EE544 PROJECT #1
Simulation of Digital Modulation Techniques

Youngseok Lee - 4930239194
Jongseok Lee - 7320360939
Yigao Shao - 6285358317

Submitted on Apr/16/2015

1. Summary

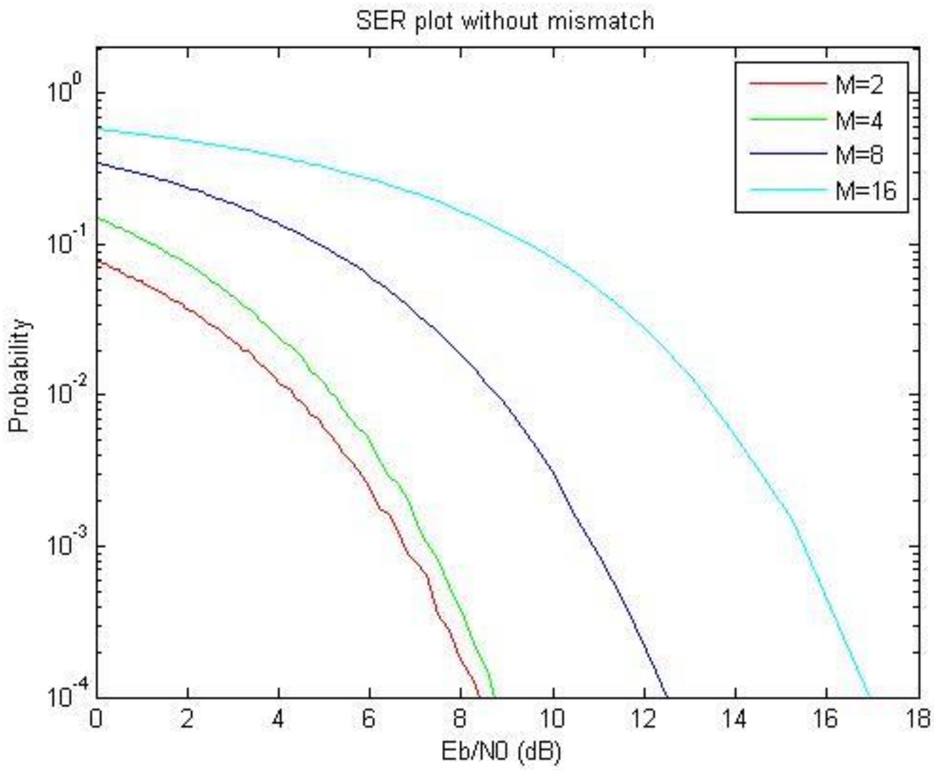
For simulations of BPSK and QPSK with amplitude and phase mismatch, the simulated results show a discrepancy that with mismatch the probability of Bit Error Rate (BER) decrease. For 8-PSK and 16-PSK, BER increase when there are phase and magnitude mismatches.

The graphs comparing the theoretical and simulated result show that the minimum probability of the simulated data can scale down only to 4*10^-6(1/250000), while the ideal qfunction results are able to scale down to infinitesimal values. This was due to the finite sample number 250000 limited the range. Except the range issue, resulted comparison data do match.

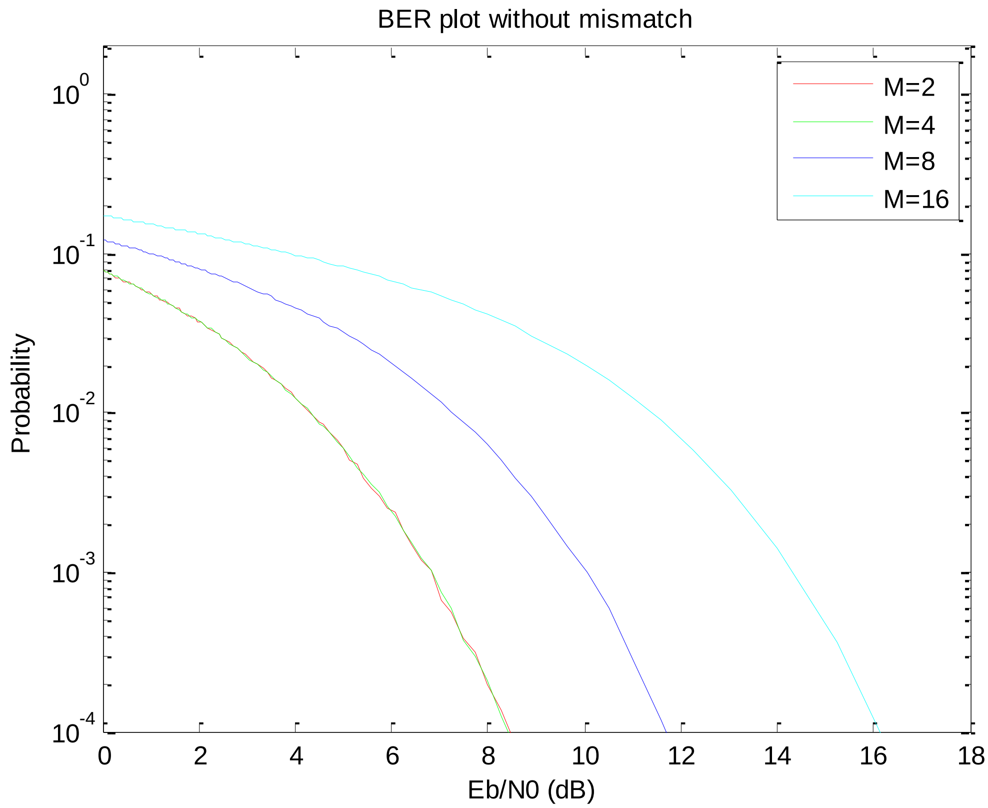
Simulink simulation results are attached in accordance with the data from Matlab M-file simulation.

2. M-ary Phase Shift Keying(M-PSK)

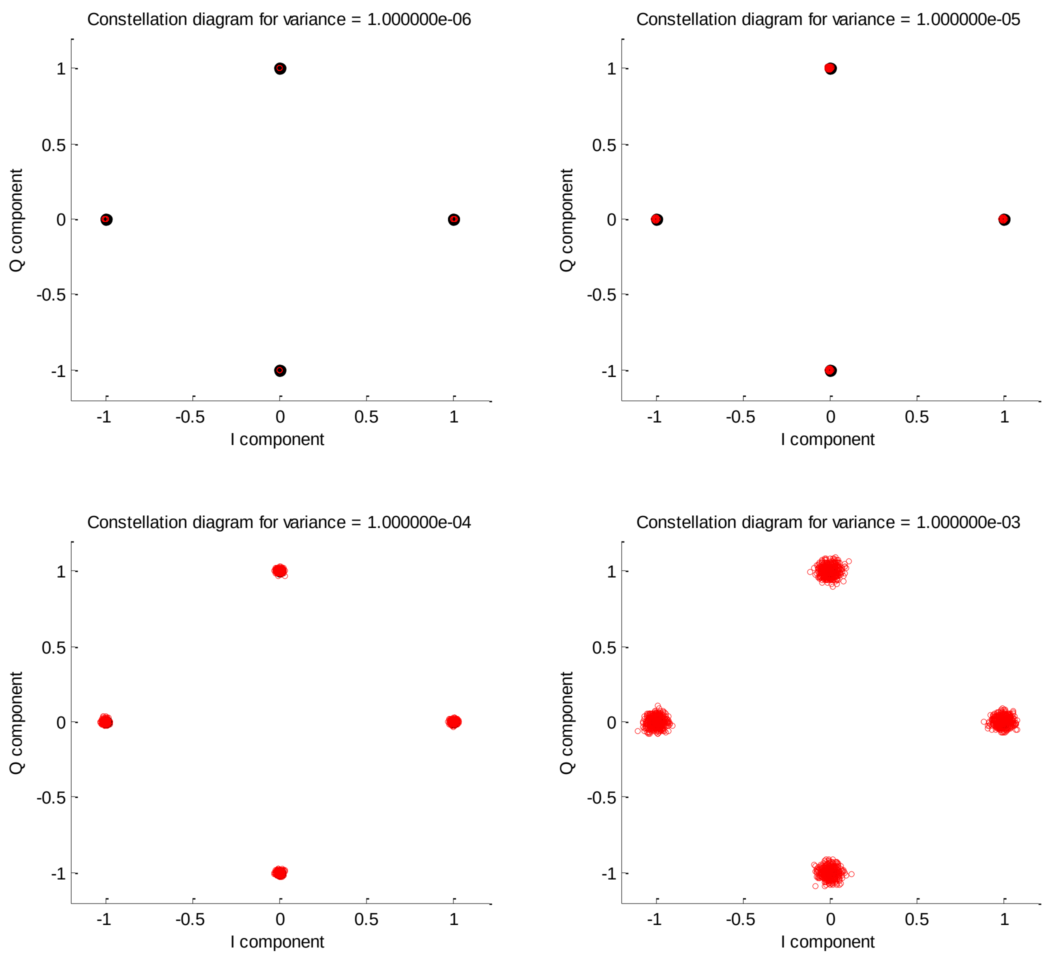
2.1 Symbol Error Rate (SER) vs Eb/N0

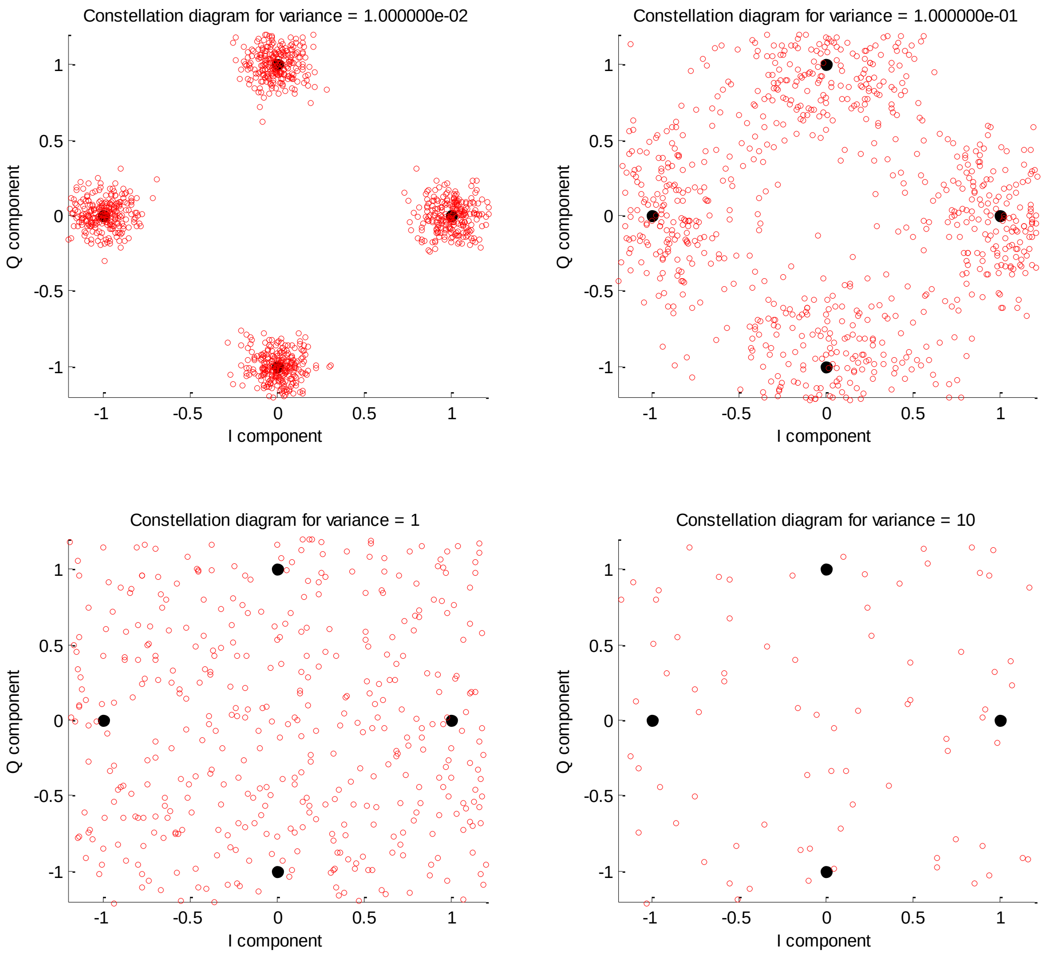


2.2 Bit Error Rate (BER) vs Eb/N0

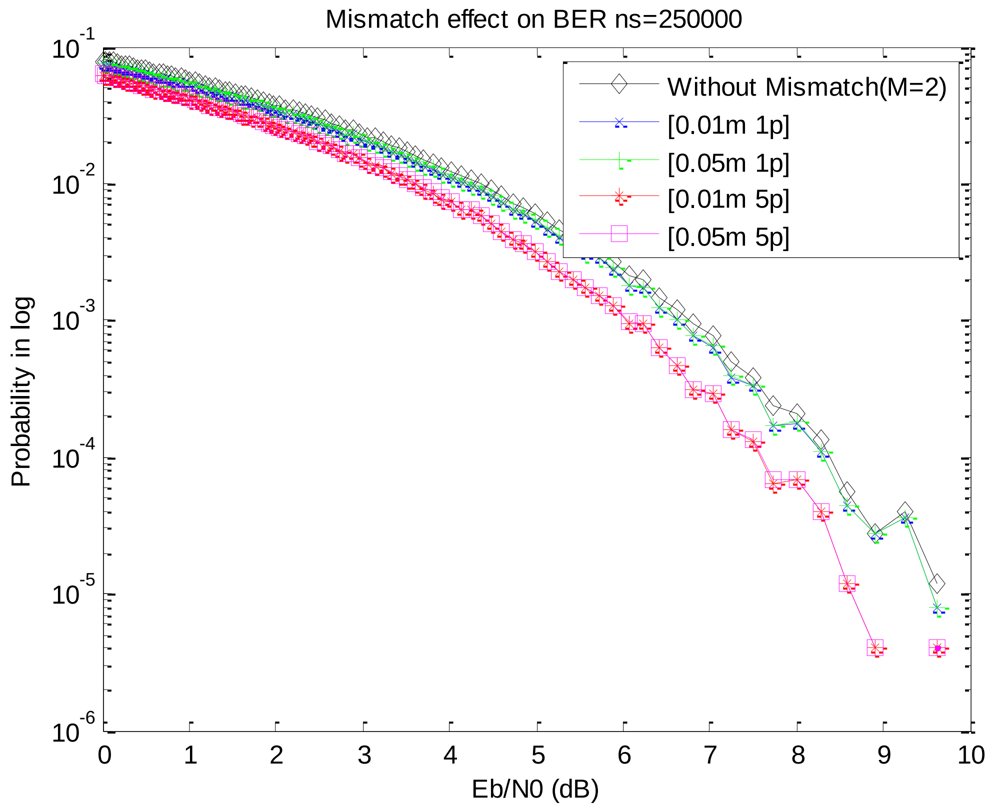


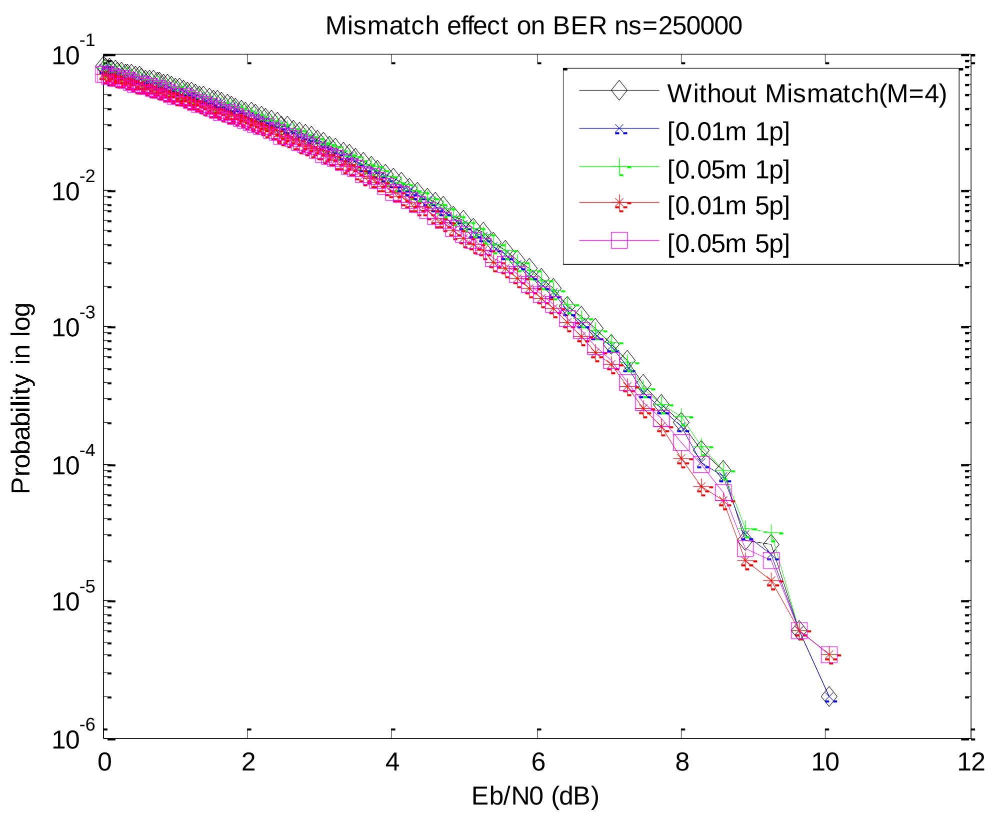
2.3 Signal Constellation in Presence of Noise

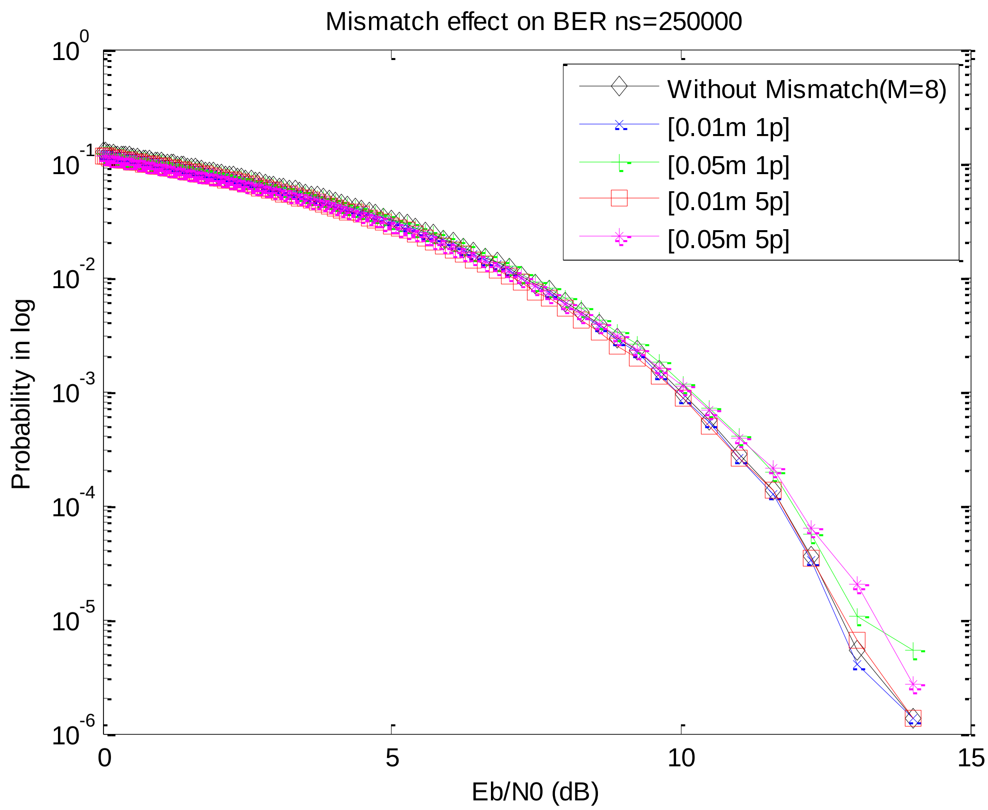


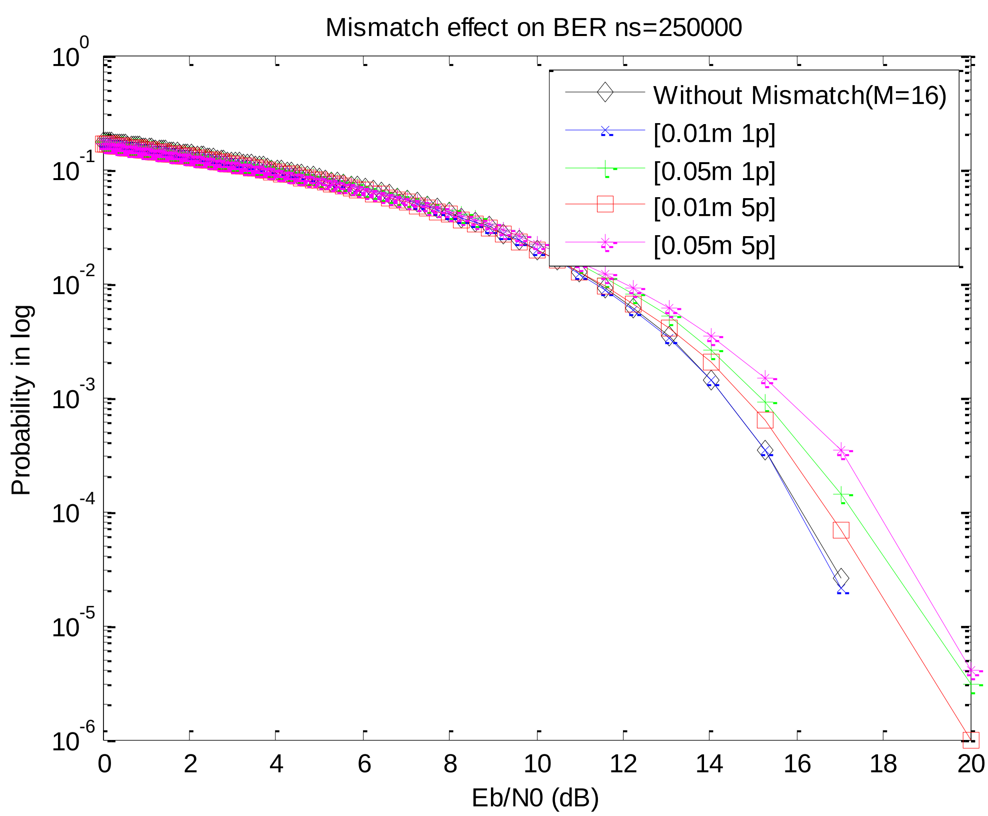

Constellation diagram for variance = 1.000000e-02
Constellation diagram for variance = 1.000000e-01
Constellation diagram for variance = 1
Constellation diagram for variance = 10
Q component
I component
1
0.5
0
-0.5
-1

2.4 Mismatch Effects (Non-ideal Transmitter)

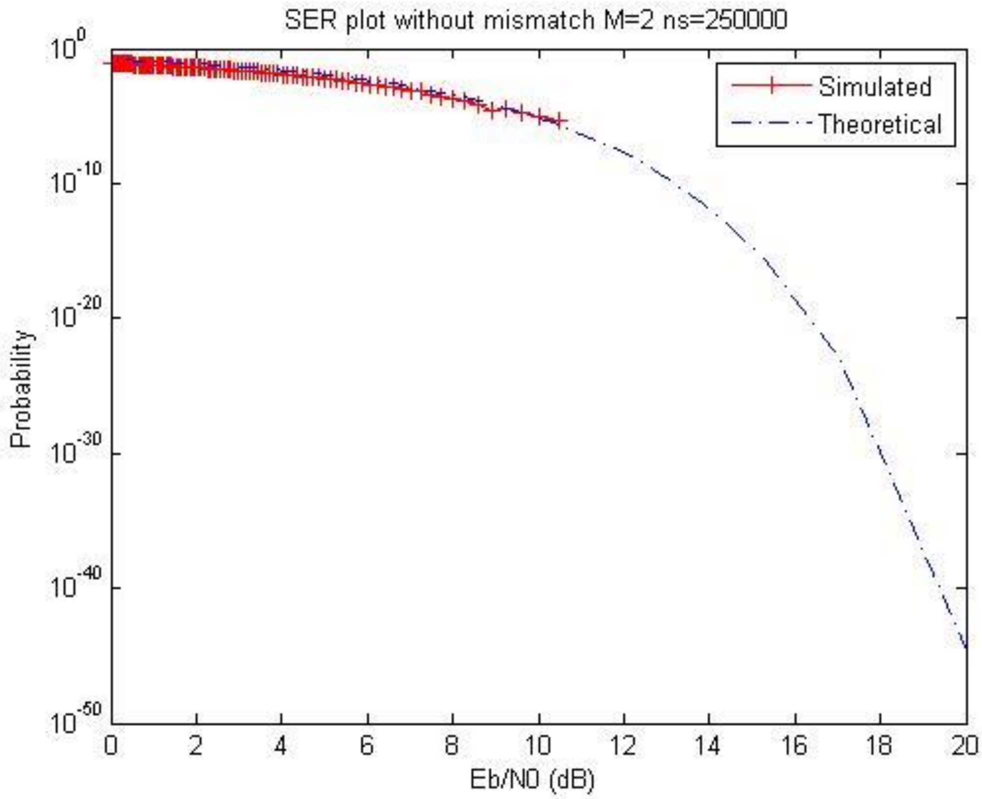


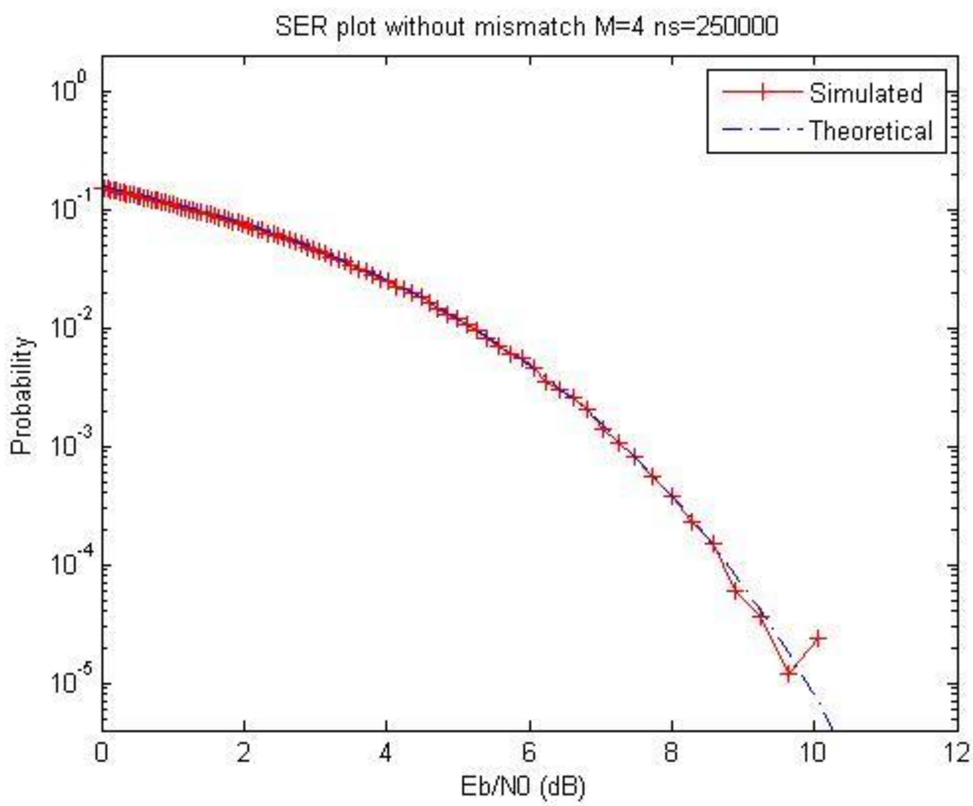


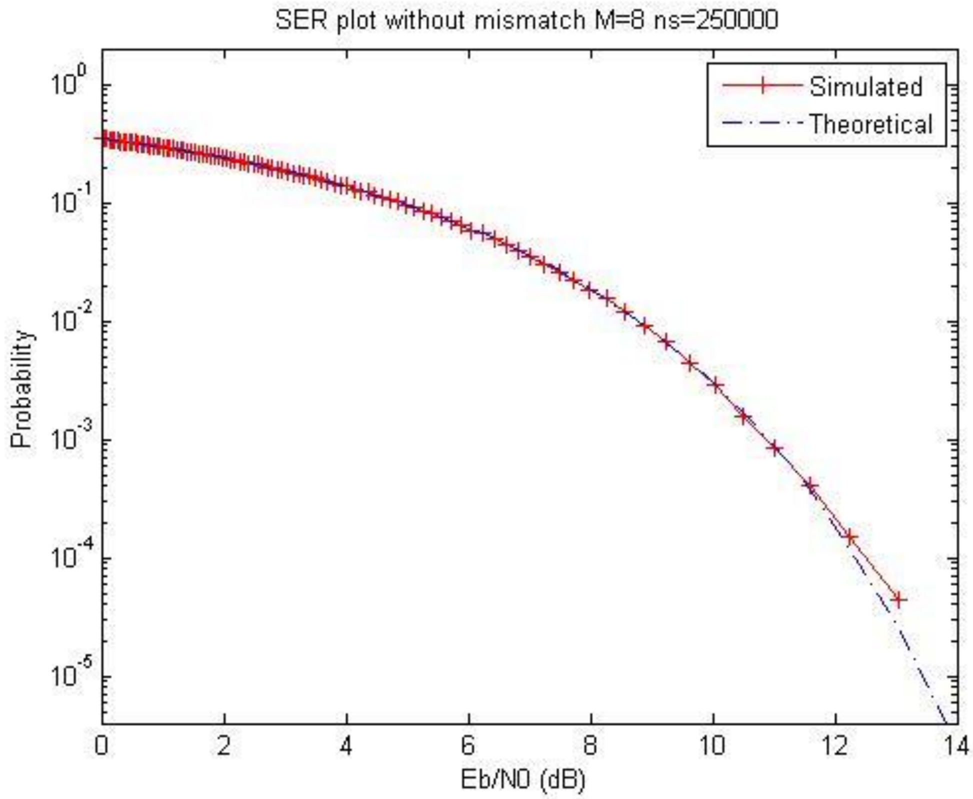

Mismatch effect on BER ns=250000
Probability in log
Eb/N0 (dB)
Without Mismatch(M=8)
[0.01m 1p]
[0.05m 1p]
[0.01m 5p]
[0.05m 5p]
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
0
5
10
15

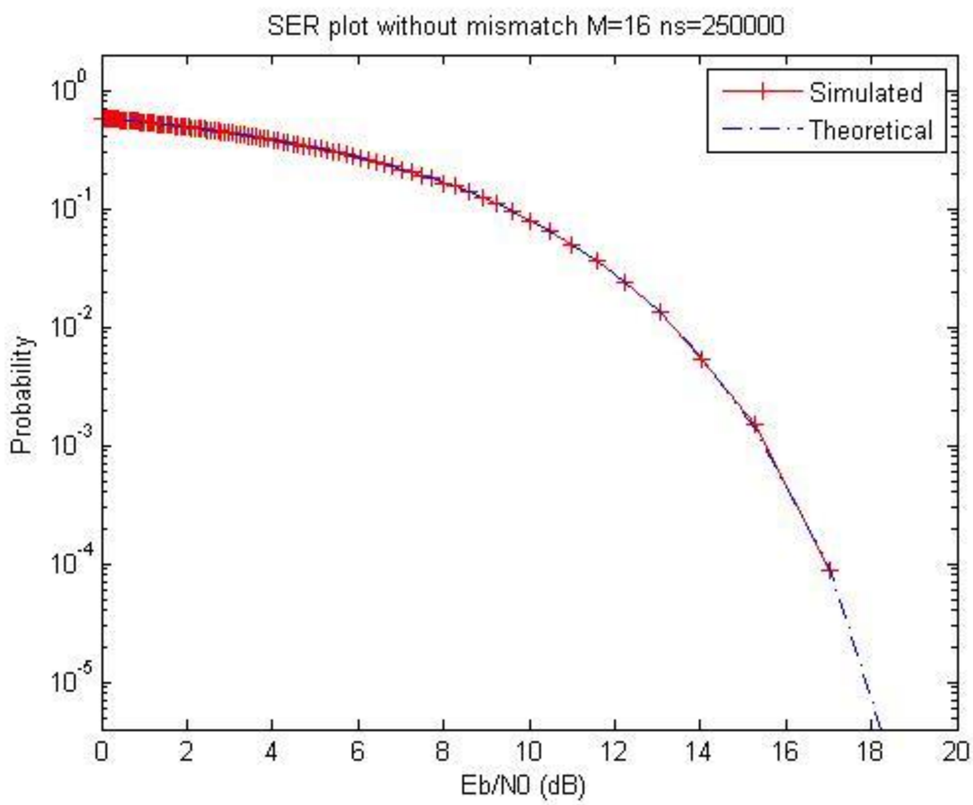

Mismatch effect on BER ns=250000
Probability in log
Eb/N0 (dB)
Without Mismatch(M=16)
[0.01m 1p]
[0.05m 1p]
[0.01m 5p]
[0.05m 5p]
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
0
2
4
6
8
10
12
14
16
18
20

2.5 Comparison Graphs (Theoretical vs Simulated Results)

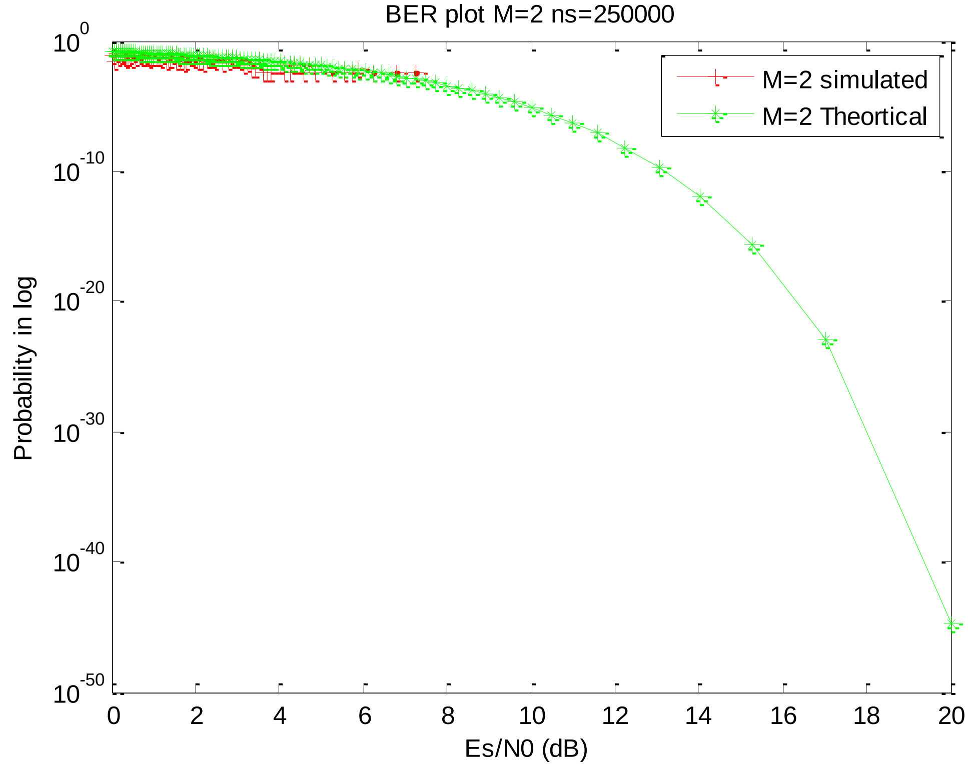


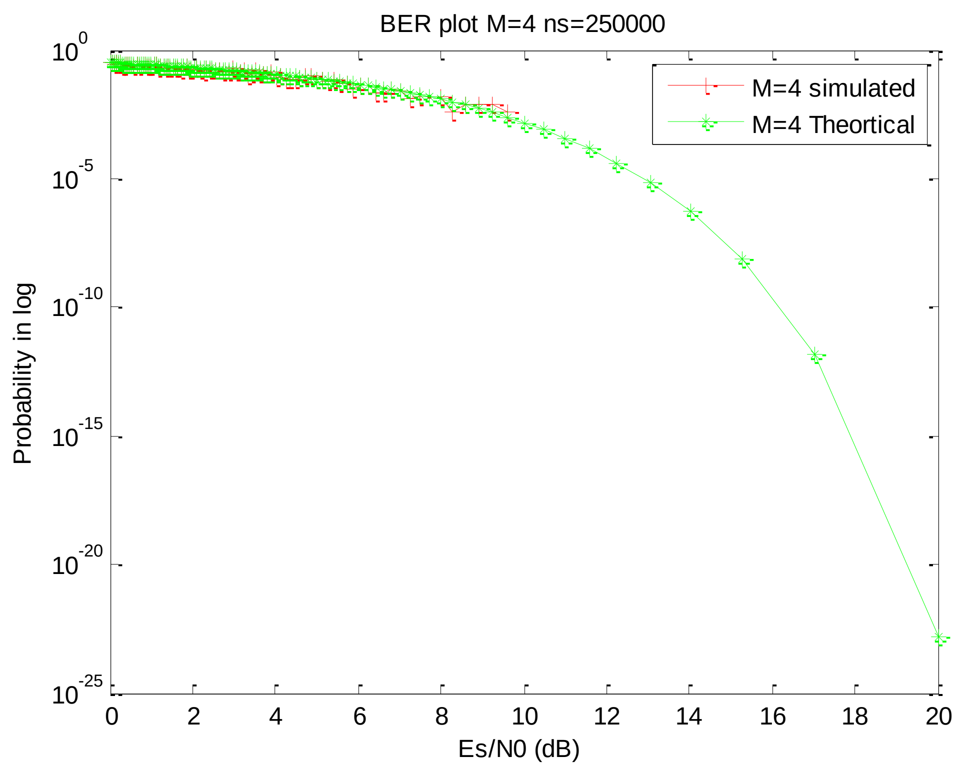


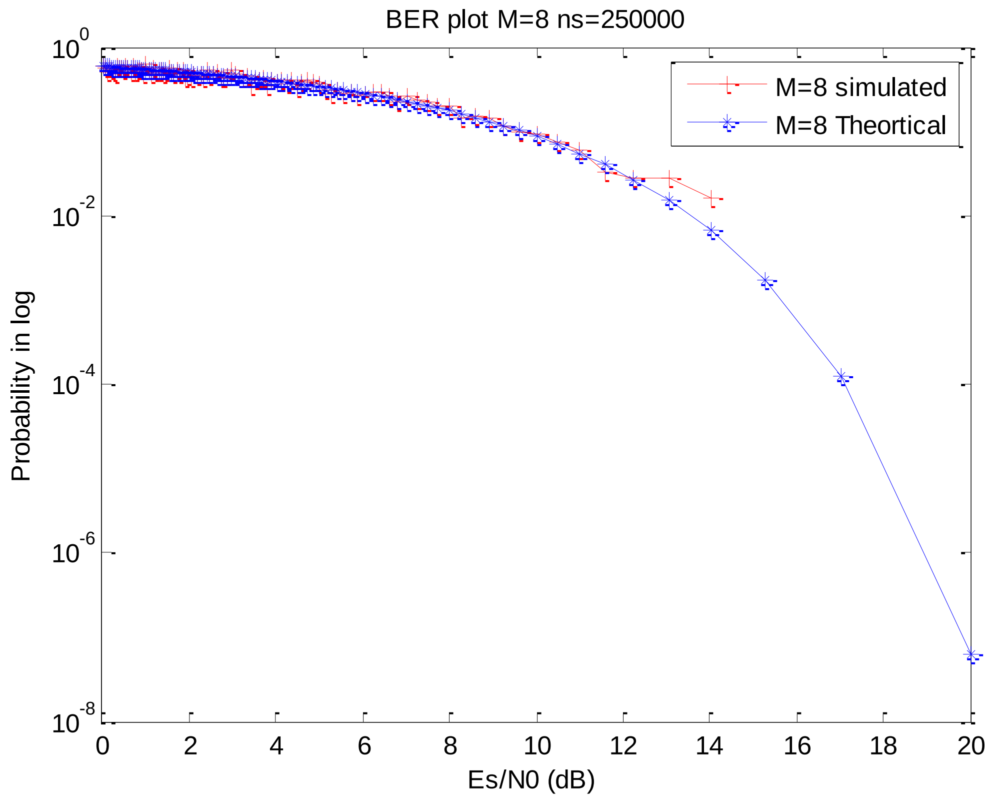

SER plot without mismatch M=8 ns=250000
Simulated
Theoretical
Probability
Eb/N0 (dB)
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
0
2
4
6
8
10
12
14

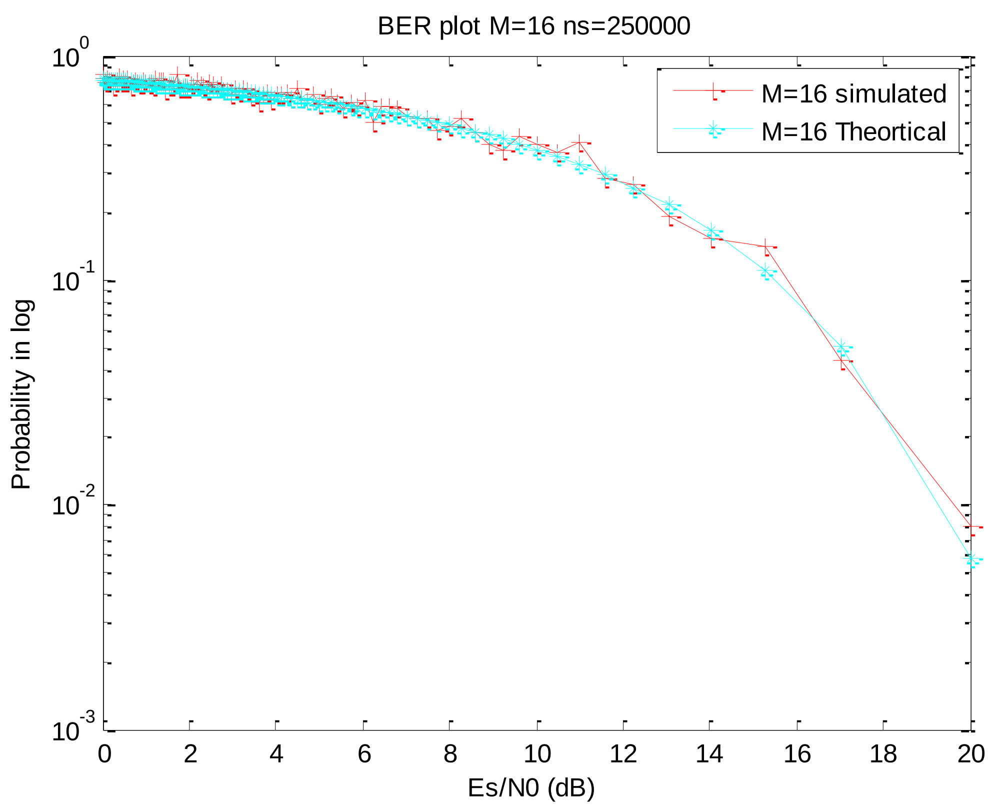

SER plot without mismatch M=16 ns=250000
Simulated
Theoretical
Probability
Eb/N0 (dB)
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
0
2
4
6
8
10
12
14
16
18
20

For number of symbols = 250000, different M with different BER are as follows,



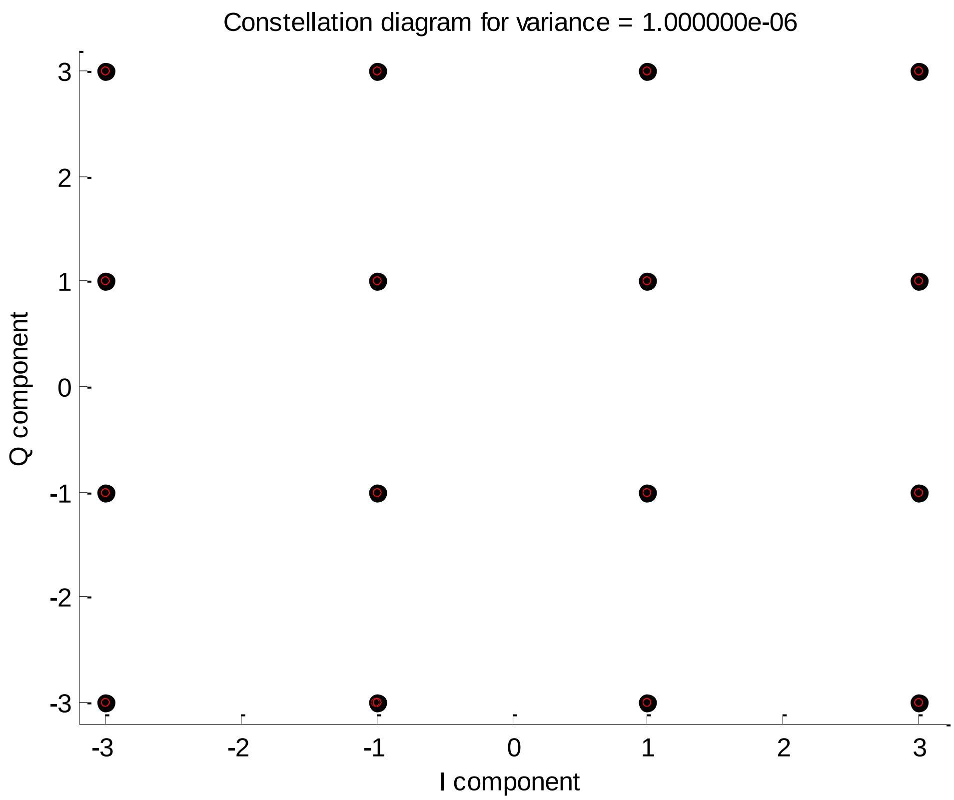


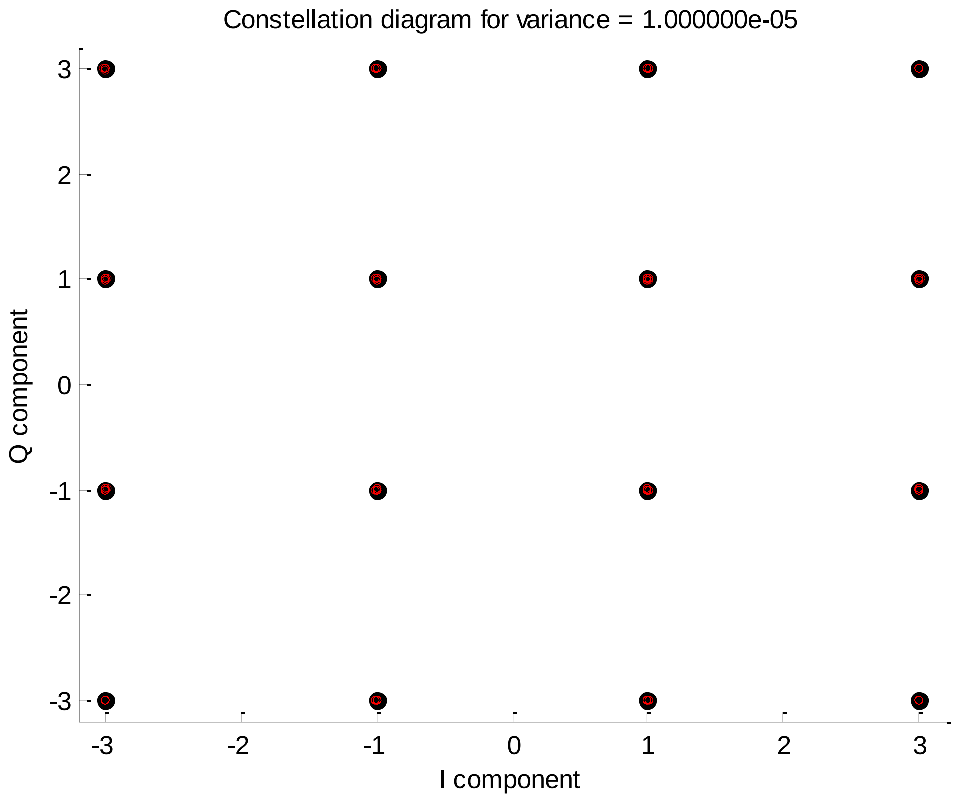

BER plot M=8 ns=250000
M=8 simulated
M=8 Theortical
Probability in log
Es/N0 (dB)

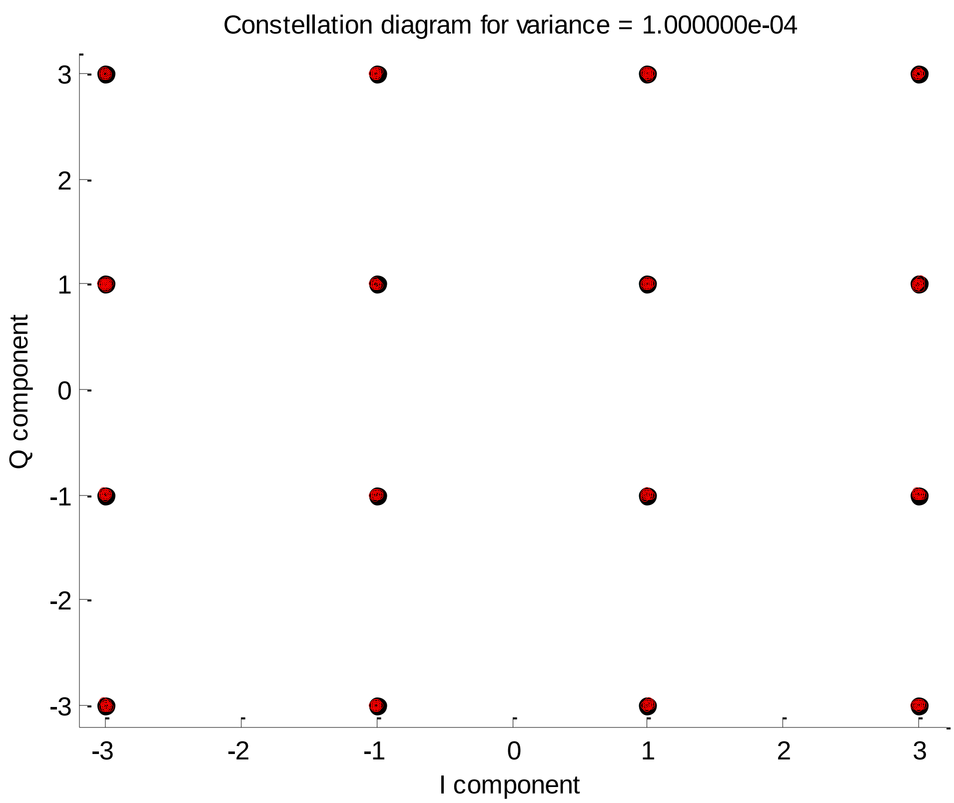

BER plot M=16 ns=250000
M=16 simulated
M=16 Theortical
Probability in log
Es/N0 (dB)

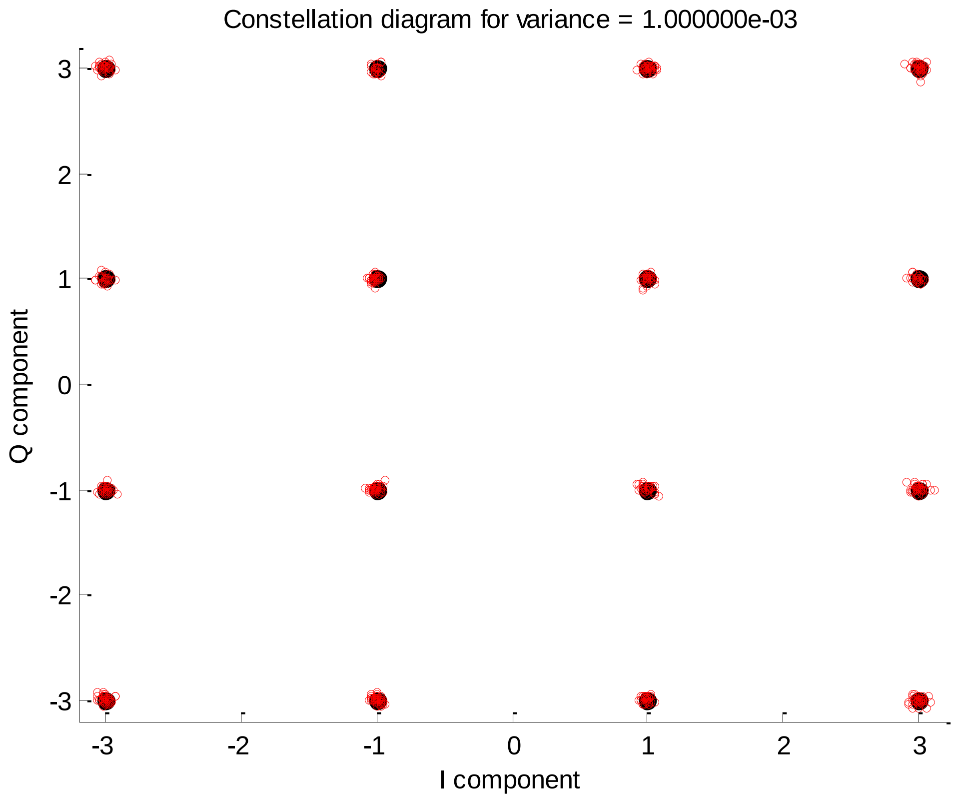
3. M-ary Quadrature Amplitude Modulation(M-QAM)

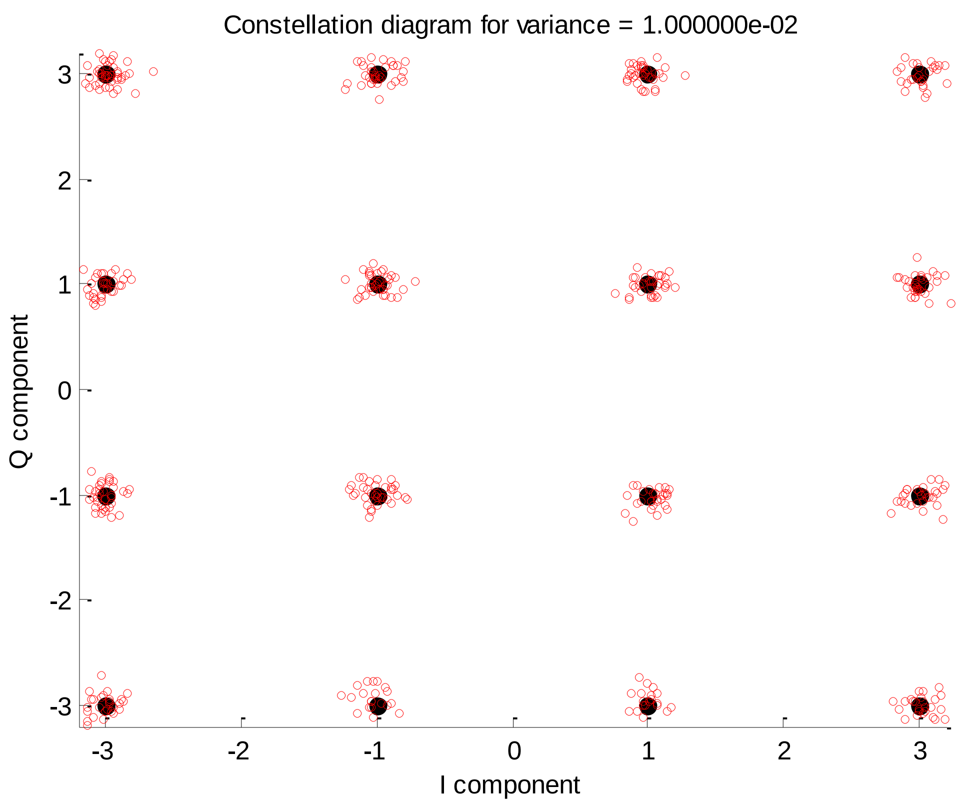
3.1 16-QAM Constellation on Effects of Noise



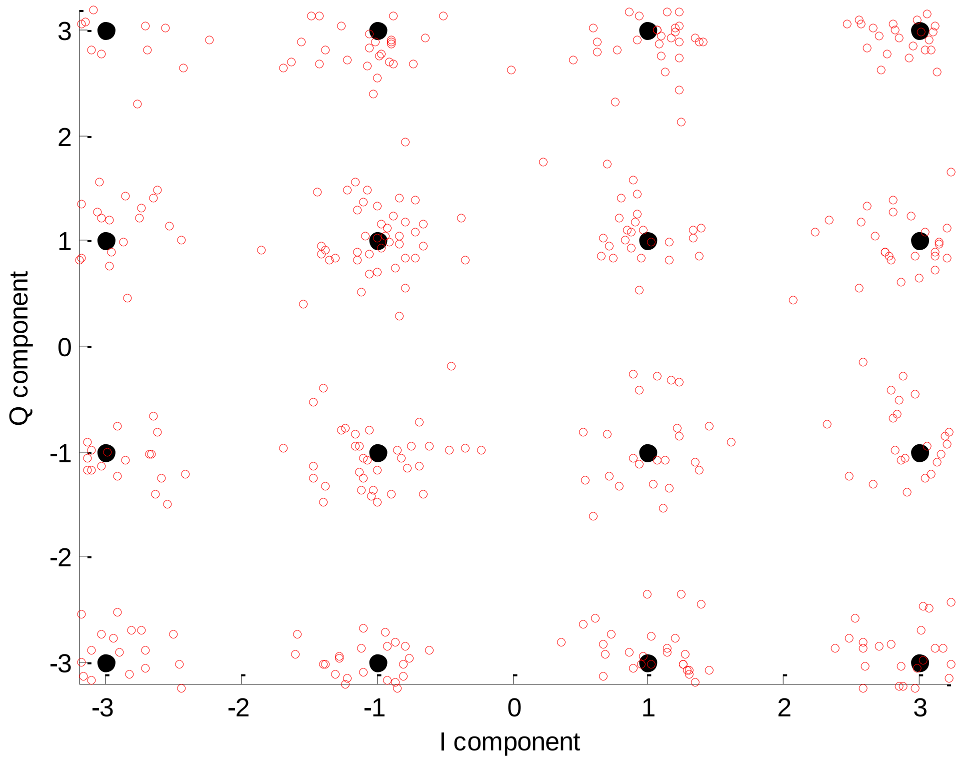

Constellation diagram for variance = 1.000000e-05
Q component
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3
I component


Constellation diagram for variance = 1.000000e-04
Q component
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3
I component

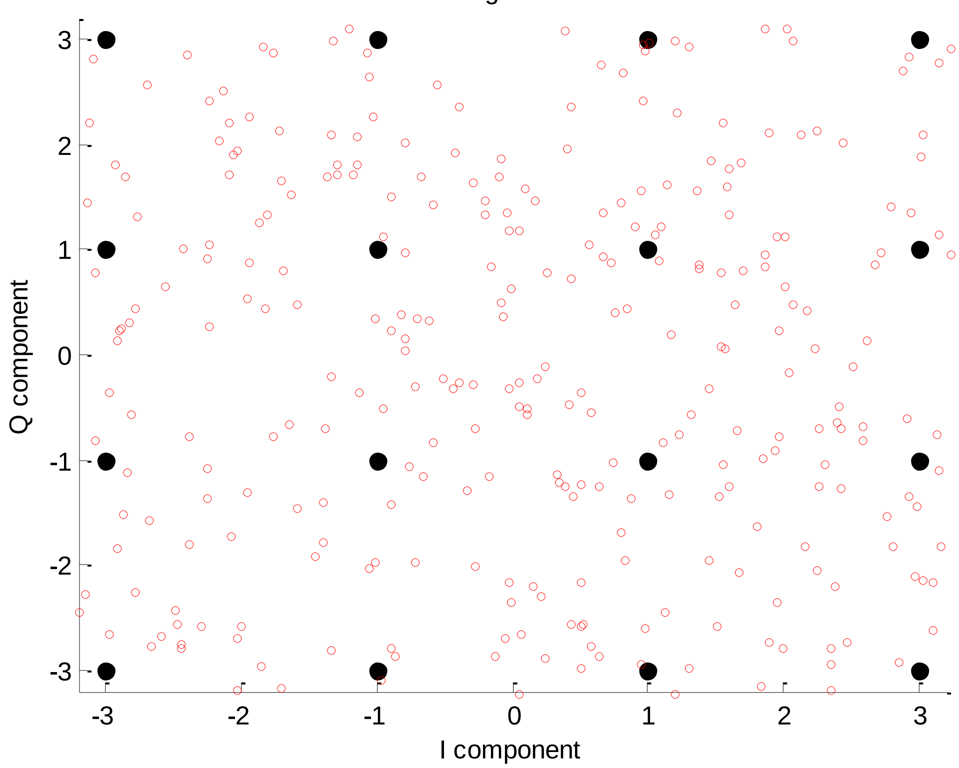

Constellation diagram for variance = 1.000000e-03
Q component
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3
I component

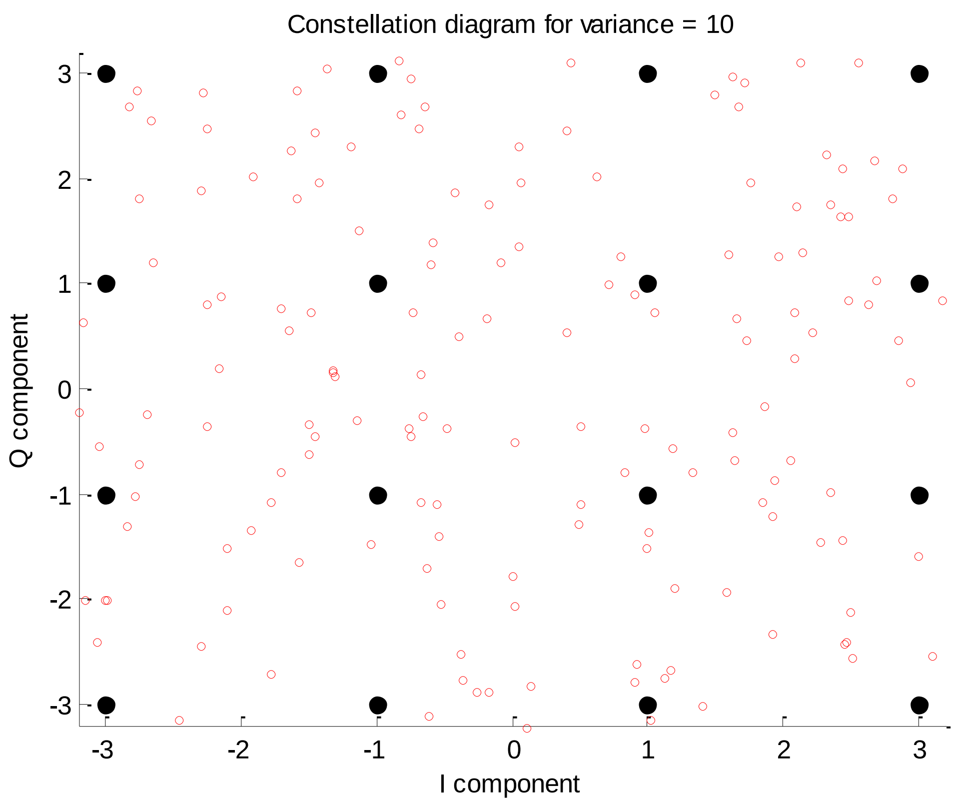

Constellation diagram for variance = 1.000000e-02
Q component
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3
I component

Constellation diagram for variance = 1.000000e-01

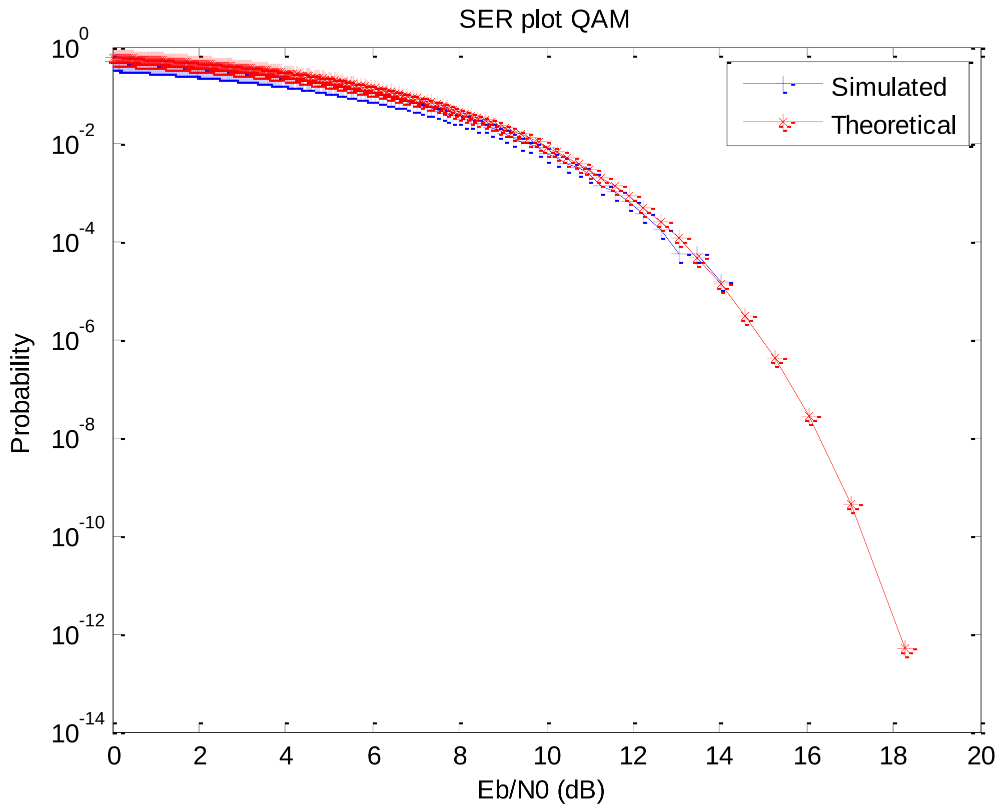


Constellation diagram for variance = 1

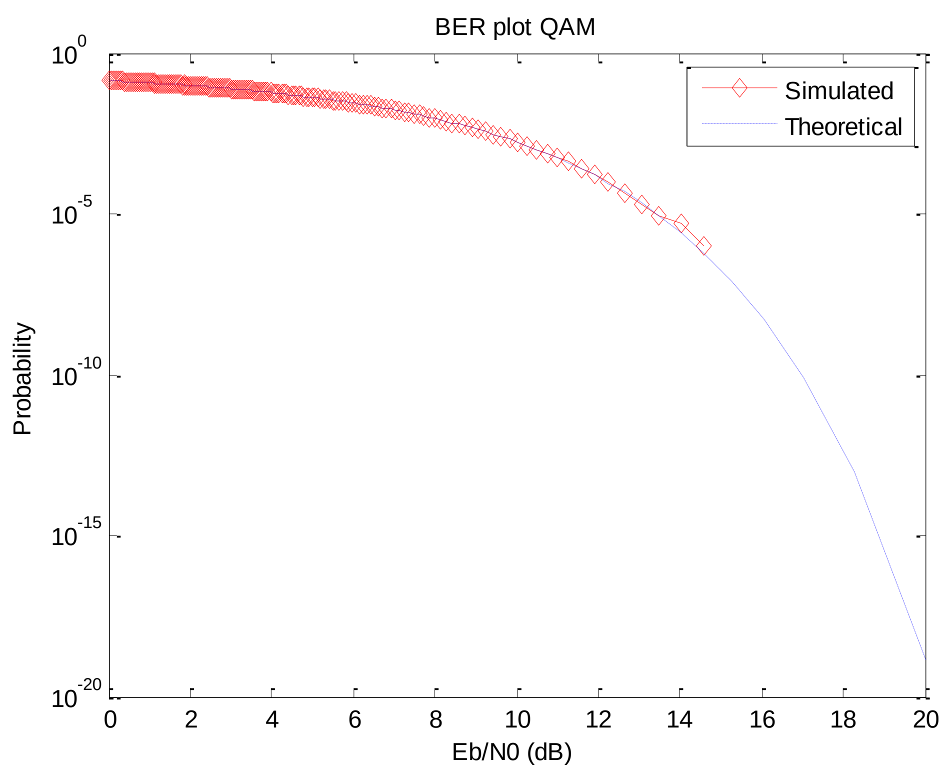



Constellation diagram for variance = 10
Q component
I component
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

3.2 16-QAM SER vs Eb/N0



3.3 16-QAM BER vs Eb/N0



3.4 Error Vector Magnitude (EVM)

Variance	EVM
0.0125000000000000	0.0373288492140046
0.0186875000000000	0.0455760355629530
0.0248750000000000	0.0526460457094564
0.0310625000000000	0.0587905977172851
0.0372500000000000	0.0642236862208964
0.0434375000000000	0.0695329521978055
0.0496250000000000	0.0744311031445799
0.0558125000000000	0.0787790681632779
0.0620000000000000	0.0830466140032277
0.0681875000000000	0.0869984927738050
0.0743750000000000	0.0909617545981184
0.0805625000000000	0.0945650297409453
0.0867500000000000	0.0980403251100570
0.0929375000000000	0.101294470922656
0.0991250000000000	0.104815328848687
0.105312500000000	0.107845769491302
0.111500000000000	0.111078703981437
0.117687500000000	0.113949460214651
0.123875000000000	0.116575198180503
0.130062500000000	0.119631640943654
0.136250000000000	0.122253128448478
0.142437500000000	0.124622861538835
0.148625000000000	0.126982129682802
0.154812500000000	0.129321068433879
0.161000000000000	0.131521084188913
0.167187500000000	0.133718434377883
0.173375000000000	0.136218683829630
0.179562500000000	0.138284442429732
0.185750000000000	0.140123387573326
0.191937500000000	0.142026520869695
0.198125000000000	0.144099699945374

0.204312500000000	0.145795884212354
0.210500000000000	0.147649545163503
0.216687500000000	0.149583722665696
0.222875000000000	0.151253344133348
0.229062500000000	0.152800047181844
0.235250000000000	0.154404830823269
0.241437500000000	0.156286728466418
0.247625000000000	0.157548713389378
0.253812500000000	0.158781060169176
0.260000000000000	0.160434854128560
0.266187500000000	0.161682983074612
0.272375000000000	0.163068642103024
0.278562500000000	0.164485942269702
0.284750000000000	0.165591078987119
0.290937500000000	0.166981212223410
0.297125000000000	0.168168656072155
0.303312500000000	0.169428038988279
0.309500000000000	0.170163739312998
0.315687500000000	0.171494641787012
0.321875000000000	0.172646341802961
0.328062500000000	0.173791568932135
0.334250000000000	0.174941478609165
0.340437500000000	0.176012197131423
0.346625000000000	0.176859697638399
0.352812500000000	0.177923028670570
0.359000000000000	0.178807392047170
0.365187500000000	0.179886702919849
0.371375000000000	0.180562192300517
0.377562500000000	0.181784644990449
0.383750000000000	0.182614214924241
0.389937500000000	0.183524887362776
0.396125000000000	0.184174879609719
0.402312500000000	0.184936604966568
0.408500000000000	0.185929598664627

0.414687500000000	0.186742372504911
0.420875000000000	0.187846014461916
0.427062500000000	0.188922485438849
0.433250000000000	0.189400330355250
0.439437500000000	0.189890092850199
0.445625000000000	0.190838517633765
0.451812500000000	0.191331072918676
0.458000000000000	0.192313902490262
0.464187500000000	0.192974387273484
0.470375000000000	0.193720706423721
0.476562500000000	0.194363255523221
0.482750000000000	0.195036663217575
0.488937500000000	0.195792175321252
0.495125000000000	0.196721133652740
0.501312500000000	0.197214481375632
0.507500000000000	0.197620212601861
0.513687500000000	0.198596960617904
0.519875000000000	0.198924998614064
0.526062500000000	0.199479041877522
0.532250000000000	0.200277664727184
0.538437500000000	0.201076756425009
0.544625000000000	0.201378373866967
0.550812500000000	0.202114125899500
0.557000000000000	0.202614326596627
0.563187500000000	0.203311721069025
0.569375000000000	0.203760647657608
0.575562500000000	0.204681676135740
0.581750000000000	0.204892186123254
0.587937500000000	0.205641671371895
0.594125000000000	0.206226238628246
0.600312500000000	0.206464555011806
0.606500000000000	0.206967030729382
0.612687500000000	0.207731377604275
0.618875000000000	0.208219776296288

0.625062500000000	0.208510122158593
0.631250000000000	0.209304698300908
0.637437500000000	0.209573256059906
0.643625000000000	0.210326361451635
0.649812500000000	0.211018556230918
0.656000000000000	0.211511423308182
0.662187500000000	0.212245515035430
0.668375000000000	0.212358541632664
0.674562500000000	0.212883447761240
0.680750000000000	0.213346775003723
0.686937500000000	0.214149825018502
0.693125000000000	0.214371509003271
0.699312500000000	0.214886378409375
0.705500000000000	0.215575479143293
0.711687500000000	0.215946822910913
0.717875000000000	0.216312199565320
0.724062500000000	0.216474392223001
0.730250000000000	0.217147460928755
0.736437500000000	0.217382151309849
0.742625000000000	0.218384250161968
0.748812500000000	0.218390110466608
0.755000000000000	0.219033999351534
0.761187500000000	0.219354351181914
0.767375000000000	0.220122196533560
0.773562500000000	0.220290224289023
0.779750000000000	0.220793432285939
0.785937500000000	0.220626340570109
0.792125000000000	0.221725648939237
0.798312500000000	0.221922477783321
0.804500000000000	0.222474394724309
0.810687500000000	0.223184850125381
0.816875000000000	0.223459434248365
0.823062500000000	0.223877968910826
0.829250000000000	0.224423567881886

0.835437500000000	0.224737700478183
0.841625000000000	0.225000695668474
0.847812500000000	0.225247747503544
0.854000000000000	0.225668491704068
0.860187500000000	0.226274261567494
0.866375000000000	0.226635202156473
0.872562500000000	0.226781334865241
0.878750000000000	0.228218305285727
0.884937500000000	0.227759958603608
0.891125000000000	0.228193956523559
0.897312500000000	0.228651698791468
0.903500000000000	0.229325771792939
0.909687500000000	0.229617930804725
0.915875000000000	0.230075486280600
0.922062500000000	0.230194463410189
0.928250000000000	0.230585656540084
0.934437500000000	0.230907911050891
0.940625000000000	0.231876407155184
0.946812500000000	0.231845743361151
0.953000000000000	0.232304980390002
0.959187500000000	0.232897847142584
0.965375000000000	0.233228499582968
0.971562500000000	0.233772181729544
0.977750000000000	0.234316569384339
0.983937500000000	0.234491797569836
0.990125000000000	0.234541801112655
0.996312500000000	0.235209855346033
1.00250000000000	0.235903307207456
1.00868750000000	0.235945260924268
1.01487500000000	0.235911827572921
1.02106250000000	0.236930461212666
1.02725000000000	0.237272131231797
1.03343750000000	0.237340001836409
1.03962500000000	0.237661443541329

1.04581250000000	0.238498157340711
1.05200000000000	0.238109082721743
1.05818750000000	0.238742001611856
1.06437500000000	0.239250157177167
1.07056250000000	0.239439800422908
1.07675000000000	0.240466187500485
1.08293750000000	0.240049811120894
1.08912500000000	0.240641460718926
1.09531250000000	0.241253301090385
1.10150000000000	0.241015610999536
1.10768750000000	0.242086321299627
1.11387500000000	0.242043581014528
1.12006250000000	0.243037529856609
1.12625000000000	0.243645753408818
1.13243750000000	0.243813631142475
1.13862500000000	0.244264551368906
1.14481250000000	0.243709580758300
1.15100000000000	0.244715563551544
1.15718750000000	0.244830478359646
1.16337500000000	0.245239097057673
1.16956250000000	0.245637286188181
1.17575000000000	0.246660400095634
1.18193750000000	0.246774680733077
1.18812500000000	0.246546033064260
1.19431250000000	0.246972496290431
1.20050000000000	0.247800877504774
1.20668750000000	0.247973720613491
1.21287500000000	0.247540212271142
1.21906250000000	0.248672083785162
1.22525000000000	0.248994996320426
1.23143750000000	0.249789016990177
1.23762500000000	0.250094992023086
1.24381250000000	0.249885007890879
1.25000000000000	0.250857737092356

4. Matlab code

Please refer to the uploaded matlab codes.

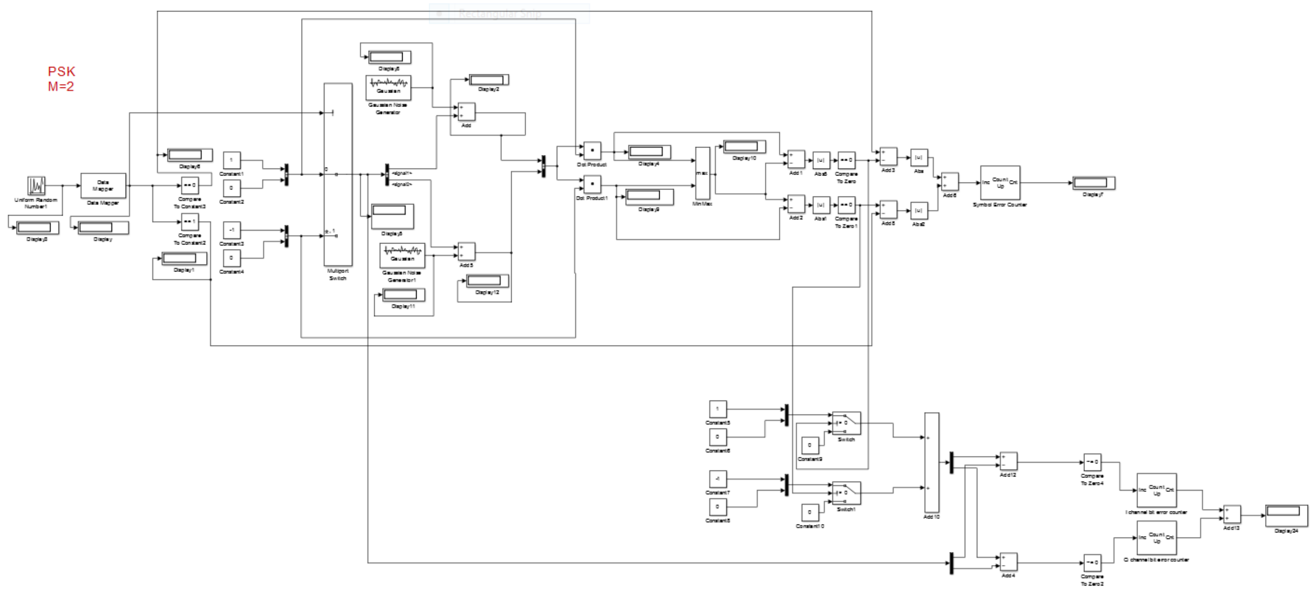
5. Conclusions

The concept and the simulation of digital modulation techniques such as MPSK and QAM are explored. To realize the simulation as close as possible with the reality communication environment, noises and mismatch components are taken into account. The simulation results are showed as self-explanatory ways by their graphs and the ones that display inconsistency with ideal results are explained in summary.

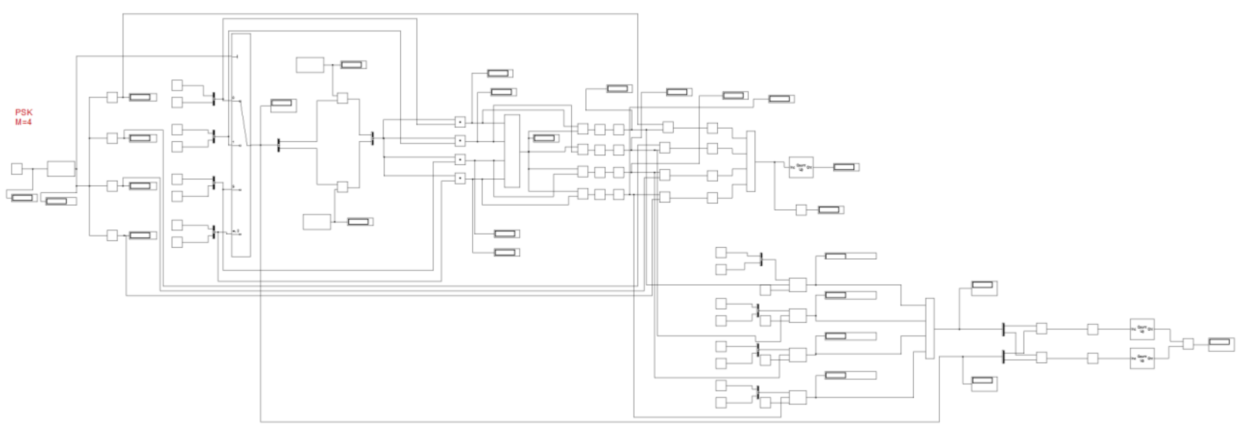
This project provides a good chance to learn about hands-on modulations techniques as well as their detailed realization methodology.

6. Simulink Simulations

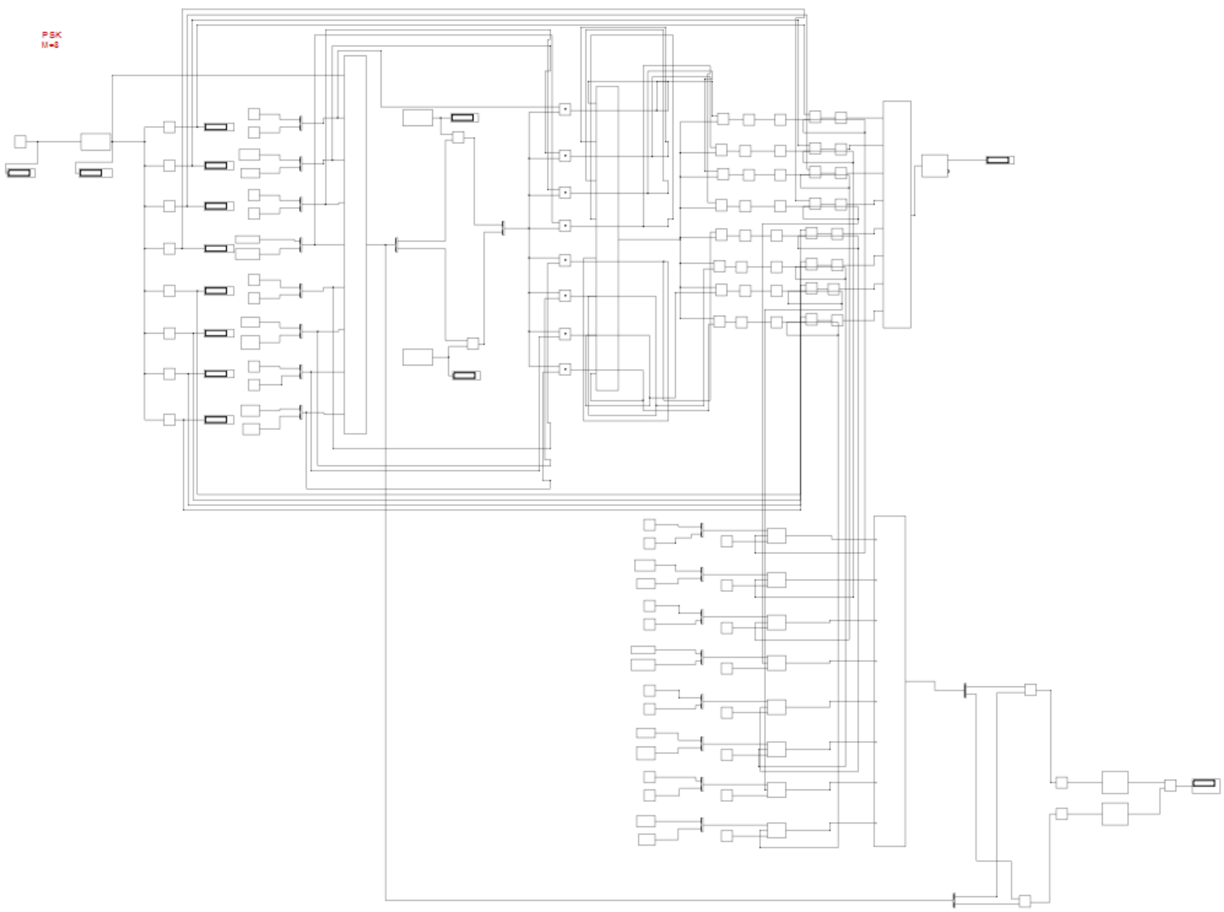
BPSK



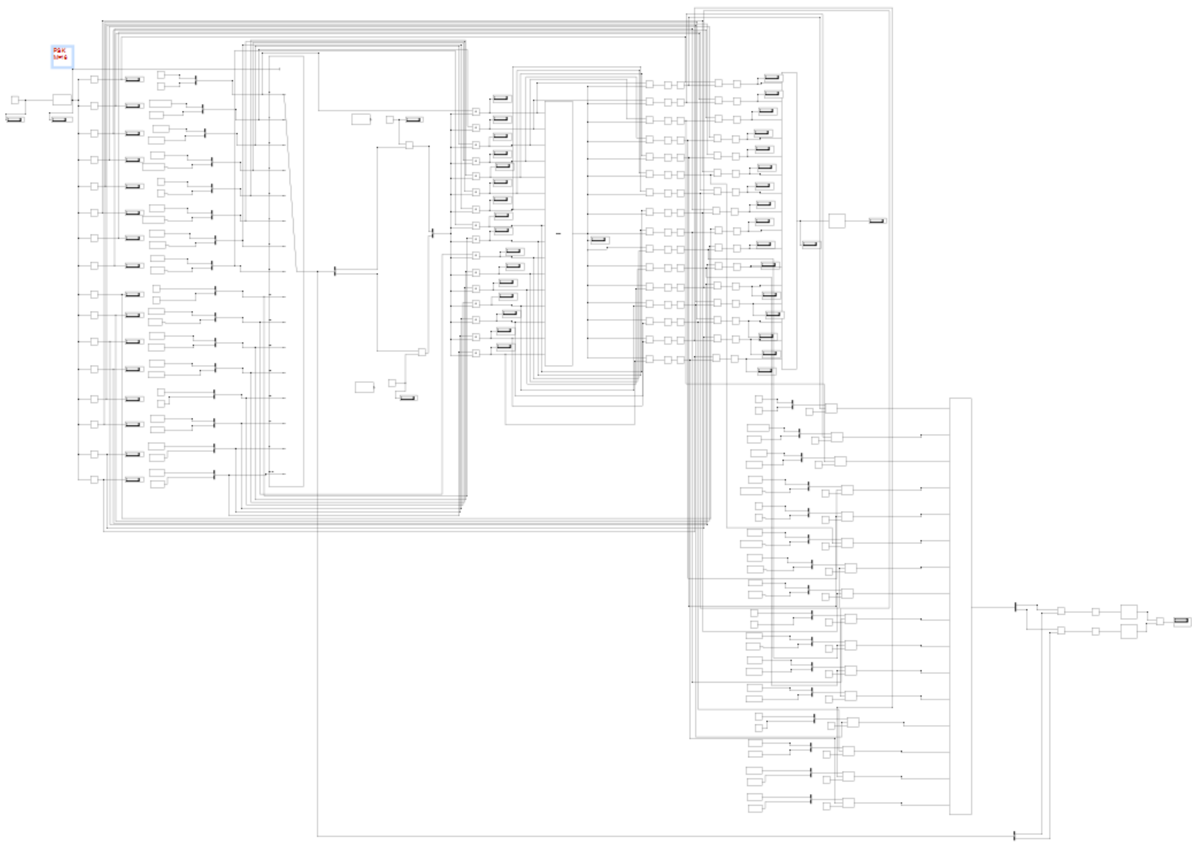
QPSK



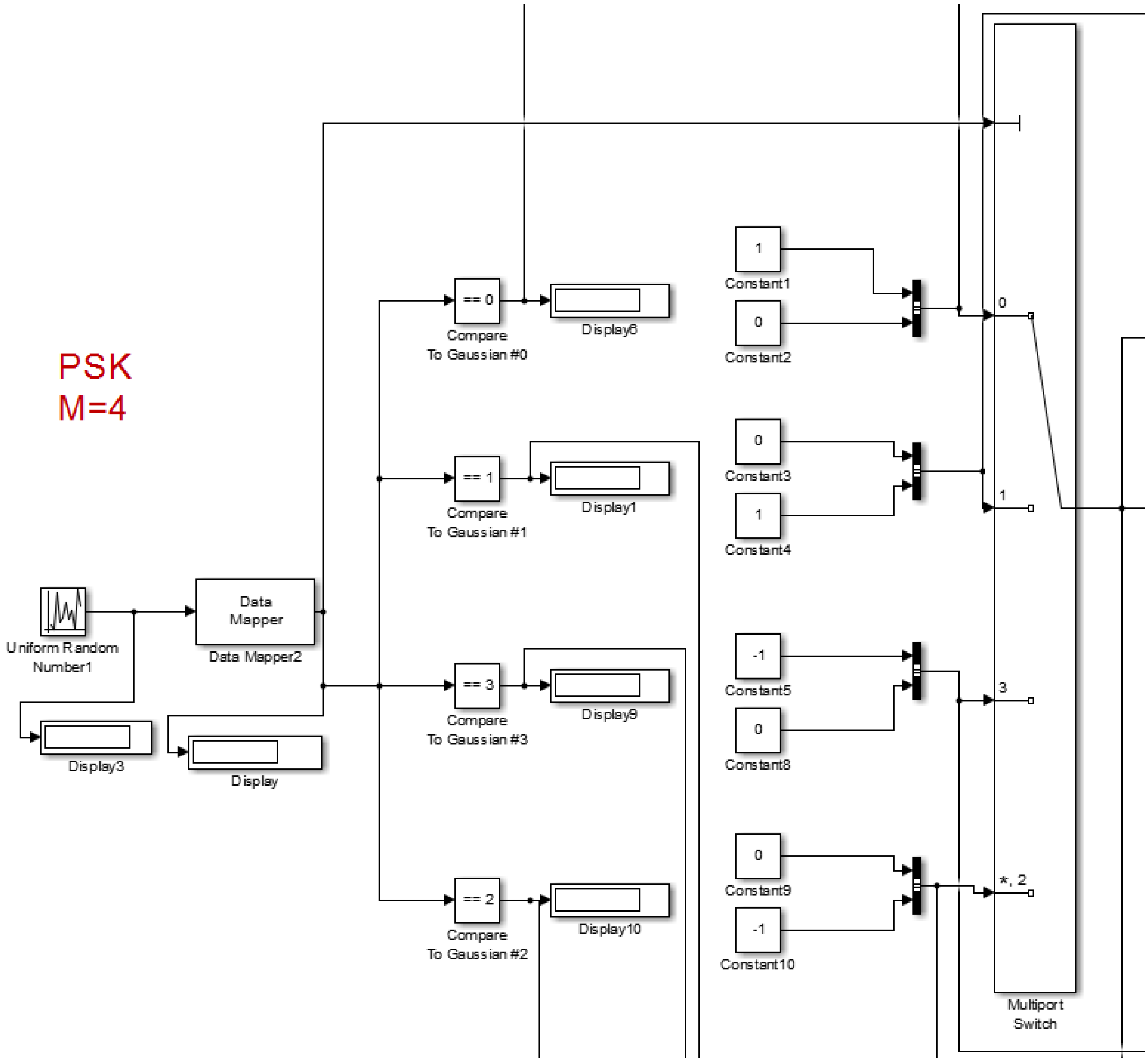
8PSK



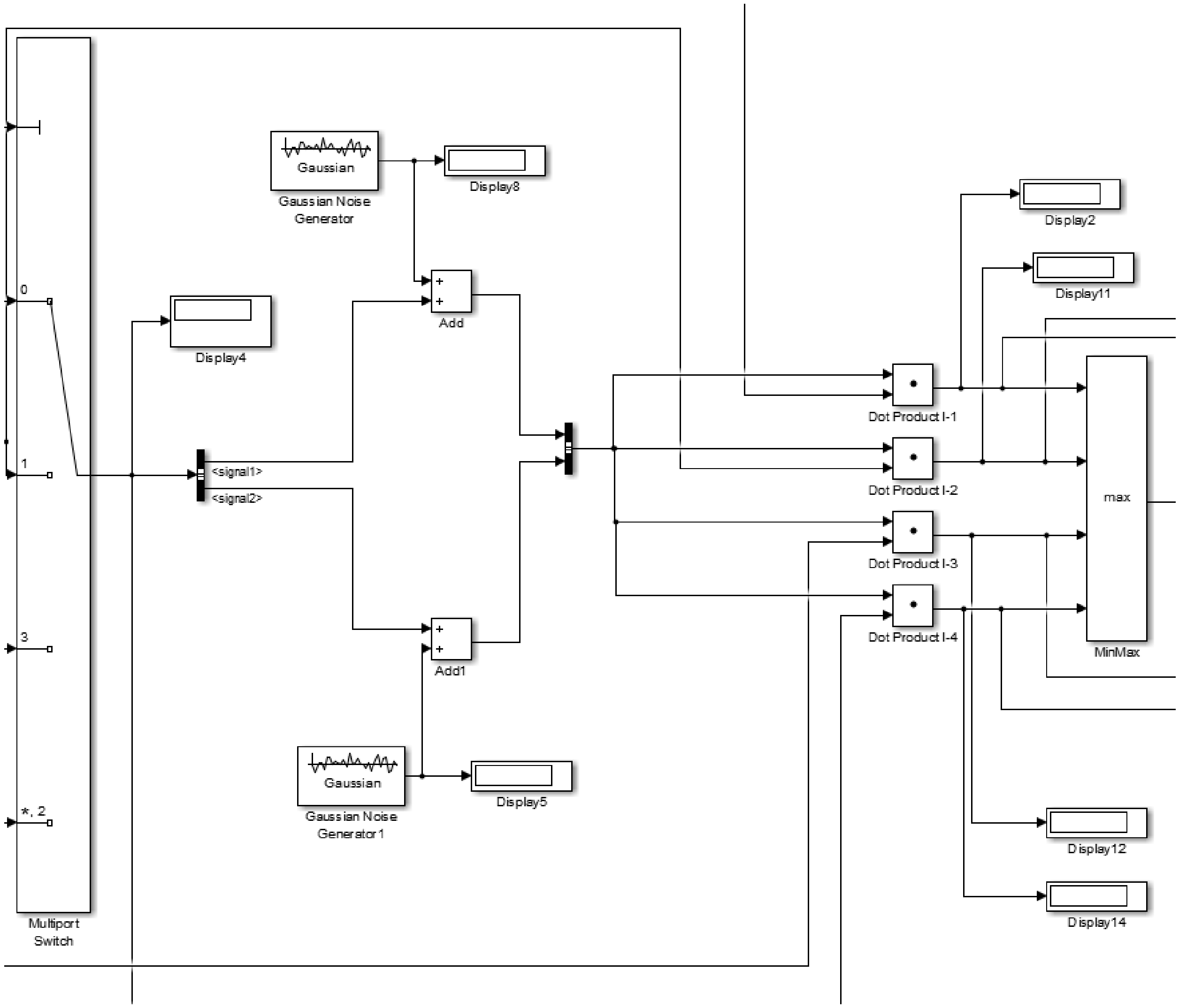
16PSK



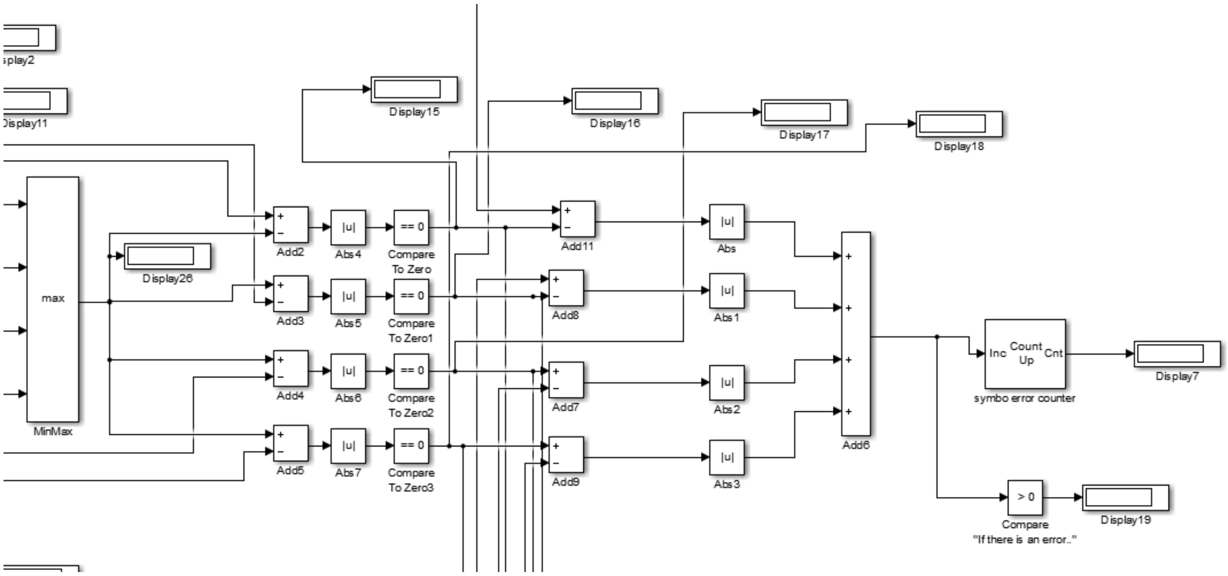
QPSK design description



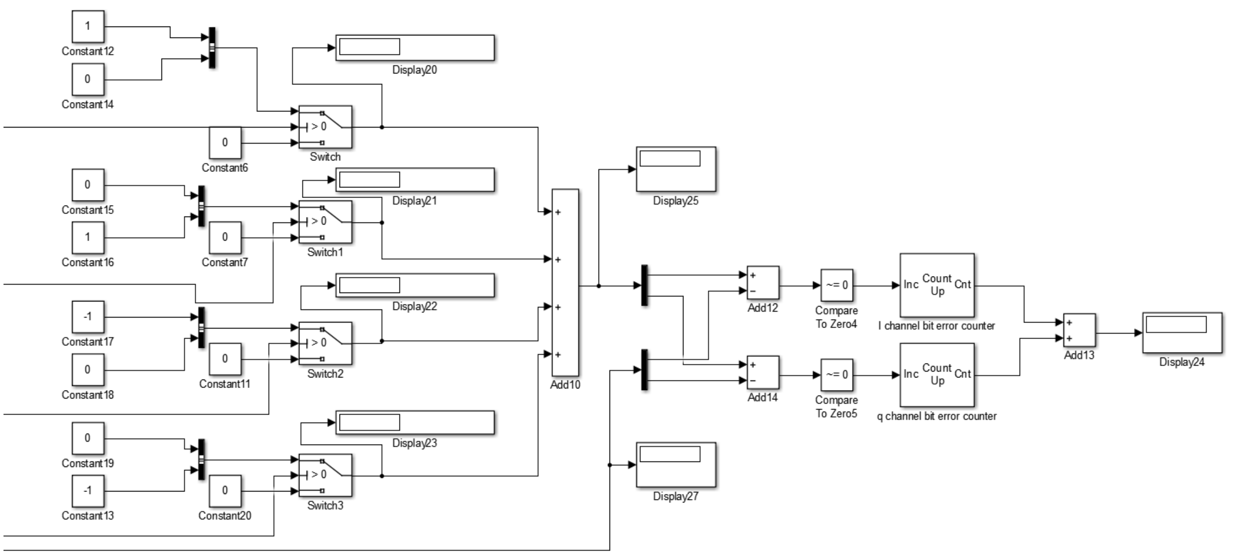
①In QPSK, Uniform Random Number generator generates from 0 to 3.9999. Data mapper maps the random numbers to Gaussian number (0, 1, 3, 2). At multiport switch, a mapped number selects I and Q coordinates i.e. Gaussian 3 → I:-1, Q:0.



②Next, Gaussian Noise Generator generates noise and added to I and Q. The results go to 'dot product' with each possible I Q coordinates (1,0 / 0,1 / -1,0 / 0,-1). Only biggest value is chosen by max block.



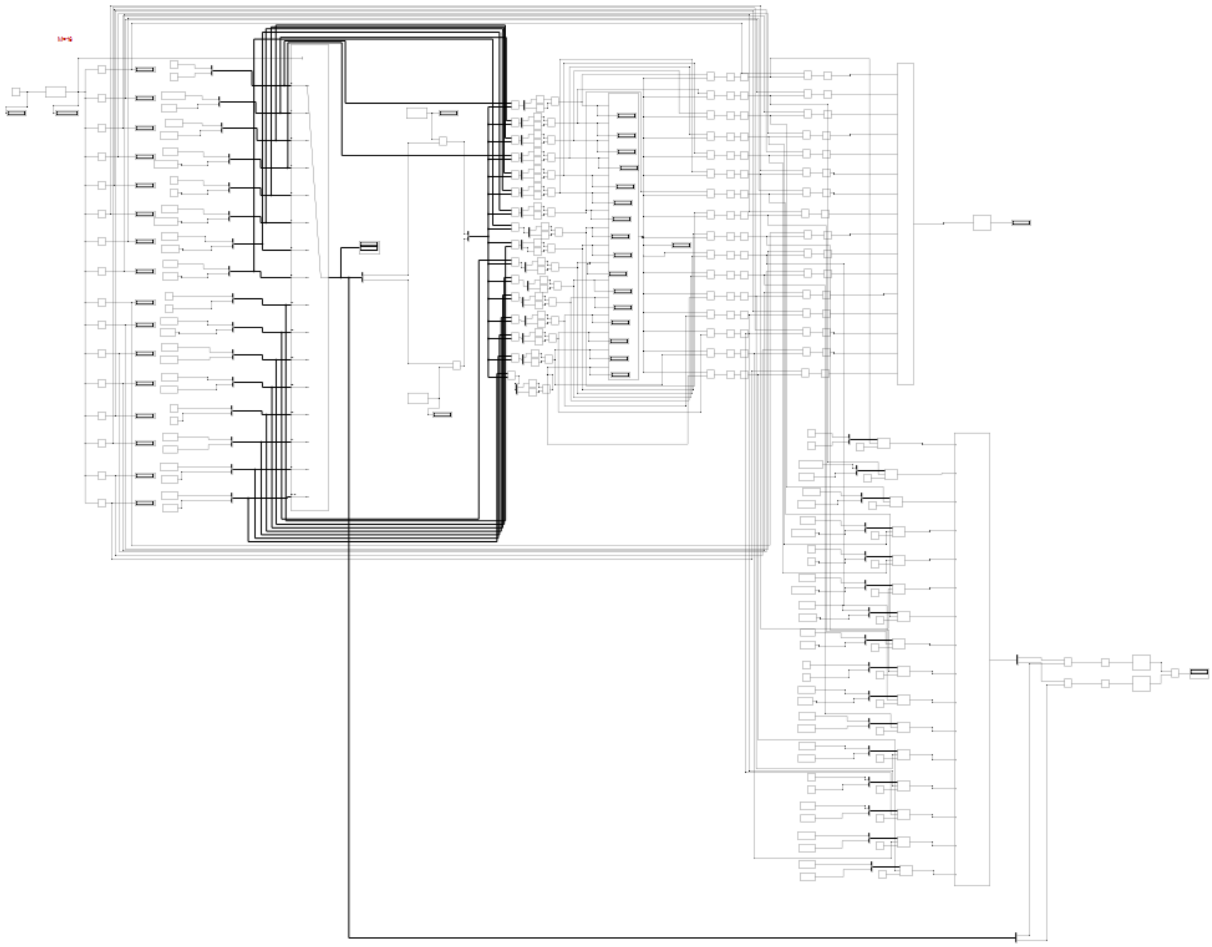
③By subtracting maximum dot product with each dot product results, and comparing it to zero, we can find the matching point. By subtracting those points with original Gaussian points, we can find if it has symbol error.



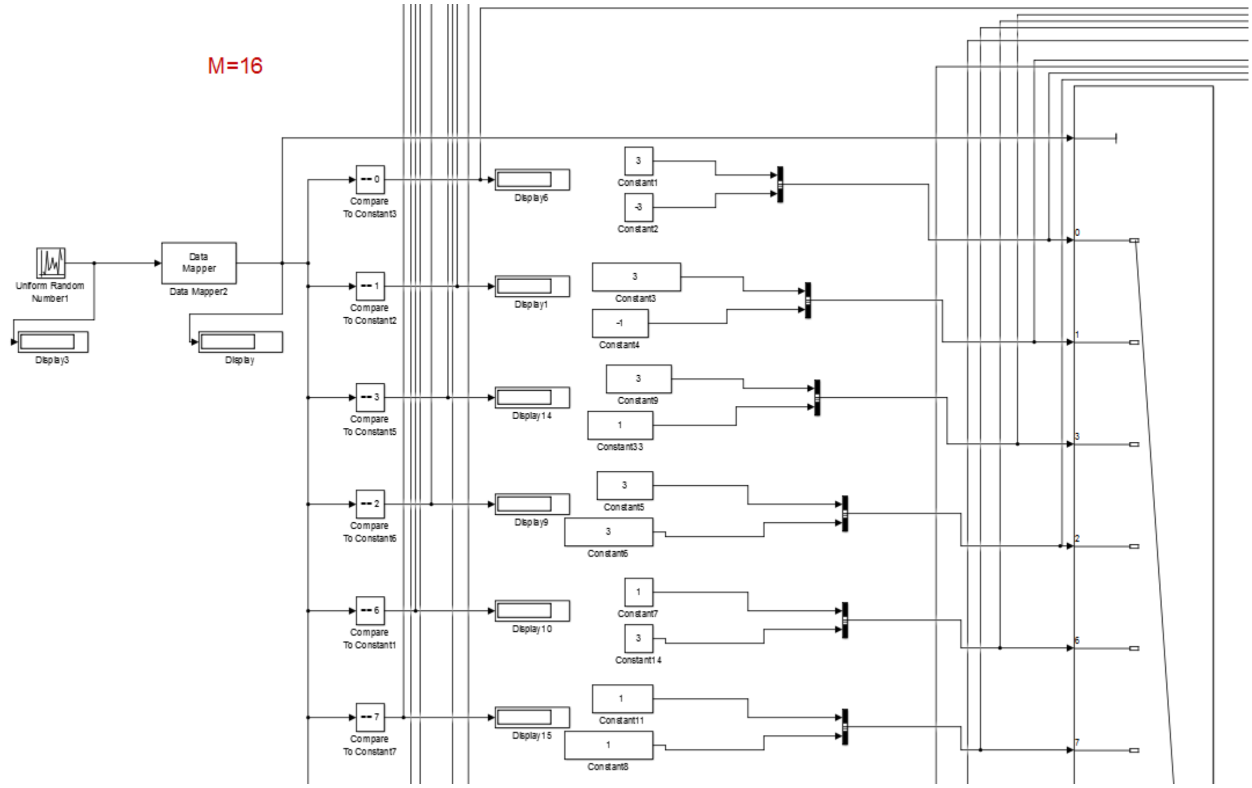
④This is a bit error counter. 4 switches selects coordinates of corresponding IQ channels (Gaussian noise added). Since rest of switch results are 0, the result of add block passes the coordinates. Next two add blocks and compare blocks compare the coordinates and the original coordinates bit by bit. If there is an error it would be counted at last.

BPSK, 8-PSK, 16-PSK have same structures but different numbers of blocks.

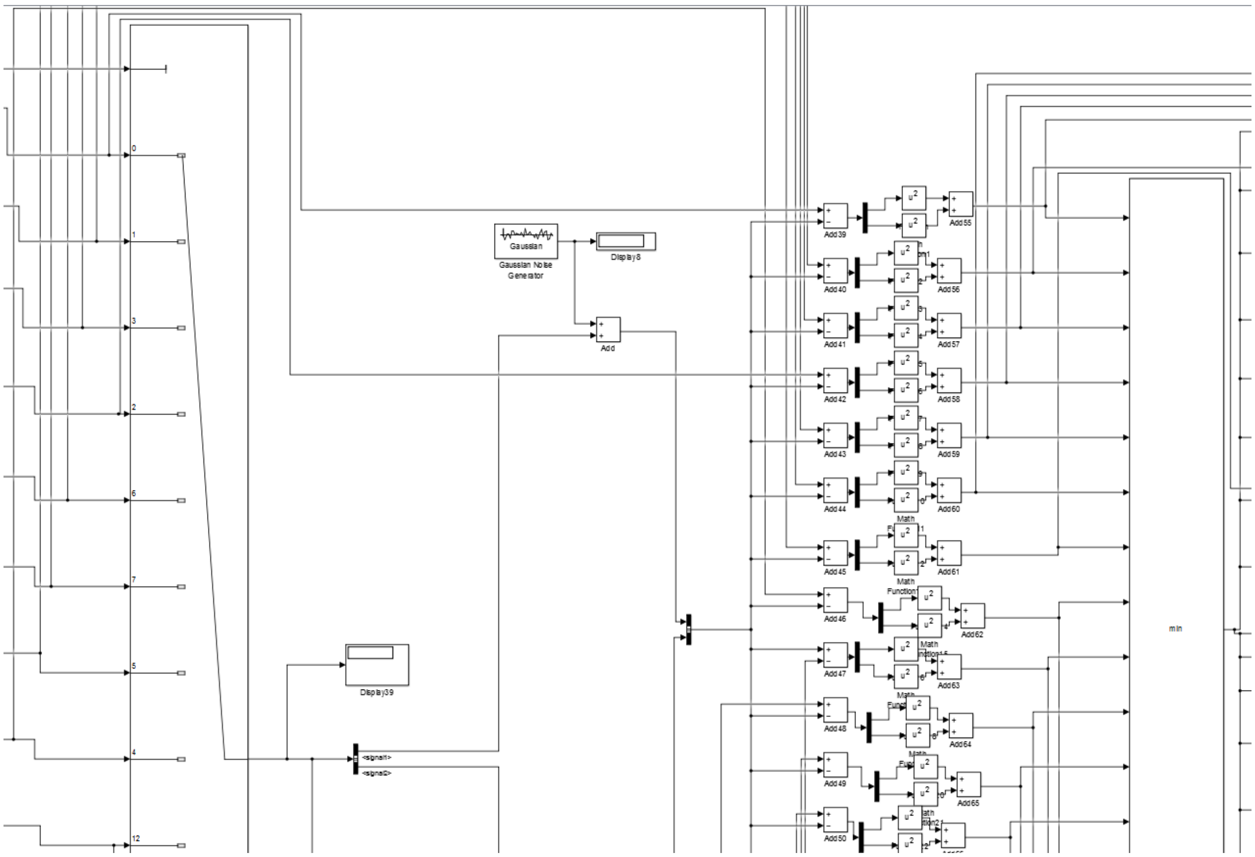
16 QAM



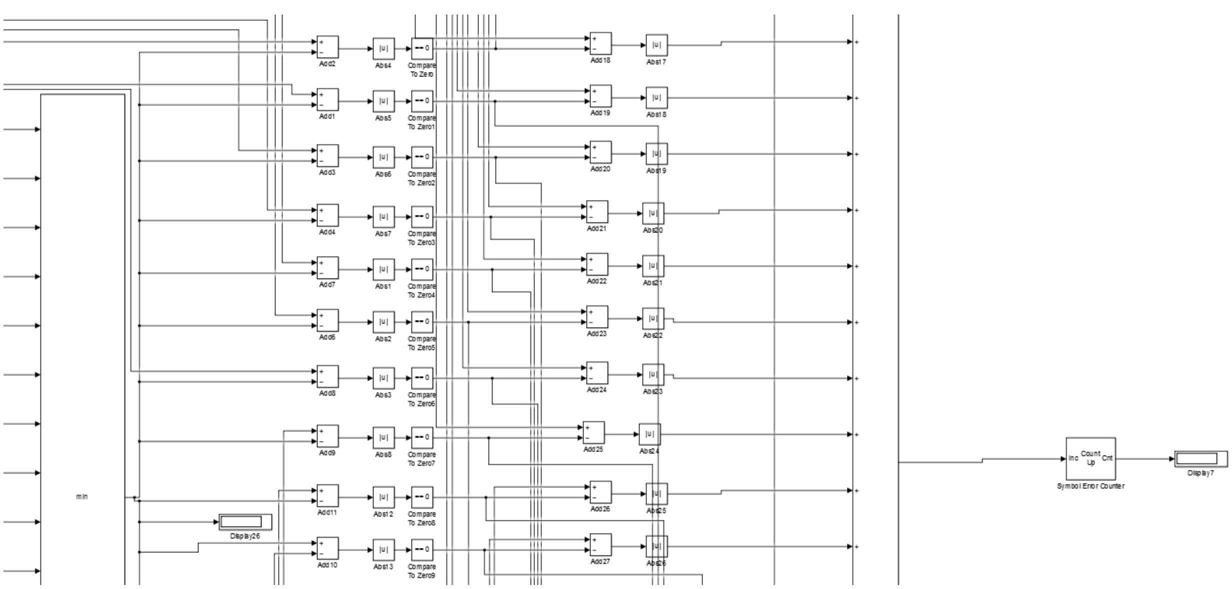
16 QAM design description



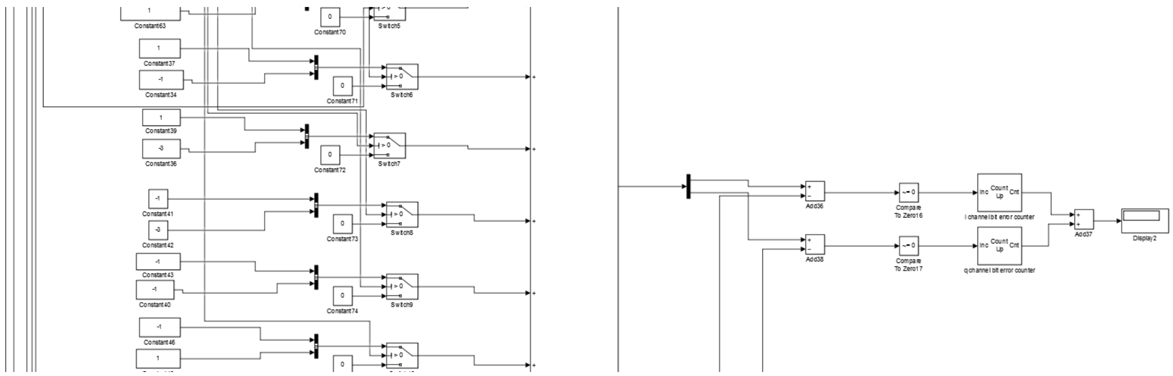
① First, random number generator generates from 0 to 15.9999. Data mapper maps the random numbers to Gaussian number (0, 1, 3, 2, 6, 7, 5, 4, 12, 13, 15, 14, 10, 11, 9, 8). At multiport switch, a mapped number selects I and Q coordinates i.e. Gaussian 11 → I:-3, Q:1.



②Next, Gaussian Noise Generator generates noise and added to I and Q. The results go to subtract and square blocks with each possible I Q coordinates (3,3 / 3,1 / 3,-1 / 3,-3 / 1,-3, 1,-1 and so on) to calculate distances. Only smallest value is chosen by min block.



③By subtracting minimum distance with distance results, and comparing it to zero, we can find the matching point. By subtracting those points with original Gaussian points (without noise), we can find if it has symbol error.



④In this bit error counter, 4 switches select coordinates of corresponding IQ channels (Gaussian noise added). Since rest of switch results are 0, the result of add block passes the coordinates. Next two add blocks and compare blocks compare the coordinates and the original coordinates bit by bit. If there is an error it would be counted at last.

Simulink simulation table

Variance	BPSK		QPSK		8-PSK		16-PSK		16-QAM	
	Symbol Error	Bit Error	Symbol Error	Bit Error	Symbol Error	Bit Error	Symbol Error	Bit Error	Symbol Error	Bit Error
0.000001	0	0	0	0	0	0	0	0	0	0
0.00001	0	0	0	0	0	0	0	0	0	0
0.0001	0	0	0	0	0	0	0	0	0	0
0.001	0	0	0	0	0	0	0	0	0	0
0.01	0	0	0	0	27	54	1.256x 10^4	2.513x 10^4	0	0
0.1	186	186	6424	1.28x 10^4	5.656x 10^4	1.127x 10^5	1.346x 10^5	2.647x 10^5	592	592
1	3.97x 10^4	3.97x 10^4	1.059x 10^5	1.976 x10^5	1.686x 10^5	3.14x 10^5	2.077x 10^5	3.945x 10^5	1.05x 10^5	1.193 x10^5
10	9.423x 10^4	9.423x 10^4	1.634x 10^5	2.843 x10^5	2.05x 10^5	3.7x 10^5	2.273x 10^5	4.29x 10^5	2.024x 10^5	2.819 x10^5

7. Matlab Code

Listing 1: de2bi.m

```
function out = de2bi(dat,nbit)
    pow2 = 2.^[0:nbit-1];
    out = zeros(1,nbit);
    c = nbit;
    while dat>0
        if dat >= pow2(c)
            dat = dat-pow2(c);
            out(c) = 1;
        end
        c = c - 1;
end
```

Listing 2: graygen.m

```
function  out = graygen(n)
    out = zeros(2^n,1)  ;
    out(2) = 1  ;
    T = 2;
    for  k = 2:n
        T2 = T*2;
        out(T+1:T2) = T + flipud(out(1:T))  ;
        T = T2  ;
end
```

Listing 3: mpsksim.m

```
function [Eb,symErrCount,bitErrCount] = mpsksim(N0,M,ns,
   Es,nonideality)

    global curCor;
    global curCorMax;
    global curCorMaxIndex;
    global din;
    global dinDe;
    global grayEncBi;
    global grayEncDe;
    global iqNoiseInChannel;
    global modTran;
    global mpskPhVec;
    global msgFromChannel;
    global nbits;
    global nVar;
    global pdin;
```

```
    global recvdMsg;

    %Parameters
    nbits=log2(M);
    Eb=Es/nbits;
    pdin = rand(ns,1);
    din = zeros(ns,nbits);
    dinDe = zeros(ns,1);
    nVar = N0/2;

    %Gray encode table generator
    grayEncDe=graygen(nbits);
    grayEncBi=zeros(M,nbits);
    for i=1:1:M
        grayEncBi(i,:)=de2bi(grayEncDe(i),nbits);
    end

    %MPSK phase vector table generator [i,q]
    mpskPhVec=zeros(M,2);
    for m=0:1:M-1
        mpskPhVec(m+1,:)=[sqrt(Es)*(1+nonideality)*cos(2*pi*m/M*(1+nonideality)),sqrt(Es)*sin(2*pi*m/M)];
    end

    %Uniform binary input generator
    for i=1:1:ns
        for k=0:1:M-1
            if (k/M<=pdin(i)) && (pdin(i)<(k+1)/M)
                din(i,:)=de2bi(k,nbits);
                dinDe(i) = k;
            end
        end
    end

    %Modulate inputs
    modTran=zeros(ns,2);
    for i=1:1:ns
        for k=1:1:M
            if isequal(din(i,:), grayEncBi(k,:))
                modTran(i,:)= mpskPhVec(k,:);
            end
        end
    end

    %Generate additive Noise in the channel
```

```
    iqNoiseInChannel=zeros(ns,2);
    for i=1:1:ns
        iqNoiseInChannel(i,:) = normrnd(0,sqrt(nVar),[1
            2]);
    end

    %Message received
    msgFromChannel = zeros(ns,2);
    msgFromChannel = iqNoiseInChannel + modTran;

    %Receiver nonideality


    %Message detector
    recvdMsg = zeros(ns,nbits);
    curCor = zeros(ns,M);
    curCorMax = zeros(ns,1);
    curCorMaxIndex = zeros(ns,1);
    for i=1:1:ns
        for k=1:1:M
            curCor(i,k) = dot(msgFromChannel(i,:),
                mpskPhVec(k,:));
            if (k == 1)
                curCorMax(i) = curCor(i,k);
                curCorMaxIndex(i) = 1;
            else
                if( curCor(i,k) > curCorMax(i))
                    curCorMax(i) = curCor(i,k);
                    curCorMaxIndex(i) = k;
                end
            end
        end
        recvdMsg(i,:) = grayEncBi(curCorMaxIndex(i),:);
    end

    %Error Counter
    symErrCount=0;
    bitErrCount=0;
    for i=1:1:ns
        if (~isequal(recvdMsg(i,:),din(i,:)))
            symErrCount = symErrCount + 1;
        end
        for k=1:1:nbits
            if(recvdMsg(i,k) ~= din(i,k))
                bitErrCount = bitErrCount + 1;
            end
```

```
        end
    end
end
```

Listing 4: mpsksimwmismatch.m

```
function [Eb,symErrCount,bitErrCount] = mpsksimwmismatch(N0,M,ns,Es,nonideality)

    global curCor;
    global curCorMax;
    global curCorMaxIndex;
    global din;
    global dinDe;
    global grayEncBi;
    global grayEncDe;
    global iqNoiseInChannel;
    global modTran;
    global mpskPhVec;
    global msgFromChannel;
    global nbits;
    global nVar;
    global pdin;
    global recvdMsg;

    listOfPhMisMatch = [0.01 0.05];
    listOfMagMisMatch = [pi/180 5*pi/180] ;

    %Parameters
    nbits=log2(M);
    Eb=Es/nbits;
    pdin = rand(ns,1);
    din = zeros(ns,nbits);
    dinDe = zeros(ns,1);
    nVar = N0/2;

    %Gray encode table generator
    grayEncDe=graygen(nbits);
    grayEncBi=zeros(M,nbits);
    for i=1:1:M
        grayEncBi(i,:)=de2bi(grayEncDe(i),nbits);
    end

    numOfNonIdealitySim = (length(listOfMagMisMatch)+length(listOfPhMisMatch)+1);
    %MPSK phase vector table generator [i,q]
    mpskPhVec=zeros(M,2,numOfNonIdealitySim);
```

```
    for m=0:1:M-1
        mpskPhVec(m+1,:,1)=[sqrt(Es)*(1)*cos(2*pi*m/M*(1)
            ),sqrt(Es)*sin(2*pi*m/M)];
    end
    if(nonideality~=0)
        for i = 1:1:length(listOfMagMisMatch)
            for k = 1:1:length(listOfPhMisMatch)
                for m=0:1:M-1
                 mpskPhVec(m+1,:,(i-1)*length(
                     listOfPhMisMatch)+k+1)=[sqrt(Es)*(1+
                     listOfMagMisMatch(i))*cos(2*pi*m/M+
                     listOfPhMisMatch(k)),sqrt(Es)*sin(2*pi
                     *m/M)];
                end
            end
        end
    end

    %Uniform binary input generator
    for i=1:1:ns
        for k=0:1:M-1
            if (k/M<=pdin(i)) && (pdin(i)<(k+1)/M)
                din(i,:)=de2bi(k,nbits);
                dinDe(i) = k;
            end
        end
    end

    %Modulate inputs
    modTran=zeros(ns,2,numOfNonIdealitySim);
    for i=1:1:ns
        for k=1:1:M
            if isequal(din(i,:), grayEncBi(k,:))
                modTran(i,:,1)= mpskPhVec(k,:,1);
            end
        end
    end
    if(nonideality~=0)
        for l = 1:1:numOfNonIdealitySim - 1
            for i=1:1:ns
                for k=1:1:M
                    if isequal(din(i,:), grayEncBi(k,:))
                        modTran(i,:,l+1)= mpskPhVec(k,:,l
                            +1);
                    end
                end
```

```
            end
        end
    end

    %Generate additive Noise in the channel
    iqNoiseInChannel=zeros(ns,2);
    for i=1:1:ns
        iqNoiseInChannel(i,:) = normrnd(0,sqrt(nVar),[1 2]);
    end


    %Message received
    msgFromChannel = zeros(ns,2,numOfNonIdealitySim);
    msgFromChannel(:,:,1) = iqNoiseInChannel + modTran(:,:,1);
    if(nonideality~=0)
        for l = 1:1:numOfNonIdealitySim - 1
            msgFromChannel(:,:,l+1) = iqNoiseInChannel + modTran(:,:,l+1);
        end
    end

    %Message detector
    recvdMsg = zeros(ns,nbits,numOfNonIdealitySim);
    curCor = zeros(ns,M,numOfNonIdealitySim);
    curCorMax = zeros(ns,1,numOfNonIdealitySim);
    curCorMaxIndex = zeros(ns,1,numOfNonIdealitySim);
    for i=1:1:ns
        for k=1:1:M
            curCor(i,k,1) = dot(msgFromChannel(i,:,1),mpskPhVec(k,:,1));
            if (k == 1)
                curCorMax(i,1) = curCor(i,k,1);
                curCorMaxIndex(i,1) = 1;
            else
                if( curCor(i,k,1) > curCorMax(i,1))
                    curCorMax(i,1) = curCor(i,k,1);
                    curCorMaxIndex(i,1) = k;
                end
            end
        end
        recvdMsg(i,:,1) = grayEncBi(curCorMaxIndex(i,1),:);
    end

```

```
    if(nonideality~=0)
        for l = 1:1:numOfNonIdealitySim - 1
            for i=1:1:ns
                for k=1:1:M
                    curCor(i,k,l+1) = dot(msgFromChannel(i,:,l+1),mpskPhVec(k,:,1));
                    if (k == 1)
                        curCorMax(i,l+1) = curCor(i,k,l+1);
                        curCorMaxIndex(i,l+1) = 1;
                    else
                         if( curCor(i,k,l+1) > curCorMax(i,l+1))
                            curCorMax(i,l+1) = curCor(i,k,l+1);
                            curCorMaxIndex(i,l+1) = k;
                        end
                    end
                end
                recvdMsg(i,:,l+1) = grayEncBi(curCorMaxIndex(i,l+1),:);
            end
        end
    end

    %Error Counter
    symErrCount(1)=0;
    bitErrCount(1)=0;
    for i=1:1:ns
        if (~isequal(recvdMsg(i,:,1),din(i,:)))
            symErrCount(1) = symErrCount(1) + 1;
        end
        for k=1:1:nbits
            if(recvdMsg(i,k,1) ~= din(i,k))
                bitErrCount(1) = bitErrCount(1) + 1;
            end
        end
    end
    if(nonideality~=0)
        for l = 1:1:numOfNonIdealitySim - 1
            symErrCount(l+1)=0;
            bitErrCount(l+1)=0;
            for i=1:1:ns
                if (~isequal(recvdMsg(i,:,l+1),din(i,:)))
                    symErrCount(l+1) = symErrCount(l+1) + 1;
```

```
                    end
                    for k=1:1:nbits
                        if(recvdMsg(i,k,l+1) ~= din(i,k))
                            bitErrCount(l+1) = bitErrCount(l
                                +1) + 1;
                        end
                    end
                end
            end
        end
    end
```

Listing 5: qamgen.m

```
function  out = qamgen(n)
    out = zeros(4*length(n)^2,2);
    input = sort(n, 'descend');
    k=1;
    for m=1:1:length(input)
        if (mod(m,2)~=0)
            for i=1:1:length(input)
                out(k,1) = input(m);
                out(k,2) = - input(i);
                k=k+1;
            end
            for i=length(input):-1:1
                out(k,1) = input(m);
                out(k,2) = input(i);
                k=k+1;
            end
        else
            for i=1:1:length(input)
                out(k,1) = input(m);
                out(k,2) = input(i);
                k=k+1;
            end
            for i=length(input):-1:1
                out(k,1) = input(m);
                out(k,2) = - input(i);
                k=k+1;
            end
        end
    end
    for m=length(input):-1:1
        if (mod(m,2)==0)
            for i=1:1:length(input)
                out(k,1) = - input(m);
```

```
                out(k,2) = - input(i);
                k=k+1;
            end
            for i=length(input):-1:1
                out(k,1) = - input(m);
                out(k,2) =  input(i);
                k=k+1;
            end
        else
            for i=1:1:length(input)
                out(k,1) = - input(m);
                out(k,2) =  input(i);
                k=k+1;
            end
            for i=length(input):-1:1
                out(k,1) = - input(m);
                out(k,2) = - input(i);
                k=k+1;
            end
        end
    end
end
```

Listing 6: qamsim.m

```
function [Eb,symErrCount,bitErrCount,EVM] = qamsim(N0,M,amp,ns)
    %Debug Parameters
    global curDis;
    global curDisMin;
    global curDisMinIndex;
    global Es;
    global din;
    global dinDe;
    global grayEncBi;
    global grayEncDe;
    global iqNoiseInChannel;
    global modTran;
    global msgFromChannel;
    global pdin;
    global qamVec;
    global recvdMsg;
    global nVar;

    %Calculate Es
    ampd=zeros(length(amp)^2,1);
    for i=1:1:length(amp)
```

```
        for k=1:1:length(amp)
            ampd((i-1)*2+k) = amp(k)^2+ amp(i)^2;
        end
    end
    Es = sum(ampd)/(length(amp)^2);

    nbits=log2(M);
    Eb=Es/nbits;
    pdin = rand(ns,1);
    din = zeros(ns,nbits);
    dinDe = zeros(ns,1);
    nVar = N0/2;

    %Gray encode table generator
    grayEncDe=graygen(nbits);
    grayEncBi=zeros(M,nbits);
    for i=1:1:M
        grayEncBi(i,:)=de2bi(grayEncDe(i),nbits);
    end

    %16QAM phase vector table generator [i,q]
    qamVec = qamgen(amp);

    %Uniform binary input generator
    for i=1:1:ns
        for k=0:1:M-1
            if (k/M<=pdin(i)) && (pdin(i)<(k+1)/M)
                din(i,:)=de2bi(k,nbits);
                dinDe(i) = k;
            end
        end
    end

    %Modulate inputs
    modTran=zeros(ns,2);
    for i=1:1:ns
        for k=1:1:M
            if isequal(din(i,:), grayEncBi(k,:))
                modTran(i,:)= qamVec(k,:);
            end
        end
    end

    %Generate additive Noise in the channel
    iqNoiseInChannel=zeros(ns,2);
    for i=1:1:ns
```

```
        iqNoiseInChannel(i,:) = normrnd(0,sqrt(nVar),[1
            2]);
    end

    %Message received
    msgFromChannel = zeros(ns,2);
    msgFromChannel = iqNoiseInChannel + modTran;

    %Message detector
    recvdMsg = zeros(ns,nbits);
    curDis = zeros(ns,M);
    curDisMin = zeros(ns,1);
    curDisMinIndex = zeros(ns,1);
    for i=1:1:ns
        for k=1:1:M
            curDis(i,k) = sqrt((msgFromChannel(i,1) -
                qamVec(k,1))^2 + (msgFromChannel(i,2) -
                qamVec(k,2))^2);
            if (k == 1)
                curDisMin(i) = curDis(i,k);
                curDisMinIndex(i) = 1;
            else
                if( curDis(i,k) < curDisMin(i))
                    curDisMin(i) = curDis(i,k);
                    curDisMinIndex(i) = k;
                end
            end
        end
        recvdMsg(i,:) = grayEncBi(curDisMinIndex(i),:);
    end

    %Error Counter
    symErrCount=0;
    bitErrCount=0;
    for i=1:1:ns
        if (~isequal(recvdMsg(i,:),din(i,:)))
            symErrCount = symErrCount + 1;
        end
        for k=1:1:nbits
            if(recvdMsg(i,k) ~= din(i,k))
                bitErrCount = bitErrCount + 1;
            end
        end
    end

    %EVM
```

```
    EVM = sqrt(sum(curDisMin.^2)/ns/(max(amp)^2 + max(amp)^2));
end
```

Listing 7: qfunc.m

```
function y = qfunc(x)
    y = 0.5 * erfc(x/sqrt(2));
    return;
```

Listing 8: PlotConstellationForMPSK.m

```
clear;
global mpskPhVec;
global msgFromChannel;

M=4;
noiseVar = [0.000001 0.00001 0.0001 0.001 0.01 0.1 1 10];
N0 = noiseVar.*2;
nonideality = 0;
ns=2500;
Es=1;



for i=1:1:length(N0)
    [Eb,symErrCount,bitErrCount] = mpsksim(N0(i),M,ns,Es,nonideality);
    subplot(2,4,i)
    hold on
    scatter(mpskPhVec(:,1),mpskPhVec(:,2),40, 'black','filled')
    scatter(msgFromChannel(1:1:1000,1),msgFromChannel(1:1:1000,2),3, 'red')
    title(sprintf('Constellation diagram for variance = %d',N0(i)/2));
    xlim([-1.2 1.2])
    ylim([-1.2 1.2])
    xlabel('I component')
    ylabel('Q component')
    hold off
end
```

Listing 9: PlotConstellationForQAM16.m

```
clear;
global msgFromChannel;
global qamVec;
```

```

M=16;
noiseVar = [0.000001, 0.00001 ,0.0001 ,0.001 ,0.01 ,0.1
    ,1 ,10];
N0 = noiseVar.*2;
nonideality = 0;
ns=250000;
Es=1;
amp = [3 1];

for i=1:1:length(N0)
    [Eb,symErrCount,bitErrCount,EVM] = qamsim(N0(i),M,amp
        ,ns);
    figure
    hold on;
    title(sprintf('Constellation diagram for variance = %
        d',N0(i)/2));
    scatter(qamVec(:,1),qamVec(:,2),40, 'black','filled')
    scatter(msgFromChannel(1:1:500,1),msgFromChannel
        (1:1:500,2),3, 'red')
    xlim([-3.2 3.2])
    ylim([-3.2 3.2])
    xlabel('I component')
    ylabel('Q component')
    hold off;
end
```

Listing 10: AnalysisForMPSKBER.m

```
colorlist=['c','b','g','r']
legendlist =['M=2','M=4','M=8','M=16']

h1=semilogy(10*log10(EbN0(:,1)),bitErrCountResult(:,1),
    strcat(colorlist(4),'-'));
hold on
h2=semilogy(10*log10(EbN0(:,2)),bitErrCountResult(:,2),
    strcat(colorlist(3),'-'));
h3=semilogy(10*log10(EbN0(:,3)),bitErrCountResult(:,3),
    strcat(colorlist(2),'-'));
h4=semilogy(10*log10(EbN0(:,4)),bitErrCountResult(:,4),
    strcat(colorlist(1),'-'));
legend([h1 h2 h3 h4],'M=2','M=4','M=8','M=16');
title('BER plot without mismatch');
ylim([10^-4 2])
xlabel('Eb/N0 (dB) ')
ylabel('Probability')
hold off
```

```
figure
h1=semilogy(10*log10(EbN0(:,1)),bitErrCountResult(:,1),
    strcat(colorlist(4),'-+'));
hold on;
h5=semilogy(10*log10(EbN0(:,1)),QresultBPSK(:,1),strcat(
    colorlist(4),'-*'));
hold off

figure
h2=semilogy(10*log10(EbN0(:,2)),bitErrCountResult(:,2),
    strcat(colorlist(4),'-+'));
hold on;
h6=semilogy(10*log10(EbN0(:,2)),QresultMPSK(:,2),strcat(
    colorlist(3),'-*'));
hold off

figure
h3=semilogy(10*log10(EbN0(:,3)),bitErrCountResult(:,3),
    strcat(colorlist(4),'-+'));
hold on;
h7=semilogy(10*log10(EbN0(:,3)),QresultMPSK(:,3),strcat(
    colorlist(2),'-*'));
hold off

figure
h4=semilogy(10*log10(EbN0(:,4)),bitErrCountResult(:,4),
    strcat(colorlist(4),'-+'));
hold on;
h8=semilogy(10*log10(EbN0(:,4)),QresultMPSK(:,4),strcat(
    colorlist(1),'-*'));
hold off
```

Listing 11: AnalysisForMPSKSER.m

```
colorlist=['c','b','g','r','m']
legendlist =['M=2','M=4','M=8','M=16']


h1=semilogy(10*log10(EbN0(:,1)),symErrCountResult(:,1),
    strcat(colorlist(4),'-'));
hold on
h2=semilogy(10*log10(EbN0(:,2)),symErrCountResult(:,2),
    strcat(colorlist(3),'-'));
h3=semilogy(10*log10(EbN0(:,3)),symErrCountResult(:,3),
    strcat(colorlist(2),'-'));
```

```
h4=semilogy(10*log10(EbN0(:,4)),symErrCountResult(:,4),strcat(colorlist(1),'-'));
legend([h1 h2 h3 h4],'M=2','M=4','M=8','M=16');
title('SER plot without mismatch');
ylim([10^-4 2])
xlabel('Eb/N0 (dB) ')
ylabel('Probability')
hold off



figure
h1=semilogy(10*log10(EbN0(:,1)),symErrCountResult(:,1),strcat(colorlist(4),'-+'));
hold on;
h5=semilogy(10*log10(EbN0(:,1)),Qresults(:,1),strcat(colorlist(2),'-.'));
legend([h1 h5],'Simulated','Theoretical');
title('SER plot without mismatch M=2 ns=250000');
xlabel('Eb/N0 (dB) ')
ylabel('Probability')
hold off

figure
h2=semilogy(10*log10(EbN0(:,2)),symErrCountResult(:,2),strcat(colorlist(4),'-+'));
hold on;
h6=semilogy(10*log10(EbN0(:,2)),Qresults(:,2),strcat(colorlist(2),'-.'));
legend([h2 h6],'Simulated','Theoretical');
title('SER plot without mismatch M=4 ns=250000');
xlabel('Eb/N0 (dB) ')
ylabel('Probability')
ylim([4*10^-6 2])
hold off

figure
h3=semilogy(10*log10(EbN0(:,3)),symErrCountResult(:,3),strcat(colorlist(4),'-+'));
hold on;
h7=semilogy(10*log10(EbN0(:,3)),Qresults(:,3),strcat(colorlist(2),'-.'));
legend([h3 h7],'Simulated','Theoretical');
title('SER plot without mismatch M=8 ns=250000');
xlabel('Eb/N0 (dB) ')
ylabel('Probability')
```

```
ylim([4*10^-6 2])
hold off

figure
h4=semilogy(10*log10(EbN0(:,4)),symErrCountResult(:,4),strcat(colorlist(4),'-+'));
hold on;
h8=semilogy(10*log10(EbN0(:,4)),Qresults(:,4),strcat(colorlist(2),'-.'));
legend([h4 h8],'Simulated','Theoretical');
title('SER plot without mismatch M=16 ns=250000');
xlabel('Eb/N0 (dB) ')
ylabel('Probability')
ylim([4*10^-6 2])
hold off
```

Listing 12: AnalysisForMPSKMismatchedBER.m

```
colorlist=['k','b','g','r','m']
legendlist =['M=2','M=4','M=8','M=16']

figure
h1=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,1,1),strcat(colorlist(1),'-d'));
hold on;
h2=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,1,2),strcat(colorlist(2),'-x'));
h3=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,1,3),strcat(colorlist(3),'-+'));
h4=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,1,4),strcat(colorlist(4),'-*'));
h5=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,1,5),strcat(colorlist(5),'-s'));
legend([h1 h2 h3 h4 h5],'Without Mismatch(M=2)','[0.01m 1p]','[0.05m 1p]','[0.01m 5p]','[0.05m 5p]')
title ('Mismatch effect on BER')
xlabel('Eb/N0 (dB)');
ylabel('Probability in log');
hold off

figure
h1=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,2,1),strcat(colorlist(1),'-d'));
hold on;
h2=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,2,2),strcat(colorlist(2),'-x'));
```

```
h3=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,2,3)
    ,strcat(colorlist(3),'-+'));
h4=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,2,4)
    ,strcat(colorlist(4),'-*'));
h5=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,2,5)
    ,strcat(colorlist(5),'-s'));
legend([h1 h2 h3 h4 h5],'Without Mismatch(M=4)','[0.01m 1
    p]','[0.05m 1p]','[0.01m 5p]','[0.05m 5p]')
title ('Mismatch effect on BER')
xlabel('Eb/N0 (dB)');
ylabel('Probability in log');
hold off

figure
h1=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,3,1)
    ,strcat(colorlist(1),'-d'));
hold on;
h2=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,3,2)
    ,strcat(colorlist(2),'-x'));
h3=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,3,3)
    ,strcat(colorlist(3),'-+'));
h4=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,3,4)
    ,strcat(colorlist(4),'-s'));
h5=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,3,5)
    ,strcat(colorlist(5),'-*'));
legend([h1 h2 h3 h4 h5],'Without Mismatch(M=8)','[0.01m 1
    p]','[0.05m 1p]','[0.01m 5p]','[0.05m 5p]')
title ('Mismatch effect on BER')
xlabel('Eb/N0 (dB)');
ylabel('Probability in log');
hold off

figure
h1=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,4,1)
    ,strcat(colorlist(1),'-d'));
hold on;
h2=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,4,2)
    ,strcat(colorlist(2),'-x'));
h3=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,4,3)
    ,strcat(colorlist(3),'-+'));
h4=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,4,4)
    ,strcat(colorlist(4),'-s'));
h5=semilogy(10*log10(EbN02(:,1)),bitErrCountResult(:,4,5)
    ,strcat(colorlist(5),'-*'));
legend([h1 h2 h3 h4 h5],'Without Mismatch(M=16)','[0.01m
    1p]','[0.05m 1p]','[0.01m 5p]','[0.05m 5p]')
```

```
title ('Mismatch effect on BER')
xlabel('Eb/N0 (dB)');
ylabel('Probability in log');
hold off
```

Listing 13: AnalysisForQAMBER.m

```
colorlist=['b','b','g','r']
counter2=1;
k = log2(M);
h1=semilogy(10*log10(EbN0),bitErrCountResult,strcat(
    colorlist(1),'-+'));
hold on

semilogy(10*log10(EbN0),Qresult,'-r*');

title('BER plot QAM');
xlabel('Eb/N0 (dB) ')
ylabel('Probability')
legend('Simulated','Theoretical');
```

Listing 14: AnalysisForQAMSER.m

```
colorlist=['b','b','g','r']
counter2=1;
k = log2(M);
h1=semilogy(10*log10(EbN0),symErrCountResult,strcat(
    colorlist(1),'-+'));
hold on

semilogy(10*log10(EbN0),Qresult,'-r*');

title('SER plot QAM');
xlabel('Eb/N0 (dB) ')
ylabel('Probability')
legend('Simulated','Theoretical');
```

Listing 15: DataCollectionSimulationForMPSKSER.m

```
clear;
global mpskPhVec;
global msgFromChannel;
M_list = [2 4 8 16];
nonideality = 0;
ns=250000;
Es=1;
counter=1;

```

```
for i=1:1:length(M_list)
    counter=1;
    k = log2(M_list(i));
    for N0=(Es/k)/10^(20/10):((Es/k)-Es/k/10^(20/10))/100:(Es/k)
        [Eb,symErrCount,bitErrCount] = mpsksim(N0,M_list(i),ns,Es,nonideality);
        EbN0(counter,i)=Eb/N0;
        symErrCountResult(counter,i)=symErrCount/ns;
        Qresults(counter,i)=2*qfunc((sqrt((2*log2(M_list(i))*(Eb/N0))))*sin(pi/M_list(i)));
        counter=counter+1
    end
end
```

Listing 16: DataCollectionSimulationForMPSKMisMatched.m

```
clear;
M_list = [2 4 8 16];
nonideality = 1;
ns=250000;
Es=1;
counter=1;

for i=1:1:length(M_list)

    counter=1;

    for N0=(Es/10^(20/10)):((Es-Es/10^(20/10))/100):(Es)
        [Eb,symErrCount,bitErrCount] = mpsksimwmismatch(N0,M_list(i),ns,Es,nonideality);
        EsN01(counter,i)=Eb*log2(M_list(i))/N0;
        EbN01(counter,i)=Eb/N0;
        for l=1:1:5
            symErrCountResult(counter,i,l)=symErrCount(l)/ns;
            bitErrCountResult(counter,i,l)=bitErrCount(l)/ns/log2(M_list(i));
        end
        counter=counter+1
    end

    counter=1;

    for N0=(Es/log2(M_list(i))/10^(20/10)):((Es/log2(M_list(i))-Es/log2(M_list(i))/10^(20/10))/100):(Es/log2(M_list(i)))
```

```
            [Eb,symErrCount,bitErrCount] = mpsksimwmismatch(N0,M_list(i),ns,Es,nonideality);
            EsN02(counter,i)=Eb*log2(M_list(i))/N0;
            EbN02(counter,i)=Eb/N0;
            for l=1:1:5
                symErrCountResult(counter,i,l)=symErrCount(l)/ns;
                bitErrCountResult(counter,i,l)=bitErrCount(l)/ns/log2(M_list(i));
            end
            counter=counter+1
        end

end
```

Listing 17: DataCollectionSimulationForMPSKBER.m

```
clear;
global mpskPhVec;
global msgFromChannel;
M_list = [2 4 8 16];
nonideality = 0;
ns=250000;
Es=1;

hold on
for i=1:1:length(M_list)
    counter=1;
    for N0=(Es/log2(M_list(i))/10^(20/10)):((Es/log2(M_list(i))-Es/log2(M_list(i))/10^(20/10))/100):(Es/log2(M_list(i)))
        [Eb,symErrCount,bitErrCount] = mpsksim(N0,M_list(i),ns,Es,nonideality);
        EbN0(counter,i)=Eb/N0;
        bitErrCountResult(counter,i)=bitErrCount/(ns*log2(M_list(i)));
        QresultBPSK(counter,i)=qfunc(sqrt((2*Eb/N0)));
        QresultMPSK(counter,i)=2*qfunc((sqrt((2*log2(M_list(i))*(Eb/N0))))*sin(pi/M_list(i)))/log2(M_list(i));
        counter=counter+1
    end
end
hold off;
```

Listing 18: DataCollectionSimulationForQAMBER.m

```
clear;
```

```
global qamVec;
global msgFromChannel;

ns=250000;
counter=1;
N0 = 2;
M = 16;
amp = [3 1];
k = log2(M);

for N0=(10/k/10^(20/10)):((10/k-10/k/10^(20/10))/200):10/
    k
    [Eb,symErrCount,bitErrCount,EVM] = qamsim(N0,M,amp,ns
        );
    EbN0(counter)=Eb/N0;
    symErrCountResult(counter)=symErrCount/ns;
    bitErrCountResult(counter)=bitErrCount/(ns*k);
    EVMResult(counter,1)=N0/2;
    EVMResult(counter,2)=EVM;
    Qresult(counter) = (4/k)*(1-1/(sqrt(M)))*qfunc(sqrt
        ((3*k*Eb)/(N0*(M-1))));
    counter=counter+1
end
```

Listing 19: DataCollectionSimulationForQAMSER.m

```
clear;
global qamVec;
global msgFromChannel;

ns=250000;
counter=1;
N0 = 2;
M = 16;
amp = [3 1];
k = log2(M);
Es=10;

for N0=(10/k/10^(20/10)):((10/k-10/k/10^(20/10))/200):10/
    k
    [Eb,symErrCount,bitErrCount,EVM] = qamsim(N0,M,amp,ns
        );
    EbN0(counter)=Eb/N0;
    symErrCountResult(counter)=symErrCount/ns;
    bitErrCountResult(counter)=bitErrCount/(ns*k);
    EVMResult(counter,1)=N0/2;
    EVMResult(counter,2)=EVM;
```

```
    Qresult(counter) = 1 - (1 - 2 * qfunc((3*Es/((M-1)*N0
        ))^0.5))^2;
    counter=counter+1
end
```