DIGITAL COMMUNICATIONS

The performance of an OFDM-based communication system.

The Communication System Consist Of

- ❑ A typical baseband OFDM-based transmitter.

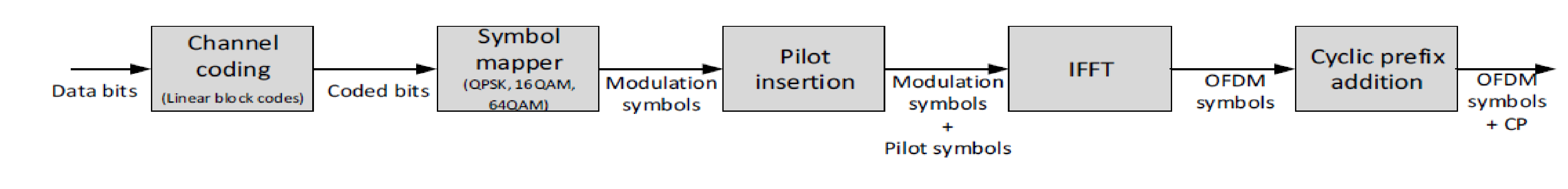
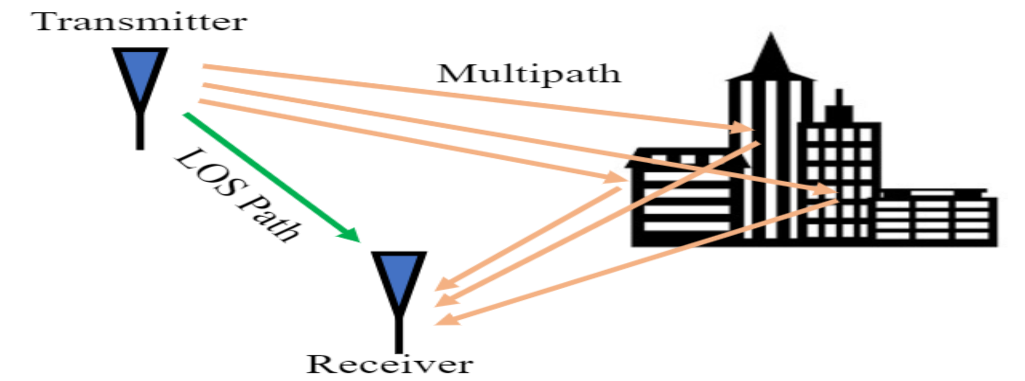


Figure 1 An OFDM-based transmitter.

- ❑ Channel.
  - AWGN
  - Multipass Fading Channel



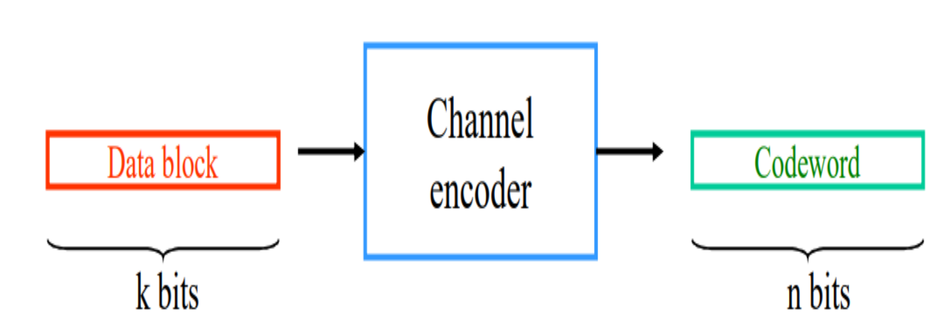
- ❑ OFDM-based receiver.

1. A typical baseband OFDM-based transmitter

- Channel Coding

❖ a (7,4) linear block code

- The information bit stream is chopped into blocks of k bits.
- Each block is encoded to a larger block of n bits.
- The coded bits are modulated and sent over channel.
- The reverse procedure is done at the receiver.



❖ At Tx

G : generatic matrix consist of

$$\mathbf{G} = [\mathbf{P} \;\vdots\; \mathbf{I}_k]$$

$$\mathbf{I}_k = k \times k \ \text{ identity matrix}$$

$$\mathbf{P}_k = k \times (n-k) \ \text{ matrix}$$

$$\boxed{\mathbf{U} = \mathbf{m}\mathbf{G}}$$

$$(u_1, u_2, \ldots, u_n) = (m_1, m_2, \ldots, m_k) \cdot \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \vdots \\ \mathbf{V}_k \end{bmatrix}$$

$$\mathbf{U} = (u_1, u_2, \ldots, u_n) = (\underbrace{p_1, p_2, \ldots, p_{n-k}}_{\text{parity bits}}, \underbrace{m_1, m_2, \ldots, m_k}_{\text{message bits}})$$

MATLAB Function for channel coding

```
function coded_bits = ChannelCoding(bit_seq ,n ,m)

%{
Inputs:
        bit_seq: Data bits
        n: Codeword length (Number of bits in the codeword)
        m: Messege block length in codeword
Outputs:
        coded_bits: Encoded data
Description:
        This function (channel coding block) takes as input the raw data bits ,performs
        a linear block coding operation which takes an input of 4-bit block and map it into a 7-bit codeword
        by multiplying by a 7 Ã— 4 generator matrix to produce a set of coded bits.
%}


% Generator matrix
I = eye(m);   % Identity matrix
A = [1 1 0; 0 1 1; 1 1 1; 1 0 1];   % Parity matrix
G = [I A];

% Split data bits into blocks of lenght m
% (pad with zeros if length(bit_seq) is not divisible by 4)
data_blocks = reshape([bit_seq(:); zeros(mod(-numel(bit_seq),m),1)], m, [])';

% Generate codewords
coded_bits = rem(data_blocks * G, 2);
coded_bits = reshape(transpose(coded_bits), 1, []);     % convert matrix to row vector
```

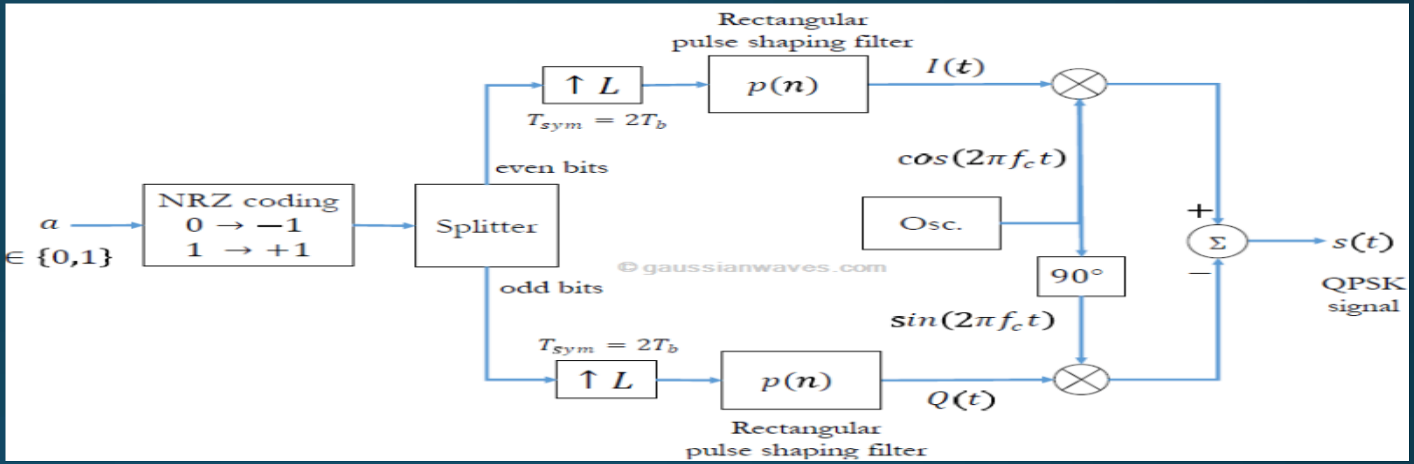
- Symbol mapper
- The symbol mapper takes the coded bits and produces a corresponding set of modulation symbols according to the modulation technique used

1) QPSK. 2) 4QAM. 3) 64QAM.

At Tx

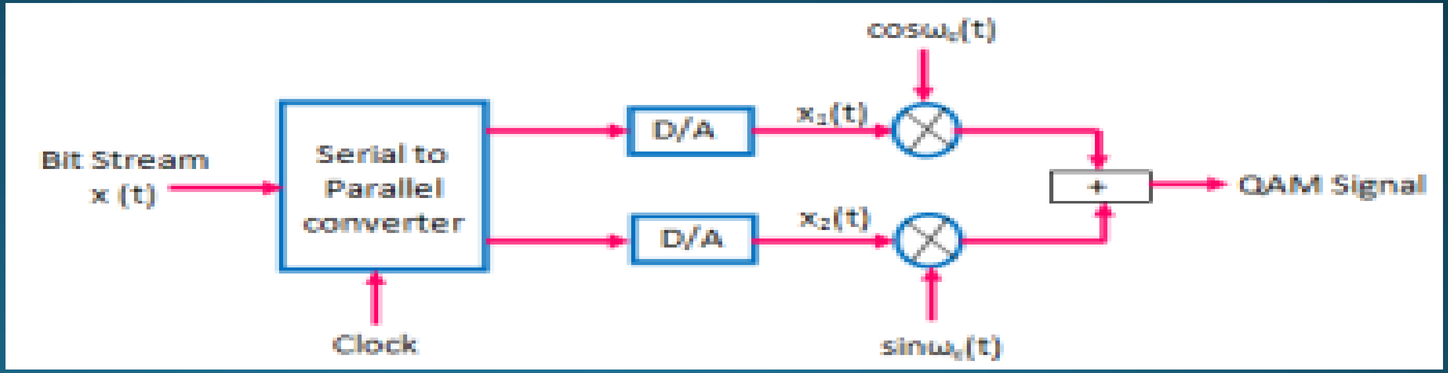
1) QPSK.

$X_{QPSK} = (A/\sqrt{2})(b_E + j\, b_o)$



2) M-QAM.

$X_{QAM} = A_c\,(A_{mi} + j\, A_{mq})$



MATLAB Function for modulation symbols

```matlab
function mod_symbols = TXSymbolMapper(coded_bits ,Mod_type)
% Inputs:
%        N_bits: Total number of bits
%        coded_bits: encoded bits to be symbols
%        Mod_type: Type of modulation technique (1: QPSK , 2: 16QAM , 3: 64QAM)
% Outputs:
%        mod_symbols: Modulated symbols
% Description:
%        This function (symbol mapper block) takes the coded bits and produces a corresponding
%        set of modulation symbols according to the modulation technique used (QPSK ,16QAM ,64QAM)
if Mod_type == 1     %QPSK
    %extend if not even
    if mod(length(coded_bits),2) == 1
        coded_bits=[coded_bits 0];
    end
     % polar NRZ
     m_polar = 2*coded_bits - 1;

     % Demultiplexing input into even and odd streams
     even_bit_stream = m_polar(1:2:end);  % even indeces
     odd_bit_stream = m_polar(2:2:end);   % odd indeces
     mod_symbols = (1/sqrt(2))*(even_bit_stream + i*odd_bit_stream);

elseif Mod_type == 2
    M = 16;              %number of symbols

elseif Mod_type == 3
    M = 64;              %number of symbols
end
```

```matlab
if (Mod_type == 2 || Mod_type == 3)      % 16QAM or 64QAM
       K = log2(M);          %number of bits per symbol
       %extend coded bits if the number of coded bits is not multiple of K
       if(mod(length(coded_bits),K) ~= 0)
           new_coded_bits = [coded_bits zeros(1,K-mod(length(coded_bits),K))];
       else
           new_coded_bits = coded_bits;
       end
       mod_symbols = zeros(1,length(new_coded_bits)/K);

       for c = 1:length(mod_symbols)
           %index of the start of each symbol
           first_num_index = (K*(c-1))+1;

           %get the first K/2 bits for the inphase carrier
           gray = reshape(char(new_coded_bits(first_num_index:first_num_index+(K/2)-1)+48),[1,K/2]);
           % convert gray symbol to decimal
           horz = gc2dec(gray);

           %get the last K/2 bits for the quadrature carrier
           gray = reshape(char(new_coded_bits(first_num_index+K/2:first_num_index+K-1)+48),[1,K/2]);
           % convert gray symbol to decimal
           vert = gc2dec(gray);

           % Getting inphase component (Branch 1) & quadrature component (Branch 2)
           inphase = 2*(horz+1)-1-sqrt(M);
           quad = 2*(vert+1)-1-sqrt(M);
           mod_symbols(c) = inphase + i*quad;
       end
end
```

- IFFT Block

Implementing the Orthognal Frequancy Division Multiplexing (OFDM) is exactly the IFFT operation .

  - The output samples from the IFFT are called The "OFDM " Symbols
  - Each one OFDM Symbol carries Nc data symbols.
  - Data symbols are thought as being specified in the frequency domain where each subcarrier carries a symbol.

- Cyclic prefix
  - It's a modification that we make to the OFDM symbol in order to combat ISI between consecutive OFDM symbols
  - It's makes each sub channel really see a flat channel.

MATLAB Function for add cyclic prefix

```matlab
function [resultVector] = addCP(Vector,meo)
    [numberOfIterations symbolSize] =size(Vector);
    %SeriesVector = reshape(Vector,1,[]);
    SeriesVector = reshape(Vector.',1,[]);
    resultVector = [];
    vectorSize= length(SeriesVector);
    for k = 0 : numberOfIterations-1
        resultVector = [resultVector SeriesVector(k*symbolSize+symbolSize-meo+1 : (k+1)*symbolSize)];
        resultVector = [resultVector SeriesVector(k*symbolSize+1 : (k+1)*symbolSize)];
    end

end
```

2.The Channel

2.1.AWGN Channel

```
function noise =AWGNwithN0(N0,L,MT,OFDMSymbol_CP,DT)

if DT==1
    if MT==1
        noise=sqrt(N0/2)*(randn(1,L)+1i*randn(1,L))+OFDMSymbol_CP;
    elseif MT==2
        noise=sqrt(N0/2)*(randn(1,L)+1i*randn(1,L))+OFDMSymbol_CP;
    elseif MT==3
        noise=sqrt(N0/2)*(randn(1,L)+1i*randn(1,L))+OFDMSymbol_CP;
    end
elseif DT==2
    if MT==1
        noise=[sqrt(N0/2)*(randn(1,L)+1i*randn(1,L));sqrt(N0/2)*(randn(1,L)+1i*randn(1,L))]+[OFDMSymbol_CP;OFDMSymbol_CP];
    elseif MT==2
        noise=[sqrt(N0/2)*(randn(1,L)+1i*randn(1,L));sqrt(N0/2)*(randn(1,L)+1i*randn(1,L))]+[OFDMSymbol_CP;OFDMSymbol_CP];
    elseif MT==3
        noise=[sqrt(N0/2)*(randn(1,L)+1i*randn(1,L));sqrt(N0/2)*(randn(1,L)+1i*randn(1,L))]+[OFDMSymbol_CP;OFDMSymbol_CP];
    end
end
end
```

2.2.AWGN Channel with Fading

```
function [noise,channelresponse, phase1, phase2]=RayleighFading(DT,OFDMSymbol_CP,N0,MT)
h1=1/sqrt(2*50)*(randn(1,50)+1i*randn(1,50));
h2=1/sqrt(2*50)*(randn(1,50)+1i*randn(1,50));
h_simo=[h1; h2];
if DT==1
    noise_symbol=conv(OFDMSymbol_CP,h1);
    noise=noise_symbol;
    noise=AWGNwithN0(N0,length(noise_symbol),MT,noise_symbol,DT);
    channelresponse=h1;
    phase1=zeros(1,length(noise_symbol));
    for i=1:length(noise_symbol)
        phase1(1,i)=CartoPolar(noise_symbol(1,i));
    end
    phase2=0;
elseif DT==2
    noise_symbol1=conv(h1,OFDMSymbol_CP);
    noise_symbol2=conv(h2,OFDMSymbol_CP);
    noise_symbol1=AWGNwithN0(N0,length(noise_symbol1),MT,noise_symbol1,1);
    noise_symbol2=AWGNwithN0(N0,length(noise_symbol2),MT,noise_symbol2,1);
    noise=[noise_symbol1;noise_symbol2];
    channelresponse=h_simo;
    phase1=zeros(1,length(noise_symbol1));
    for i=1:length(noise_symbol1)
        phase1(1,i)=CartoPolar(noise_symbol1(1,i));
    end
    phase2=zeros(1,length(noise_symbol2));
    for i=1:length(noise_symbol2)
        phase1(1,i)=CartoPolar(noise_symbol2(1,i));
    end
```

MATLAB Function for No calculation

```
function No=Calc_No(Eb_No_dB,Mod_type)
% assumption : A^2 & Ts/2 = 1
Eb_No_linear = 10.^(Eb_No_dB./10);       % linear scale

if Mod_type == 1      % QPSK
    M=4;
     Eb = 1/(log2(M));
 elseif Mod_type == 2    % 16QAM
     M=16;
     Eb = 2*(M-1)/(3*log2(M));
elseif Mod_type == 3    %64QAM
    M=64;
    Eb = 2*(M-1)/(3*log2(M));
end

No = Eb./Eb_No_linear;     % PSD noise (No)

end
```

3. OFDM-based receiver.

- Remove Cyclic Prefix
- FFT Block
- Demodulat The received symbol

At Rx

1) QPSK.

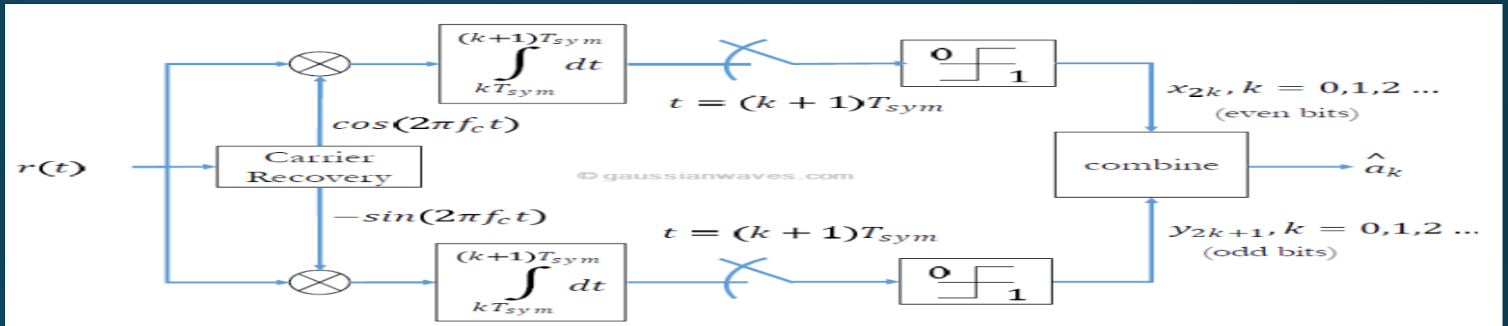
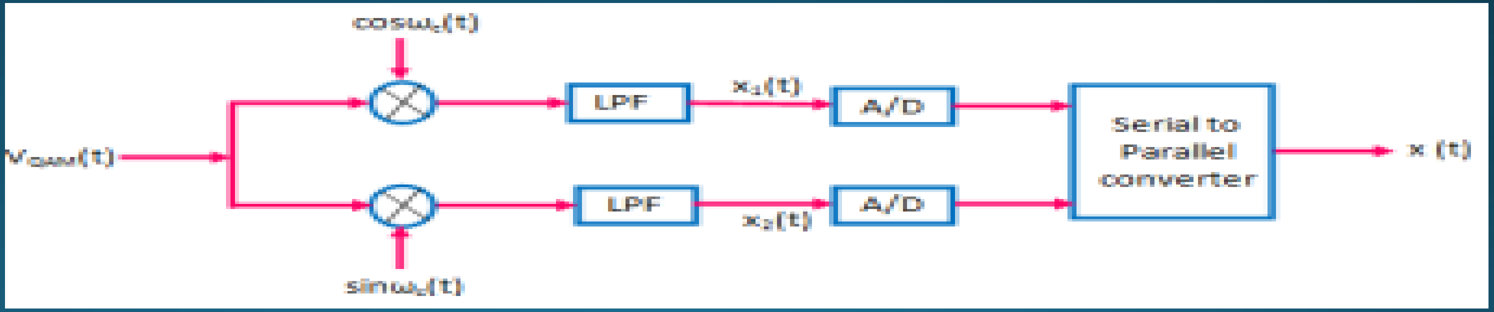


Figure 3: Waveform simulation model for QPSK demodulation

2) M-QAM.



MATLAB code for getting the received signal by SIMO system

```
function r= SimoSystem(y,h)
    %h1=1/sqrt(2*50)*(randn(1,50)+1i*randn(1,50));
    %h2=1/sqrt(2*50)*(randn(1,50)+1i*randn(1,50));
    %h = [h1 h2];
    w = h./norm(h);
    r= w'.*y;
    norm(w);
end
```

MATLAB Function for remove cyclic prefix

```
function [ recoveredVector] = removeCP(Vector,symbolSize,meo)
    unit = symbolSize + meo;
    recoveredVector = [];
     recoveredVector2 = [];
    vectorSize= length(Vector);
    numberOfIterations = vectorSize / unit;


    for k = 0 : numberOfIterations-1
       recoveredVector = [recoveredVector; Vector(k*unit+meo+1 : (k+1)*unit)];
    end
%     for i = 0 : numberOfIterations-1
%        recoveredVector2 = [recoveredVector2 Vector(2,i*unit+meo+1 : (i+1)*unit)];
%     end
%     recoveredVector = [recoveredVector1;recoveredVector2];
%     if DT==1
%         recoveredVector = [reshape(recoveredVector1',[], 100)];
%     else
%         recoveredVector = [reshape(recoveredVector1',[], 100)  reshape(recoveredVector2',[] , 100) ];
%     end
    %recoveredVector2 = reshape(recoveredVector2,[],symbolSize);
end
```

MATLAB Function for demodulation symbols

```matlab
function demod_data = RXSymbolMapper(N_bits ,n ,m ,mod_symbols ,Mod_type)
% Inputs:
%        N_bits: Total number of bits
%        n: Codeword length (Number of bits in the codeword)
%        m: Messege block length in codeword
%        mod_symbols: Received symbols to be decoded
%        Mod_type: Type of demodulation technique (1: QPSK , 2: 16QAM , 3: 64QAM)
% Outputs:demod_data: Decoded data
% Description:
%        This function (demodulation block) takes as input received symbols ,
%        demodulate them to codewords
if Mod_type == 1       %QPSK
    r1 = real(mod_symbols);
    r2 = imag(mod_symbols);
    threshold = 0;
    r1_demod = r1>threshold;
    r2_demod = r2>threshold;
    % Multiplexing output odd and even streams
    %demod_data = zeros(1,1024*2);
    demod_data = zeros(1,length(mod_symbols)*2);
    demod_data(1:2:end) = r1_demod;
    demod_data(2:2:end) = r2_demod;
    if length(demod_data)~= ceil(N_bits+N_bits/m*(n-m))
        demod_data = demod_data(1:ceil(N_bits+N_bits/m*(n-m)));
    end
elseif Mod_type == 2
   M = 16;                 %number of symbols
elseif Mod_type == 3
   M = 64;                 %number of symbols
end
```

```matlab
if (Mod_type == 2 || Mod_type == 3)       % 16QAM or 64QAM
        K = log2(M);              %number of bits per symbol
        demod_qam = mod_symbols;
        estimated = -(sqrt(M)-1):2:(sqrt(M)-1);
        threshold = estimated(1,2:end)-1;
        threshold = [-Inf threshold Inf];
        demod_data = zeros(1,length(mod_symbols)*K);
        check = 0;
        for c = 1:length(demod_qam)
            first_num_index = (K*(c-1))+1;
            inphase = real(demod_qam(c));
            quad = imag(demod_qam(c));
            for j=1:sqrt(M)
                if(inphase >= threshold(j) && inphase < threshold(j+1))
                    inphase = estimated(j);
                    check = check+1;
                end
                if(quad >= threshold(j) && quad < threshold(j+1))
                    quad = estimated(j);
                    check = check+1;
                end
                if(check == 2)
                    check=0;
                    break;
                end
            end
            horz = ((inphase+sqrt(M)+1)/2)-1;
            vert = ((quad+sqrt(M)+1)/2)-1;
            horz_gc = dec2gc(horz,K/2);
```

horz_gc = dec2gc(horz,K/2);

```matlab
            vert_gc = dec2gc(vert,K/2);
            for j=1:K/2
                demod_data(first_num_index+j-1) = double(horz_gc(j)-48);
            end
            for j=(K/2)+1:K
                demod_data(first_num_index+j-1) = double(vert_gc(j-(K/2))-48);
            end
        end
        if length(demod_data)~= ceil(N_bits+N_bits/m*(n-m))
            demod_data = demod_data(1:ceil(N_bits+N_bits/m*(n-m)));
        end
end
```

- The Channel decoding

H : Parity check matrix consist of

$$\mathbf{H} = [\mathbf{I}_{n-k} \;\vdots\; \mathbf{P}^T]$$

$$\mathbf{G}\mathbf{H}^T = \mathbf{0}$$

$$\mathbf{r} = \mathbf{U} + \mathbf{e}$$

$\mathbf{r} = (r_1, r_2, \ldots, r_n)$ received codeword or vector

$\mathbf{e} = (e_1, e_2, \ldots, e_n)$ error pattern or vector

Syndrome testing

S is syndrome of r, corresponding to the error pattern e.

$$\mathbf{S} = \mathbf{r}\mathbf{H}^T = \mathbf{e}\mathbf{H}^T$$

MATLAB Function for channel decoding

```
function decoded_bits = ChannelDecoding(n ,m ,demod_data)
% Inputs:
%         n: Codeword length (Number of bits in the codeword)
%         m: Messege block length in codeword
%         coded_bits: Received bits to be decoded
% Outputs: decoded_bits: Decoded data
% Description:
%         This function (channel decoding block) takes as input received codewords decode them to
%         4-bit blocks and compute syndrom on the data to correct errors.
% Parity check matrix
r = demod_data;    % received data
r = reshape([r'; zeros(mod(-numel(demod_data),n),1)], n, [])';      % convert row vector to matrix
% Parity check matrix
I = eye(n-m);
A = [1 1 0; 0 1 1; 1 1 1; 1 0 1];
A_T = transpose(A);
H = [A_T I];
% Syndrome
S = rem(H*transpose(r), 2);
% Correct errors (in case of 1 error)
for i = 1:size(S, 2)
    for j = 1:size(H, 2)
        if S(:, i) == H(:, j)
            r(i, j) = ~r(i, j);
        end
    end
end
rec_data_blocks = r(:, 1:4);
decoded_bits = reshape(transpose(rec_data_blocks), 1, []);   % convert matrix to row vector
```

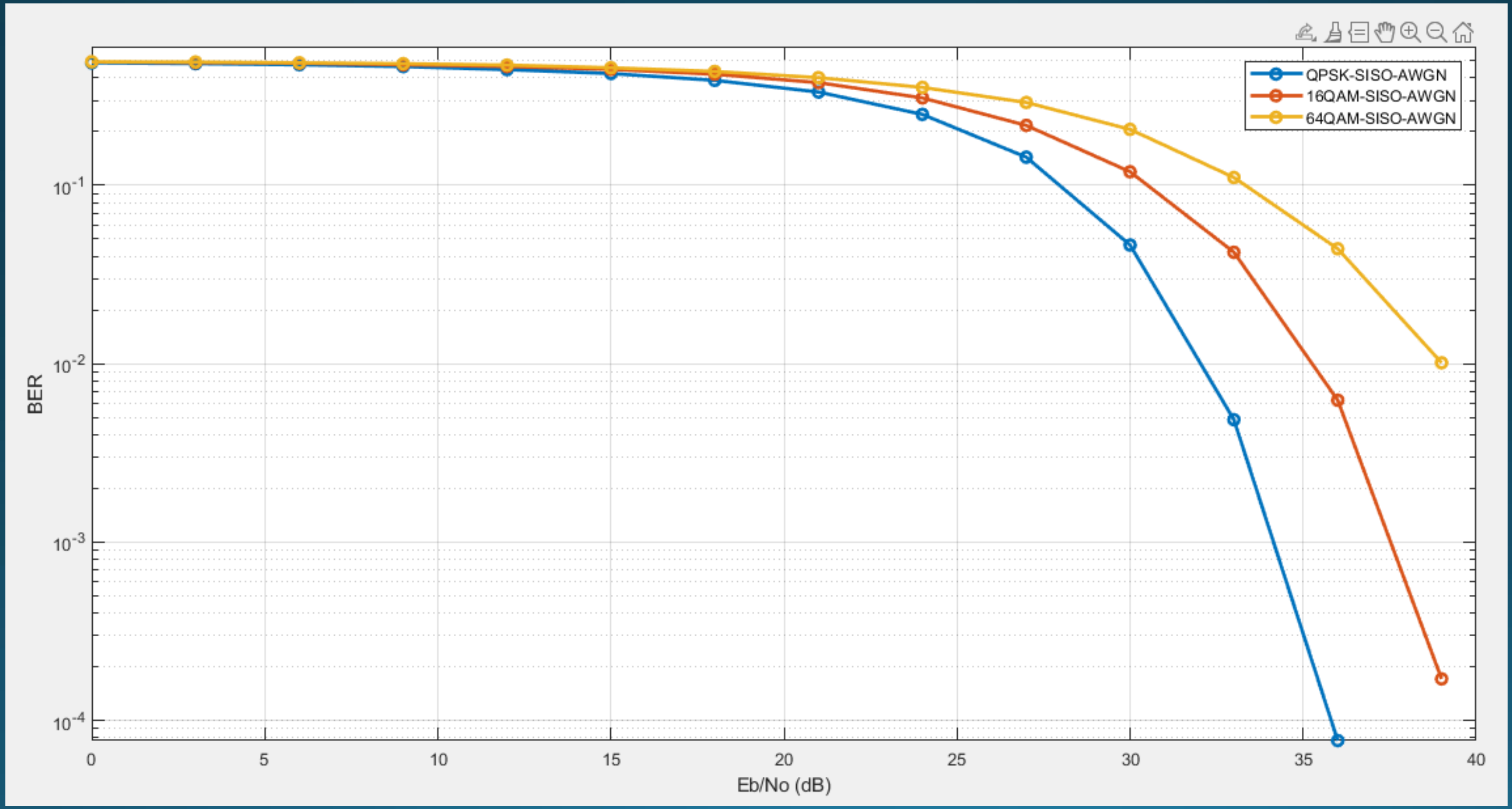
• Steps of code

1. Generating a sequence of bits equal to the total number of bits.
2. Pass the coded bits through the channel coding and symbol mapper blocks.
3. Generating OFDM Symbols.
4. Adding Cyclic prefix.
5. Pass OFDM Symbols through the Channel.
    a) AWGN Channel with zero mean and variance No/2.
    b) Multipath Fading Channel with zero mean and variance 1.
6. Get The received signal by
    a) Single Antenna.(SISO System)
    b) Multiple Antennas.(SIMO System)
7. Removing Cyclic prefix.
8. Demodulate Codewords from received symbols then decode bits from demodulated codeword.
9. Finally Compute Eb/No.

Results

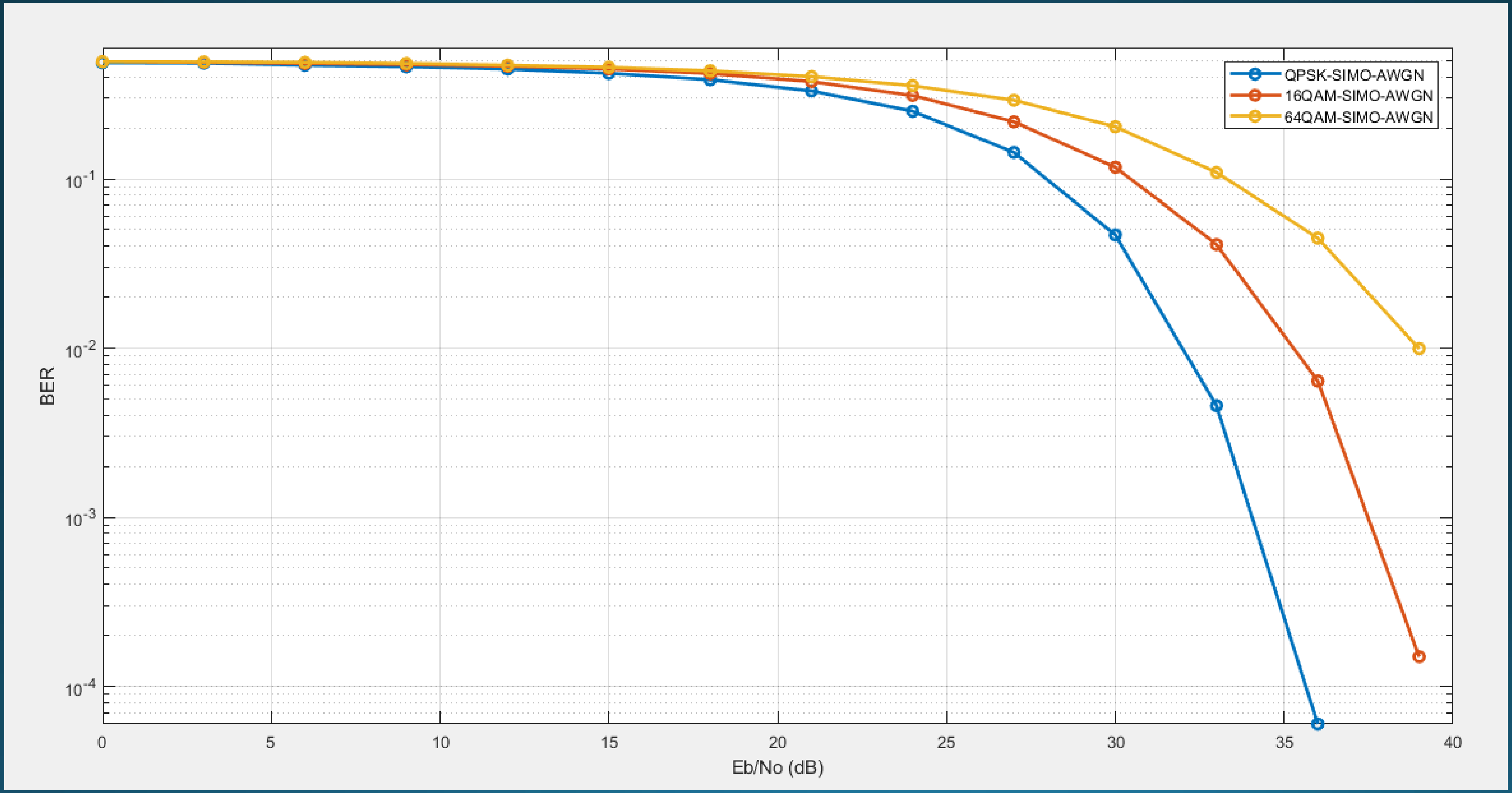
Plot BER VS Eb/No (db)

(QPSK/16QAM/64QAM)_SISO _AWGN



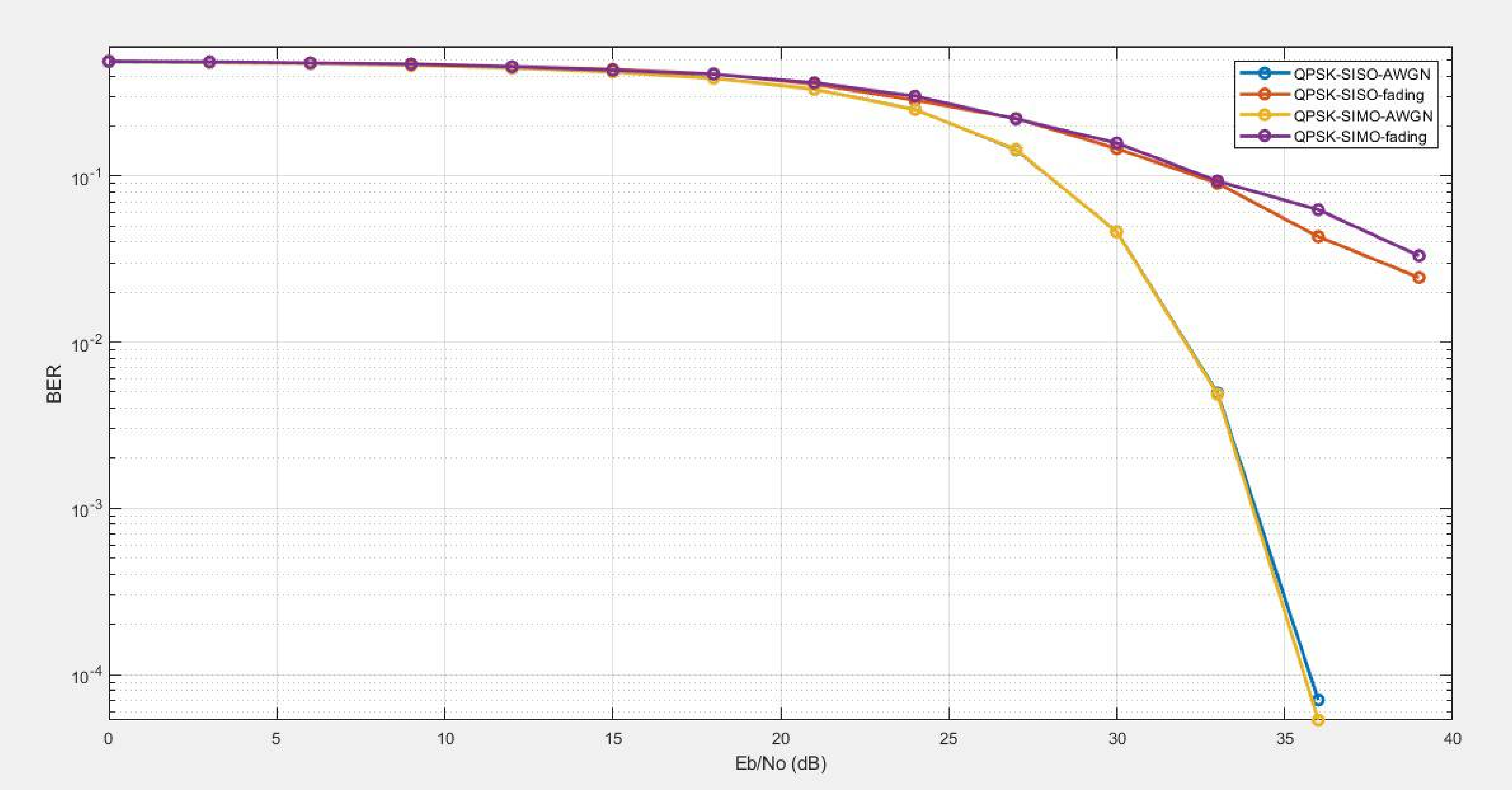
Plot BER VS Eb/No (db)

(QPSK/16QAM/64QAM)_SIMO _AWGN



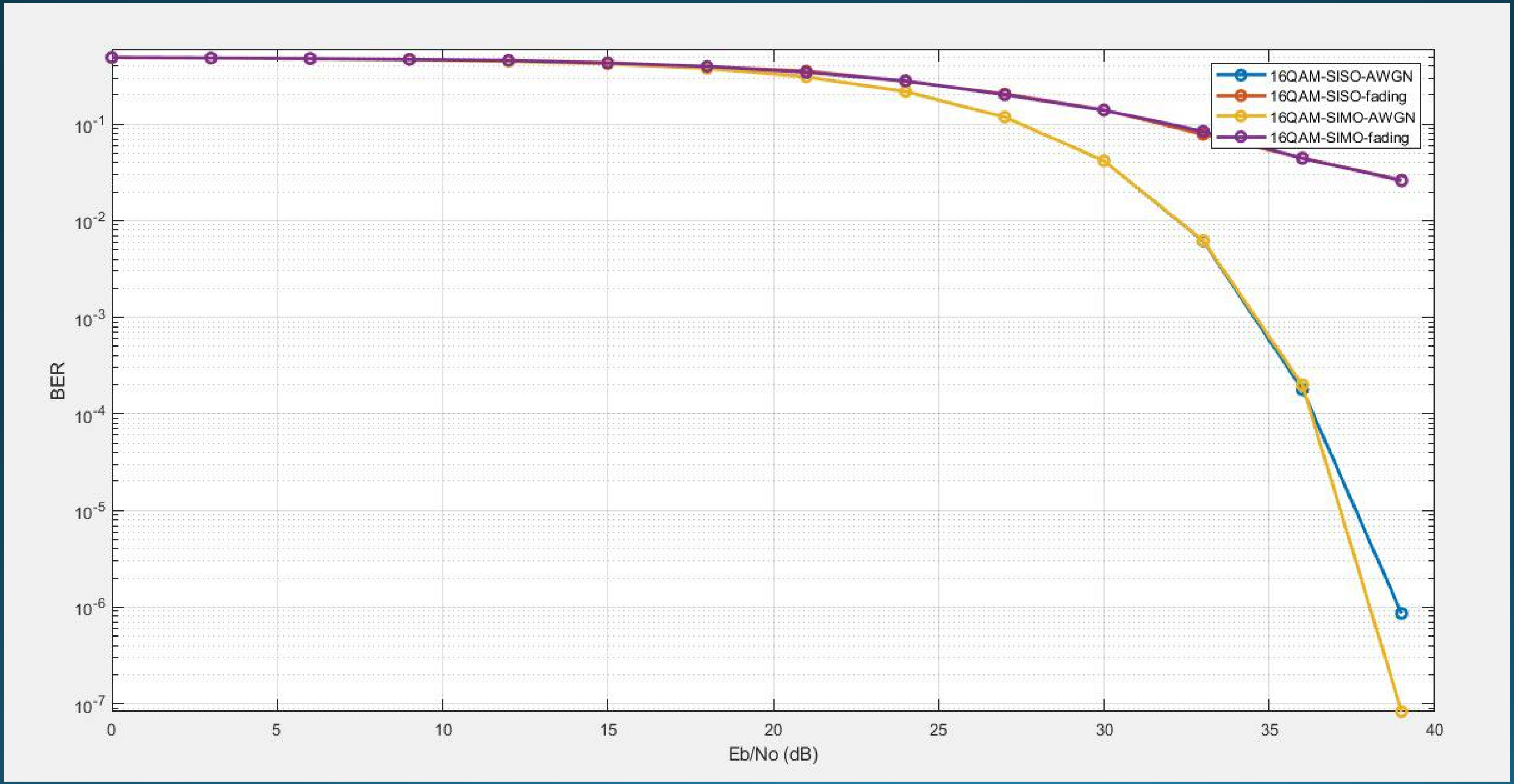
Plot BER VS Eb/No (db)

QPSK _(SISO /SIMO)_(AWGN/Fading)



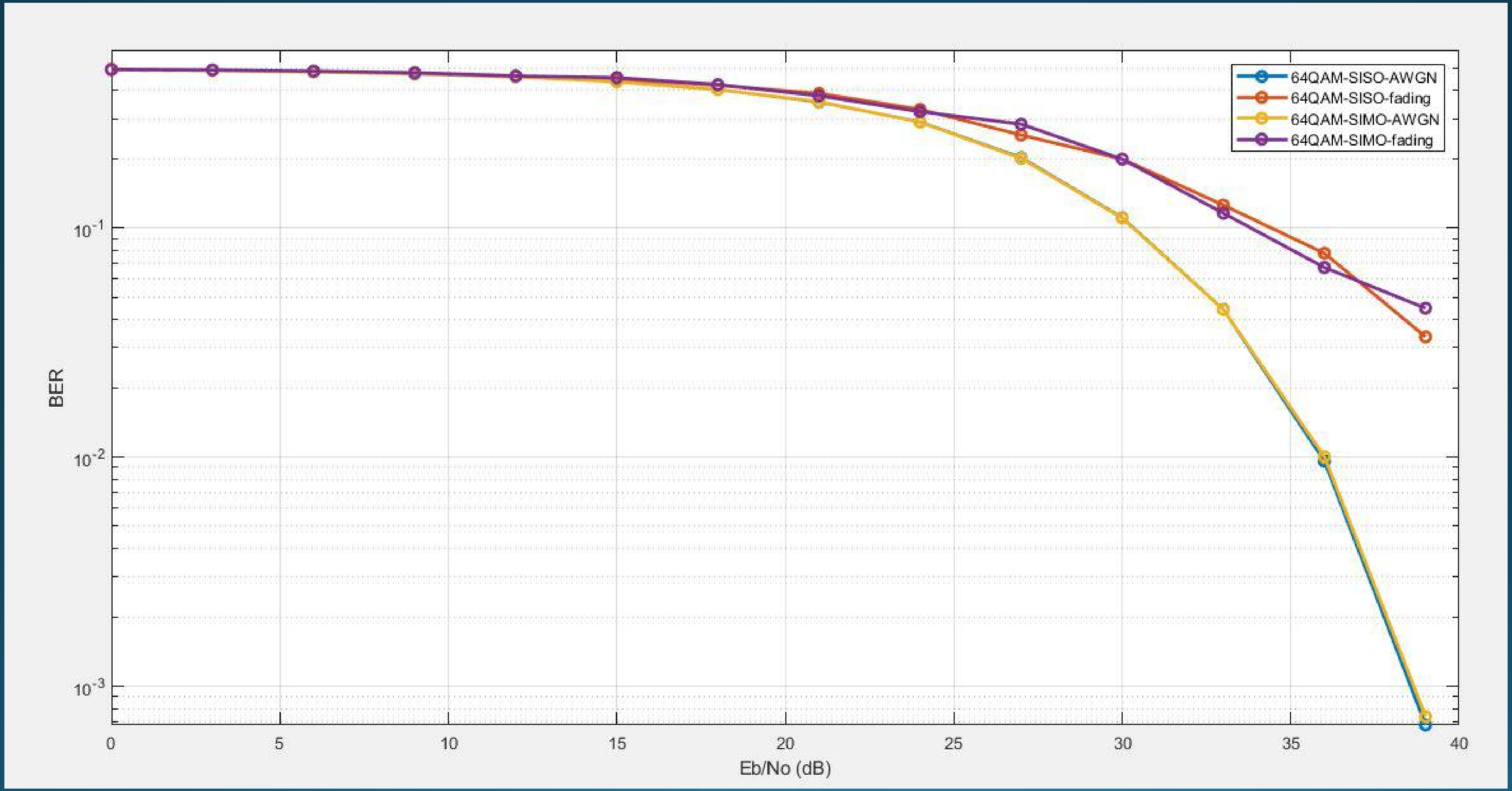
Plot BER VS Eb/No (db)

16QAM _(SISO /SIMO)_(AWGN/Fading)



Plot BER Vs Eb/No (db)

64QAM _(SISO /SIMO)_(AWGN/Fading)



ID	SEC	الاسم	
18011867	8	منة الله محمد عبدالهادى عبدالفتاح طه	1
18010344	8	أسماء جمال عبدالله إبراهيم سالم	2
18010513	8	جهاد جمال محمد محمد الكومى	3
18010001	8	ابانوب إبراهيم ينى عبدالملاك	4
15010473	2	أسماء جمال عبدالحليم مبروك ناجى	5
18010835	8	سيمون جوزيف شحاته عبدالملاك	6
18011303	8	مارك جورج لويز بطرس	7
18011937	8	ميرنا منصور كمال	8
18011310	8	مارينا فؤاد عزيز	9
18010788	1	سعيد مبروك سعيد الشيخ	10