EXP2

SECTORING  
clc;

close all;

clear all;

sectoring\_type = input('Enter the type of sectoring (360, 120, or 60 degrees): ', 's');

if strcmp(sectoring\_type, '360')

N = 12; io = 6; n = 4;

S\_I = (sqrt(3 \* N))^n / io;

fprintf('For 360-degree sectoring:\nS/I in standard scale: %.4f\nS/I in dB scale: %.2f dB\n', S\_I, 10 \* log10(S\_I));

elseif strcmp(sectoring\_type, '120')

N = 7; io = 2; n = 4;

S\_I = (sqrt(3 \* N))^n / io;

fprintf('For 120-degree sectoring:\nS/I in dB scale: %.2f dB\n', 10 \* log10(S\_I));

elseif strcmp(sectoring\_type, '60')

N = 4; io = 1; n = 4;

S\_I = (sqrt(3 \* N))^n / io;

fprintf('For 60-degree sectoring:\nS/I in dB scale: %.2f dB\n', 10 \* log10(S\_I));

else360

disp('Invalid sectoring type. Please enter "360", "120", or "60".');

end

EXP3

EXP4

**a. To predict the type of small scale fading channel according to coherence bandwidth.**

**b. To predict the type of small scale fading channel according to Doppler spread.**

clc;

clear all;

close all;

% Part A: Coherence Bandwidth and Channel Type Prediction

% Step 1: Take user-defined values of RMS delay spread (τ\_rms) and frequency correlation (fc)

tau\_rms = input('Enter the RMS delay spread (in seconds): ');

fc = input('Enter the frequency correlation (fc): ');

% Validate inputs

if tau\_rms <= 0

fprintf('Error: RMS delay spread must be a positive number.\n');

elseif fc < 0.5 || fc > 1

fprintf('Error: Frequency correlation (fc) must be between 0.5 and 1.\n');

else

% Step 2: Find coherence bandwidth (Bc)

if fc >= 0.9

Bc = 1 / (50 \* tau\_rms);

elseif fc >= 0.5 && fc < 0.9

Bc = 1 / (5 \* tau\_rms);

end

fprintf('The coherence bandwidth (Bc) is: %.2f Hz\n', Bc);

% Step 3: Take user-defined values of signal bandwidth (Bs) and symbol period (Ts)

Bs = input('Enter the signal bandwidth (in Hz): ');

Ts = input('Enter the symbol period (in seconds): ');

% Predict the type of channel based on coherence bandwidth and signal bandwidth

if Bs <= Bc

fprintf('The channel is a Flat Fading Channel.\n');

else

fprintf('The channel is a Frequency Selective Fading Channel.\n');

end

end

% Part B: Doppler Shift, Coherence Time, and Channel Type Prediction

% Step 1: Take user-defined velocity (v) in m/s and carrier frequency (f\_c)

v = input('Enter the velocity (in m/s): ');

f\_c = input('Enter the carrier frequency (in Hz): ');

% Validate inputs

if v < 0

fprintf('Error: Velocity must be a non-negative number.\n');

elseif f\_c <= 0

fprintf('Error: Carrier frequency must be a positive number.\n');

else

% Step 2: Calculate maximum Doppler shift (f\_d)

c = 3e8; % Speed of light in m/s

f\_d = (v / c) \* f\_c;

fprintf('The maximum Doppler shift (f\_d) is: %.2f Hz\n', f\_d);

% Step 3: Calculate coherence time (Tc)

Tc = (0.423 / f\_d);

fprintf('The coherence time (Tc) is: %.2f seconds\n', Tc);

% Step 4: Take user-defined values of signal bandwidth (Bs) and symbol period (Ts)

Bs = input('Enter the signal bandwidth (in Hz): ');

Ts = input('Enter the symbol period (in seconds): ');

% Predict the type of channel based on coherence time and symbol period

if Ts <= Tc

fprintf('The channel is a Slow Fading Channel.\n');

else

fprintf('The channel is a Fast Fading Channel.\n');

end

end

EXP5

Part A: The total spectrum allocation is 12.5 MHZ and the guard band allocated at the edge of the spectrum is 10KHz for a AMPS system. Determine the number of channels available for a FDMA system.

Code: clc; clear all; close all; Bt = 12.5\*10e6; Bg = 10\*10e3; Bc = 30\*10e3; N = (Bt-2\*Bg)/Bc; fprintf('Number of available channels: %d\n', floor(N));

Consider Global System for Mobile, which is a TDMA system that uses 25 MHz for the forward link, which is broken into radio channels of 200 kHz. If 8 speech channels are supported on a single radio channel, and if no guard band is assumed, find the number of simultaneous users that can be accommodated in GSM.

Code: clc; clear all; close all; Bt = 25\*10e6; Bc = 200\*10e3; speech\_channels = 8; N\_Radio\_Channels = Bt/Bc; Total\_Users = N\_Radio\_Channels\*speech\_channels; fprintf("Total number of simultaneous users: %d\n", Total\_Users);

ii) If GSM uses a frame structure where each frame consists of 8 time slots, and each time slot contains 156.25 bits, and data is transmitted at 270.833 kbps in the channel, find (a) the time duration of a bit, (b) the time duration of a slot, (c) the time duration of a frame, (d) how long must a user occupying a single time slot must wait between two simultaneous solutions. Code:

clc; clear all; close all; data\_rate\_kbps = 270.833\*10e3; bits\_per\_slot = 156.25; slots\_per\_frame = 8; time\_duration\_of\_a\_bit = 1 / data\_rate\_kbps; time\_duration\_of\_a\_slot = bits\_per\_slot\*time\_duration\_of\_a\_bit; time\_duration\_of\_a\_frame = slots\_per\_frame\*time\_duration\_of\_a\_slot; wait\_time\_between\_transmissions = time\_duration\_of\_a\_frame-time\_duration\_of\_a\_slot; fprintf('Time Duration of a Bit: %.8f seconds\n', time\_duration\_of\_a\_bit); fprintf('Time Duration of a Slot: %.8f seconds\n', time\_duration\_of\_a\_slot); fprintf('Time Duration of a Frame: %.8f seconds\n', time\_duration\_of\_a\_frame); fprintf('Wait Time Between Two Simultaneous Transmissions: %.8f seconds\n', wait\_time\_between\_transmissions);

iii) If a normal GSM time slot consists of 6 trailing bits, 8.25 guard bits, 26 training bits, and 2 traffic bursts of 58 bits of data, find the frame efficiency. Code: clc;

clear all; close all; trailing\_bits = 6; guard\_bits = 8.25; training\_bits = 26; data\_bits\_per\_burst = 58; number\_of\_bursts = 2; total\_data\_bits = number\_of\_bursts \* data\_bits\_per\_burst; total\_bits\_in\_time\_slot = trailing\_bits + guard\_bits + training\_bits + total\_data\_bits; frame\_efficiency = total\_data\_bits / total\_bits\_in\_time\_slot; fprintf('Frame Efficiency: %.4f\n', frame\_efficiency \* 100);

EXP6

function exp6()

% Main script to take user input and generate PN code

disp('Enter a 4-bit binary array (e.g., [1 0 1 0]):');

array = input('', 's'); % Read user input as a string

array = str2num(array); % Convert string to numeric array

% Debugging output to check the input array

disp('Input array:');

disp(array);

% Validate input

if length(array) != 4

error('Input must be a 4-bit binary array.');

endif

% Validate if all elements are binary

if any(array < 0 | array > 1)

error('Array must contain only binary digits (0 or 1).');

endif

% Initialize PN code and shift register

pn\_code = [];

shift\_register = array;

% Generate PN code (length = 15 for a 4-bit shift register)

for i = 1:15

% Append the leftmost bit to the PN code

pn\_code = [pn\_code, shift\_register(1)];

% Calculate the feedback bit using XOR of specific bits

feedback\_bit = xor(shift\_register(1), shift\_register(2)); % Simple feedback for example

% Shift the register

shift\_register = [shift\_register(2:4), feedback\_bit];

endfor

% Display the PN code and its length

disp('Generated PN code:');

disp(pn\_code);

disp(['Length of PN code: ', num2str(length(pn\_code))]);

endfunction

EXP7   
Task 1.

1. Let N be the number of users in a system, who are allotted Walsh codes.

2. Initialize a variable w=1.

Construct a 2x2 Walsh matrix H. (Hint: H=[w,w,w,~w]

3. Let the number of times this construction be repeated for N users be m. Note that m=ceil(log2(N)).

4. Ok Repeat construction of H m times to accommodate N users.

5. Display H.

CODE:

N = 16;

w = 1;

m = ceil(log2(N));

H = w;

for i = 1:m

H = [H, H; H, -H];

end

disp('Walsh Matrix H:');

disp(H);

TASK 2.

1. Generate the orthogonality test matrix ‘O’ in which the result of orthogonality is stored. For example, the matrix should look as follows for 4 users:

𝑂=[1001000000001001]

Code :

%continued below previous code

O = zeros(N, N);

for i = 1:N

for j = 1:N

O(i, j) = dot(H(i, :), H(j, :)) / N;

end

end

disp('Walsh Matrix H:');

disp(H)

disp('Orthogonality Matrix O:');

disp(O);

**EXP8**

**Write an Octave code to convert given serial bit-stream into QAM symbols and view their modulation by subcarrier of given frequency, with reference to OFDM system.**

clc;

clear all;

close all;

% Define the mapping table for 16-QAM

symbol\_map = {

[0, 0, 0, 0], -3 - 3j;

[0, 0, 0, 1], -3 - 1j;

[0, 0, 1, 0], -3 + 3j;

[0, 0, 1, 1], -3 + 1j;

[0, 1, 0, 0], -1 - 3j;

[0, 1, 0, 1], -1 - 1j;

[0, 1, 1, 0], -1 + 3j;

[0, 1, 1, 1], -1 + 1j;

[1, 0, 0, 0], 3 - 3j;

[1, 0, 0, 1], 3 - 1j;

[1, 0, 1, 0], 3 + 3j;

[1, 0, 1, 1], 3 + 1j;

[1, 1, 0, 0], 1 - 3j;

[1, 1, 0, 1], 1 - 1j;

[1, 1, 1, 0], 1 + 3j;

[1, 1, 1, 1], 1 + 1j;

};

% Step 2: Generate a 64-bit random stream

bit\_stream = round(rand(1, 64)); % Generate random bits (0 or 1)

% Divide the 64 bits into 16 symbols (4 bits each)

symbols = reshape(bit\_stream, 4, 16)';

% Map the 4-bit symbols to QAM symbols using the provided map

function qam\_symbol = map\_to\_qam(symbol, symbol\_map)

idx = find(cellfun(@(x) isequal(x, symbol), symbol\_map(:, 1)));

if ~isempty(idx)

qam\_symbol = symbol\_map{idx, 2};

else

error('Symbol not found in the mapping table.');

end

end

% Convert each 4-bit symbol to the corresponding QAM symbol

qam\_symbols = zeros(1, 16);

for i = 1:16

qam\_symbols(i) = map\_to\_qam(symbols(i, :), symbol\_map);

end

disp('QAM symbols: ');

disp(qam\_symbols);

% Plot 64 serial bits

figure;

subplot(3,1,1);

stem(bit\_stream, 'filled');

title('64-bit Serial Bit Stream');

xlabel('Bit Index');

ylabel('Bit Value');

% Plot real and imaginary parts of the QAM symbols

subplot(3,1,2);

stem(real(qam\_symbols), 'filled');

title('Real Part of QAM Symbols');

xlabel('Symbol Index');

ylabel('Real Part');

subplot(3,1,3);

stem(imag(qam\_symbols), 'filled');

title('Imaginary Part of QAM Symbols');

xlabel('Symbol Index');

ylabel('Imaginary Part');

% Carrier signal generation

fo = 24000000;

to = 1 / fo;

t = 0.01 \* to:0.01 \* to:8 \* to;

carr\_real = cos(2 \* pi \* fo \* t);

carr\_imaj = sin(2 \* pi \* fo \* t);

% Plot the real and imaginary carriers

figure;

plot(t, carr\_real, 'r', t, carr\_imaj, 'b');

title('Real and Imaginary Carriers');

xlabel('Time (s)');

ylabel('Amplitude');

legend('Real Carrier', 'Imaginary Carrier');

% Step 5: Calculate and plot the channel symbols

figure; % Create a new figure for the subplots

for i = 1:16

% Compute the channel symbol for each QAM symbol

real\_part = real(qam\_symbols(i)) \* carr\_real;

imag\_part = imag(qam\_symbols(i)) \* carr\_imaj;

channel\_symbol = real\_part - imag\_part; % Channel symbol

% Plot the channel symbol in a subplot for each QAM symbol

subplot(4, 4, i); % Create a 4x4 grid of subplots

plot(t, channel\_symbol);

title(['QAM Symbol ' num2str(i)]);

xlabel('Time (s)');

ylabel('Amplitude');

end