = mean(bk_hat.' ~= bits);

        %% ==================================================
        %% ===================== 16-QAM =====================
        %% ==================================================
        bits = bk(1:4*floor(length(bk)/4));
        b = reshape(bits,4,[]).';

        I = (2*b(:,1)-1) .* (2 - (2*b(:,3)));
        Q = (2*b(:,2)-1) .* (2 - (2*b(:,4)));
        xk = (I + 1j*Q) / sqrt(10);

        h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt
           (2);
        n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(
           xk),1));
```

```

        y = h .* xk + n;
        y_eq = y ./ h;

        I_hat = real(y_eq);
        Q_hat = imag(y_eq);

        b1 = I_hat > 0;
        b2 = Q_hat > 0;
        b3 = abs(I_hat) < 2;
        b4 = abs(Q_hat) < 2;

        bk_hat = reshape([b1 b2 b3 b4].',[],1);
        BER_16QAM_unc(idx) = mean(bk_hat ~= bits);

        % ----- Repetition 16-QAM -----
        bits_rep = repelem(bits, R);
        b = reshape(bits_rep,4,[]).';

        I = (2*b(:,1)-1) .* (2 - (2*b(:,3)));
        Q = (2*b(:,2)-1) .* (2 - (2*b(:,4)));
        xk = (I + 1j*Q) / sqrt(10);

        h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt
           (2);
        n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(
           xk),1));

        y = h .* xk + n;
        y_eq = y ./ h;

        I_hat = real(y_eq);
        Q_hat = imag(y_eq);

        b1 = I_hat > 0;
        b2 = Q_hat > 0;
        b3 = abs(I_hat) < 2;
        b4 = abs(Q_hat) < 2;

        bk_hat_rep = reshape([b1 b2 b3 b4].',[],1);
        bk_hat_mat = reshape(bk_hat_rep, R, []);
```

```
        bk_hat = sum(bk_hat_mat,1) >= ceil(R/2);

        BER_16QAM_rep(idx) = mean(bk_hat.' ~= bits);
    end

    %% ===================== FIGURES =====================

    % ---- BPSK ----
    fig = plot_rayleigh(EbN0_dB, BER_BPSK_unc, BER_BPSK_rep,
        Nbits, 'BPSK');
    save_figure_png(fig, ...
        'Q2_BPSK_Rayleigh_Uncoded_vs_Repetition', ...
        'figures');

    % ---- QPSK ----
    fig = plot_rayleigh(EbN0_dB, BER_QPSK_unc, BER_QPSK_rep,
        Nbits, 'QPSK');
    save_figure_png(fig, ...
        'Q2_QPSK_Rayleigh_Uncoded_vs_Repetition', ...
        'figures');

    % ---- 16-QAM ----
    fig = plot_rayleigh(EbN0_dB, BER_16QAM_unc, BER_16QAM_rep,
        Nbits, '16-QAM');
    save_figure_png(fig, ...
        'Q2_16QAM_Rayleigh_Uncoded_vs_Repetition', ...
        'figures');

    % ---- Combined Uncoded Comparison ----
    fig = figure;

    semilogy(EbN0_dB, BER_BPSK_unc, 'r-o', 'LineWidth', 1.8);
        hold on;
    semilogy(EbN0_dB, BER_QPSK_unc, 'b-s', 'LineWidth', 1.8);
    semilogy(EbN0_dB, BER_16QAM_unc,'k-^', 'LineWidth', 1.8);

    grid on; grid minor;

    legend('BPSK','QPSK','16-QAM','Location','southwest');
    xlabel('E_b / N_0 (dB)','FontWeight','bold');
    ylabel('BER','FontWeight','bold');
```

```
    title('Uncoded Modulation Comparison over Rayleigh Flat
       Fading Channel', ...
          'FontWeight','bold');

    set(gca,'YScale','log','FontSize',11);

    save_figure_png(fig, ...
        'Q2_Uncoded_BPSK_QPSK_16QAM_Comparison', ...
        'figures');
    end

%====================== Q2 Helper functions
   ==============================
%% Plot BER
function fig = plot_rayleigh(EbN0_dB, BER_uncoded, BER_rep, Nbits
   , modType)
% PLOT_Q2_BPSK_RAYLEIGH
% Plots uncoded and repetition-coded transmission over Rayleigh
   flat fading

    fig = figure;

    semilogy(EbN0_dB, BER_uncoded, ...
        'ro--', 'LineWidth', 1.8, 'MarkerSize', 7);
    hold on;

    semilogy(EbN0_dB, BER_rep, ...
        'bs--', 'LineWidth', 1.8, 'MarkerSize', 7);

    grid on;
    grid minor;

    xlabel('E_b / N_0 (dB)', ...
        'FontSize', 12, 'FontWeight', 'bold');

    ylabel('Bit Error Rate (BER)', ...
        'FontSize', 12, 'FontWeight', 'bold');

    title([modType ' over Rayleigh Flat Fading Channel'], ...
        'FontSize', 14, 'FontWeight', 'bold');

```

```
    legend( ...
        ['Uncoded ' modType], ...
        [modType ' with Repetition (Rate 1/5)'], ...
        'Location','southwest');

    set(gca, 'FontSize', 11);
    set(gca, 'YScale', 'log');

    % Info box
    annotation(fig,'textbox', ...
        [0.15 0.18 0.38 0.18], ...
        'String',{ ...
            ['Modulation: ' modType], ...
            'Channel: Rayleigh Flat Fading', ...
            'Coding: Repetition (Rate 1/5)', ...
            sprintf('# bits: %d', Nbits)}, ...
        'FitBoxToText','on', ...
        'BackgroundColor','white', ...
        'EdgeColor','black', ...
        'FontSize',10);
end


%% Q3
% ==============================================================
% ===================== QUESTION 3 ============================
% ==============================================================
function Q3

    %% ================= PARAMETERS =================
    Nfft = 256;
    Eb = 1;
    EbN0_dB = 0:2:20;
    EbN0_lin = 10.^(EbN0_dB/10);
    No = Eb ./ EbN0_lin;                 % noise spectral density

    mods = {'BPSK','QPSK','16QAM'};
    R = 5;                           % repetition factor
    Nsym = 150;                      % OFDM symbols per SNR (runtime safe)

```

```
    % Two Channels
    h_flat = (randn + 1j*randn)/sqrt(2);
    H = (randn(Nfft,1) + 1j*randn(Nfft,1)) / sqrt(2);

    %% ================= LOOP OVER MODULATIONS =================
    for m = 1:length(mods)

        modType = mods{m};

        %% ===== OFDM FRAME DEFINITION =====
        switch modType
            case 'BPSK'
                rows = 32; cols = 8;   bits_ps = 1;
            case 'QPSK'
                rows = 32; cols = 16;  bits_ps = 2;
            case '16QAM'
                rows = 32; cols = 32;  bits_ps = 4;
        end

        bits_per_symbol = rows * cols;   % mapper input size

        % ---- Uncoded system ----
        bits_uncoded = bits_per_symbol;  % full OFDM payload

        % ---- Repetition-coded system ----
        bits_info_rep = floor(bits_per_symbol / R);  %
           information bits
        bits_coded    = bits_info_rep * R;            % after
           repetition




        BER_flat_unc = zeros(size(EbN0_dB));
        BER_flat_rep = zeros(size(EbN0_dB));
        BER_freq_unc = zeros(size(EbN0_dB));
        BER_freq_rep = zeros(size(EbN0_dB));

        for snr = 1:length(EbN0_dB)

            err_fu = 0; err_fc = 0;
```

```
        err_su = 0; err_sc = 0;
        bits_unc_cnt = 0; bits_rep_cnt = 0;

        for sym = 1:Nsym

            %% ========== PART 1: CODING ==========

            % ---- Uncoded ----
            info_unc = randi([0 1], bits_uncoded, 1);

            % ---- Repetition coded ----
            info_rep  = randi([0 1], bits_info_rep, 1);
            coded_rep = repelem(info_rep, R);

            % ---- Padding to OFDM size ----
            info_unc_p  = info_unc;  % already
               bits_per_symbol
            coded_rep_p = [coded_rep; zeros(bits_per_symbol -
                length(coded_rep),1)];

            %% ========== PART 2: INTERLEAVER ==========
            info_unc_i  = ofdm_interleave(info_unc_p,
               modType);
            coded_rep_i = ofdm_interleave(coded_rep_p,
               modType);

            %% ========== PART 3: MAPPER ==========
            X_unc = ofdm_mapper(info_unc_i, modType, Eb);
            X_rep = ofdm_mapper(coded_rep_i, modType, Eb);


            %% ========== PART 4a: IFFT ==========
            x_unc = ifft(X_unc);
            x_rep = ifft(X_rep);

            %% ========== PART 5a: FLAT FADING ==========

            y_unc_flat = h_flat * x_unc;
            y_rep_flat = h_flat * x_rep;


```

```
        %% ========== AWGN ==========
        sigma = sqrt(Eb ./ (2*EbN0_lin(snr)));

        noise_unc = sigma * (randn(size(y_unc_flat)) + 1j
            *randn(size(y_unc_flat)));
        noise_rep = sigma * (randn(size(y_rep_flat)) + 1j
            *randn(size(y_rep_flat)));

        y_unc_flat = y_unc_flat + noise_unc;
        y_rep_flat = y_rep_flat + noise_rep;

        %% ========== PART 5b: FREQUENCY SELECTIVE
            ==========

        y_unc_sel = ifft(X_unc .* H);
        y_rep_sel = ifft(X_rep .* H);

        %% ========== AWGN ==========
        sigma = sqrt(Eb ./ (2*EbN0_lin(snr)));

        noise_unc = sigma * (randn(size(y_unc_sel)) + 1j*
            randn(size(y_unc_sel)));
        noise_rep = sigma * (randn(size(y_rep_sel)) + 1j*
            randn(size(y_rep_sel)));

        y_unc_sel = y_unc_sel + noise_unc;
        y_rep_sel = y_rep_sel + noise_rep;


        %% ========== PART 6: RECEIVER (FLAT) ==========
        err_fu = err_fu + ofdm_receiver(y_unc_flat,
            h_flat, info_unc, modType, false, R);
        err_fc = err_fc + ofdm_receiver(y_rep_flat,
            h_flat, info_rep, modType, true,  R);

        %% ========== PART 6: RECEIVER (SELECTIVE)
            ==========
        err_su = err_su + ofdm_receiver(y_unc_sel, H,
            info_unc, modType, false, R);
```

```
                err_sc = err_sc + ofdm_receiver(y_rep_sel, H,
                    info_rep, modType, true,  R);

                bits_unc_cnt = bits_unc_cnt + length(info_unc);
                bits_rep_cnt = bits_rep_cnt + length(info_rep);

            end

            BER_flat_unc(snr) = err_fu / bits_unc_cnt;
            BER_flat_rep(snr) = err_fc / bits_rep_cnt;
            BER_freq_unc(snr) = err_su / bits_unc_cnt;
            BER_freq_rep(snr) = err_sc / bits_rep_cnt;
        end

        %% ========== PLOT ==========
        fig = plot_ofdm(EbN0_dB, ...
            BER_flat_unc, BER_flat_rep, ...
            BER_freq_unc, BER_freq_rep, modType);

        save_figure_png(fig,['Q3_OFDM_' modType],'figures');
    end

end
%====================== Q3 Helper functions
    =============================

%% Coding
function [info_unc, info_rep, coded_rep, pad_unc, pad_rep] =
    ofdm_coding(modType,Nfft,R)

    bps = strcmp(modType,'BPSK') + ...
          2*strcmp(modType,'QPSK') + ...
          4*strcmp(modType,'16QAM');

    Ninfo_unc = floor(Nfft/bps);
    Ninfo_rep = floor(Nfft/(R*bps));

    info_unc = randi([0 1],Ninfo_unc*bps,1);
    info_rep = randi([0 1],Ninfo_rep*bps,1);

    coded_rep = repelem(info_rep,R);
```

```

    % ---- padding length (DO NOT append yet) ----
    pad_unc = Nfft*bps - length(info_unc);
    pad_rep = Nfft*bps - length(coded_rep);
end

%% InterLeaver
function out = ofdm_interleave(bits, modType)

    switch modType
        case 'QPSK'
            assert(length(bits)==512, 'QPSK interleaver requires
               512 bits');
            out = reshape(bits,32,16).';
            out = out(:);

        case '16QAM'
            assert(length(bits)==1024,'16QAM interleaver requires
                1024 bits');
            out = reshape(bits,32,32).';
            out = out(:);

        otherwise  % BPSK
            out = bits;
    end
end

%% Mapper
function X = ofdm_mapper(bits, modType, Eb)

    switch modType

        case 'BPSK'
            % Average symbol energy = Eb
            X = sqrt(Eb) * (2*bits - 1);

        case 'QPSK'
            % Average symbol energy = 2Eb     scaled to Eb
            b = reshape(bits,2,[]).';
            X = sqrt(Eb) * ( ...
                (2*b(:,1)-1) + 1j*(2*b(:,2)-1) );
```

```
        case '16QAM'
            % Average symbol energy = 10     normalized by 2.5
            b = reshape(bits,4,[]).';
            I = (2*b(:,1)-1).*(2-(2*b(:,3)));
            Q = (2*b(:,2)-1).*(2-(2*b(:,4)));
            X = sqrt(Eb/2.5) * (I + 1j*Q);

    end
end

%% Recevier
function err = ofdm_receiver(y, h, info_bits, modType, isRep, R)

    %% FFT
    Y = fft(y);

    %% Equalization
    if numel(h) > 1         % frequency selective
        Yeq = Y ./ (h + 1e-12);
    else                    % flat fading
        Yeq = Y / h;
    end

    %% Demapper
    switch modType
        case 'BPSK'
            rx_bits = real(Yeq) > 0;

        case 'QPSK'
            rx_bits = reshape([real(Yeq)>0 imag(Yeq)>0].',1,[]);

        case '16QAM'
            rx_bits = reshape([ ...
                real(Yeq)>0 imag(Yeq)>0 ...
                abs(real(Yeq))<2 abs(imag(Yeq))<2].',1,[]);
    end

    %% De-interleaver
    switch modType
        case 'QPSK'
```

```
                rx_bits = reshape(rx_bits,16,32).';
                rx_bits = rx_bits(:).';

            case '16QAM'
                rx_bits = reshape(rx_bits,32,32).';
                rx_bits = rx_bits(:).';
    end

    %% Decode (Repetition)
    if isRep
        dec = zeros(1,length(info_bits));
        for k = 1:length(info_bits)
            dec(k) = sum(rx_bits((k-1)*R+1:k*R)) > R/2;
        end
        rx_bits = dec;
    else
        rx_bits = rx_bits(1:length(info_bits));
    end

    %% Error count
    rx_bits  = rx_bits(:);
    info_bits = info_bits(:);

    err = sum(rx_bits ~= info_bits);

end

%% Noise
function n = awgn_noise(x, Eb, EbN0)
    N0 = Eb / EbN0;
    n  = sqrt(N0/2) * (randn(size(x)) + 1j*randn(size(x)));
end


%% Plot
function fig = plot_ofdm(EbN0_dB,fU,fC,sU,sC,modType)
    fig = figure;
    semilogy(EbN0_dB,fU,'r-o','LineWidth',1.6); hold on;
    semilogy(EbN0_dB,fC,'b-s','LineWidth',1.6);
    semilogy(EbN0_dB,sU,'k-^','LineWidth',1.6);
    semilogy(EbN0_dB,sC,'m-d','LineWidth',1.6);
```

```
    grid on; grid minor;
    legend('Flat-NoCode','Flat-Rep','Freq-NoCode','Freq-Rep','
       Location','southwest');
    xlabel('E_b/N_0 (dB)');
    ylabel('BER');
    title(['OFDM ' modType ' over Flat & Frequency Selective
       Fading']);
end
```