Digital Communication

**Lab Assignment: 9**

**Name: Vinit Mehta**

**Student ID: 202101477**

**Lab Group: 6**

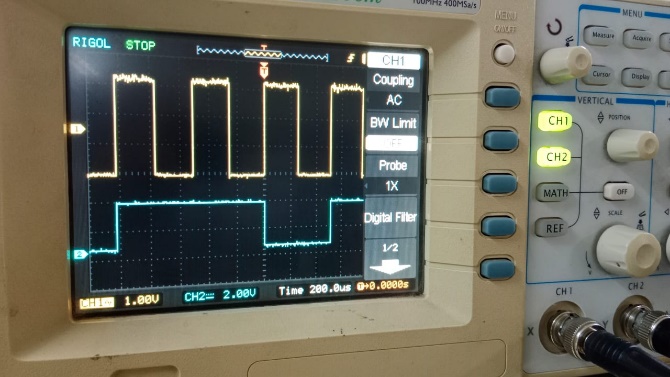
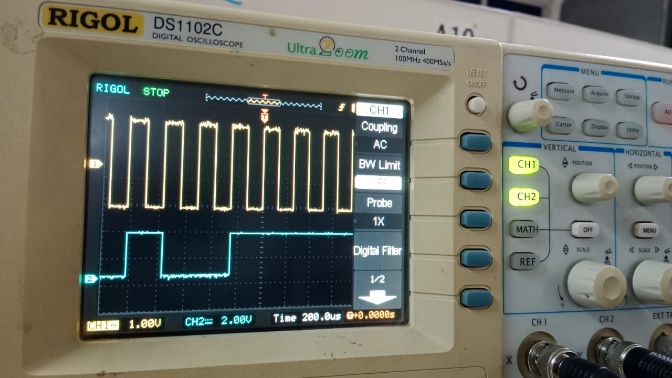
**Experiment 1:**

**Exercise: 1:**

Data Bit: 32-bit Data Bit: 32-bit

Data Rate: 2KHz Data Rate: 4KHz  
CH1: TP1 CH1: TP1

CH2: TP2 CH2:TP2

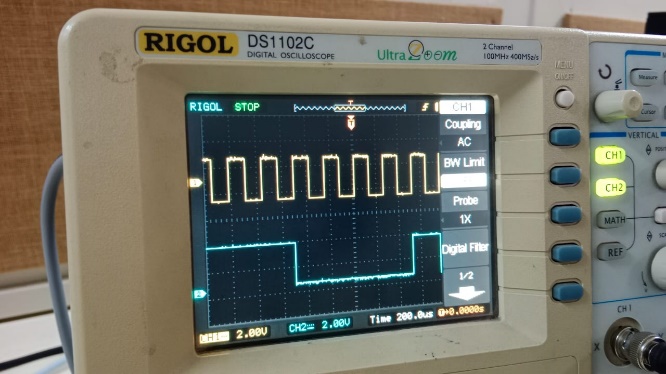
 

Data Bit: 16-bit

Data Rate: 4KHz

CH1: TP1

CH2: TP2



**Exercise: 2:**

Data Bit: 8-bit

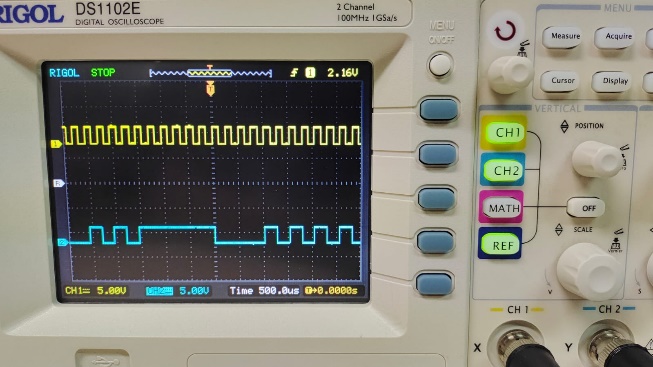
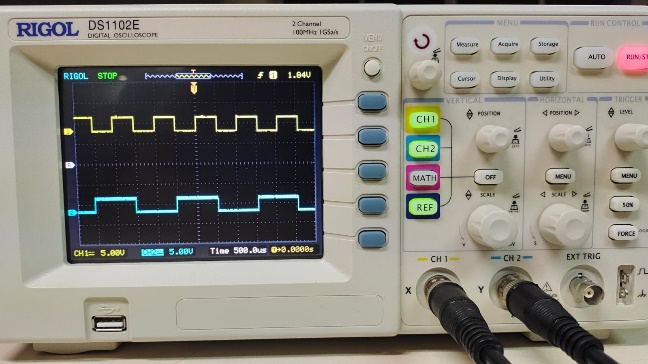
Data Rate: 4KHz

CH1: TP1

CH2: TP2

CH3:TP3

CH4:TP4

Data Bit: 32-bit

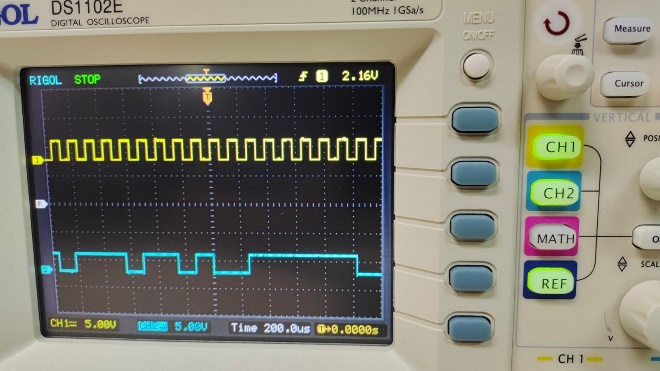
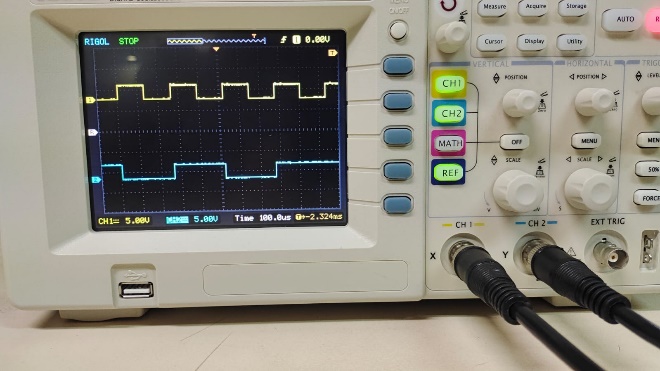
Data Rate: 8KHz

CH1: TP1

CH2: TP2

CH3:TP3

CH4:TP4

Data Bit: 64-bit

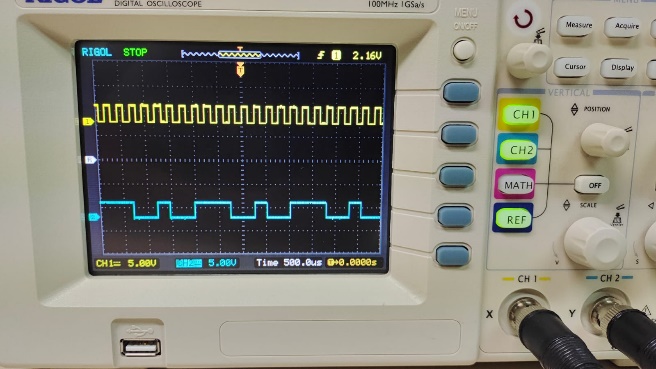
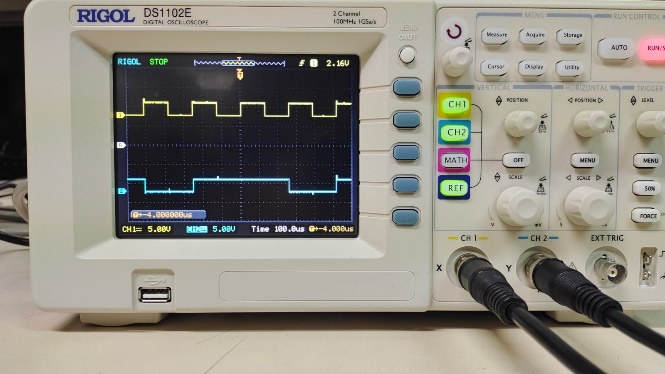
Data Rate: 4KHz

CH1: TP1

CH2: TP2

CH3:TP3

CH4:TP4

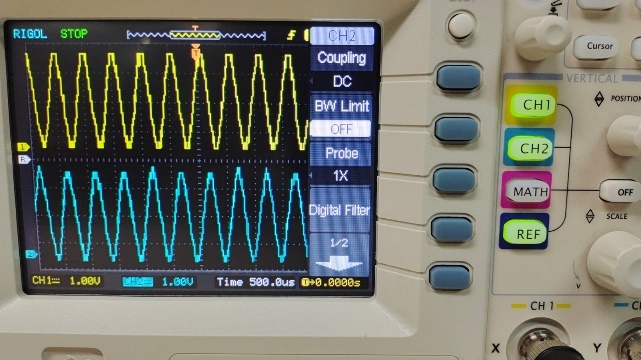
**Exercise: 3:**

Data Bit: 8-bit

Data Rate: 2KHz

CH1: TP5

CH2: TP6

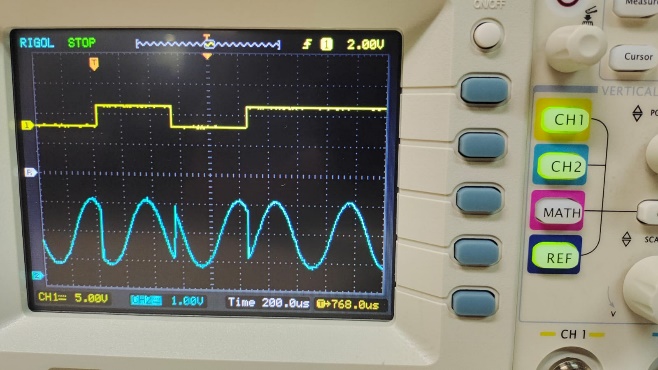
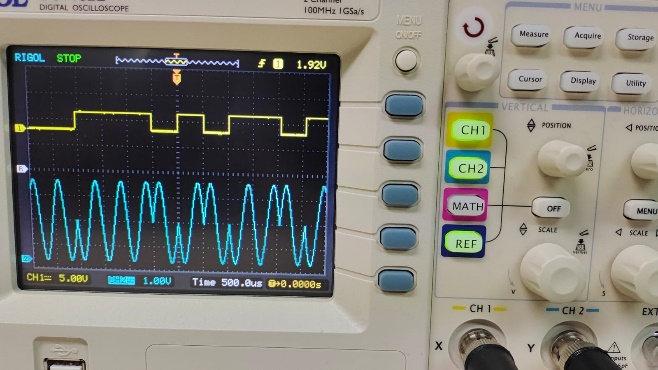


**Exercise: 4:**

Data Bit: 32-bit Data Bit: 32-bit

Data Rate: 4KHz Data Rate: 4KHz  
CH1: TP3 CH1: TP4

CH2: TP7 CH2:TP8

**** 

**Exercise: 5:**

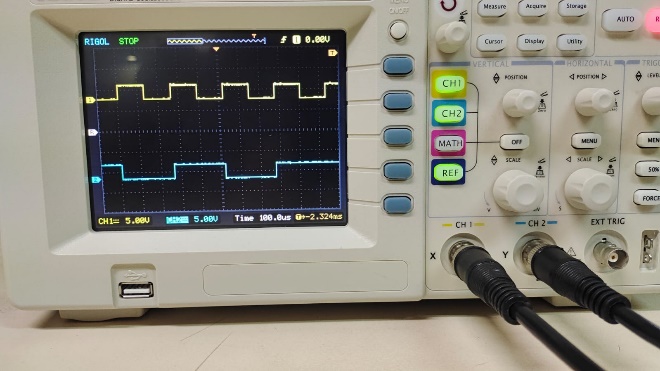
Data Bit: 8-bit

Data Rate: 16KHz

CH1: TP3

CH2: TP4

CH3:TP9

 ****

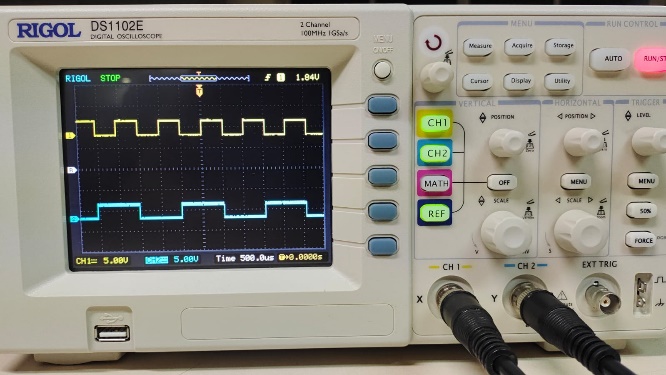
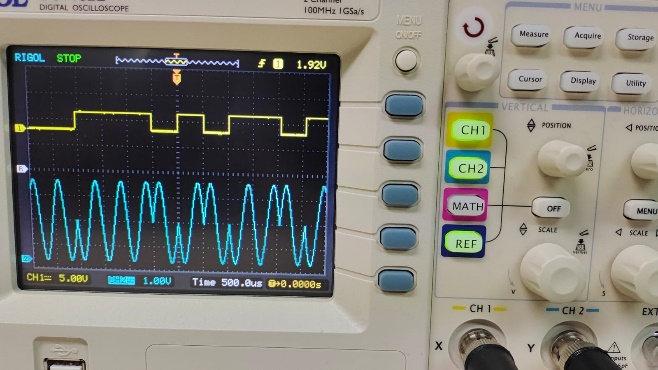
Data Bit: 32-bit

Data Rate: 4KHz

CH1: TP3

CH2: TP4

CH3:TP9

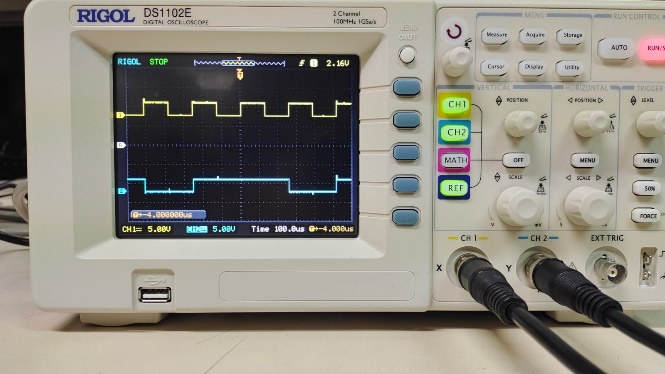
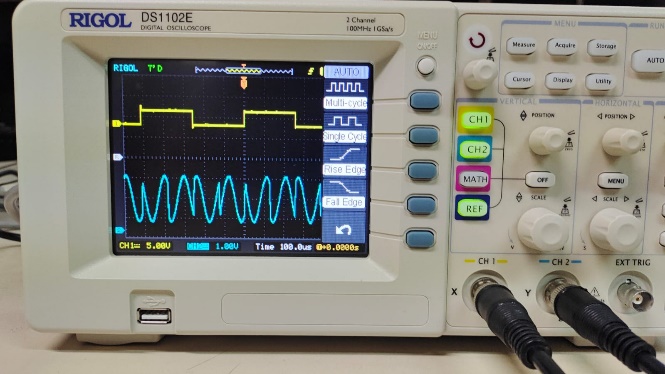
Data Bit: 32-bit

Data Rate: 16KHz

CH1: TP3

CH2: TP4

CH3:TP9

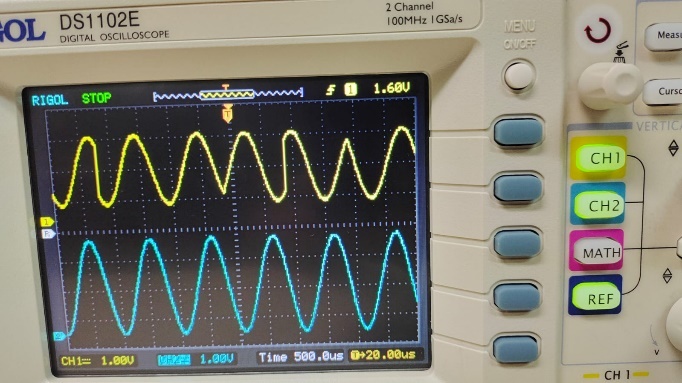
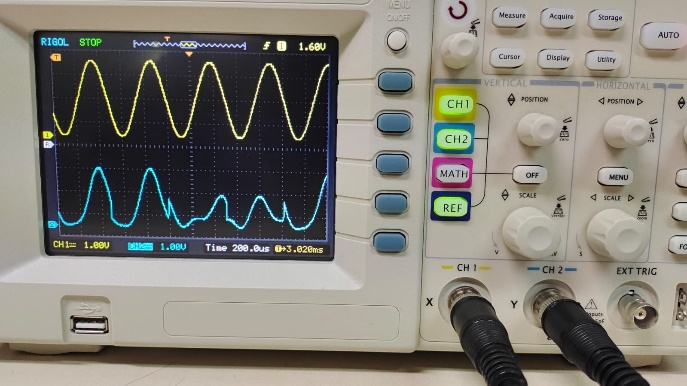
 

**Exercise: 6:**

Data Bit: 8-bit Data Bit: 16-bit

Data Rate: 2KHz Data Rate: 2KHz  
CH1: TP9 CH1: TP9

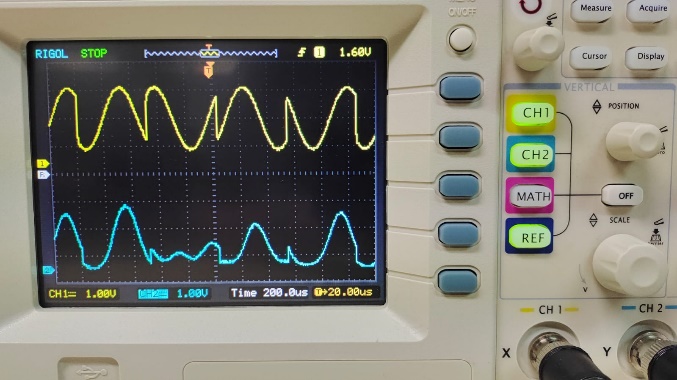
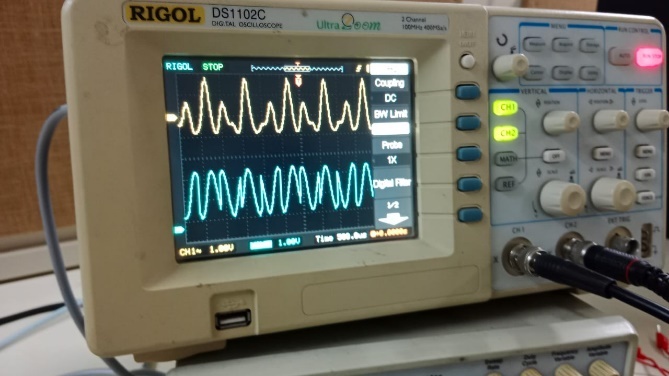
CH2: TP10 CH2:TP11

Data Bit: 8-bit Data Bit: 16-bit

Data Rate: 4KHz Data Rate: 2KHz  
CH1: TP9 CH1: TP9

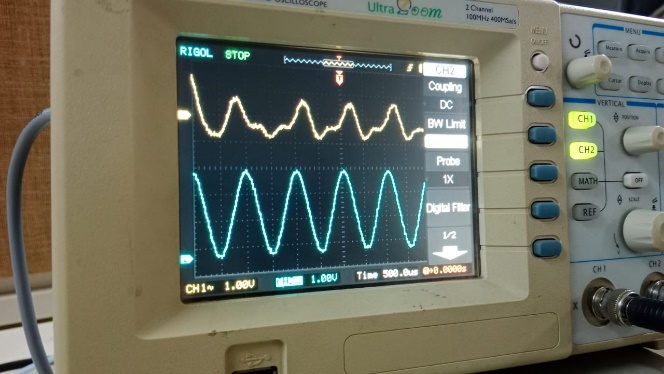
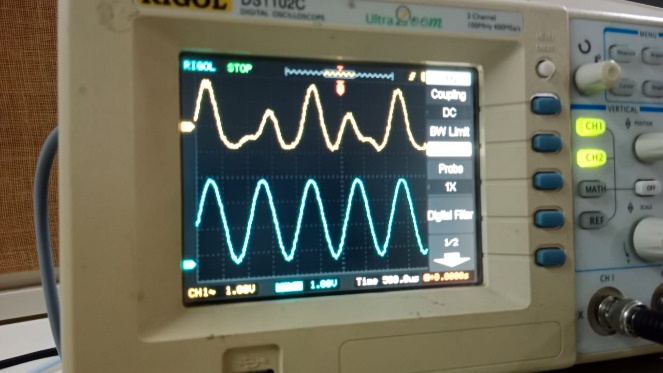
CH2: TP12 CH2:TP12

Data Bit: 8-bit Data Bit: 16-bit

Data Rate: 2KHz Data Rate: 2KHz  
CH1: TP10 CH1: TP10

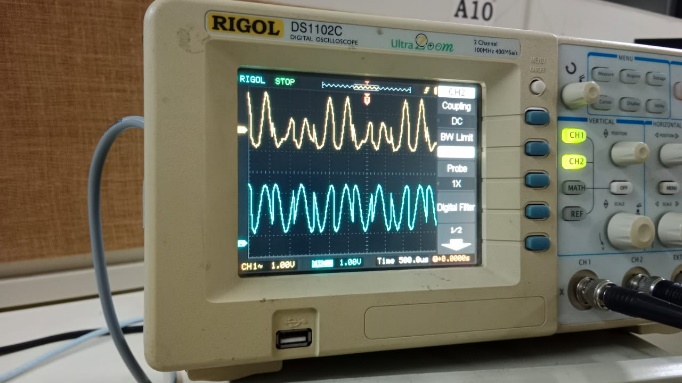
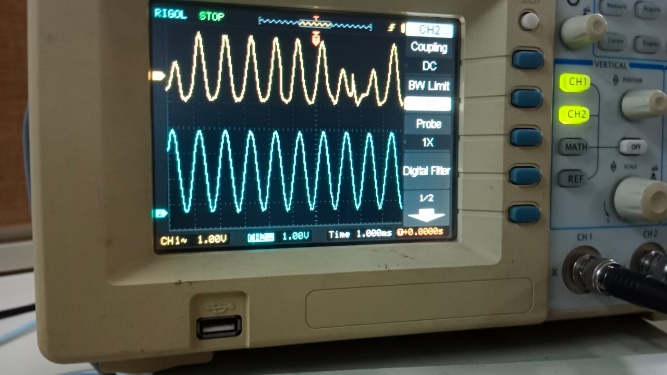
CH2: TP12 CH2:TP12

Data Bit: 8-bit Data Bit: 16-bit

Data Rate: 4KHz Data Rate: 2KHz  
CH1: TP9 CH1: TP9

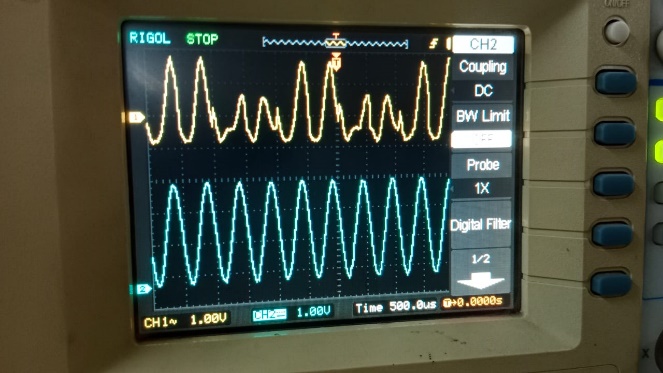
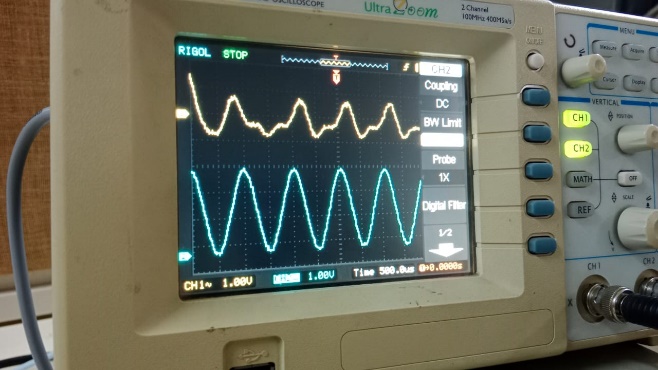
CH2: TP13 CH2:TP18

Data Bit: 8-bit Data Bit: 16-bit

Data Rate: 4KHz Data Rate: 2KHz  
CH1: TP11 CH1: TP11

CH2: TP13 CH2:TP13

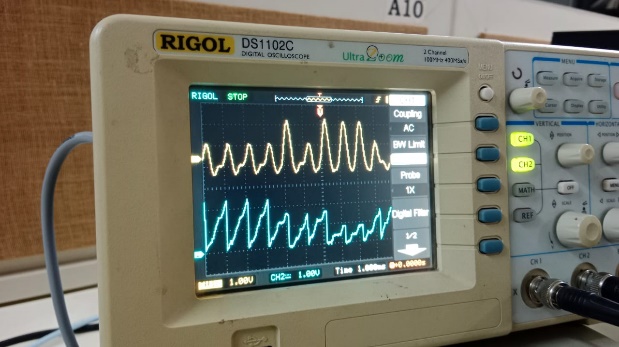
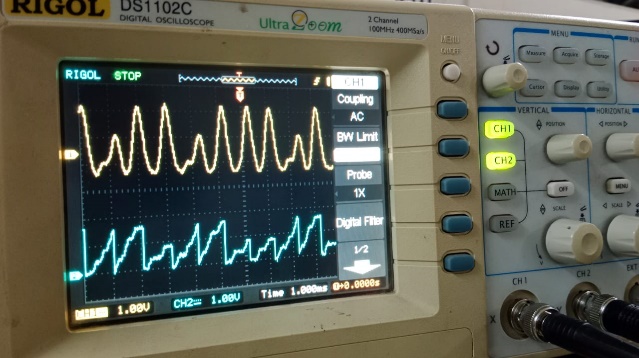
 

**Exercise: 7:**

Data Bit: 32-bit Data Bit: 64-bit

Data Rate: 2KHz Data Rate: 2KHz  
CH1: TP12 CH1: TP12

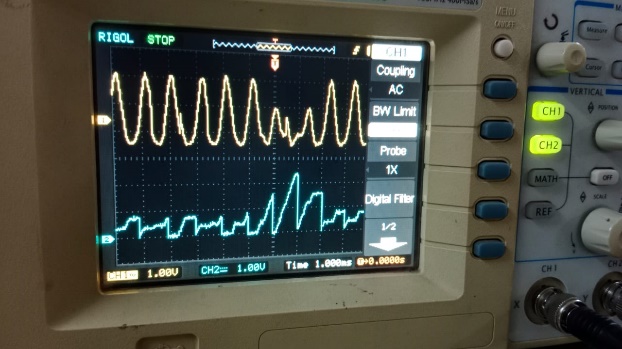
CH2: TP14 CH2:TP14

Data Bit: 32-bit Data Bit: 64-bit

Data Rate: 2KHz Data Rate: 2KHz  
CH1: TP13 CH1: TP13

CH2: TP15 CH2:TP15

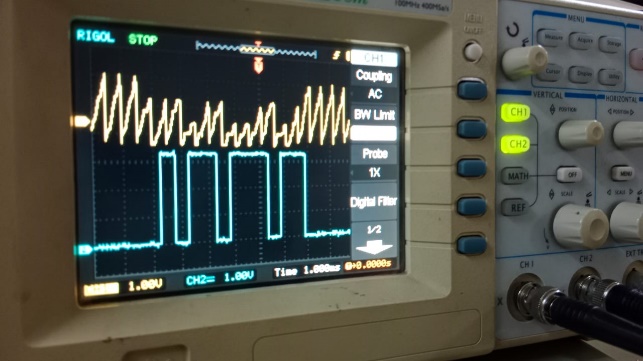
 

**Exercise: 8:**

Data Bit: 32-bit Data Bit: 64-bit

Data Rate: 2KHz Data Rate: 2KHz  
CH1: TP14 CH1: TP14

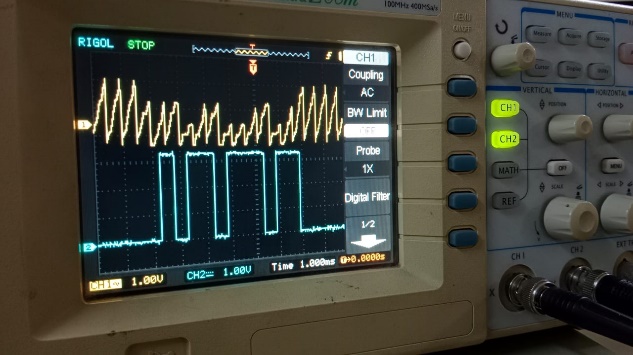
CH2: TP16 CH2:TP16

Data Bit: 64-bit Data Bit: 32-bit

Data Rate: 4KHz Data Rate: 4KHz  
CH1: TP14 CH1: TP15

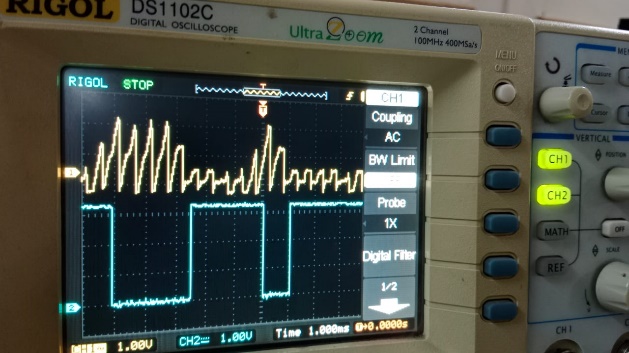
CH2: TP16 CH2:TP17

Data Bit: 64-bit Data Bit: 64-bit

Data Rate: 2KHz Data Rate: 4KHz  
CH1: TP15 CH1: TP15

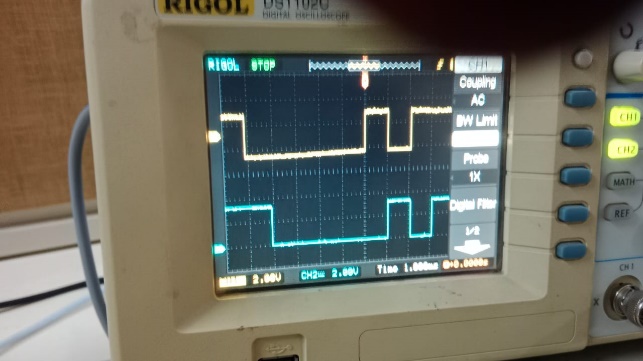
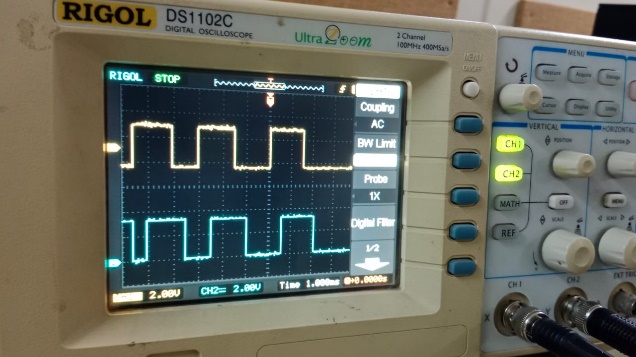
CH2: TP17 CH2:TP17

Data Bit: 32-bit Data Bit: 64-bit

Data Rate: 2KHz Data Rate: 4KHz  
CH1: TP3 CH1: TP3

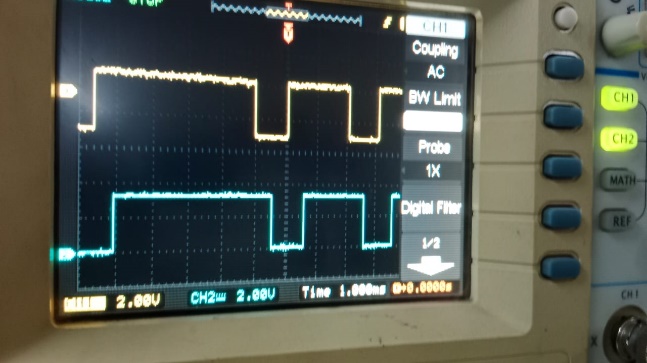
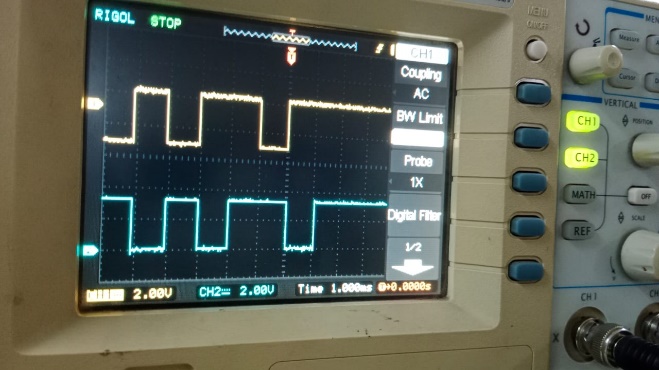
CH2: TP16 CH2:TP16

Data Bit: 32-bit Data Bit: 64-bit

Data Rate: 2KHz Data Rate: 4KHz  
CH1: TP4 CH1: TP4

CH2: TP17 CH2:TP17

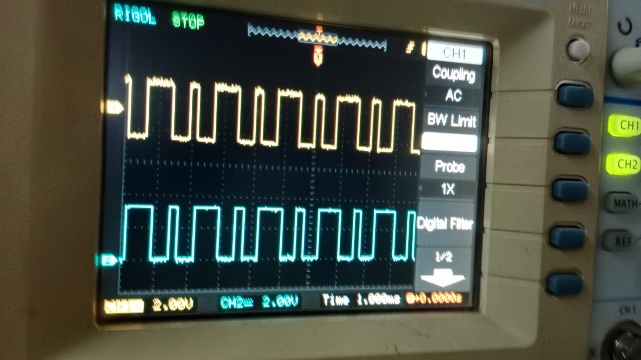
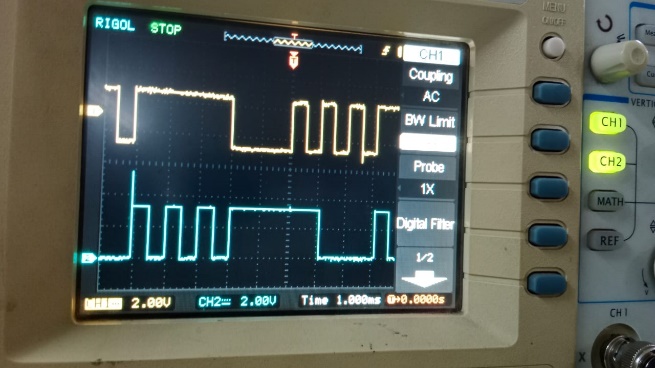
 

**Exercise: 9:**

Data Bit: 8-bit Data Bit: 16-bit

Data Rate: 4KHz Data Rate: 2KHz  
CH1: TP2 CH1: TP2

CH2: TP18 CH2:TP18

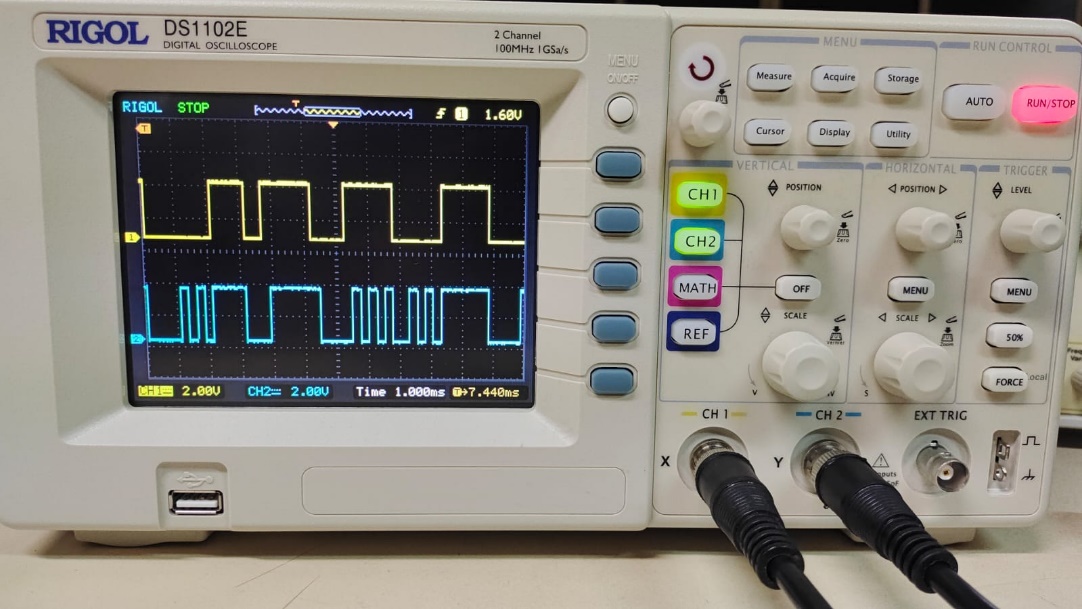
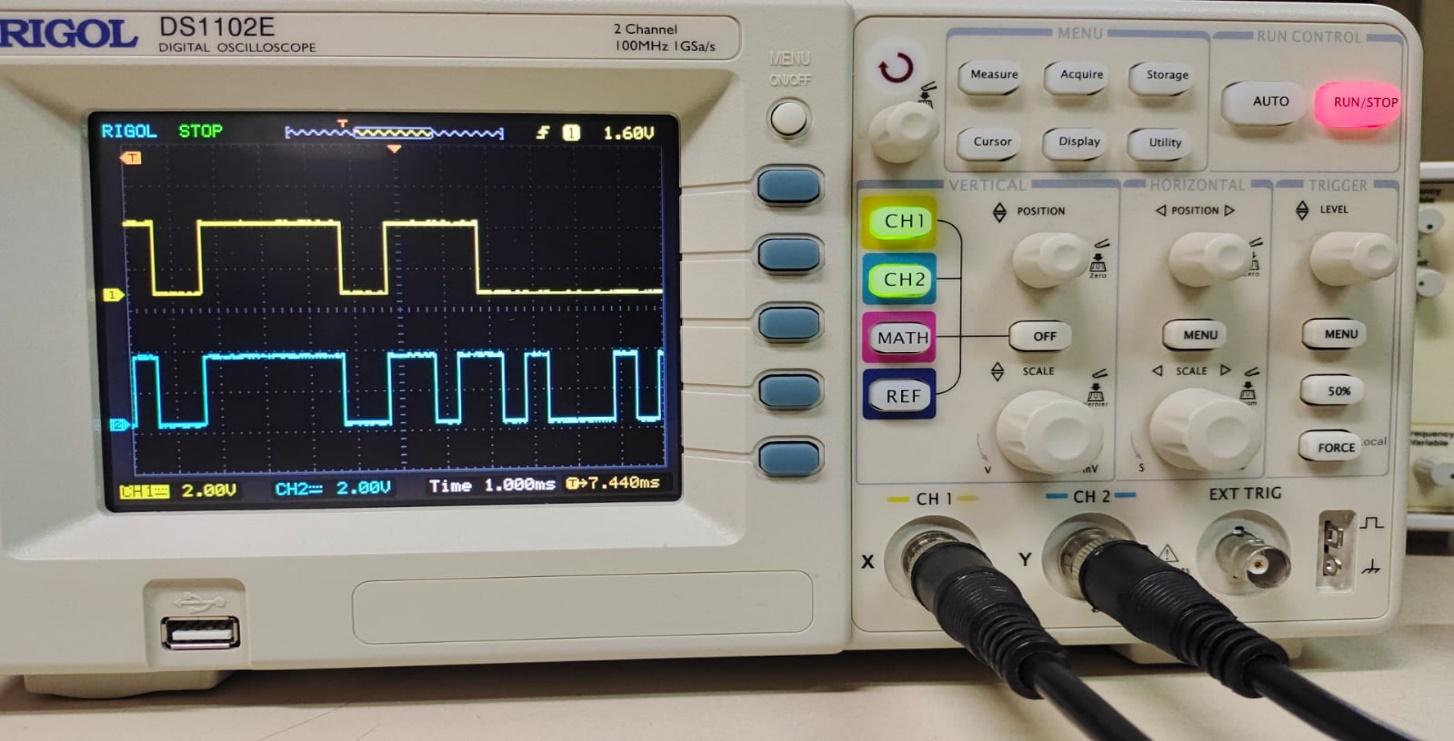
Data Bit: 32-bit

Data Rate: 2KHz

CH1: TP16

CH2: TP17

CH3:TP18

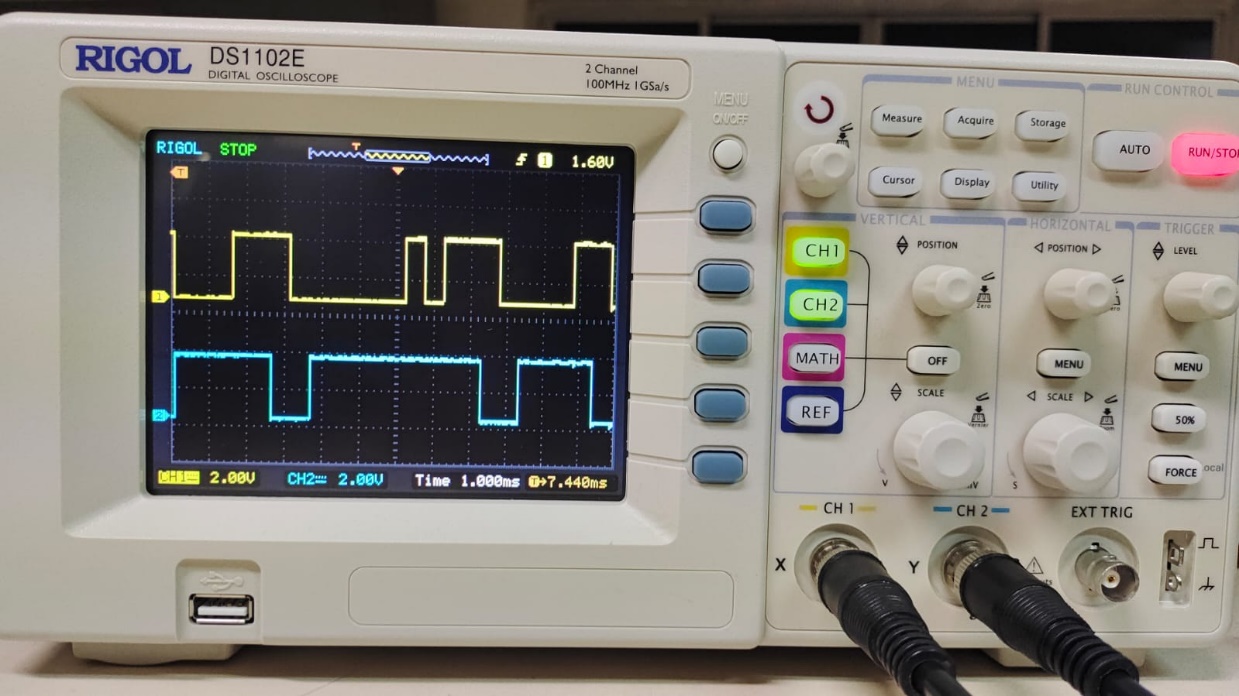
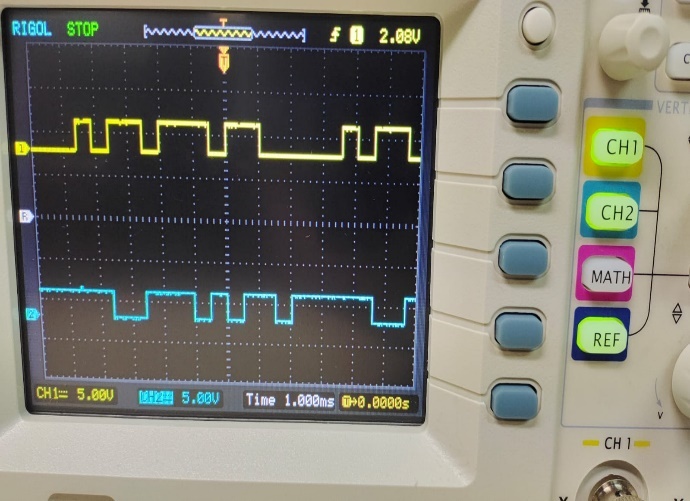
Data Bit: 64-bit

Data Rate: 4KHz

CH1: TP16

CH2: TP17

CH3:TP18

**Experiment 2:**

**Problem 5.1:**

% Define parameters

A = 1; % Amplitude

Tb = 1; % Bit interval duration

numSamples = 10; % Number of samples per bit interval

sigma\_squared\_values = [0, 0.1, 0.5, 1.0]; % Values of noise variance

% Generate the received signal sequences for s0(t) and s1(t) for each noise variance

for sigma\_squared = sigma\_squared\_values

% Generate the noise sequence

nk = sqrt(sigma\_squared) \* randn(1, numSamples \* 2); % Generate Gaussian noise

% Generate received signal sequence for s0(t)

s0\_samples = [A \* ones(1, numSamples), -A \* ones(1, numSamples)]; % Sampled version of s0(t)

received\_s0 = A + nk; % Received signal when s0(t) is transmitted

% Generate received signal sequence for s1(t)

s1\_samples = [A \* ones(1, numSamples), -A \* ones(1, numSamples)]; % Sampled version of s1(t)

received\_s1 = [A + nk(1:numSamples), -A + nk(numSamples+1:end)]; % Received signal when s1(t) is transmitted

% Perform correlation with s0(t)

correlation\_s0 = zeros(1, numSamples);

for k = 1:numSamples

correlation\_s0(k) = sum(received\_s0 .\* circshift(s0\_samples, [0, k-1]));

end

% Perform correlation with s1(t)

correlation\_s1 = zeros(1, numSamples);

for k = 1:numSamples

correlation\_s1(k) = sum(received\_s1 .\* circshift(s1\_samples, [0, k-1]));

end

% Plot correlator outputs for s0(t) and s1(t) at time instants k = 1, 2, 3, ..., 10

figure;

subplot(2, 1, 1);

stem(1:numSamples, correlation\_s0);

title(['Correlation with s0(t) for \sigma^2 = ' num2str(sigma\_squared)]);

xlabel('Time instants (k)');

ylabel('Correlation output');

subplot(2, 1, 2);

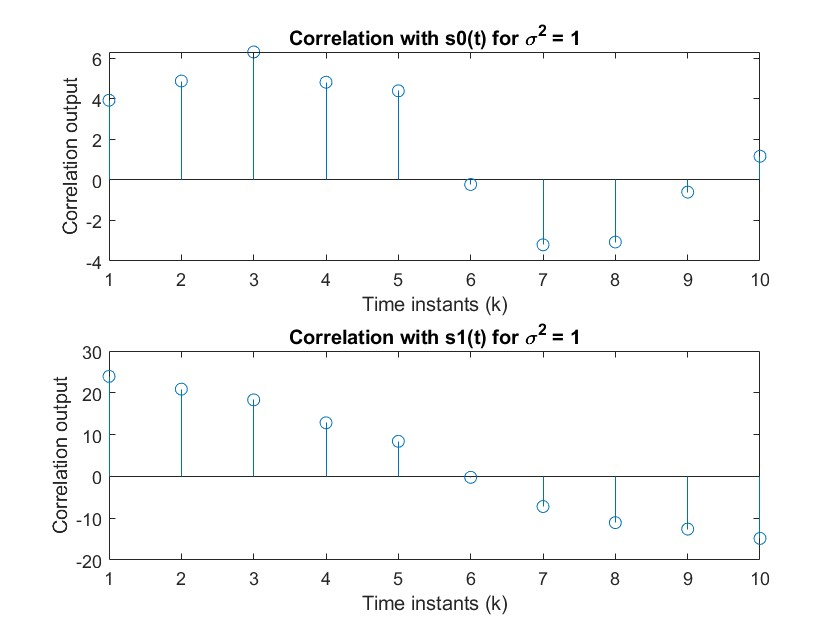
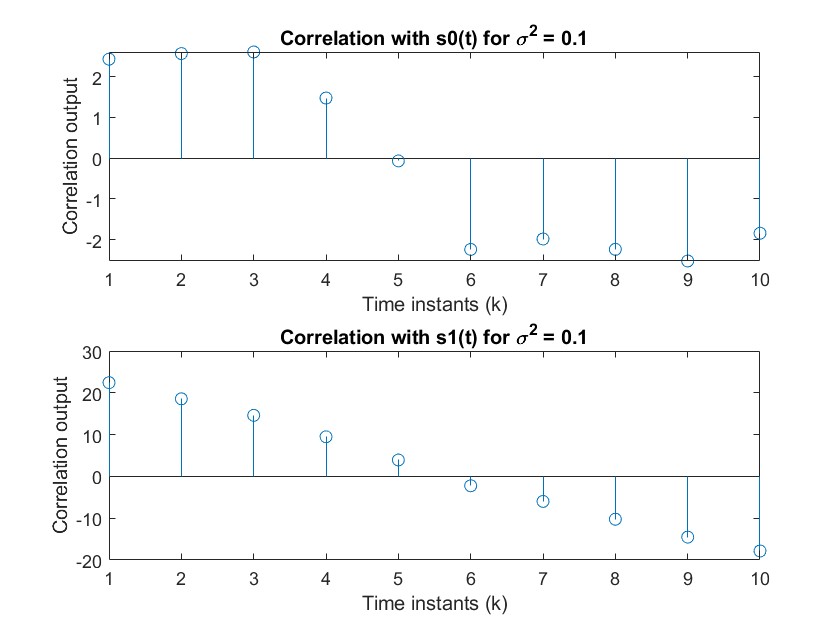
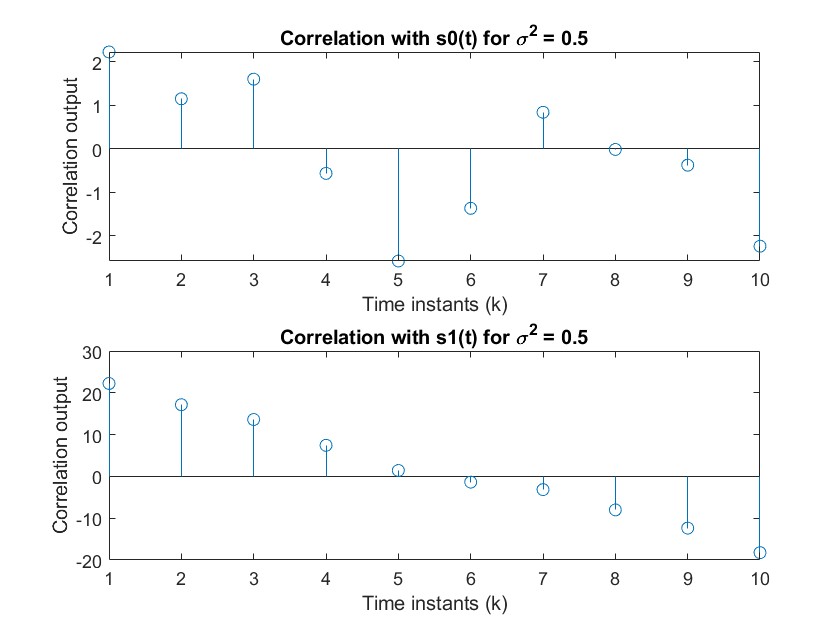
stem(1:numSamples, correlation\_s1);

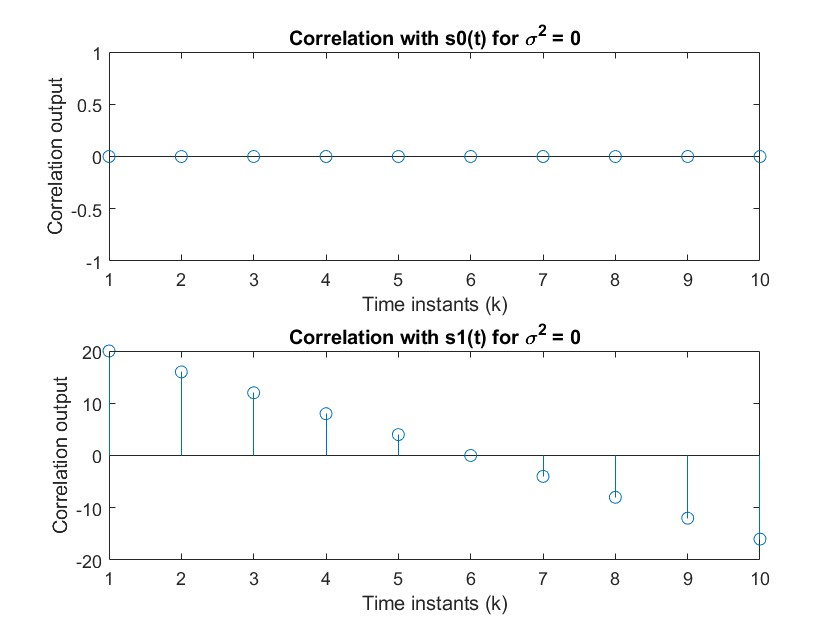
title(['Correlation with s1(t) for \sigma^2 = ' num2str(sigma\_squared)]);

xlabel('Time instants (k)');

ylabel('Correlation output');

end





**Problem 5.11:**

% Number of symbols

num\_symbols = 10000;

% Generate random quaternary symbols (-3, -1, 1, 3)

symbols = 2 \* (randi([0, 1], 1, num\_symbols) \* 2 - 1);

% Noise variance

noise\_variance = 0; % Setting noise variance to 0

% Generate Gaussian noise with zero mean and specified variance

noise = sqrt(noise\_variance) \* randn(1, num\_symbols);

% Add noise to the symbols

received\_symbols = symbols + noise;

% Decision rule: Demodulation

decoded\_symbols = zeros(size(received\_symbols));

decoded\_symbols(received\_symbols < -2) = -3;

decoded\_symbols(received\_symbols >= -2 & received\_symbols < 0) = -1;

decoded\_symbols(received\_symbols >= 0 & received\_symbols < 2) = 1;

decoded\_symbols(received\_symbols >= 2) = 3;

% Calculate the symbol error rate from simulation

symbol\_errors = sum(decoded\_symbols ~= symbols);

simulated\_symbol\_error\_rate = symbol\_errors / num\_symbols;

% Theoretical symbol error rate for QPAM with no noise

theoretical\_symbol\_error\_rate = 0;

% Display simulated and theoretical symbol error rates

fprintf('Simulated Symbol Error Rate: %.6f\n', simulated\_symbol\_error\_rate);

fprintf('Theoretical Symbol Error Rate: %.6f\n', theoretical\_symbol\_error\_rate);

% Plot 1000 received signal-plus-noise samples

samples\_to\_plot = 1000;

if samples\_to\_plot > num\_symbols

samples\_to\_plot = num\_symbols;

end

figure;

stem(1:samples\_to\_plot, received\_symbols(1:samples\_to\_plot));

xlabel('Sample Index');

ylabel('Received Signal + Noise');

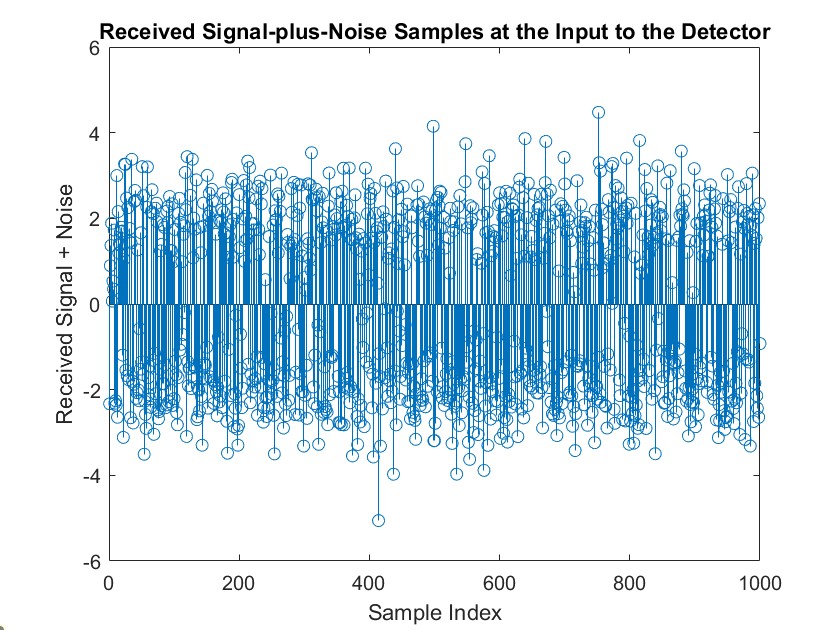
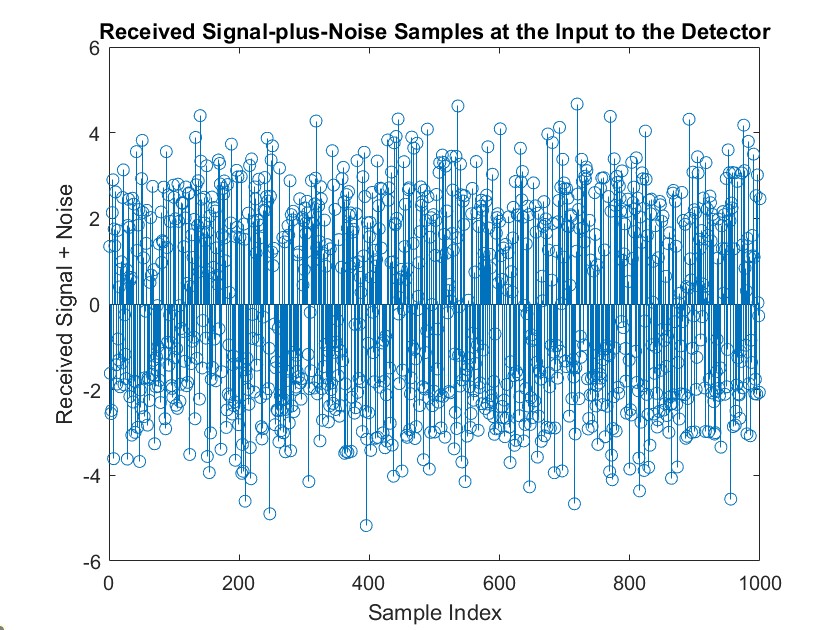
title('Received Signal-plus-Noise Samples at the Input to the Detector');

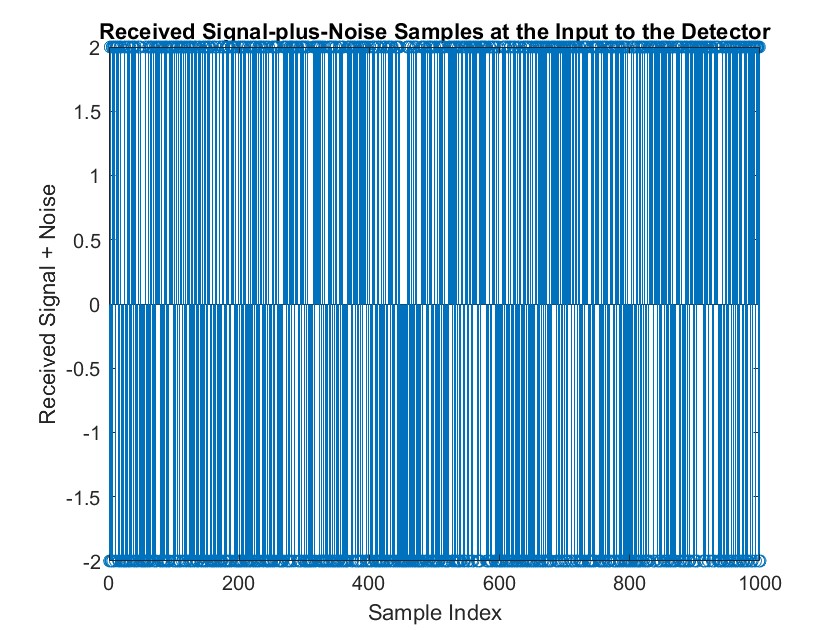
**Output:**

Sigma^2=0, 0.1, 0.5, 1

Simulated Symbol Error Rate: 1.000000

Theoretical Symbol Error Rate: 0.000000

Sigma=1Sigma=0.5Sigma=0.1

Sigma=0

**Experiment: 3:**

**Problem 7.1:**

% Define parameters

ex = 0.5; % Roll-off factor for square-root raised-cosine filter

T = 1; % Time period

fc = 40 / T; % Carrier frequency

% Frequency range

f = linspace(-10/T, 10/T, 1000);

% Calculate baseband signal spectrum (PAM with square-root raised-cosine filter)

X\_baseband = sinc(f \* T) .\* cos(pi \* ex \* f \* T) ./ (1 - (2 \* ex \* f \* T).^2);

% Calculate carrier spectrum (impulses at +/-fc)

carrier = zeros(size(f));

carrier(f == fc) = 1;

carrier(f == -fc) = 1;

% Calculate amplitude-modulated PAM signal spectrum

Y\_AM\_PAM = conv(X\_baseband, carrier, 'same');

% Plotting the spectra

figure;

% Baseband signal spectrum

subplot(2, 1, 1);

plot(f, abs(X\_baseband), 'b');

title('Baseband Signal Spectrum (PAM with SRRC filter)');

xlabel('Frequency');

ylabel('Amplitude');

grid on;

% AM-PAM signal spectrum

subplot(2, 1, 2);

plot(f, abs(Y\_AM\_PAM), 'r');

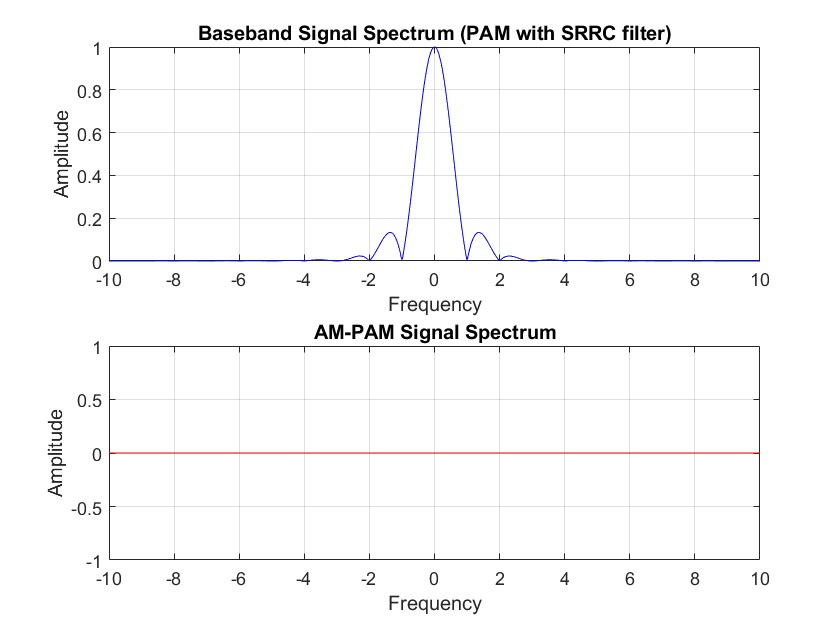
title('AM-PAM Signal Spectrum');

xlabel('Frequency');

ylabel('Amplitude');

grid on;

% Show plots

****

**Problem 7.7:**

% Define parameters

T = 1; % Time period

fc = 6 / T; % Carrier frequency

M = 8; % Number of phases for PSK

t = linspace(0, T, 1000); % Time vector

% Define GT(t) pulse shape function

GT = @(t) 0.5 \* (1 - cos(2 \* pi \* t / T)) .\* (0 <= t & t <= T);

% Generate 8-PSK waveforms

theta = linspace(0, 2 \* pi, M + 1); % Phase angles

theta = theta(1:M); % Remove the last angle to avoid overlap

PSK\_waveforms = zeros(M, length(t));

for i = 1:M

PSK\_waveforms(i, :) = cos(2 \* pi \* fc \* t + theta(i)) .\* GT(t);

end

% Plotting the 8-PSK signal waveforms

figure;

for i = 1:M

subplot(M/2, 2, i);

plot(t, PSK\_waveforms(i, :));

title(['Phase ' num2str(i)]);

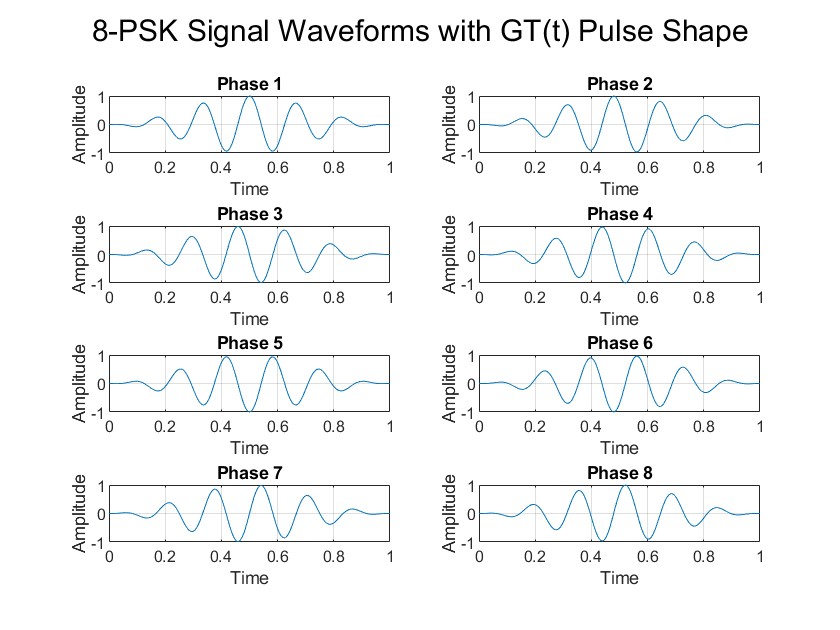
xlabel('Time');

ylabel('Amplitude');

grid on;

end

sgtitle('8-PSK Signal Waveforms with GT(t) Pulse Shape');



**Problem 7.9:**

% Define parameters

M = 4; % Number of phases for PSK

Eb\_N0\_dB = 0:5:20; % Eb/N0 values in dB

num\_symbols = 10^5; % Number of symbols to simulate

constellation = exp(1i \* (0:2\*pi/M:2\*pi\*(1-1/M))); % PSK constellation

% Monte Carlo simulation loop for different Eb/N0 values

for k = 1:length(Eb\_N0\_dB)

Eb\_N0 = 10^(Eb\_N0\_dB(k) / 10); % Convert dB to linear scale

% Generate random information symbols

info\_symbols = randi([1 M], 1, num\_symbols);

transmitted\_symbols = constellation(info\_symbols); % Map symbols to constellation

% AWGN channel

noise\_var = 1 / (2 \* Eb\_N0); % Calculate noise variance for given Eb/N0

noise = sqrt(noise\_var) \* (randn(1, num\_symbols) + 1i \* randn(1, num\_symbols)); % Complex Gaussian noise

received\_symbols = transmitted\_symbols + noise; % Received symbols with noise

% Detector: Find closest symbol to the received symbol

detected\_symbols = zeros(1, num\_symbols);

for i = 1:num\_symbols

received\_phase = angle(received\_symbols(i));

[~, index] = min(abs(received\_phase - angle(constellation)));

detected\_symbols(i) = constellation(index);

end

% Calculate symbol error rate (SER)

symbol\_errors = nnz(detected\_symbols - transmitted\_symbols);

symbol\_error\_rate(k) = symbol\_errors / num\_symbols;

end

% Plot Symbol Error Rate (SER) vs Eb/N0

figure;

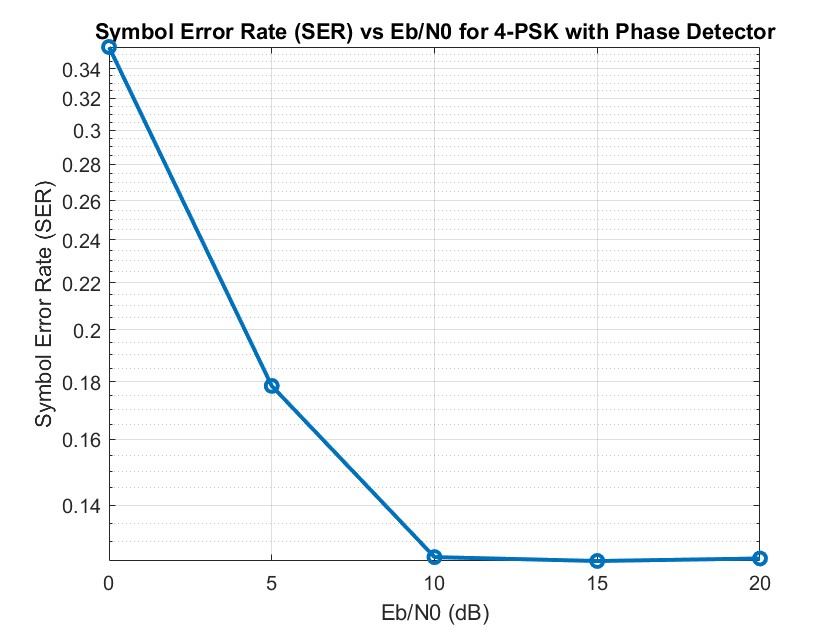
semilogy(Eb\_N0\_dB, symbol\_error\_rate, 'o-', 'linewidth', 2);

title('Symbol Error Rate (SER) vs Eb/N0 for 4-PSK with Phase Detector');

xlabel('Eb/N0 (dB)');

ylabel('Symbol Error Rate (SER)');

grid on;



**Problem 7.12:**

% Define parameters

T = 1; % Time period

fc = 8 / T; % Carrier frequency

A = sqrt(2) - 1; % Signal constellation amplitude factor

t = linspace(0, T, 1000); % Time vector

% Define 8-QAM constellation points (in-phase and quadrature components)

constellation = [

-A - 1i \* A;

-A + 1i \* A;

A - 1i \* A;

A + 1i \* A;

-A - 3i \* A;

-A + 3i \* A;

A - 3i \* A;

A + 3i \* A

];

% Generate and plot 8-QAM signal waveforms

figure;

for i = 1:size(constellation, 1)

% Generate waveform for each constellation point using rectangular pulse

waveform = real(constellation(i)) \* cos(2 \* pi \* fc \* t) + imag(constellation(i)) \* sin(2 \* pi \* fc \* t);

% Plot each waveform

subplot(4, 2, i);

plot(t, waveform);

title(['8-QAM Waveform ' num2str(i)]);

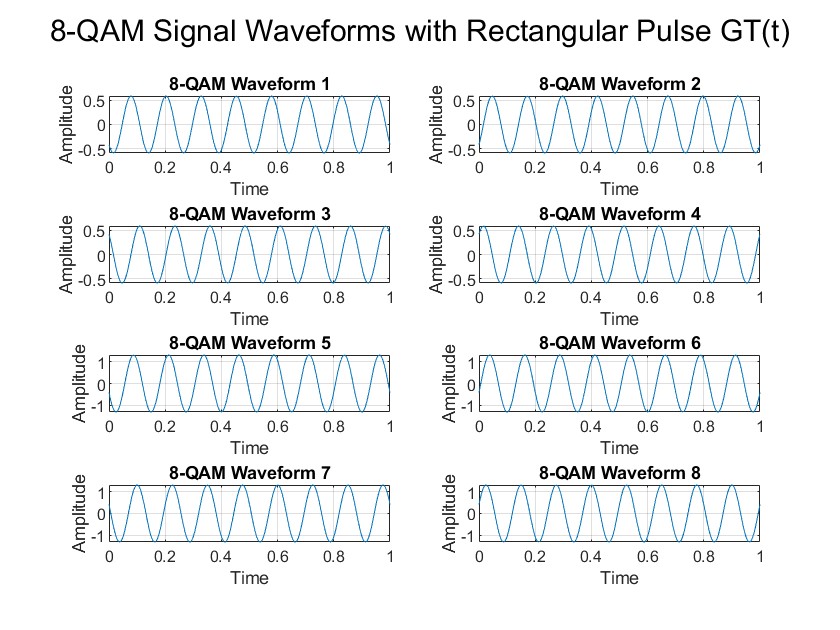
xlabel('Time');

ylabel('Amplitude');

grid on;

end

sgtitle('8-QAM Signal Waveforms with Rectangular Pulse GT(t)');



**Problem 7.17:**

% Define parameters

M = 8; % Number of symbols in 8-QAM

num\_symbols = 10000; % Number of symbols to simulate

Eb\_N0\_dB = 0:2:20; % Eb/N0 values in dB

Eb\_N0 = 10.^(Eb\_N0\_dB / 10); % Convert Eb/N0 values to linear scale

% Constellation points for 8-QAM

constellation = [

-3-3i;

-3-1i;

-3+3i;

-3+1i;

-1-3i;

-1-1i;

-1+3i;

-1+1i

];

% Initialize arrays to store symbol error rates (SER)

SER\_simulated = zeros(1, length(Eb\_N0));

SER\_upper\_bound = zeros(1, length(Eb\_N0));

% Monte Carlo simulation loop for different Eb/N0 values

for k = 1:length(Eb\_N0)

% Generate random information symbols

info\_symbols = randi([1 M], 1, num\_symbols);

transmitted\_symbols = constellation(info\_symbols); % Map symbols to constellation

% Normalization of energy to unity

normalized\_symbols = transmitted\_symbols / sqrt(mean(abs(transmitted\_symbols).^2));

% Generate noise components

sigma = 1 / sqrt(2 \* Eb\_N0(k)); % Calculate noise standard deviation

noise = sigma \* (randn(1, num\_symbols) + 1i \* randn(1, num\_symbols)); % Complex Gaussian noise

% Received symbols with noise

received\_symbols = normalized\_symbols + noise;

% Demodulate received symbols (nearest neighbor detection)

detected\_symbols = zeros(size(received\_symbols));

for i = 1:num\_symbols

[~, index] = min(abs(received\_symbols(i) - constellation));

detected\_symbols(i) = constellation(index);

end

% Calculate symbol error rate (SER)

symbol\_errors = nnz(detected\_symbols - transmitted\_symbols);

SER\_simulated(k) = symbol\_errors / num\_symbols;

% Theoretical upper bound on symbol error rate

SER\_upper\_bound(k) = 2 \* qfunc(sqrt(3 \* Eb\_N0(k))); % Theoretical upper bound formula

end

% Plotting the results

figure;

semilogy(Eb\_N0\_dB, SER\_simulated, 'o-', 'linewidth', 2, 'DisplayName', 'Simulated SER');

hold on;

semilogy(Eb\_N0\_dB, SER\_upper\_bound, 'r--', 'linewidth', 2, 'DisplayName', 'Theoretical Upper Bound');

title('Symbol Error Rate (SER) vs Eb/N0 for 8-QAM System');

xlabel('Eb/N0 (dB)');

ylabel('Symbol Error Rate (SER)');

legend('Location', 'best');

grid on;

