Cairo University Faculty of Engineering

Electronics and Electrical Communications Department

Mobile Communications Assignment 2: Optical OFDM

Name	I.D.
يوسف حسام سعد الدين	9211396
خالد وليد سمير متولي	9202515

This report is submitted to **Prof. Mai B. Kafafy** as a part of the course work for [ELC4017 – Mobile Communications] course.

Table of Contents

List of Figures

List of Tables

Introduction

Orthogonal Frequency Division Multiplexing (OFDM) is widely used in RF communication due to its robustness against multipath fading. However, in optical communication, especially for visible light and fiber-optic systems, Optical OFDM is gaining traction. Unlike RF OFDM, which transmits complex-valued signals, Optical OFDM operates with real, non-negative signals because light intensity cannot be negative. To adapt OFDM for optical systems, techniques like DC-Biased Optical OFDM (DCO-OFDM), Asymmetrically Clipped Optical OFDM (ACO-OFDM), and Unipolar OFDM (U-OFDM), or Flip-OFDM, have been developed. These schemes ensure the transmission of positive signals while addressing challenges such as DC bias effects and maintaining spectral efficiency. This report compares the performance of DCO-OFDM, ACO-OFDM, and U-OFDM.

DC-Biased Optical OFDM (DCO-OFDM)

DCO-OFDM modulates data on half the subcarriers and adds the other half as complex conjugate symbols of the first half, producing a complex Hermitian-symmetric frequency domain signal to ensure real-valued output after IFFT. A positive DC bias is added to shift the entire signal above zero, making it suitable for optical transmission. While effective, this method introduces additional power overhead due to the bias and suffers from clipping distortion at low bias levels.

Asymmetrically Clipped Optical OFDM (ACO-OFDM)

ACO-OFDM transmits data only on the odd subcarriers and relies on the anti-symmetric nature of the time-domain signal to ensure that clipping the negative values (setting them to zero) does not distort the information-bearing subcarriers. This allows for bias-free and power-efficient implementation, though it utilizes only quarter of the available subcarriers, hence reducing spectral efficiency.

Unipolar OFDM (U-OFDM)

U-OFDM combines two separate OFDM symbols (one positive, one inverted negative) in time to form a unipolar waveform without requiring a DC bias. It avoids clipping distortion by careful signal construction and reconstruction at the receiver, but this comes at the cost of increased signal duration.

Results

Figure 1 illustrates the BER performance of various optical OFDM schemes. For DCO-OFDM, increasing the DC-bias level improves the BER up to a certain threshold, beyond which further increases have negligible effect. ACO-OFDM using odd subcarrier modulation achieves a BER performance comparable to that of U-OFDM. In contrast, ACO-OFDM with even subcarrier modulation results in the poorest performance, with BER values approaching those of a random signal.

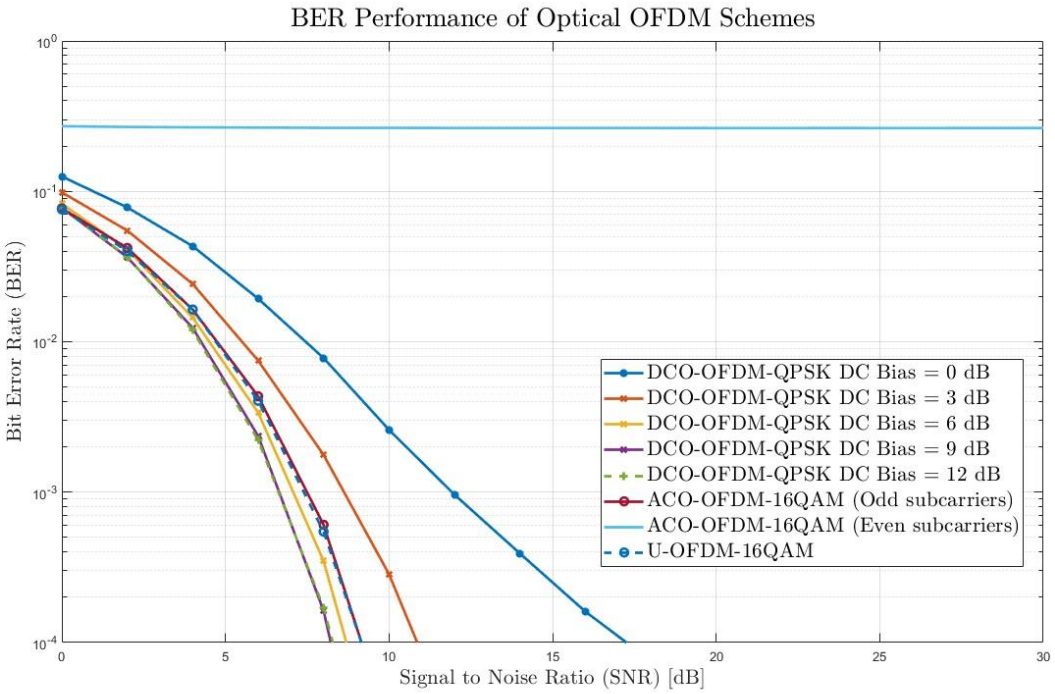

Figure 1: *BER vs SNR for ACO, DCO and U-OFDM.*

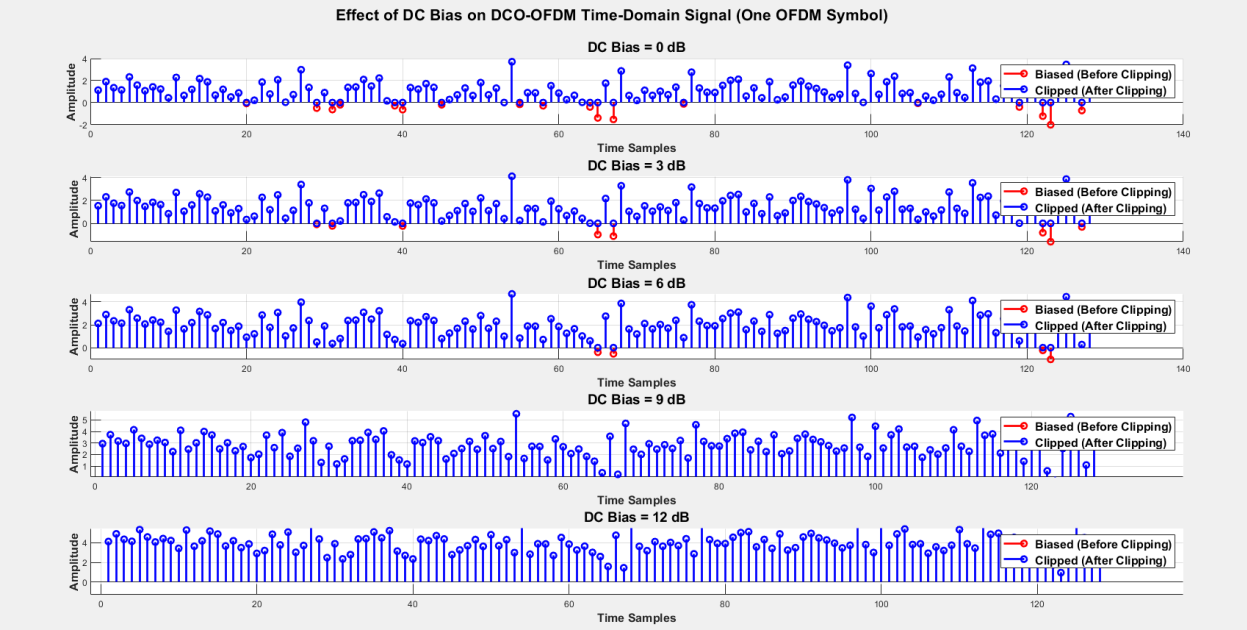

Figure 2: *Effect of DC Bias on DCO-OFDM Time-Domain Signal (One OFDM Symbol).*

How DC-bias value affects the BER of the DCO-OFDM and Why and how to get the RMS of the OFDM symbol.

As illustrated in **Figure 1**, increasing the DC bias leads to a reduction in BER, which aligns with expectations. A higher DC bias shifts more of the OFDM signal above zero, thereby reducing the impact of clipping since less information is lost in the process. When the bias is sufficiently large, the clipping noise becomes negligible. Beyond a certain threshold specifically, once the entire signal lies above zero, further increases in the DC bias no longer yield performance improvements. This behavior is evident in the BER curves for DC bias levels of 9 dB and 12 dB, which are nearly identical. Additionally, **Figure 2** demonstrates this effect on a single OFDM symbol: starting from a bias of 9 dB, the signal becomes almost entirely positive, and hence clipping noise can be ignored so any further increase in the DC bias doesn't have much effect on the BER performance.

The RMS (Root Mean Square) of the OFDM symbol is defined as the RMS value of the signal before adding any DC bias. Since OFDM signals are composed of many subcarriers with random modulated data so we can assume the samples follow a Gaussian distribution and estimate RMS as the standard deviation.

```
% Compute RMS using standard deviation
RMS = std(ofdm_signal(:));
```

Here, ofdm_signal is the time-domain output from the IFFT block, and (:) reshapes the signal into a column vector to compute the standard deviation across all samples.

BER vs the power-efficiency of the DCO-OFDM

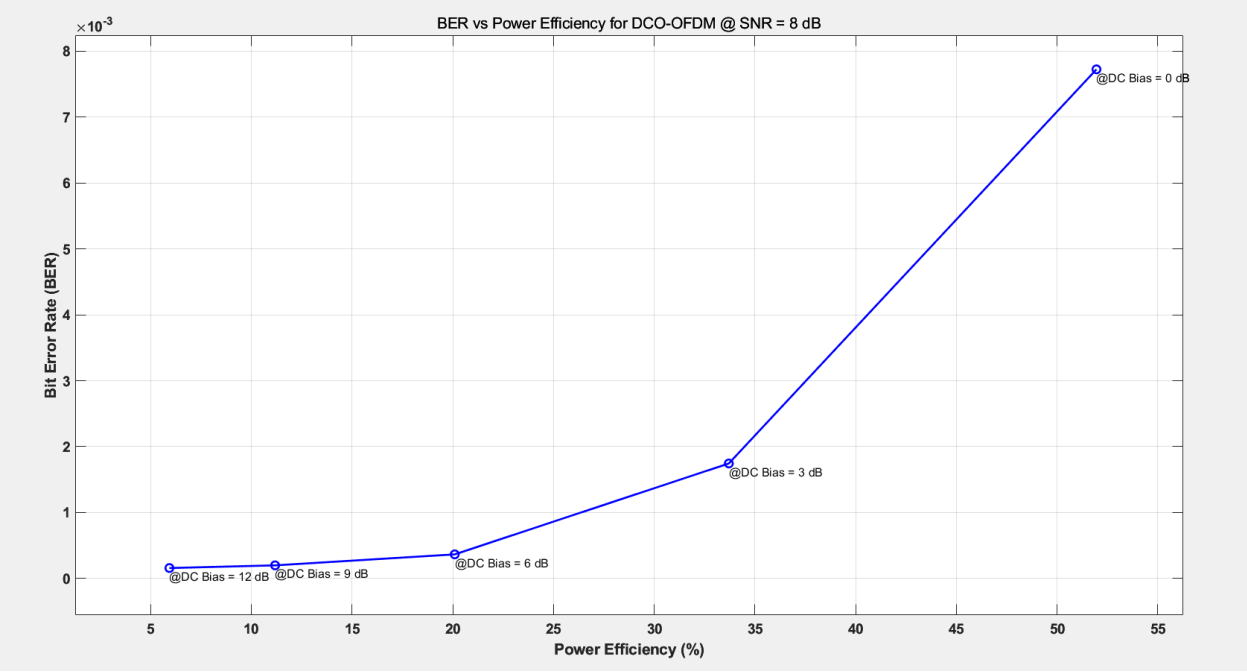

Figure 3: *BER vs Power Efficiency for DCO-OFDM at SNR = 8 dB.*

Based on the created BER vs the power-efficiency graph in the previous point, what DC bias value would we use and why? How much does this value cover of the Gaussian distribution of the samples

From **Figure 3** we can see that as the power efficiency decreases the BER also decreases as we increase the added DC bias hence clipping noise is reduced but this relation saturates as the DC value shifts the whole signal above 0 making clipping noise negligible so adding more DC decreases power efficiency without much improvement in the BER. Based on the figure, we would choose a DC bias value of 9 dB, as it achieves a good balance between BER performance and power efficiency. Beyond this value (e.g., at 9 dB or more), further increases in DC bias result in negligible BER improvement but significantly worse power efficiency.

To analyze how much of the Gaussian distribution this value covers, we note that the OFDM signal before clipping follows a zero-mean Gaussian distribution. A zero-mean Gaussian random variable lies within the range of ±2σ with a 97.8% probability. This implies that if we set the DC bias to 2σ, almost the entire signal is shifted above zero, making clipping nearly negligible. We normalize the OFDM signal such that its RMS (i.e., σ) is 1, a DC bias of 2σ corresponds to 2 units of amplitude, hence

$$\mathbf{20\log_{10}2 \approx 6.02\,dB}$$

In our case, we selected a DC bias of 9 dB, which corresponds to

$$\boldsymbol{Bias = 10^{\frac{9}{20}} \approx 2.82\sigma}$$

This means that our chosen DC bias of 9 dB shifts the signal by about 2.82 standard deviations, which covers more than 99% of the Gaussian distribution. As a result, almost all negative samples are shifted above zero, effectively eliminating clipping noise but at the cost of reduced power efficiency.

Why Choose 16-QAM for ACO-OFDM and U-OFDM?

To ensure a fair comparison between DCO-OFDM, ACO-OFDM, and U-OFDM, it is important that all schemes transmit the same number of bits per OFDM symbol. In DCO-OFDM, half of the subcarriers are used for carrying data, while the other half are reserved to satisfy Hermitian symmetry by placing the complex conjugates of the data symbols. Using QPSK modulation (2 bits per symbol) with 128 subcarriers results in 64 data subcarriers, yielding 128 bits per OFDM symbol.

In ACO-OFDM, only the odd-indexed subcarriers are used for data to maintain the necessary anti-symmetry for proper signal reconstruction. Hermitian symmetry is also applied, so effectively only one-quarter of the subcarriers carry information. To match the bit rate of DCO-OFDM, 16-QAM (which carries 4 bits per symbol) is used, ensuring that the same 128 bits are transmitted per OFDM symbol.

For U-OFDM, although half of the subcarriers can be used for data, the scheme requires sending a second OFDM symbol that contains the flipped and inverted version of the original signal. As a result, two OFDM symbols are transmitted for each block of data. To maintain the same data rate as QPSK-modulated DCO-OFDM, 16-QAM must be used in U-OFDM as well.

Comparison of the BER of the two cases of the ACO-OFDM (when we modulated the odd-subcarriers only and when we modulated the even-subcarriers only) and which case is valid for Optical OFDM modulation, and which is not

From the resulting BER curves, we observe that when odd subcarriers are modulated, the BER performance is consistent with expected ACO-OFDM behavior and shows a valid decreasing trend with increasing SNR. When even subcarriers are modulated, the BER is significantly worse or even random, indicating that the received data cannot be reliably decoded. Only modulating the odd subcarriers is valid for ACO-OFDM in optical communication systems. ACO-OFDM relies on anti-symmetry in the time domain to create a real and non-negative signal after clipping. This anti-symmetry is achieved when only the odd subcarriers are modulated, due to the specific structure of the Hermitian symmetry and the inverse FFT. Because of this structure, any negative value that was clipped can be recovered from its corresponding positive value in the second half of the OFDM symbol. Specifically, the time-domain anti-symmetry guarantees:

$$S_H\left[i+\frac{N_{SC}}{2}\right]=\sum_{n=-\frac{N_{SC}}{2}}^{\frac{N_{SC}}{2}-1} C_n[i] * e^{j2\pi*\frac{ni}{N_{SC}}} * e^{j\pi n} = -S_H[i]$$

This property was used during decoding. In our implementation, we reconstruct the clipped signal at the receiver using:

```
% Use the anti-symmetry to reconstruct the signal
y1 = aco_ofdm_odd_channel_signal(1:half_N, :);
y2 = aco_ofdm_odd_channel_signal(half_N+1:end, :);
ofdm_signal_ACO_odd_reconstructed = [y1 - y2; -(y1 - y2)];
```

However, if even subcarriers are modulated, the resulting time-domain signal does not exhibit the necessary anti-symmetry, so clipping at zero introduces distortion directly onto the modulated subcarriers, severely degrading the signal and making reliable demodulation impossible.

Comparison of U-OFDM and ACO-OFDM in terms of power efficiency and spectral efficiency

Spectral Efficiency

In Both schemes we use only half the subcarriers to achieve Hermitian symmetry. ACO-OFDM uses only the odd-indexed subcarriers for data transmission. This is done to preserve the anti-symmetry in the time domain, which allows for unipolar signal generation after clipping. Hence overall quarter of the total subcarriers are used.

U-OFDM, on the other hand, uses all usable subcarriers in a single OFDM symbol (after constructing Hermitian symmetry). However, it achieves unipolarity by transmitting two time-domain OFDM symbols, one for the positive part of the signal, and one for the negative part (flipped).

Hence, ACO-OFDM sacrifices frequency to maintain signal unipolarity while U-OFDM sacrifices time (doubling duration) to achieve the same spectral efficiency given by:

$$SE = \frac{1}{2} * \frac{1}{2} * \#bits/symbol$$

The first 1/2 accounts for Hermitian symmetry; the second 1/2 arises from using only odd subcarriers (ACO) or duplicating the symbol in time (U-OFDM).

Power Efficiency

Both ACO-OFDM and U-OFDM are more power-efficient because they do not require DC bias unlike DCO-OFDM. And they both have similar power efficiency. However, ACO-OFDM is more energy-efficient than U-OFDM. This is because U-OFDM transmits two OFDM symbols per data frame (one for the original signal and another for its flipped negative counterpart) effectively doubling the transmission time for the same amount of data.

***Table 1**: Summary of comparison between various parameters of ACO-OFDM and U-OFDM.*

Parameters	**ACO-OFDM**	**U-OFDM**
Spectral efficiency	$SE = \frac{1}{2} * \frac{1}{2} * \#bits/symbol$	$SE = \frac{1}{2} * \frac{1}{2} * \#bits/symbol$
Power efficiency	Same	Same
BER at same SNR	Same	Same
Energy efficiency	More	Less

MATLAB code

```
%% Define parameters
SNR_dB = 0:2:30;                                % SNR range in dB
N = 128;                                        % Subcarriers (same N for FFT/IFFT size)
half_N = N/2;                                   % Half subcarriers
numSymbols = 32*32*2;                           % Number of OFDM symbols
numBits = 2*2*(N/2 - 1)*numSymbols;    % number of bits to be multiple of subcarriers
% Generate the binary data
binary_data = randi([0 1], numBits, 1);

%% DCO-OFDM (DC-biased Optical OFDM)
fprintf('************************************************************** \n');
fprintf('DCO-OFDM Simulation started... \n');
bias_dB = 0:3:12;                                          % DC Bias range in dB
BER_all_biases = zeros(length(bias_dB), length(SNR_dB)); % Store BER for all biases
power_efficiency = zeros(length(bias_dB), 1);
%% Transmitter
% Generate QPSK symbols
dataSymbols = QPSK_Gray_Mapper(binary_data);
% Reshape for OFDM symbols
dataSymbols = reshape(dataSymbols, N/2 - 1, []);
% Construct the Hermitian symmetric OFDM frame
X = zeros(N, size(dataSymbols, 2));
% U_k for k=1 to N/2-1
X(2:N/2, :) = dataSymbols;
X(1, :) = 0;       % 0 DC
X(N/2 + 1, :) = 0; % 0 Nyquist
% Define indices for k = 2 to N/2 - 1
data_indices = 2:N/2;
% Their Hermitian counterparts: N - k + 2
hermitian_indices = N - data_indices + 2;
% Apply symmetry
X(hermitian_indices, :) = conj(X(data_indices, :));
% Apply IFFT to get real-valued time domain signal
ofdm_signal = ifft(X, N);
% Normalize the OFDM Signal
ofdm_signal = ofdm_signal / sqrt(mean(abs(ofdm_signal(:)).^2));
% Compute RMS using standard deviation
RMS = std(ofdm_signal(:));
% Compute useful power from original time-domain OFDM signal (unbiased)
P_useful = mean(abs(ofdm_signal(:)).^2);
for i = 1:length(bias_dB)
    BER_values = zeros(1, length(SNR_dB)); % Store BER for each SNR value
    dc_bias = (10^(bias_dB(i)/20))*RMS;
    OFDM_Signal_Biased = ofdm_signal + dc_bias;
    % Clipping step (this will introduce clipping noise If the DC
    % bias is high enough, the clipping noise can be neglected)
    OFDM_Signal_Biased_clipped = max(OFDM_Signal_Biased, 0);
    % Compute total power after DC bias and clipping
    P_total = mean(abs(OFDM_Signal_Biased_clipped(:)).^2);
    % Calculate power efficiency
    power_efficiency(i) = (P_useful / P_total)*100;
```

```
    fprintf('============================================== \n');
    fprintf('Iteration number %d \n',i);
    fprintf('bias_dB = %.4f \n',bias_dB(i));
    fprintf('RMS = %.4f \n', RMS);
    fprintf('DC Bias = %.4f \n',dc_bias);

    for j = 1:length(SNR_dB)
        %% Channel
        % Convert SNR from dB to linear
        SNR_linear = 10^(SNR_dB(j)/10);
        N0 = 1/SNR_linear;
        % Variance of the Gaussian noise
        variance = N0/2;
        % Generate the noise vector
        noise = sqrt(variance) * randn(size(OFDM_Signal_Biased_clipped));
        % Assuming AWGN channel so noise is simply added
        ofdm_channel_signal = OFDM_Signal_Biased_clipped + noise;

        %% Receiver
        OFDM_Signal_Received  = ofdm_channel_signal - dc_bias;
        Y = fft(OFDM_Signal_Received, N); % Convert to frequency domain
        Y_data = Y(2:N/2, :);  % Keep only the first half (excluding DC and Nyquist)
        binary_data_received = QPSK_Gray_Demapper(Y_data(:));
        BER_all_biases(i, j) = sum(binary_data ~= binary_data_received) /
length(binary_data_received);
    end
end
% Plots colors
colors = lines(8);
figure;
semilogy(SNR_dB, BER_all_biases(1,:), '-x', 'LineWidth', 2, 'DisplayName', 'DCO-OFDM-
QPSK DC Bias = 0 dB', 'Color', colors(1, :));
hold on;
semilogy(SNR_dB, BER_all_biases(2,:), '-x', 'LineWidth', 2, 'DisplayName', 'DCO-OFDM-
QPSK DC Bias = 3 dB', 'Color', colors(2, :));
hold on;
semilogy(SNR_dB, BER_all_biases(3,:), '-x', 'LineWidth', 2, 'DisplayName', 'DCO-OFDM-
QPSK DC Bias = 6 dB', 'Color', colors(3, :));
hold on;
semilogy(SNR_dB, BER_all_biases(4,:), '-x', 'LineWidth', 2, 'DisplayName', 'DCO-OFDM-
QPSK DC Bias = 9 dB', 'Color', colors(4, :));
hold on;
semilogy(SNR_dB, BER_all_biases(5,:), '--x', 'LineWidth', 2, 'DisplayName', 'DCO-
OFDM-QPSK DC Bias = 12 dB', 'Color', colors(5, :));
hold on;

%% ACO-OFDM (Asymmetrically Clipped Optical OFDM)
fprintf('************************************************************ \n');
fprintf('ACO-OFDM Simulation started... \n');
BER_values_odd = zeros(length(SNR_dB), 1);  % Store BER for each SNR
BER_values_even = zeros(length(SNR_dB), 1); % Store BER for each SNR
for j = 1:length(SNR_dB)
    %% Transmitter
    % Generate 16-QAM symbols for ACO-OFDM
    dataSymbols_ACO = QAM16_Mapper(binary_data);
```

```
    % Reshape for OFDM symbols (ensuring correct size)
    dataSymbols_ACO = reshape(dataSymbols_ACO, (N/4), []);
    % Initialize OFDM frame with zeros
    X_ACO_odd = zeros(N, size(dataSymbols_ACO, 2));
    X_ACO_even= zeros(N, size(dataSymbols_ACO, 2));
    % Note: MATLAB uses 1-based indexing, while in comm subcarrier indexing starts
from 0.
    % So, comm subcarrier index 0 (DC) → MATLAB index 1
    %     comm subcarrier index 1 (1st odd) → MATLAB index 2
    %     comm subcarrier index 2 (1st even) → MATLAB index 3
    % This means to modulate odd subcarriers (in comm terms), we use even MATLAB
indices (2, 4, 6, ...)
    % and for even comm subcarriers, we use odd MATLAB indices (3, 5, 7, ...).
    odd_indices = 2:2:(N/2);
    even_indices = 1:2:(N/2);
    % Load data on odd subcarriers only
    X_ACO_odd(odd_indices, :) = dataSymbols_ACO;
    % Load data onto even subcarriers
    X_ACO_even(even_indices, :) = dataSymbols_ACO;
    % Hermitian symmetry for real-valued IFFT
    hermitian_odd = N - odd_indices + 2;
    X_ACO_odd(hermitian_odd, :) = conj(X_ACO_odd(odd_indices, :));
    %hermitian_even = N - even_indices + 2;
    %X_ACO_even(hermitian_even, :) = conj(X_ACO_even(even_indices, :));
    % IFFT to generate real-valued time-domain signal
    ofdm_signal_ACO_odd = ifft(X_ACO_odd, N);
    ofdm_signal_ACO_even = ifft(X_ACO_even, N, 'symmetric');
    % Clipping at zero
    ofdm_signal_ACO_odd_clipped = max(ofdm_signal_ACO_odd, 0);
    ofdm_signal_ACO_even_clipped = max(ofdm_signal_ACO_even, 0);
    %% Channel
    SNR_linear = 10^(SNR_dB(j)/10);
    % Noise power
    N0 = 1/SNR_linear;
    % Variance of the Gaussian noise
    variance = 2.5*N0/2;
    % Generate the noise vector
    n_dsc = length(odd_indices);
    n = sqrt(n_dsc/(N*N))*sqrt(variance) * randn(size(ofdm_signal_ACO_odd_clipped));
    % Assuming AWGN channel so noise is simply added
    aco_ofdm_odd_channel_signal = ofdm_signal_ACO_odd_clipped + n;
    aco_ofdm_even_channel_signal = ofdm_signal_ACO_even_clipped + n;
    %% Receiver
    % Use the anti-symmetry to reconstruct the signal (modulated on odd-subcarriers)
    y1 = aco_ofdm_odd_channel_signal(1:half_N, :);
    y2 = aco_ofdm_odd_channel_signal(half_N+1:end, :);
    ofdm_signal_ACO_odd_reconstructed = [y1 - y2; -(y1 - y2)];
    % Convert to frequency domain
    Y_odd = fft(ofdm_signal_ACO_odd_reconstructed, N);
    Y_even = fft(aco_ofdm_even_channel_signal, N);
    % Obtain the received symbols
    received_symbols_odd = Y_odd(odd_indices, :);
    received_symbols_even = Y_even(even_indices, :);
    % Reshape to a column vector for demodulation
    received_symbols_odd = received_symbols_odd(:);
```

```
    received_symbols_even = received_symbols_even(:);
    % QAM Demapping
    received_bits_odd = QAM16_Demapper(received_symbols_odd);
    received_bits_even = QAM16_Demapper(received_symbols_even);
    BER_values_odd(j) = sum(binary_data ~= received_bits_odd) / length(binary_data);
    BER_values_even(j) = sum(binary_data ~= received_bits_even)/ length(binary_data);
end
% Plot the BER curves
semilogy(SNR_dB, BER_values_odd, '-o', 'LineWidth', 2, 'DisplayName', 'ACO-OFDM-16QAM
(Odd subcarriers)', 'Color', colors(7, :));
hold on;
semilogy(SNR_dB, BER_values_even, '-o', 'LineWidth', 2, 'DisplayName', 'ACO-OFDM-
16QAM (Even subcarriers)', 'Color', colors(6, :));
hold on;
%% U-OFDM (Unipolar OFDM) (Flip-OFDM)
fprintf('************************************************************ \n');
fprintf('U-OFDM Simulation started... \n');
BER_values_UOFDM = zeros(length(SNR_dB), 1); % Store BER for each SNR
for j = 1:length(SNR_dB)
    %% Transmitter
    % Generate 16-QAM symbols for ACO-OFDM
    dataSymbols_U = QAM16_Mapper(binary_data);
    % Reshape for OFDM symbols (ensuring correct size)
    dataSymbols_U = reshape(dataSymbols_U, N/2 - 1, []);
    % Construct the Hermitian symmetric OFDM frame
    X = zeros(N, size(dataSymbols_U, 2));
    % U_k for k=1 to N/2-1
    X(2:N/2, :) = dataSymbols_U;
    X(1, :) = 0;       % 0 DC
    X(N/2 + 1, :) = 0; % 0 Nyquist
    % Define indices for k = 2 to N/2 - 1
    data_indices = 2:N/2;
    % Their Hermitian counterparts: N - k + 2
    hermitian_indices = N - data_indices + 2;
    % Apply symmetry
    X(hermitian_indices, :) = conj(X(data_indices, :));
    % Apply IFFT to get real-valued time domain signal
    ofdm_signal = ifft(X, N);
    x_pos_tx = max(ofdm_signal, 0); % Positive part
    x_neg = min(ofdm_signal, 0);    % Negative part (still negative)
    x_neg_tx = -x_neg;              % Flip negative to positive for transmission
    %% Channel
    SNR_linear = 10^(SNR_dB(j)/10);
    % Noise power
    N0 = 1/SNR_linear;
    % Variance of the Gaussian noise
    variance = 2.5*N0/2;
    n_dsc = length(data_indices);
    % Generate 2 noise vectors because we are sending 2 frames
    n1 = sqrt(n_dsc/(N*N))*sqrt(variance).*randn(size(x_pos_tx));
    n2 = sqrt(n_dsc/(N*N))*sqrt(variance).*randn(size(x_neg_tx));
    % Assuming AWGN channel so noise is simply added
    u_ofdm_frame_1 = x_pos_tx + n1;
    u_ofdm_frame_2 = x_neg_tx + n2;
```

```
%% Receiver
    y_pos = u_ofdm_frame_1;          % First half (original positive)
    y_neg_flipped = u_ofdm_frame_2;  % Second half (flipped negative)
    % Re-flip the negative part to its original polarity
    y_neg = -y_neg_flipped;
    % Reconstruct the full bipolar OFDM signal
    y_reconstructed = y_pos + y_neg;
    % Step 4: FFT to go to frequency domain
    Y = fft(y_reconstructed, N);
    % Reshape to a column vector for demodulation
    received_symbols_u = Y(2:N/2, :);
    received_symbols_u = received_symbols_u(:);
    % QAM Demapping
    received_bits_u = QAM16_Demapper(received_symbols_u);
    BER_values_UOFDM(j) = sum(binary_data ~= received_bits_u)/ length(binary_data);
end
% Plot the BER curves
semilogy(SNR_dB, BER_values_UOFDM, '-.o', 'LineWidth', 2, 'DisplayName', 'U-OFDM-
16QAM', 'Color', colors(8, :));
grid on;
% Add plot formatting
xlabel('Signal to Noise Ratio (SNR) dB');
ylabel('Bit Error Rate (BER)');
title('BER Performance of Optical OFDM Schemes (DCO-OFDM, ACO-OFDM and U-OFDM)');
legend('Location', 'southwest');
ylim([1e-4 1]);

% DCO Power efficiency plot
target_snr_idx = find(SNR_dB == 8);
BER_vs_efficiency = BER_all_biases(:, target_snr_idx);
figure;
plot(power_efficiency, BER_vs_efficiency, '-x', 'LineWidth', 2);
xlabel('Power Efficiency (%)');
ylabel('Bit Error Rate (BER)');
title('BER vs Power Efficiency for DCO-OFDM');
grid on;
set(gca, 'FontSize', 12, 'FontWeight', 'bold');
grid on;
box on;

% Add DC bias labels to each marker
for i = 1:length(bias_dB)
    text(power_efficiency(i), BER_vs_efficiency(i), ...
        sprintf('@DC Bias = %.0f dB', bias_dB(i)), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'left', ...
        'FontSize', 12);
end
fprintf('************************************************************ \n');
fprintf('All Simulations finished succesfully. \n');
```

```
%% Required Fuctions
% 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.
% Notes:
%   - The QPSK constellation table used in this function is defined as:
%     QPSK_Table = [-1-1i, -1+1i, 1-1i, 1+1i]
function QPSK_Data_Mapped = QPSK_Gray_Mapper(binary_data)
% Reshape the binary data to have each 2 consecutive bits as 1 symbol
QPSK_Table = [-1-1i, -1+1i, 1-1i, 1+1i]/sqrt(2);
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

% 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.
% Notes:
%   - The QPSK constellation table used in this function is defined as:
%     QPSK_Table = [-1-1i, -1+1i, 1-1i, 1+1i];
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]/sqrt(2);
% 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
```

```
% 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
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

% 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
```