QAM-16 M-ary Signaling with Quadrature

Aim- To simulate M-ary signaling for a coherent transmitter and receiver. Our implementation specifically applies Quadrature Amplitude Modulation (QAM) with 4 in-phase and 4-quadrature signal amplitudes. The amplitudes will be +-1p(t) and +-3p(t) for both in-phase and quadrature components. Our signaling pulse, p(t), will be the frequency-and-space efficient "raised cosine filter".

The transmitter transmits 50000 up-sampled QAM symbols. At receiver side, Matched filter is applied, and output of matched filter is sampled. Threshold detection is based on energies of the signal. We, for our case, calculated three thresholds to detect the four symbols in both real and imaginary part of sampled signal.

Threshold (1) = (Ep+(3*Ep))/2

= 2Ep =12

Threshold (2) = (Ep+(-Ep))/2

= 0

Threshold (3) = (-Ep+(-3*Ep))/2

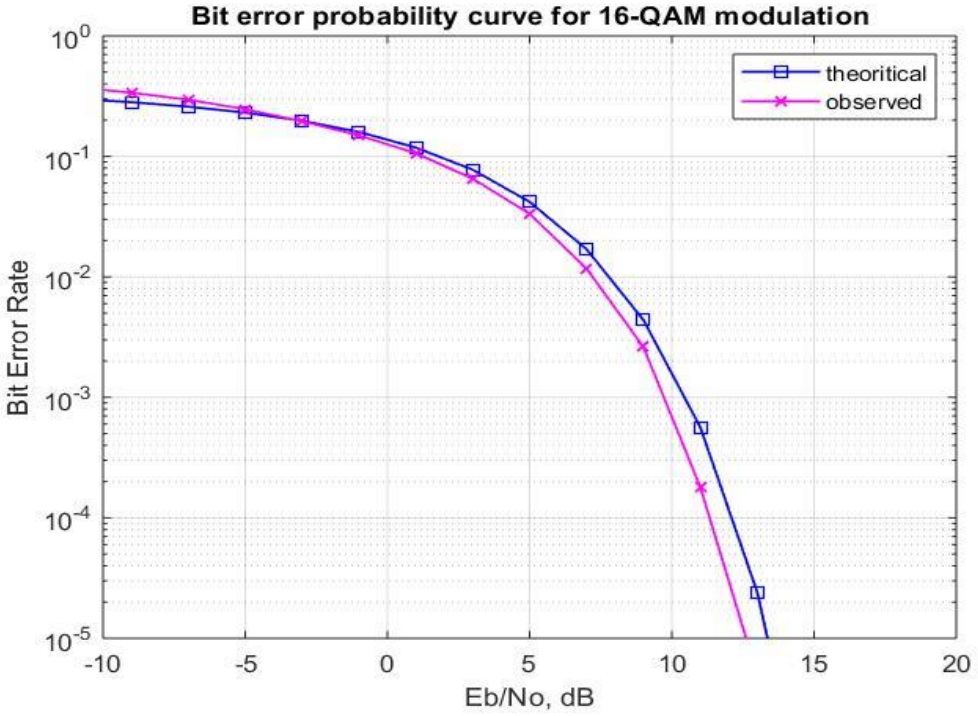
= -2Ep = -12

After Thresholds are applied and symbols are decoded back to corresponding binary bits, Bit error rate is calculated and plotted with theoretical bit error rate for SNR -10 dB to 20 dB.

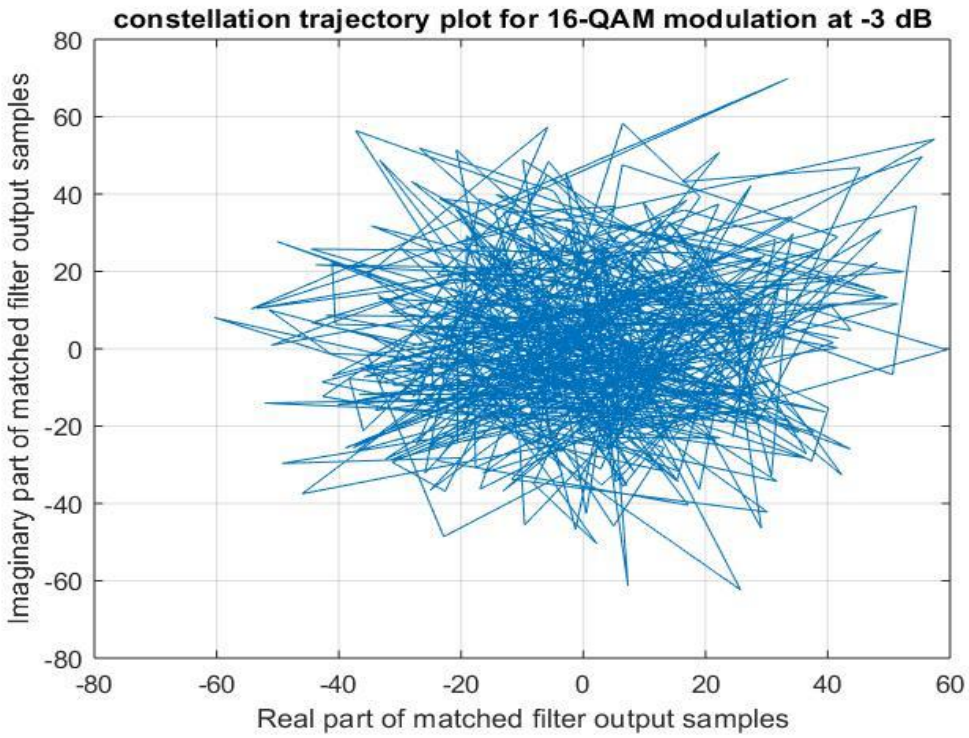
Also, plotted the scatter plot, constellation trajectory plot and eye diagram for -3dB and 20 db. As we have taken pulse energy(Ep) as 6, hence we get scatter and constellation plot energies near to 18, 6, -6, and 18.

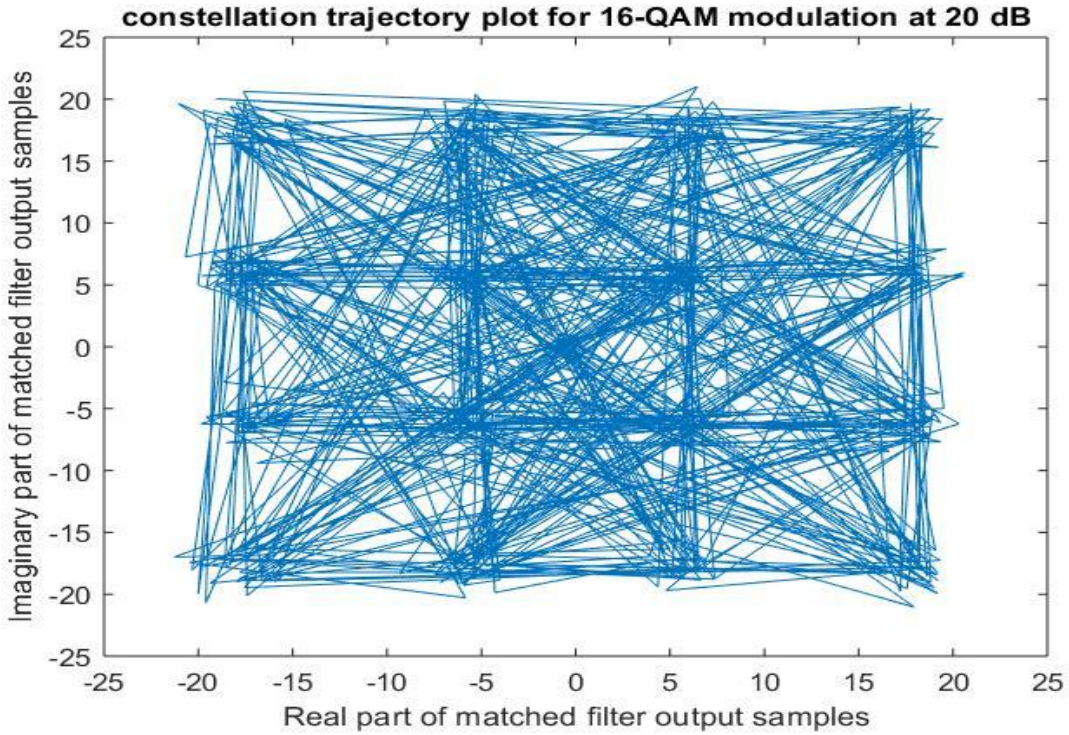
Simulation Results:

1. Bit error rate plot for -10 dB to 20 dB.

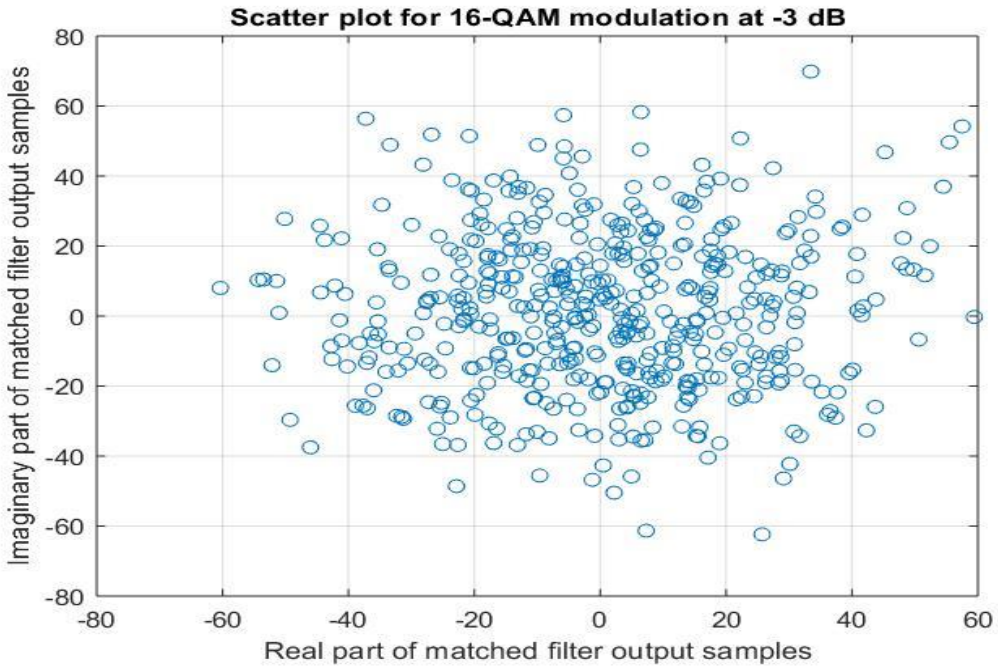


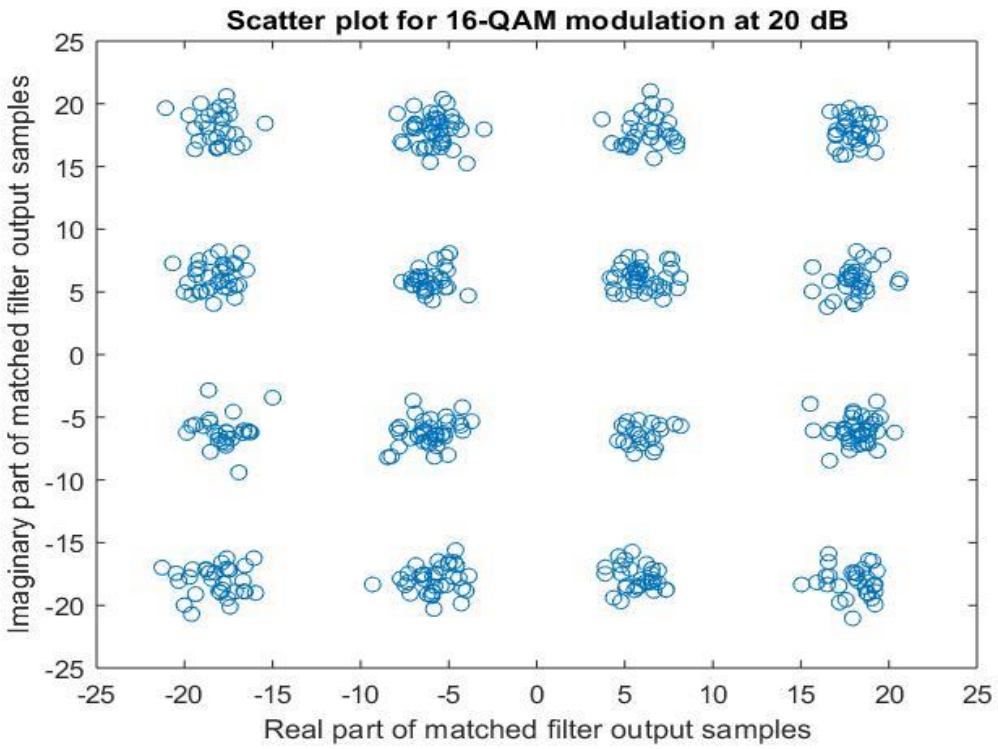
2. Constellation trajectory plot for -3 dB and 20 dB.





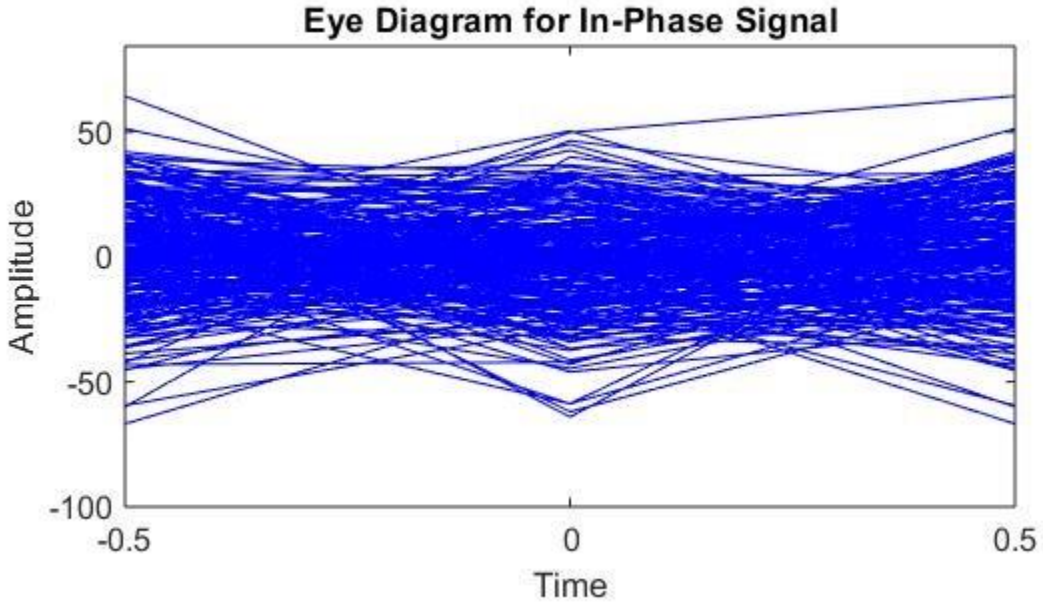
3. Scatter plot of matched filter output for -3dB and 20 dB.



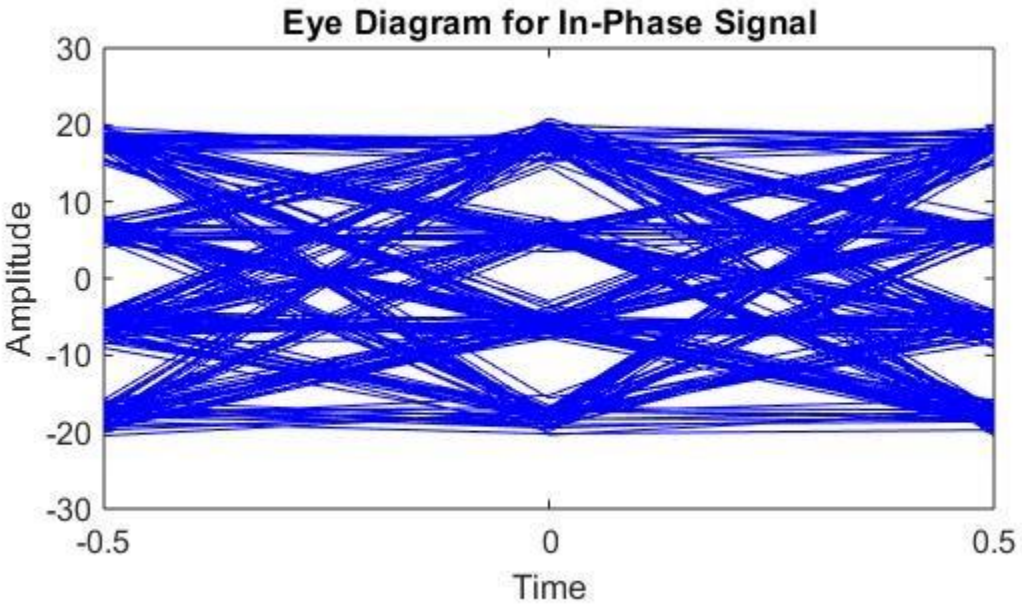


4.Eye diagram for -3 dB and 20 dB.

For SNR -3dB



For SNR 20 dB



Matlab code:

```
clc;
clear;
clear all;
samplesPerSymbol=8;
p_pulseEnergy = 6;
numSymbols = 50000;
numSimulations = 1;
noiseMean = 0;
numPlotSym=500;
f_oversamp=8;
group_delay=4;
p_of_t=rcosflt([1], 1, f_oversamp, 'sqrt', 0.5,
group_delay)';
p_of_t=p_of_t(1:end-8);
p_of_t = p_of_t*(sqrt(p_pulseEnergy/(p_of_t*p_of_t')));
transmittedSymbols=rand(1,4*numSymbols);
transmittedSymbols(transmittedSymbols>0.5)=1;
transmittedSymbols(transmittedSymbols<=0.5)=0;
idx7 = 1;
idx8 = 1;
for ii = -5:1:10

idx = 1;
```

```
for i = 1:2:length(transmittedSymbols)-1
if ((transmittedSymbols(i) == 0) &&
(transmittedSymbols(i+1) == 0))
    tx_sig(idx) = -3;
end
if ((transmittedSymbols(i) == 0) &&
(transmittedSymbols(i+1) == 1))
    tx_sig(idx) = -1;
end
if ((transmittedSymbols(i) == 1) &&
(transmittedSymbols(i+1) == 0))
    tx_sig(idx) = 3;
end
if ((transmittedSymbols(i) == 1) &&
(transmittedSymbols(i+1) == 1))
    tx_sig(idx) = 1;
end
idx = idx + 1;
end
idx1 = 1;
for k =1:2:length(tx_sig)-1
 tx_sym(idx1)= tx_sig(k) + j*tx_sig(k+1);
 idx1 = idx1 +1;
end
up_sampled = upsample(tx_sym,f_oversamp);
delay_fun = 2* group_delay * f_oversamp;
pulseshaped_signalling = conv(p_of_t,up_sampled);
signal_length = length(pulseshaped_signalling);
eb2n(idx7) = ((ii)*2)+10*log10(1/(sqrt(10)));
eb_num(idx7) = 10.^(eb2n(idx7)/10);
noiseVariance(idx7) = 26/(2*eb_num(idx7));
noise = noiseMean +
sqrt(noiseVariance(idx7)/2)*((randn(1,signal_length)) +
(j*randn(1,signal_length)));
transmitted_signal = pulseshaped_signalling + noise;
idx2 =  1;
idx9 = 1;
for l = 1:8:((length(transmitted_signal)- 71)+1)
    op = conv(transmitted_signal(l:l+70),fliplr(p_of_t));
    rx_op1(idx9:idx9+133) = op;
    sam_op(idx2) = op(64);
    idx2 = idx2 + 1;
    idx9 = idx9 + 134;
end
```

```
threshold_1 = 2*p_pulseEnergy;
threshold_2 = -2*p_pulseEnergy;
idx3 = 1;
idx5 = 1;
idx6 = 1;
for v = 1:1:length(sam_op)
    if (real(sam_op(v)) > threshold_1)
        rx_sym(idx3) = 3;
        rx_bit(idx6:idx6+1)= [1,0];
    end
    if (real(sam_op(v)) <= threshold_1 && real(sam_op(v))>
0)
        rx_sym(idx3) = 1;
        rx_bit(idx6:idx6+1)= [1,1];
    end
     if (real(sam_op(v)) >= threshold_2 && real(sam_op(v))<
0)
        rx_sym(idx3) = -1;
        rx_bit(idx6:idx6+1)= [0,1];
     end
    if (real(sam_op(v))< threshold_2 )
        rx_sym(idx3) = -3;
        rx_bit(idx6:idx6+1)= [0,0];
    end
    if (imag(sam_op(v)) > threshold_1)
        rx_sym_imag(idx5) = 3;
        rx_bit(idx6+2:idx6+3)= [1,0];
    end
    if (imag(sam_op(v)) <= threshold_1 && imag(sam_op(v))>
0)
        rx_sym_imag(idx5) = 1;
        rx_bit(idx6+2:idx6+3)= [1,1];
    end
     if (imag(sam_op(v)) >= threshold_2 && imag(sam_op(v))<
0)
        rx_sym_imag(idx5) = -1;
        rx_bit(idx6+2:idx6+3)= [0,1];
     end
    if (imag(sam_op(v)) < threshold_2 )
        rx_sym_imag(idx5) = -3;
        rx_bit(idx6+2:idx6+3)= [0,0];
    end
    idx3 = idx3+1;
    idx5 = idx5 +1;
```

```
    idx6 = idx6 + 4;
end
    y = bitxor(transmittedSymbols,rx_bit);
    bit_err(idx7) = (numel(find(y)))/(4*numSymbols);
    theoryBer(idx7) =
(1/4)*(3/2)*erfc(sqrt(4*0.1*eb_num(idx7)));

idx4 = 1;
idx8 = idx8+1;
for b = 1:1:length(rx_sym)
    rx_op(idx4) = rx_sym(b) + j*(rx_sym_imag(b));
    idx4 = idx4 +1;
end

idx7 = idx7 +1;
end
figure(1),semilogy(eb2n,theoryBer,'bs-','LineWidth',1);
hold on;
figure(1),semilogy(eb2n,bit_err,'mx-','LineWidth',1);
axis([-10 20 10^-5 1])
grid on
xlabel('Eb/No, dB')
ylabel('Bit Error Rate')
title('Bit error probability curve for 16-QAM modulation')
legend('theoritical','observed');
figure(2),plot(real(sam_op(1:500)),imag(sam_op(1:500)),'o')
;
grid on
xlabel('Real part of matched filter output samples')
ylabel('Imaginary part of matched filter output samples')
title('Scatter plot for 16-QAM modulation at 20 dB')
figure(3),plot(real(sam_op(1:500)),imag(sam_op(1:500)));
grid on
xlabel('Real part of matched filter output samples')
ylabel('Imaginary part of matched filter output samples')
title('constellation trajectory plot for 16-QAM modulation
at 20 dB')
eyediagram(sam_op(1:numPlotSym),2);
```