ÔN TẬP CUỐI KỲ THỰC HÀNH TRUYỀN THÔNG SỐ VÀ DỮ LIỆU

Câu 1 (Bài 2): Viết chương trình thực hiện mã hóa NRZ-L/ Bipolar AMI/ Manchester.

```
% NRZ-L
%% ======== Init
bitstream = [0 1 0 0 1 0 1 1 0 1]
pulse_high = 5;
pulse_low = 0;

%% ======== Create signal
for bit = 1:length(bitstream)
    %set bit time
    bt = bit-1:0.001:bit;
    if bitstream(bit) == 1
        y = (bt<bit)*pulse_high;
    else
        y = (bt<bit)*pulse_low;
    end
    try
        if bitstream(bit+1)==1
            y(end) = pulse_high;
        end
    catch e
        y(end) = pulse_high;
    end
    % draw pulse and label
    figure(1)
    plot(bt, y, 'LineWidth', 2);
    text(bit-0.5,pulse_high+0.5, num2str(bitstream(bit)), 'FontWeight', 'bold');
    hold on;
end

% draw grid
grid on;
axis([0 length(bitstream) pulse_low-1 pulse_high+1]);
set(gca,'YTick', [pulse_low pulse_high])
set(gca,'XTick', 1:length(bitstream))
title('Unipolar Non-Return-to-Zero Level')
```

```
% AMI
bitstream = [0 1 0 0 1 0 1 1 0 1]
pulse = 5;
current_pulse = -pulse;
for bit = 1:length(bitstream)
    bit_time = bit-1:0.001:bit;
    if bitstream(bit) == 0
        y = (bit_time<bit)*0;
    else
        current_pulse = -current_pulse;
        y = (bit_time<bit)*current_pulse;
    end
    try
        if bitstream(bit+1) == 1
            y(end) = -current_pulse;
```

```
        end
    catch e
        y(end) = -current_pulse;
    end
    plot(bit_time, y, 'LineWidth', 2);
    text(bit-0.5,pulse+0.5, num2str(bitstream(bit)), 'FontWeight', 'bold');
    hold on;
end
grid on;
axis([0 length(bitstream) -5 6]);
set(gca,'YTick', [-pulse pulse])
set(gca,'XTick', 1:length(bitstream))
title('Bipolar - AMI')
```

```
% Manchester
bitstream = [0 1 0 0 1 0 1 1 0 1]
pulse = 1;
yy = [];
for bit = 1:length(bitstream)
    bt = bit-1:0.01:bit;
    if bitstream(bit) == 1
        y = (bt<bit)*pulse - (bt<bit-0.5)*2*pulse;
        current_level = pulse;
    else
        y = -((bt<bit)*pulse - (bt<bit-0.5)*2*pulse);
        current_level = -pulse;
    end
    try
        if bitstream(bit+1) == bitstream(bit)
            y(end) = -current_level;
        else
            y(end) = current_level;
        end
    catch e
        y(end) = current_level;
    end
    plot(bt, y, 'LineWidth', 2);
    text(bit-0.5,pulse+0.25, num2str(bitstream(bit)), 'FontWeight', 'bold');
    hold on;
end
% draw grid
grid on;
axis([0 length(bitstream) -1 1.5]);
set(gca,'YTick', [-pulse pulse])
set(gca,'XTick', 1:length(bitstream))
title('Manchester')
```

Câu 2 (Bài 2): Viết chương trình vẽ phổ tín hiệu mã hóa NRZ-L/ Bipolar AMI/ Manchester cho chuỗi 20Kbits.

```
clear ; clc
bitstream = randi([0 1], 1, 20000); % So luong bit
fb=100; %(bps)
pulse_high = 1;
pulse_low = -1;
yy=[];
ts = 0.25; % Toc do lay mau
for bit = 1:length(bitstream)
    % set bit time
    bt = bit-1:ts:(bit-0.25);
    if bitstream(bit) == 0
        % low level pulse
        y = (bt<bit)*pulse_low;
    else
        % high level pulse
        y = (bt<bit) * pulse_high;
    end
    yy=[yy y];
end
k=zeros(100,800); % tong so mau: (1/ts)*so luong bit
for j=1:1:100
    k(j,:)=yy(800*j-799:800*j);
    sep(j,:)=fft(k(j,:),128);
end
n=size(sep,2);
fs=4*fb;
f = (0:n-1)*(fs/n); % frequency range
m_sep=mean((abs(sep).^2),1)/fs;
figure(1);
plot(f,m_sep);
% draw grid
grid on;
ylabel('Mean square voltage');
xlabel('Frequency');
title('Polar Non-Return-to-Zero Level');
```

```
bitstream = randi([0 1], 1, 20000);
fb=100;
yy=[];
pulse = 5;
ts = 0.25;
current_level = -pulse;
for bit = 1:length(bitstream)
    % set bit time
    bt = bit-1:ts:(bit-0.25);
    if bitstream(bit) == 0
        % binary 0, set to zero
        y = (bt<bit)*0;
    else
        % each binary 1 has the opposite pulse level from the previous
        current_level = -current_level;
        y = (bt<bit)*current_level;
```

```
    end
    % assign last pulse point by inspecting the following bit
    try
        % we care only about ones as they use alternate levels
        if bitstream(bit+1) == 1
            y(end) = -current_level;
        end
    catch e
        % bitstream end; assume next bit is 0
        y(end) = -current_level;
    end
    yy=[yy y];
end
k=zeros(100,800);
for j=1:1:100
    k(j,:)=yy(800*j-799:800*j);
    sep(j,:)=fft(k(j,:),128);
end
n=size(sep,2);
fs=4*fb;
f = (0:n-1)*(fs/n); % frequency range
m_sep=mean((abs(sep).^2),1)/fs;
figure(1);
plot(f,m_sep);
grid on;
xlabel('Frequency')
title('Bipolar - AMI')
```

```
num_bit = 20000;
bitstream = randi([0 1], 1, num_bit);
fb=100;
fs = 4;
pulse = 1;
yy=[];
for bit = 1:length(bitstream)
    % set bit time
    bt = bit-1:1/fs:(bit-0.25);
    if bitstream(bit) == 1
        % low -> high
        y = (bt<bit) * pulse - 2*pulse * (bt < bit- 0.5);
        % set last pulse point to high level
        current_level = pulse;
    else
        % high -> low
        y = -(bt<bit) * pulse + 2*pulse*(bt < bit - 0.5);
        % set last pulse point to low level
        current_level = -pulse;
    end
    try
        % if the next bit is the same as this one
        %change the level
        if bitstream(bit+1) == bitstream(bit)
            y(end) = -current_level;
        else
            y(end) = current_level;
        end
```

```
    catch e
        % assume next bit is the same as the last one
        y(end) = current_level;
    end
    yy=[yy y];
end
num_sample = fs*num_bit/100;
k=zeros(100,num_sample);
for j=1:1:100
    k(j,:)=yy(num_sample*j-(num_sample-1):num_sample*j);
    sep(j,:)=fft(k(j,:),128);
end
n=size(sep,2);
fs=4*fb;
f = (0:n-1)*(fs/n); % frequency range
m_sep=mean((abs(sep).^2),1)/fs;
figure(1);
plot(f,m_sep);
grid on;
xlabel('Frequency')
title('Manchester')
```

Câu 3 (Bài 4): Viết chương trình thực hiện mã hóa CRC cho dữ liệu 10 bit qua kênh nhị phân có xác suất lỗi là 0,1. Phía thu kiểm tra dữ liệu đúng hay sai dựa vào giải mã CRC và xuất ra thông báo là "TRANSMISSION SUCCESSFUL" hoặc "Retransmission Required".

```
clear;clc
data=randi([0 1],1, 10);
addbit = [0 0 0];
bit_data = [data addbit];
div=[1 0 1 1];
[q,r]=deconv(bit_data,div);
r = mod(r,2);
tx_data = bitxor(bit_data,r)
p_error = 0.1; %xac suat loi
rx_data = bsc(tx_data,p_error);
[qcheck, rcheck] = deconv(rx_data,div);
rcheck = mod(rcheck,2);
check = sum(rcheck);
if check ~= 0
    disp("Retransmission Required");
else
    disp("TRANSMISSION SUCCESSFUL");
end
```

Câu 4 (Bài 4): Viết chương trình mô phỏng mã hóa chập sử dụng điều chế QPSK. Vẽ giản đồ BER với EbNo = 0:2:18 dB và so sánh hai trường hợp có mã hóa và không có mã hóa.

```
clear; clc
rng default
M = 4; % Modulation order
k = log2(M); % Bits per symbol
EbNoVec = (0:2:18); % Eb/No values (dB)
numSymPerFrame = 1000; % Number of QAM symbols per frame
berEstHard = zeros(size(EbNoVec));
trellis = poly2trellis(7,[171 133]);
tbl = 32;
rate = 1/2;
for n = 1:length(EbNoVec)
    % Convert Eb/No to SNR
    snrdB = EbNoVec(n) + 10*log10(k*rate);
    % Noise variance calculation for unity average signal power.
    noiseVar = 10.^(-snrdB/10);
    % Reset the error and bit counters
    [numErrsHard,numBits] = deal(0);
    while numErrsHard < 100 && numBits < 1e7
        % Generate binary data and convert to symbols
        dataIn = randi([0 1],numSymPerFrame*k,1);
        % Convolutionally encode the data
        dataEnc = convenc(dataIn,trellis);
        % QAM modulate
        txSig = qammod(dataEnc,M,'InputType','bit','UnitAveragePower', true);
        % Pass through AWGN channel
        rxSig = awgn(txSig,snrdB,'measured');
        % Demodulate the noisy signal using harddecision (bit) and
        % soft decision (approximate LLR) approaches.
        rxDataHard = qamdemod(rxSig,M,'OutputType','bit','UnitAveragePower',true);
        % Viterbi decode the demodulated data
        dataHard = vitdec(rxDataHard,trellis,tbl,'cont','hard');
        % Calculate the number of bit errors in the frame. Adjust for the
        % decoding delay, which is equal to the traceback depth.
        numErrsInFrameHard = biterr(dataIn(1:end-tbl),dataHard(tbl+1:end));
        % Increment the error and bit counters
        numErrsHard = numErrsHard + numErrsInFrameHard;
        numBits = numBits + numSymPerFrame*k;
    end
    % Estimate the BER for both methods
    berEstHard(n) = numErrsHard/numBits;
    disp(['Done SNR = ',num2str(snrdB)])
end
%Plot the estimated hard and soft BER data. Plot the
%theoretical performance for an uncoded 64-QAM channel.
semilogy(EbNoVec, [berEstHard],'-*')
hold on
semilogy(EbNoVec,berawgn(EbNoVec,'qam',M))
legend('Hard','Uncoded','location','best')
grid
xlabel('Eb/No (dB)')
ylabel('Bit Error Rate')
```

Câu 5 (Bài 5): Viết chương trình mô phỏng giao thức Stop-and-Wait ARQ trường hợp truyền không bị sai, chờ ACK truyền frame tiếp theo.

```
function [pac] = MakeFrame(data,div)
    bit_data = [data, zeros(1,3)];
    [q,r]=deconv(bit_data,div);
    r = mod(r,2);
    pac = bitxor(bit_data,r);
    %pac=[0 1 1 1 1 1 1 0 pac];
end
```

```
clear;clc
%stop n wait protocol
pass=0; % The total number of transmitted frames
m=10; % The number of frames
n=7; % The frame length
tx=zeros(m,m);
RequestToSend = true;
Arrivaltx = false;
arivalrx=false;
canSend = true;
sn=1;
rn = 1;
div=[1 0 0 1];
msg=randi([0,1],m,n);
pac=[];
msgrx=[];
tx = [];
px=0.1;
%timer = 0.002; %timeout
while(sn<=m)
    pass=pass+1;
    %=============Transmitter
    if (RequestToSend&&canSend)
        pac(sn,:)=MakeFrame(msg(sn,:),div);
        tx(sn,:)= pac(sn,:);
        fprintf('Tx - Truyen frame thu %d \n',sn); %tic
        cn = sn;
        sn =sn+1;
        canSend = false;
        arivalrx=true;
    end
    %================Channel
    msgrx(cn,:)=bsc(tx(cn,:),px);

    %================Receiver
    if (arivalrx)
        [q2,r2]=deconv(msgrx(cn,:),div);
        r2(1,:)=mod(r2(1,:),2);
        arivalrx=false;
        canSend = true;
        if r2==0
            rn=rn+1;
            fprintf('Rx - San sang nhan frame thu %d \n',rn);
```

```
        else
            rn=rn+1;
            fprintf('Rx - Sai khong truyen lai %d\n',rn);
        end
    end
end
```

Câu 6 (Bài 5): Viết chương trình mô phỏng giao thức Stop-and-Wait ARQ trường hợp truyền bị mất frame, hết 0,07s đầu phát gửi lại frame đó.

```
function [pac] = MakeFrame(data,div)
    bit_data = [data, zeros(1,3)];
    [q,r]=deconv(bit_data,div);
    r = mod(r,2);
    pac = bitxor(bit_data,r);
    pac=[0 1 1 1 1 1 1 0 pac];
end
```

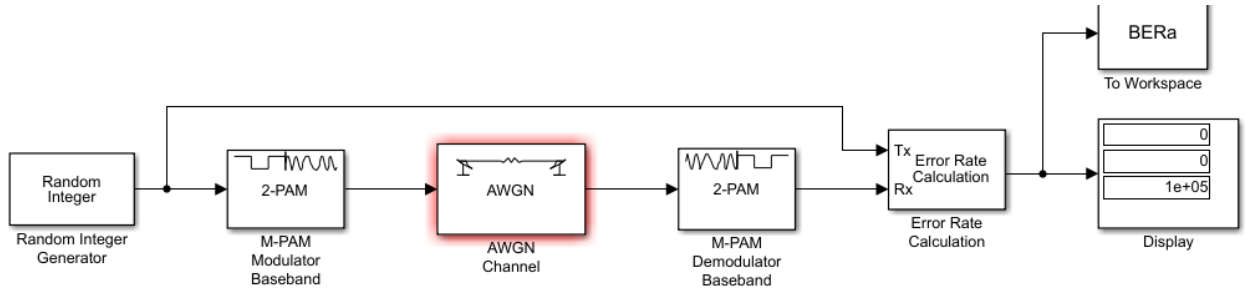
```
clear;clc
%stop n wait protocol
pass=0; % The total number of transmitted frames
m=10; % The number of frames
n=7; % The frame length
tx=zeros(m,m);
RequestToSend = true;
Arrivaltx = false;
arivalrx=false;
canSend = true;
sn=1;
rn = 1;
div=[1 0 0 1];
msg=randi([0,1],m,n);
pac=[];
msgrx=[];
tx = [];
px=1;
timer = 0.002; %timeout
while(sn<=m)
    pass=pass+1;
    %=============Transmitter
    if (RequestToSend&&canSend)
        pac(sn,:)=MakeFrame(msg(sn,:),div);
        tx(sn,:)= pac(sn,:);
        fprintf('Tx - Truyen frame thu %d \n',sn); tic
        cn = sn;
        sn =sn+1;
        canSend = false;
        arivalrx=true;
    end
    %================Channel
    msgrx(cn,:)=bsc(tx(cn,:),px);

    %================Set timeout
```

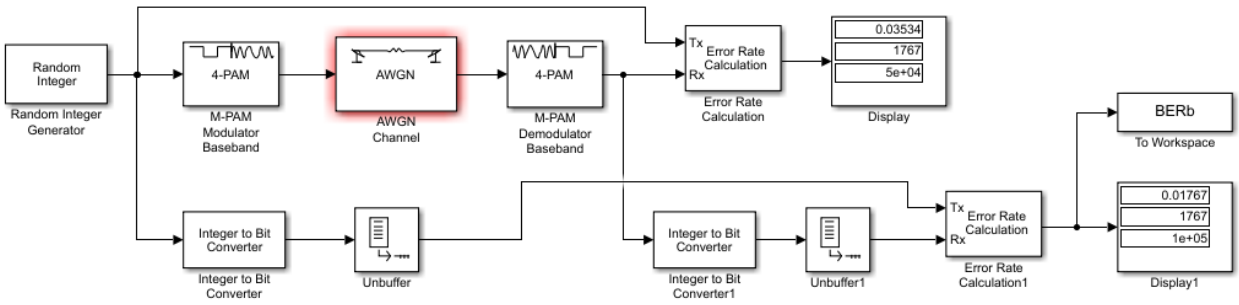
```
    if (toc > timer)
        canSend = true;
        px=0;
        sn = sn-1;
    end
    %================Receiver
    if (arivalrx)
        if (msgrx(cn,1:8)==[0 1 1 1 1 1 1 0])
            fprintf('Rx - Nhan duoc frame %d \n',rn);
            [q2,r2]=deconv(msgrx(cn,:),div);
            r2(1,:)=mod(r2(1,:),2);
            arivalrx=false;
            canSend = true;
            if r2==0
                rn=rn+1;
                fprintf('Rx - San sang nhan frame thu %d \n',rn);
            else
                rn=rn+1;
                fprintf('Rx - Sai khong truyen lai %d\n',rn);
            end
        else
            fprintf('Rx - Khong nhan duoc frame %d \n',rn);
        end
    end
end
```

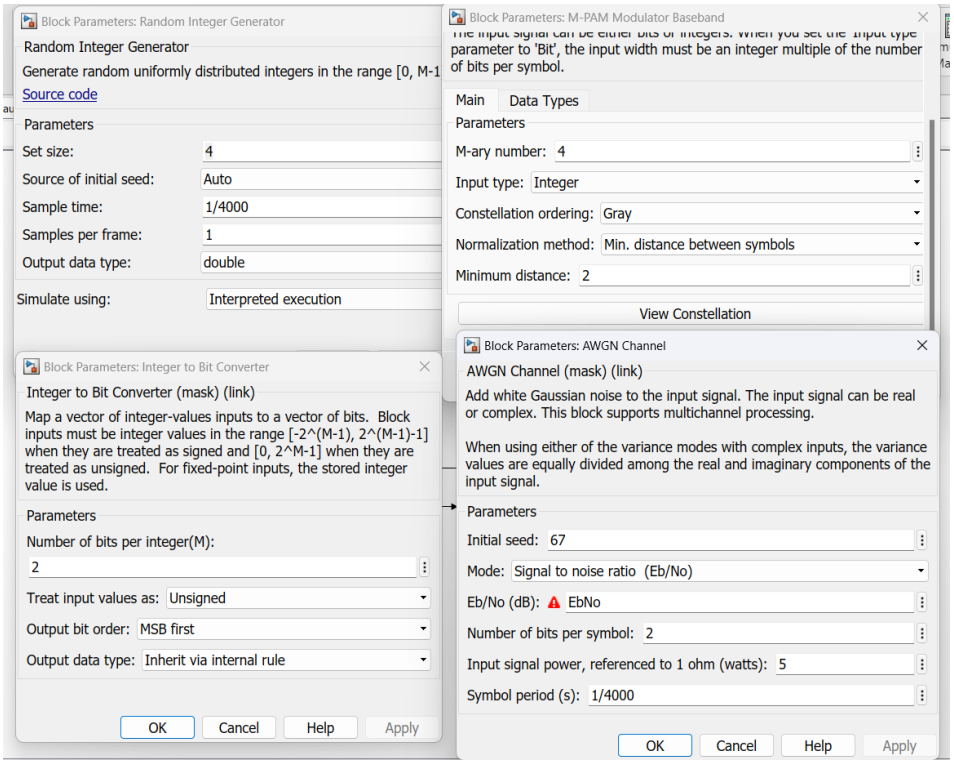
Câu 7 (Bài 6): Thiết kế mô hình thực hiện điều chế và giải điều chế 2-PAM/ 4-PAM/ QPSK với tốc độ bit phát là 4Kbps, kênh truyền có nhiễu AWGN. Sử dụng bertool để vẽ tỉ lệ lỗi bit (BER) và tỉ lệ lỗi symbol (SER) của mô hình và so sánh với BER lý thuyết. So sánh BER của hệ thống trong 2 trường hợp sử dụng điều biến theo mã Gray và mã Binary.

- 2-PAM

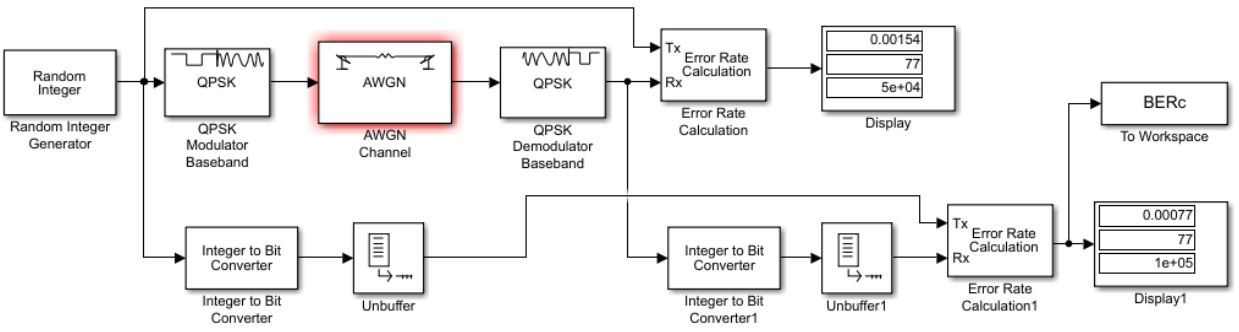


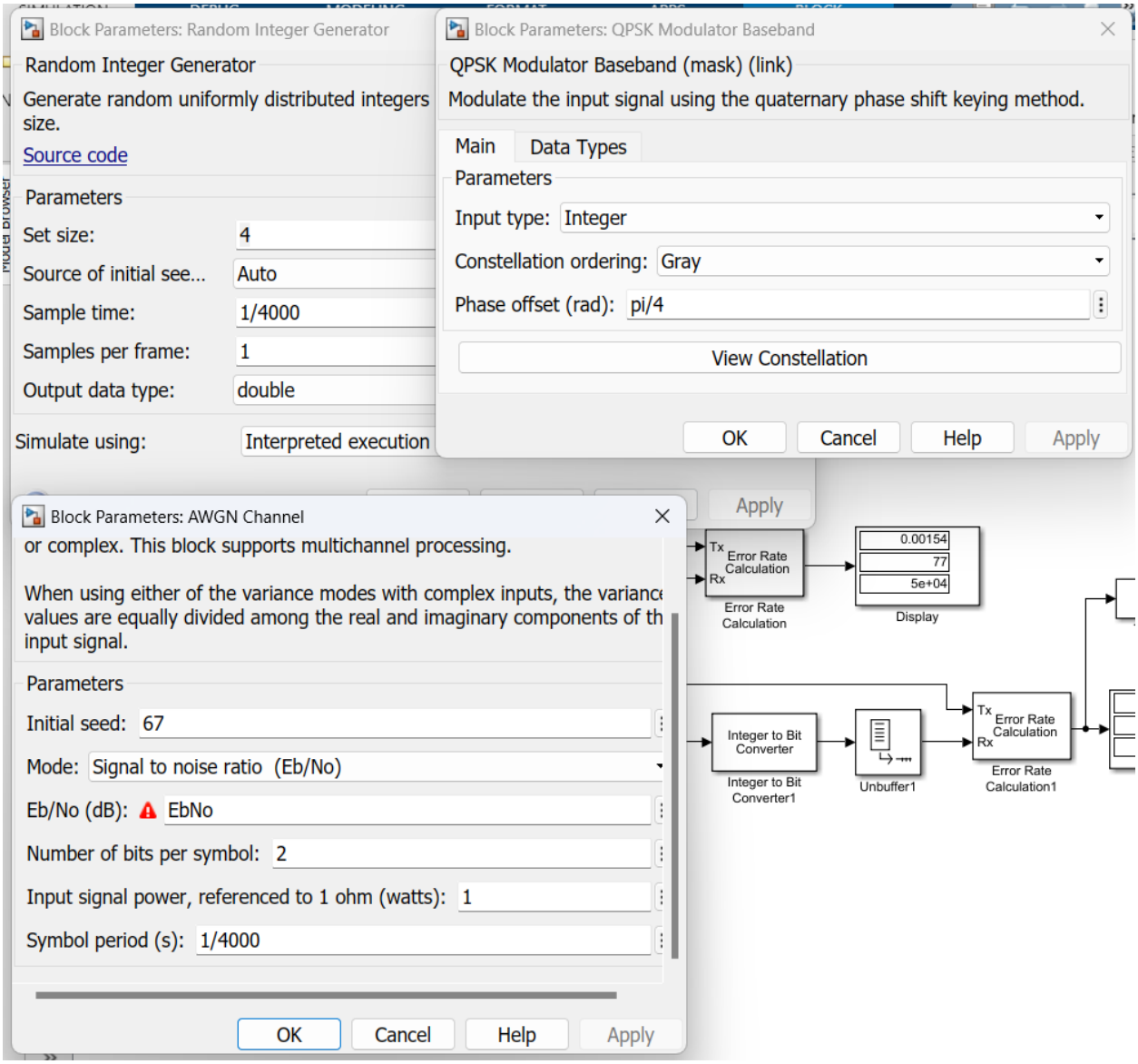
- 4-PAM


Random Integer
Random Integer Generator
4-PAM
M-PAM Modulator Baseband
AWGN
AWGN Channel
4-PAM
M-PAM Demodulator Baseband
Tx
Rx
Error Rate Calculation
Error Rate Calculation
0.03534
1767
5e+04
Display
BERb
To Workspace
Integer to Bit Converter
Integer to Bit Converter
Unbuffer
Integer to Bit Converter
Integer to Bit Converter1
Unbuffer1
Error Rate Calculation
Error Rate Calculation1
0.01767
1767
1e+05
Display1

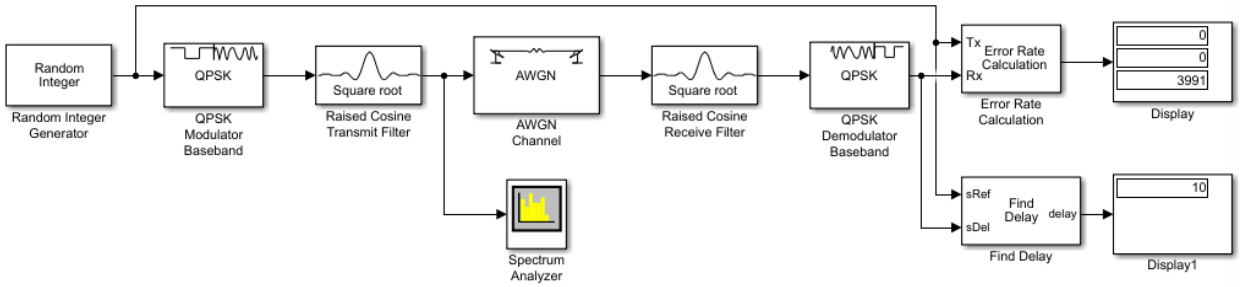

Block Parameters: Random Integer Generator
Random Integer Generator
Generate random uniformly distributed integers in the range [0, M-1
Source code
Parameters
Set size: 4
Source of initial seed: Auto
Sample time: 1/4000
Samples per frame: 1
Output data type: double
Simulate using: Interpreted execution
Block Parameters: M-PAM Modulator Baseband
parameter to 'Bit', the input width must be an integer multiple of the number of bits per symbol.
Main
Data Types
Parameters
M-ary number: 4
Input type: Integer
Constellation ordering: Gray
Normalization method: Min. distance between symbols
Minimum distance: 2
View Constellation
Block Parameters: Integer to Bit Converter
Integer to Bit Converter (mask) (link)
Map a vector of integer-values inputs to a vector of bits. Block inputs must be integer values in the range [-2^(M-1), 2^(M-1)-1] when they are treated as signed and [0, 2^M-1] when they are treated as unsigned. For fixed-point inputs, the stored integer value is used.
Parameters
Number of bits per integer(M): 2
Treat input values as: Unsigned
Output bit order: MSB first
Output data type: Inherit via internal rule
OK
Cancel
Help
Apply
Block Parameters: AWGN Channel
AWGN Channel (mask) (link)
Add white Gaussian noise to the input signal. The input signal can be real or complex. This block supports multichannel processing.
When using either of the variance modes with complex inputs, the variance values are equally divided among the real and imaginary components of the input signal.
Parameters
Initial seed: 67
Mode: Signal to noise ratio (Eb/No)
Eb/No (dB): EbNo
Number of bits per symbol: 2
Input signal power, referenced to 1 ohm (watts): 5
Symbol period (s): 1/4000
OK
Cancel
Help
Apply

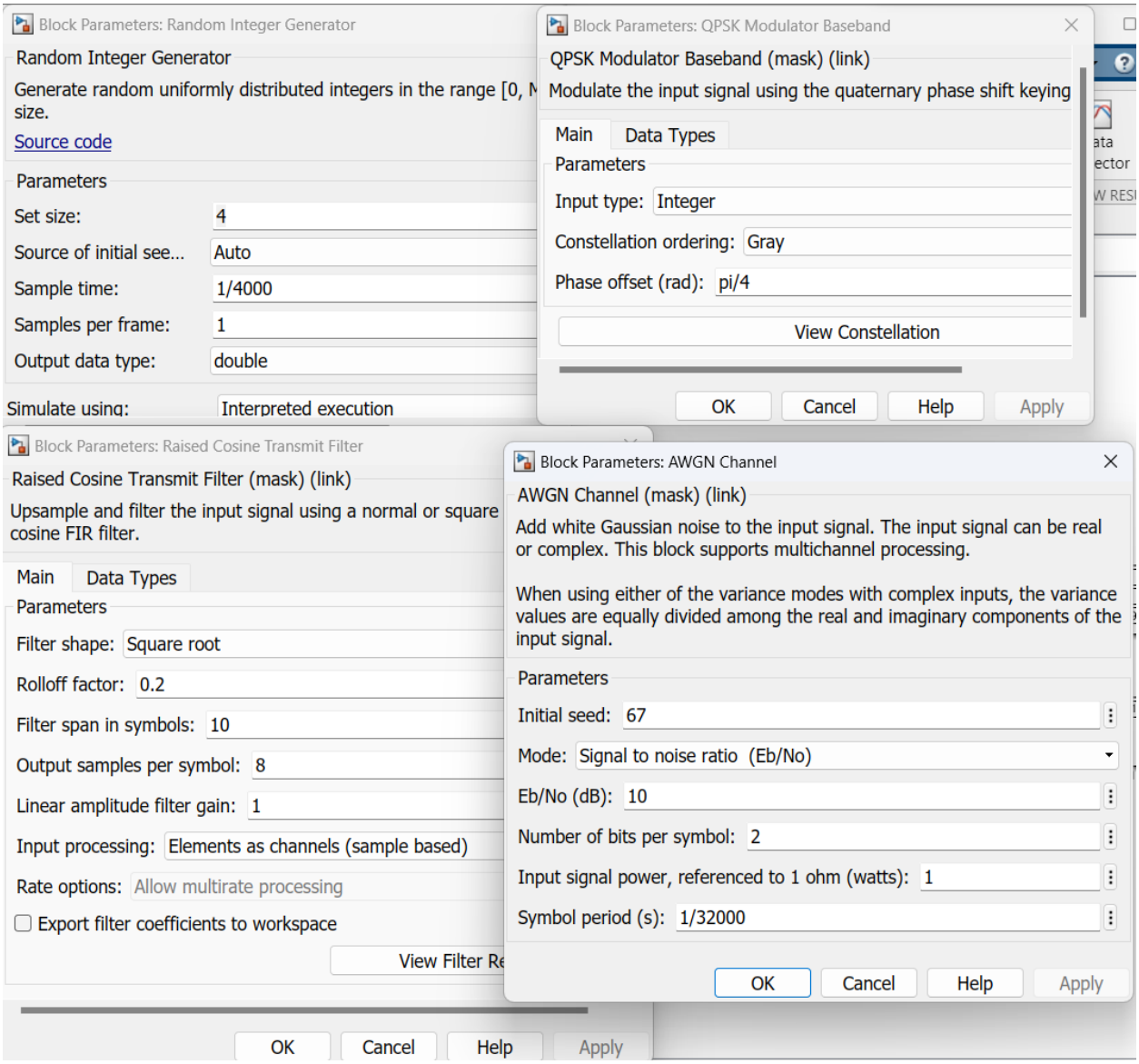
- QPSK


Random Integer
Random Integer Generator
QPSK
QPSK Modulator Baseband
AWGN
AWGN Channel
QPSK
QPSK Demodulator Baseband
Tx
Rx
Error Rate Calculation
Error Rate Calculation
0.00154
77
5e+04
Display
BERc
To Workspace
Integer to Bit Converter
Integer to Bit Converter
Unbuffer
Integer to Bit Converter
Integer to Bit Converter1
Unbuffer1
Error Rate Calculation
Error Rate Calculation1
0.00077
77
1e+05
Display1



Câu 8 (Bài 7): Thiết kế mô hình thực hiện lọc tạo dạng xung raised-cosin và sử dụng bộ thu matched filter cho tín hiệu QPSK truyền dẫn dải gốc, tốc độ bit là 4Kbps. Thiết kế bộ đếm lỗi; quan sát băng thông tín hiệu





Câu 9 (Bài 7): Viết chương trình thực hiện lọc tạo dạng xung raised-cosin và sử dụng bộ thu matched filter trong hệ thống truyền dẫn dải gốc sử dụng điều biến 4-PAM

```
clear all;clc
pulseCenter=10;
alpha=0.25;
Nspsym = 4;% Number of sample per symbol
N=10; % Number of bit
power_of_noise = 5;

%% ============Transmit
data=randi([0 1],1,N) % Generate data

% PAM modulation
```

```
data2 = pammod(data,4);

% Raised cosine
data_up= upsample(data2,4);
pulseTx = srrc(-pulseCenter:pulseCenter, alpha, 3);
s=conv(data_up,pulseTx,'full');

% Channel
ynoisy = awgn(s,power_of_noise,'measured');

%% =============Receive
r=ynoisy((pulseCenter+1):Nspsym:(end-pulseCenter))
r_de=pamdemod(r,4)
y_mf=conv(ynoisy,pulseTx,'full');
r_mf=y_mf((2*pulseCenter+1):Nspsym:(end-2*pulseCenter))
r_de_mf=pamdemod(r_mf,4)

%% =====Power
r_power = mean(abs(r).^2)
r_mf_power = mean(abs(r_mf).^2)

%% =======Plot
figure(1);
plot(ynoisy)
figure(2);
plot(y_mf)

%% Raised Cosine
t = -10:10
x = srrc(t, 0.5, 3)
plot(t,x);
```

```
function X = srrc(t, alpha, Ts)
%SRRC Returns a square-root raised cosine pulse
X = (sin((1 - alpha) * pi / Ts * t) ...
+ 4 * alpha / Ts * t .* cos((1 + alpha) * pi / Ts * t)) ...
./ (pi / sqrt(Ts) * t .* (1 - (4 * alpha / Ts * t) .^ 2));
X(t == 0) = (1 - alpha + 4 * alpha / pi) / sqrt(Ts);
X(abs(t) == Ts / 4 / alpha) = alpha / sqrt(2 * Ts) * ...
((1 + 2 / pi) * sin(pi / 4 / alpha) + ...
(1 - 2 / pi) * cos(pi / 4 / alpha));
end
```