MAHA BARATHI ENGINEERING COLLEGE

NH-79, SALEM-CHENNAI HIGHWAY, A.VASUDEVANUR, CHINNASALEM (TK), KALLAKURICHI (DT) 606 201.
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai
Accredited by NAAC and Recognized under section 2(f) & 12(B) status of UGC, New Delhi

www.mbec.ac.in | Ph: 04151-256333, 257333 | E-mail: mbec123@gmail.com

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CEC331- 4G/5G Communication Networks

III Year/ VI Semester B.E ECE

Regulation 2021
(As Per Anna University, Chennai syllabus)

MAHA BARATHI ENGINEERING COLLEGE

NH-79, SALEM-CHENNAI HIGHWAY, A.VASUDEVANUR, CHINNASALEM (TK), KALLAKURICHI (DT) 606 201.

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

Accredited by NAAC and Recognized under section 2(f) & 12(B) status of UGC, New Delhi

www.mbec.ac.in | Ph: 04151-256333, 257333 | E-mail: mbec123@gmail.com

BONAFIDE CERTIFICATE

Certified that this is a bonafide record of workdone by Selvan/Selvi…………………………………………………………. ……………

Reg.No.……………………………..of……**Sixth**…………………Semester…………

Electronics and Communication Engineering ……Branch of ……B.E………..

Degree Examination in the subject……………**CEC331- 4G/5G Communication Networks**……

Staff In charge

Date:

Head of the Department

Submitted for Anna University Practical Examination conducted on…………………

Internal Examiner

External Examiner

CONTENTS

LIST OF EXPERIMENTS

SIMULATION USING MATLAB

1. 5G-Compliant waveform generation and testing
2. Modelling of 5G Synchronization signal blocks and bursts
3. Channel Modelling in 5G networks
4. Multiband OFDM demodulation
5. Perfect Channel estimation
6. Development of 5g New Radio Polar Coding

EXPT NO: 1 Date:	5G-COMPLIANT WAVEFORM GENERATION AND TESTING

AIM:

To perform the 5G- compliant waveform generation and testing in Matlab software.

SOFTWARE USED:

PC with MATLAB Software

PROCEDURE:

1. Setting Parameters for Carrier frequency, Sample rate, Number of samples in the waveform, Signal-to-noise ratio
2. Generate random binary data for QPSK modulation.
3. QPSK Modulation
4. Create a time vector based on the number of samples and sample rate
5. Combine the in-phase and quadrature components to form the QPSK signal.
6. Generate the carrier signal.
7. Modulate the QPSK symbols with the carrier signal to get the transmitted signal.
8. Add AWGN (Additive White Gaussian Noise) to the transmitted signal.
9. Divide the received signal by the carrier signal to perform demodulation.
10. Extract the phase information from the received signal.
11. Convert the demodulated symbols back to bits by comparing the phase.
12. Plot the transmitted signal, and received signal with noise, and the comparison of transmitted and decoded data.

MATLAB CODE:

```
clc;
clear all;
close all;
% Parameters
carrier-frequency = 3.5e9; % Carrier frequency in Hz (e.g., 3.5 GHz for sub-6GHz 5G)
Sample Rate = 30.72e6;     % Sample rate in Hz
Num Samples = 1024;    % Number of samples in the waveform
snr = 20;   % Signal-to-noise ratio in dB

% Generate a simple 5G waveform (QPSK modulation)
data = randi([0, 1], 2, numSamples); % Generate random bits for QPSK modulation
qpskSymbols = 2 * data - 1;    % Map bits to QPSK symbols (-1, 1)

% Create a time vector
time = (0:numSamples - 1) / sampleRate;

% Modulate the QPSK symbols
qpskSignal = qpskSymbols(1, :) + 1j * qpskSymbols(2, :);

% Generate the carrier signal
carrierSignal = exp(1j * 2 * pi * carrierFrequency * time);

% Generate the transmitted signal
transmittedSignal = qpskSignal .* carrierSignal;

% Add noise to the transmitted signal
noisySignal = awgn(transmittedSignal, snr, 'measured');
```

% Receiver

```
received signal = noisy signal / carrier signal;
```

% Demodulate the received signal

```
demodulatedSymbols = angle(receivedSignal);
```

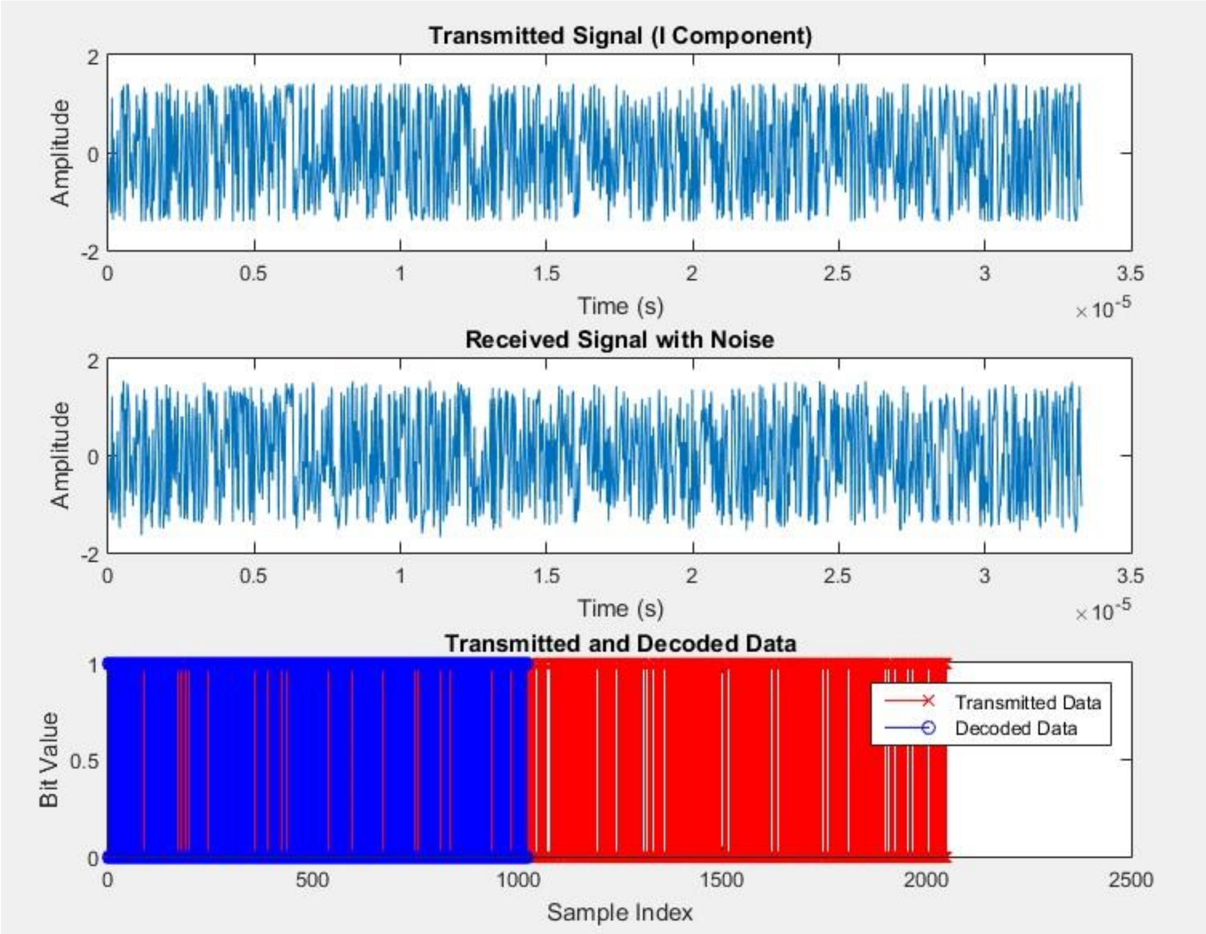
% Decode the demodulated symbols back to bits

```
decodedData = demodulatedSymbols> 0;
```

% Plot the results

```
subplot(3, 1, 1);
plot(time, real(transmittedSignal));
title('Transmitted Signal (I Component)');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3, 1, 2);
plot(time, real(noisySignal));
title('Received Signal with Noise');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3, 1, 3);
stem(data(:), 'rx');
hold on;
stem(decodedData(:), 'bo');
title('Transmitted and Decoded Data');
xlabel('Sample Index');
ylabel('Bit Value');
legend('Transmitted Data', 'Decoded Data');
```

Output:



Result:

Thus the 5G-Compliant Waveform Generation and Testing in MATLAB was successfully executed.

EXPT NO: 2 Date:	MODELING OF 5G SYNCHRONIZATION SIGNAL BLOCKS AND BURSTS

AIM:

To perform the Model 5G Synchronization Signal Blocks (SSBs) and Bursts in MATLAB software.

SOFTWARE USED:

PC with MATLAB Software

PROCEDURE:

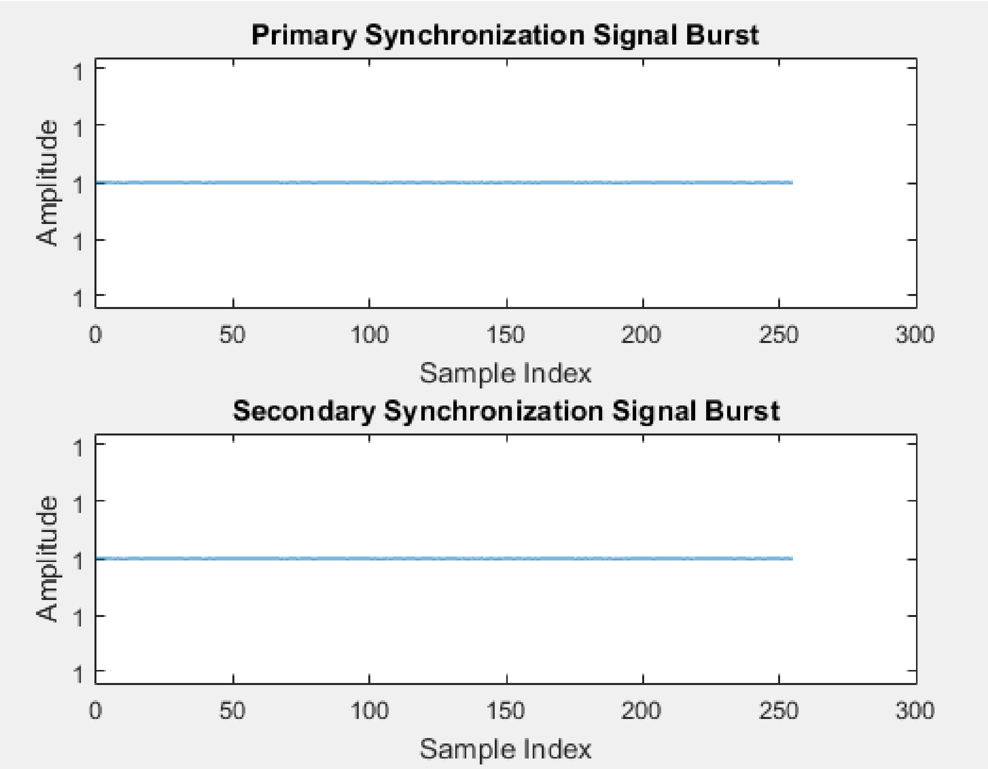
1. Set Parameters for sampling frequency, cyclic prefix duration, sampling period
2. Generate Primary Synchronization Signal (PSS)
3. Generate Secondary Synchronization Signal (SSS)
4. Repeat the PSS and SSS sequences to construct the full synchronization signal burst.
5. Modulate the PSS and SSS sequences with the specified cyclic prefix duration to create the burst signals.
6. Visualize the amplitude of the PSS and SSS bursts
7. Generate a simulated received signal by adding noise to the sum of PSS and SSS bursts
8. Perform cross-correlation of the received signal with PSS and SSS bursts to detect synchronization
9. Visualize the correlation results for PSS and SSS

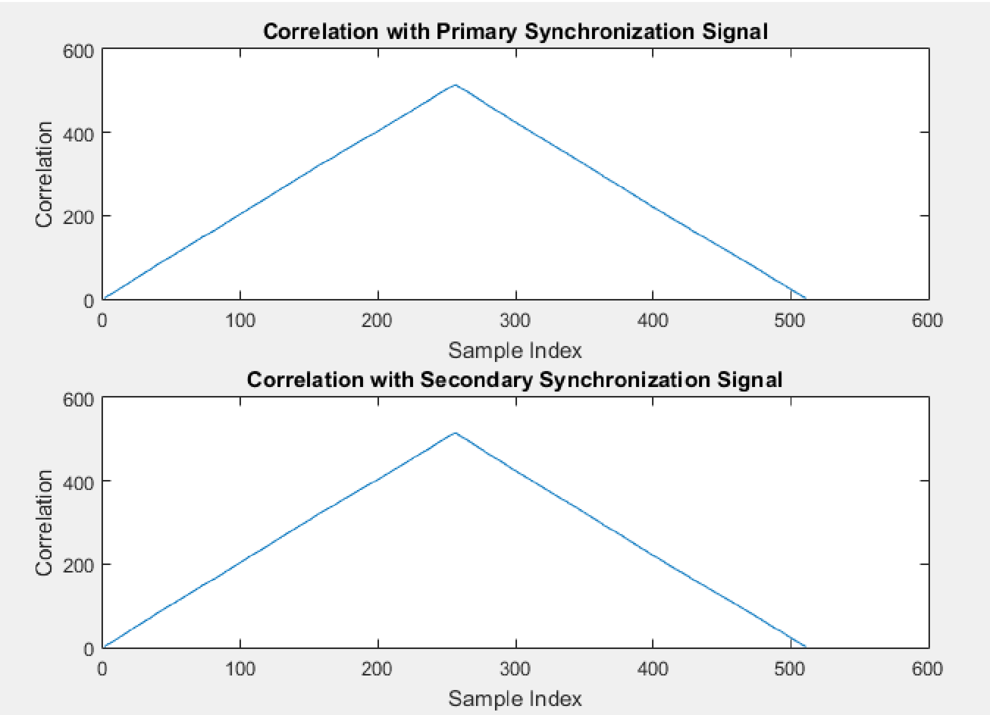
MATLAB CODE:

```
clc;
clear all;
close all;

% Parameters
fs = 30.72e6; % Sampling frequency (Hz)
Tc = 1/4.6e6; % Cyclic Prefix duration (s)
Ts = 1/30.72e6; % Sampling period (s)

% Generate Primary Synchronization Signal (PSS)
N_id_1 = 0; % PSS identity (0 to 127)
n = 0:127;
pss = exp(1j * pi * N_id_1 * (n.*(n+1)/2));

% Generate Secondary Synchronization Signal (SSS)
N_id_2 = 0; % SSS identity (0 or 1)
m = 0:31;
sss = exp(1j * pi * N_id_2 * m);

% Generate Burst
n_burst = 0:255; % Burst duration in samples
pss_sequence = repmat(pss, 1, length(n_burst)/length(pss));
sss_sequence = repmat(sss, 1, length(n_burst)/length(sss));

% Construct full synchronization signal burst
pss_burst = pss_sequence .* exp(1j * 2 * pi * n_burst * Tc / Ts);
sss_burst = sss_sequence .* exp(1j * 2 * pi * n_burst * Tc / Ts);

% Plot the bursts
figure;
subplot(2,1,1);
```

```
plot(n_burst, abs(pss_burst));
title('Primary Synchronization Signal Burst');
xlabel('Sample Index');
ylabel('Amplitude');
subplot(2,1,2);
plot(n_burst, abs(sss_burst));
title('Secondary Synchronization Signal Burst');
xlabel('Sample Index');
ylabel('Amplitude');

% Correlation with received signal (for synchronization detection)
received_signal = awgn(pss_burst + sss_burst, 10); % Simulated received signal with noise

% Correlation with PSS
correlation_pss = abs(xcorr(received_signal, pss_burst));
figure;
subplot(2,1,1);
plot(correlation_pss);
title('Correlation with Primary Synchronization Signal');
xlabel('Sample Index');
ylabel('Correlation');

% Correlation with SSS
correlation_sss = abs(xcorr(received_signal, sss_burst));
subplot(2,1,2);
plot(correlation_sss);
title('Correlation with Secondary Synchronization Signal');
xlabel('Sample Index');
ylabel('Correlation');
```

Output:





RESULT:

Thus the Modeling 5G Synchronization Signal Blocks (SSBs) and Bursts using MATLAB was successfully executed.

EXPT NO: 3 Date:	CHANNEL MODELLING IN 5G NETWORKS

AIM:

To simulate and analyze the propagation characteristics of wireless signals in 5G networks using MATLAB.

SOFTWARE USED:

PC with MATLAB Software

PROCEDURE:

1. Set simulation parameters.
2. Define channel impulse response and normalize it.
3. Generate random QPSK symbols for transmission.
4. Convolve QPSK symbols with channel impulse response.
5. Add AWGN noise to the received waveform.
6. Plot transmitted QPSK symbols and received waveform.

MATLAB CODE:

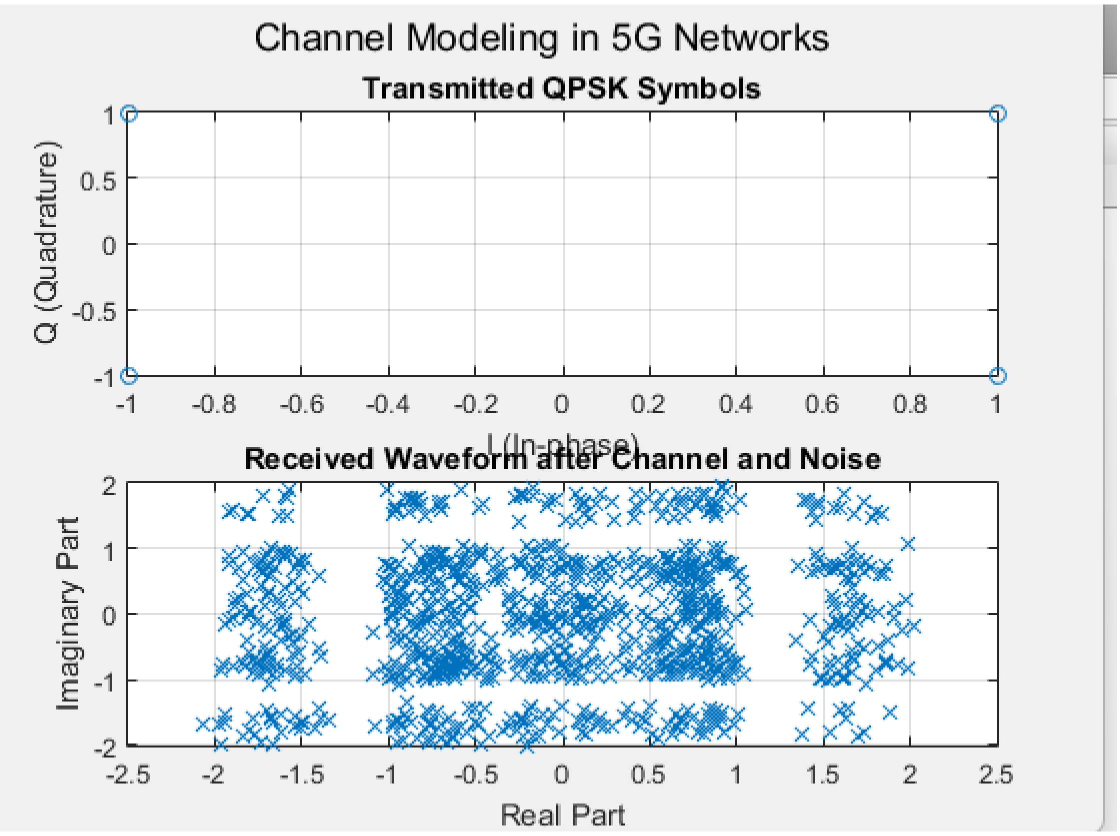
```
clc;
clear all;
close all;

% Parameters
num_samples = 1000;     % Number of samples in the waveform
snr_db = 20;            % Signal-to-noise ratio in dB

% Generate a simple channel impulse response
channel_impulse_response = [0.1, 0.5, 0.8, 0.5, 0.1]; % Example channel
coefficients

% Normalize the channel response
channel_impulse_response = channel_impulse_response /
norm(channel_impulse_response);

% Generate a random QPSK waveform
qpsk_symbols = 2 * (randi([0, 1], 1, num_samples) - 0.5) +
            1j * (2 * (randi([0, 1], 1, num_samples) - 0.5));

% Convolve the waveform with the channel impulse response
received_waveform = conv(qpsk_symbols, channel_impulse_response);

% Add AWGN noise
noise_power = 10^(-snr_db / 10);
noise = sqrt(noise_power / 2) * (randn(size(received_waveform)) + 1j *
randn(size(received_waveform)));
received_waveform = received_waveform + noise;

% Display the transmitted and received waveforms
figure;
subplot(2, 1, 1);
plot(real(qpsk_symbols), imag(qpsk_symbols), 'o');
title('Transmitted QPSK Symbols');
xlabel('I (In-phase)');
ylabel('Q (Quadrature)');
grid on;

subplot(2, 1, 2);
plot(real(received_waveform), imag(received_waveform), 'x');
title('Received Waveform after Channel and Noise');
```

```
xlabel('Real Part');
ylabel('Imaginary Part');
grid on;
```

% Add a title for the entire figure

```
figure_title = 'Channel Modeling in 5G Networks';
h = suptitle(figure_title);
set(h, 'FontSize', 14);
```

Output:



Result:

Thus the Channel modeling in 5G Networks using MATLAB was successfully executed.

EXPT NO: 4 Date:	MULTIBAND OFDM DEMODULATION

AIM:

To implement efficient and accurate demodulation of Multi-Band Orthogonal Frequency Division Multiplexing (MB-OFDM) signals using MATLAB.

SOFTWARE USED:

PC with MATLAB Software

PROCEDURE:

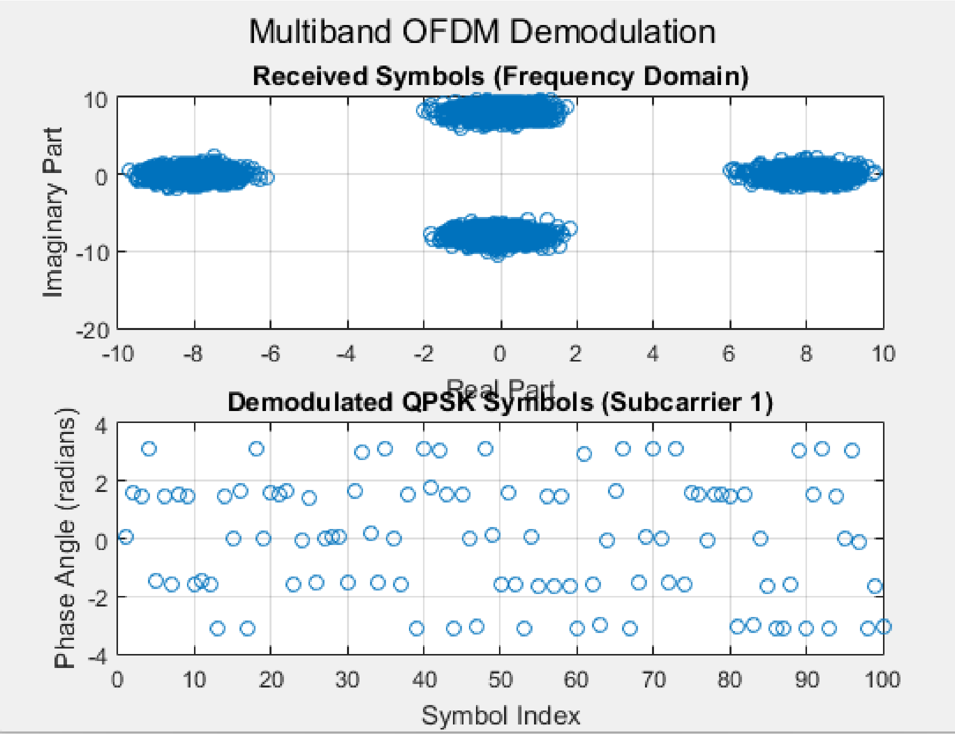
1. Set simulation parameters.
2. Generate Random QPSK Symbols
3. Perform IFFT
4. Add a cyclic prefix to the time-domain waveform.
5. Up sample the waveform to account for oversampling.
6. Generate and Add Noise to the Transmitted Waveform.
7. Down sample the received waveform to account for the oversampling
8. Remove the cyclic prefix from the received waveform.
9. Perform a Fast Fourier Transform (FFT) on the received waveform & Demodulate QPSK Symbols.

MATLAB CODE:

```
clc;
clear all;
close all;

% Parameters
num_subcarriers = 64;          % Number of subcarriers
num_symbols = 100;             % Number of symbols
oversampling_factor = 4;     % Oversampling factor
sampling_rate = 1e6;          % Sampling rate in Hz
symbol_rate = 10e3;           % Symbol rate in  Hz
snr_db = 20;                 % Signal-to-noise ratio in dB

% Generate random QPSK symbols
qpsk_symbols = randi([0, 3], num_subcarriers, num_symbols);
qpsk_symbols = exp(1j * pi / 2 * qpsk_symbols);

% Perform IFFT to generate time-domain waveform
time_domain_waveform = ifft(qpsk_symbols, num_subcarriers) *
sqrt(num_subcarriers);

% Add cyclic prefix (CP)
cp_length = 16;
cp = time_domain_waveform(end - cp_length + 1:end, :);
time_domain_waveform_with_cp = [cp; time_domain_waveform];

% Upsample the waveform
tx_waveform = upsample(time_domain_waveform_with_cp,
oversampling_factor);

% Generate AWGN noise
noise_power = 10^(-snr_db / 10);
noise = sqrt(noise_power / 2) * (randn(size(tx_waveform)) + 1j *
randn(size(tx_waveform)));

% Add noise to the transmitted waveform
rx_waveform = tx_waveform + noise;

% Downsample the received waveform
rx_waveform_downsampled = downsample(rx_waveform,
oversampling_factor);
```

```
% Remove cyclic prefix
rx_waveform_no_cp = rx_waveform_downsampled(cp_length + 1:end, :);

% Perform FFT to obtain frequency-domain symbols
rx_freq_symbols = fft(rx_waveform_no_cp, num_subcarriers);

% Demodulate QPSK symbols
rx_qpsk_symbols = angle(rx_freq_symbols);

% Choose a subcarrier index for plotting (e.g., the first subcarrier)
subcarrier_index = 1;

% Display the received symbols before and after demodulation
subplot(2, 1, 1);
plot(real(rx_freq_symbols(:)), imag(rx_freq_symbols(:)), 'o');
title('Received Symbols (Frequency Domain)');
xlabel('Real Part');
ylabel('Imaginary Part');
grid on;

subplot(2, 1, 2);

% Plot the demodulated QPSK symbols for the chosen subcarrier
plot(1:num_symbols, rx_qpsk_symbols(subcarrier_index, :), 'o');
title(['Demodulated QPSK Symbols (Subcarrier ', num2str(subcarrier_index),
')']);
xlabel('Symbol Index');
ylabel('Phase Angle (radians)');
grid on;

% Add a title for the entire figure
figure_title = 'Multiband OFDM Demodulation';
h = suptitle(figure_title);
set(h, 'FontSize', 14);
```

Output:



Result:

Thus the Demodulation of Multi-Band Orthogonal Frequency Division Multiplexing (MB-OFDM) signals using MATLAB was successfully executed.

EXPT NO: 5 Date:	PERFECT CHANNEL ESTIMATION

AIM:

To achieve accurate and optimal channel estimation for 5G communication systems using MATLAB

SOFTWARE USED:

PC with MATLAB Software

PROCEDURE:

1. Set simulation parameters.
2. Generate Random Channel Matrix & Random Symbols
3. Modulate the symbols using QPSK modulation.
4. Transmit the modulated symbols through the generated channel matrix.
5. Add Noise to Received Symbols.
6. Perform Channel Estimation & Calculate Mean square error (MSE).
7. Plot the true and estimated channel matrices.

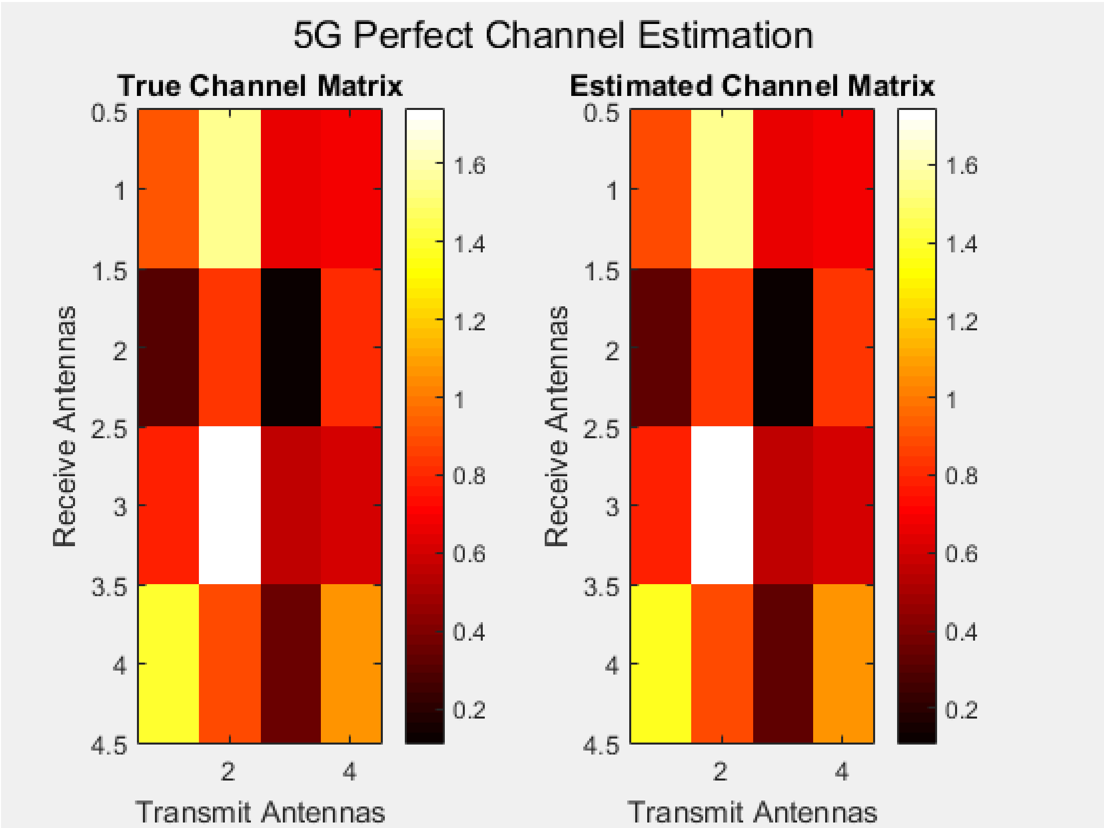
Matlab Code:

```
clc;
clear all;
close all;

% Parameters
num_antennas = 4;    % Number of antennas at the transmitter / receiver
num_symbols = 100;        % Number of symbols
snr_db = 20;              % Signal-to-noise ratio in dB

% Generate random channel matrix
H_true = (randn(num_antennas, num_antennas) + 1i *
randn(num_antennas, num_antennas)) / sqrt(2);

% Generate random symbols
symbols = randi([0, 1], num_antennas, num_symbols);

% Modulate symbols (e.g., using QPSK)
modulated_symbols = 2 * symbols - 1;

% Transmit symbols through channel
received_symbols = H_true * modulated_symbols;

% Add noise to received symbols
noise_power = 10^(-snr_db / 10);
noise = sqrt(noise_power/2) * (randn(num_antennas, num_symbols) + 1i *
randn(num_antennas, num_symbols));
received_symbols_with_noise = received_symbols + noise;

% Perform channel estimation using received and transmitted symbols
estimated_H = received_symbols_with_noise * pinv(modulated_symbols);

% Calculate mean squared error (MSE) of the estimated channel
mse = mean(mean(abs(H_true - estimated_H).^2, 'omitnan'));

% Display MSE
disp(['Mean Squared Error: ', num2str(mse)]);

% Plot the true and estimated channel matrices
subplot(1, 2, 1);
imagesc(abs(H_true));
colormap('hot');
```

```
colorbar;
title('True Channel Matrix');
xlabel('Transmit Antennas');
ylabel('Receive Antennas');

subplot(1, 2, 2);
imagesc(abs(estimated_H));
colormap('hot');
colorbar;
title('Estimated Channel Matrix');
xlabel('Transmit Antennas');
ylabel('Receive Antennas');

% Add a title for the entire figure
figure_title = '5G Perfect Channel Estimation';
h = suptitle(figure_title);
set(h, 'FontSize', 14);
```

Output:

Mean Squared Error: **0.00010273**



Result:

Thus the accurate and optimal channel estimation for 5G communication systems using MATLAB was successfully executed.

EXPT NO: 6 Date:	DEVELOPMENT OF 5G NEW RADIO POLAR CODING

AIM:

To perform Polar Coding and Decoding using MATLAB.

SOFTWARE USED:

PC with MATLAB Software

PROCEDURE:

1. Initializes the parameters for the polar code, including the length of information bits and CRC bits
2. Generate Random information bits are generated for transmission
3. Create Polar Code
4. Encode Information Bits
5. Add CRC Bits to the encoded codeword for error detection
6. Simulate Channel Errors by adding AWGN (Additive White Gaussian Noise) to the transmitted codeword.
7. Perform Polar Decoding to recover the information bits, considering the CRC for error checking.
8. Check CRC for Error Detection
9. Display transmitted and received information bits

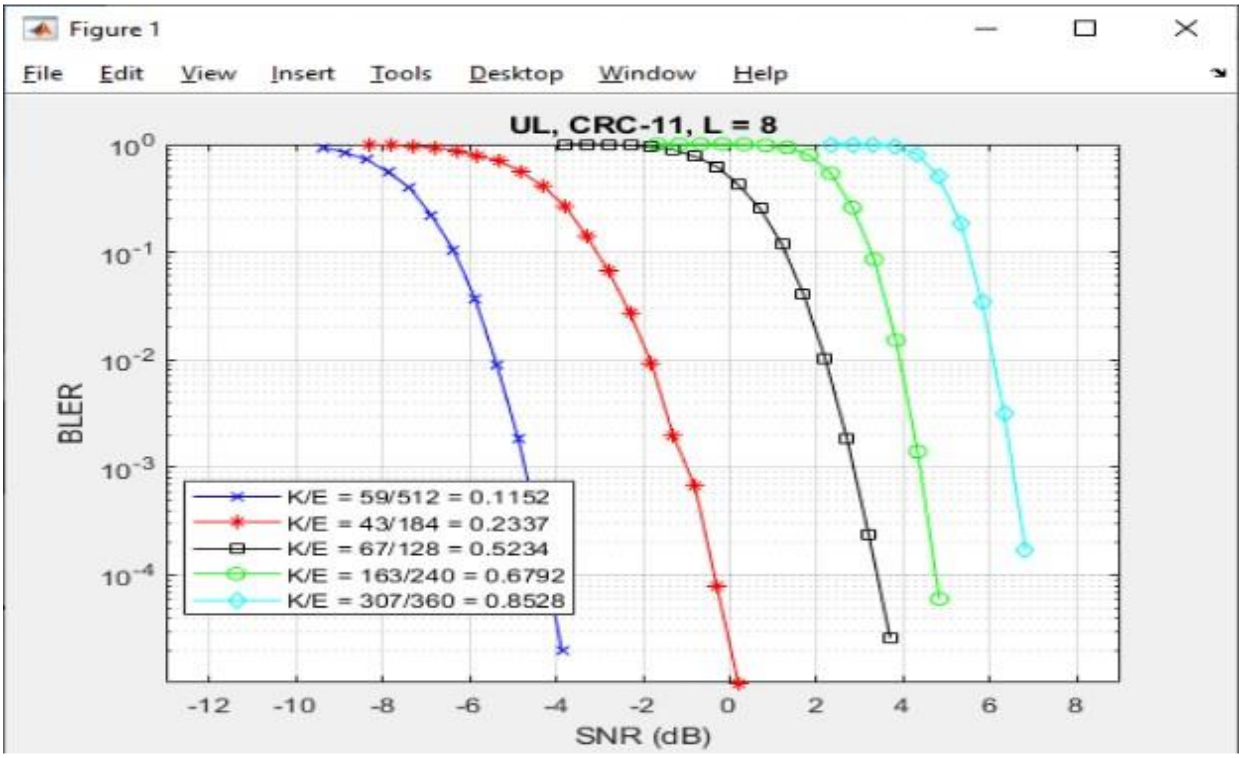
Matlab Code:

```
clc;
clear all;
close all;

% Set the parameters for polar coding
infoLength = 16; % Number of information bits
crcLength = 8; % Number of CRC bits

% Generate random information bits (0s and 1s)
infoBits = randi([0, 1], infoLength, 1);

% Create a polar code using the 5G NR code construction rules
polarCode = nrPolarCode(infoLength, infoLength + crcLength);

% Encode the information bits using the polar code
codeword = nrPolarEncode(infoBits, polarCode);

% Add CRC bits to the codeword
crc = comm.CRCGenerator('Polynomial', 'z^8 + z^2 + 1');
crcBits = crc(codeword);

% Simulate channel errors (for demonstration purposes)
receivedCodeword = awgn(codeword, 10); % Add AWGN noise (10 dB SNR)

% Perform polar decoding to recover the information bits
decodedInfoBits = nrPolarDecode(receivedCodeword, polarCode, crc);

% Check CRC to verify the correctness of the decoded information bits
crcDetector = comm.CRCDetector('Polynomial', 'z^8 + z^2 + 1');
isCRCValid = crcDetector(receivedCodeword);
```

% Display the results

```
disp('Transmitted Information Bits:');
disp(infoBits.');
disp('Received Information Bits:');
disp(decodedInfoBits.');
if isCRCValid
disp('CRC Check: Passed (Decoded information is correct).');
else
disp('CRC Check: Failed (Decoded information has errors).');
end
```

Output:



Result:

Thus the Polar Coding and Decoding using MATLAB was successfully executed.