Digital Radio Design with Matlab

Budi Mulyanto

***Abstract*— In this work, a system of Digital Radio or Software-defined Radio was built and simulated. The approach was by building function blocks one-by-one. The simulation was done in Matlab. Impairments were included to make the simulation closer to the real life. The solution for each impairments or nonideal factors were provided and used to neutralize the nonideal effect.**

***Index Terms*— Digital Radio, Software-defined Radio, Digital communication.**

I. INTRODUCTION

DESIGN was done in two main phases. In the first phase, it was assumed that all the models were ideal. The main purpose of this phase is to make sure that all the blocks in the transmitter and receiver, including the channel, work. This idealized design also gave the big picture how the transmitter and receiver work. Figure 1 shows the block diagram of the design. The second phase includes some nonideal factors into the building blocks introduced in the first phase. This phase tried to simulate the implementation of digital radio in the real life with its nonideal issues, e.g., nonideal channel, noise, etc. This report mainly summarizes the result of the simulation for nonideal case. Table 1 summarizes all the nonideal factors discussed and its solutions.

Nonideal factors	**Solutions**
Phase offset	Carrier recovery
Noise	Matched filter and coding
Time offset of pulse shaping	Clock recovery
channel	equalizer

Table 1. Nonideal factors and its solutions.

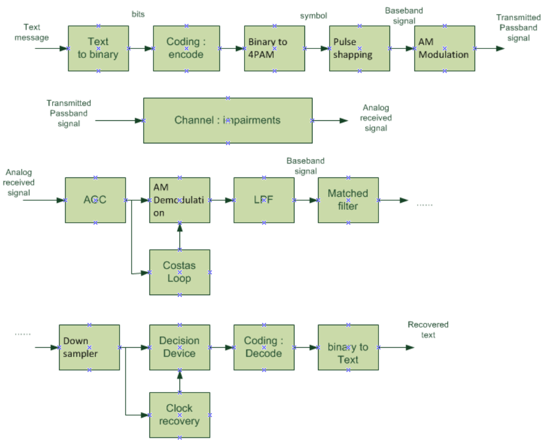


Figure 1. The block diagram of the software defined radio. A larger version is available at the end of this report.

Specification	
Symbol	4 PAM
Carrier Frequency	20 Hz
Oversampling factor	100
SRRC's width	0.5
F cut-off LPF	22 Hz
LPF's length	100
Coding	blockcode (5,2)

Table 2. Specification.

The rest of this report is organized as follows. Section II discusses the detail of each building blocks of the Software-defined Radio (SDR). Section III shows simulation result. In this section, a simple text is used as an input which later is transmitted, received, and decoded to get the original message. At the end, a short summary closes this report.

II. BUILDING BLOCKS

The detail of each building blocks is described here. The building blocks can be listed as follow.

A. TRANMITTER
   1. Text to binary
   2. Encode
   3. Binary to 4PAM
   4. Pulse shaping
   5. AM modulation

B. CHANNEL : impairments
   1. ISI
   2. Gaussian noise
   3. Frequency offset
   4. Phase offset
   5. Symbol period offset

C. RECEIVER
   1. AGC
   2. Costas Loop
   3. AM demodulation
   4. LPF
   5. Matched filter
   6. Clock recovery
   7. Down sampler
   8. Decision device
   9. Decode
   10. Binary to text

The discussion follows the flow from Figure 1.

A. TRANMITTER
   1. Text to binary
      Function: Translate the text into its binary representation (ASCII).

2. Encode
   Function: Encode the binary into a certain format which is more noise resistance. Encoding results redundancy because the data is larger than the original. However, with this encoding, errors can be detected and corrected.
   The format used in this encoding is called blockcode(5,2). This encoding forms packages from the original data. Each package contains 2 bits. Packages is multiplied with a matrix generator G. The multiplication will result unique 5 bit binary. The pairs are as follow.
   00 ←→00000
   01 ←→01011
   10 ←→10101
   11 ←→11110

   The matrix generator:
   $$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

   This discussion will be continued later in the decoding part.

3. Binary to 4PAM
   Function: translate 2 bit binary into 4PAM.
   If the transmission were done using binary format, it would be hard to distinguish between data '0' (bit zero) and no transmission. For this reason, 4 PAM format is used.
   The conversion is described as follow.
   00 ←→-3
   01 ←→-1
   10 ←→1
   11 ←→3

4. Pulse shaping
   Function: converting the digital data into analog signal.
   Before transmitting the digital data, it must be converted into analog signal. Pulse shaping generates an analog signal whose values represent the digital data. In this design, the Square Root Raised Cosine (SRRC) function was used to shape the pulse. The SRRC equation is written as follow.

$$srrc(t) = \begin{cases} \dfrac{1}{\sqrt{T}} \dfrac{\sin\left(\dfrac{\pi(1-\beta)t}{T}\right) + \left(\dfrac{4\beta t}{T}\right)\cos\left(\dfrac{\pi(1+\beta)t}{T}\right)}{\left(\dfrac{\pi t}{T}\right)\left(1-\left(\dfrac{\beta t}{T}\right)^2\right)} \text{............} t \neq 0, t \neq \pm\dfrac{T}{4\beta} \\ \dfrac{1}{\sqrt{T}}\left(1-\beta+\left(\dfrac{4\beta}{\pi}\right)\right) \text{.....................................} t = 0 \\ \dfrac{\beta}{\sqrt{2T}}\left[\left(1+\dfrac{2}{\pi}\right)\sin\left(\dfrac{\pi}{4\beta}\right)+\left(1-\dfrac{2}{\pi}\right)\cos\left(\dfrac{\pi}{4\beta}\right)\right] \text{......} t = \pm\dfrac{T}{4\beta} \end{cases}$$

5. AM modulation
   Function: modulate the analog signal into a passband signal.
   Efficient transmission requires an antennae with length of 1/10 of the wavelength. So higher carrier frequency transmission requires shorter antennae. However, simulation using higher carrier frequency require more computational recourse. In this work, the carrier frequency was 20 Hz, much lower than the recommended carrier frequency (2 MHz according to M6 Specification).

B. CHANNEL : impairments
   1. Fading
      Fading effect simulate the moving transmitter and/or receiver.
   2. Inter-symbol Interference (ISI)
      ISI describe the overlapping of 2 symbols due to wider pulse width during pulse shaping.
   3. Gaussian noise
      Noise in the channel which is modeled as a Gaussian noise.
   4. Frequency offset
      This impairment describes the difference between the frequency in the transmitter and receiver.
   5. Phase offset
      This impairment describes the difference between the phase in the transmitter and receiver.
   6. Symbol period offset
      This impairment can cause inaccurate sampling time.

C. RECEIVER
   1. AGC
      Function: neutralize the fading effect.
      AGC neutralize the fading by keeping the signal at a certain power.
   2. Costas Loop
      Function: estimate the phase of the transmitter oscillator so the receiver oscillator has negligible phase difference.
      The Costas Loop diagram is shown in Figure 2.

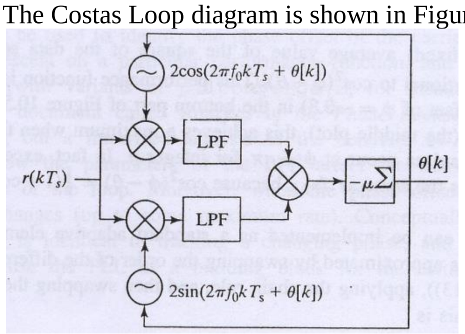


      Figure 2. Costas Loop diagram.
   3. AM demodulation
      Function: demodulate passband signal into baseband signal. The phase estimated by the Costas Loop is used here.
   4. LPF
      Function: filtering the baseband signal.
      The demodulation creates the baseband signal and some artifact at higher frequency. This LPF is used to get the baseband signal.
      The transfer function of the LPF is written below.
      $$Y(z) = \frac{b(1) + b(2)z^{-1} + \ldots + b(nb+1)z^{-nb}}{1 + a(2)z^{-1} + \ldots + a(na+1)z^{-na}} X(z)$$

      The cut off frequency of this filter is 22 Hz

5. Matched filter
   Function: increase SNR.
   To maximize the SNR, the same filter used in the pulse shaping in the transmitter must be used in the receiver. In this case, it is the SCCR function.
6. Clock recovery
   Function: estimate the frequency offset.
   It is important to sampling the signal at the right time to get a correct information. Frequency offset can make the system sample at the wrong time.
7. Down sampler
   Function: converting the baseband signal into the symbol frequency.
   This is required so the amplitude of the signal can be determined later by the decision device.
8. Decision device
   Function: translate the amplitude value into 4 PAM.
9. Decode
   Function: translate the 4 PAM into binary.
   The binary data is multiplied with matrix H^T.

$$H^T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

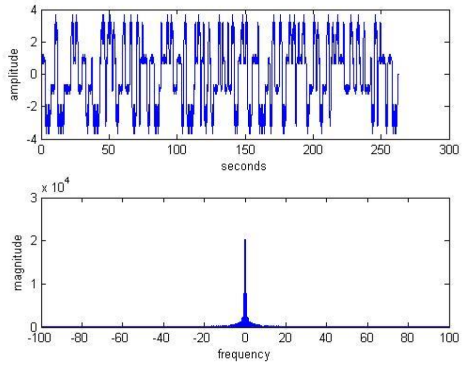
   If there is no errors, the product of two matrix would result 0. In case there are some errors, subtract the corresponded value found in the error table from the multiplication result. This subtraction result would be one of the four possible 5-digit values describe in the transmitter. (See Encode part.)
   This method can also correct maximum 1 wrong bit out of 5. Coding also increase the security of the transmission.
10. Binary to text
    Function: translate the ASCII into text.

III. SIMULATION

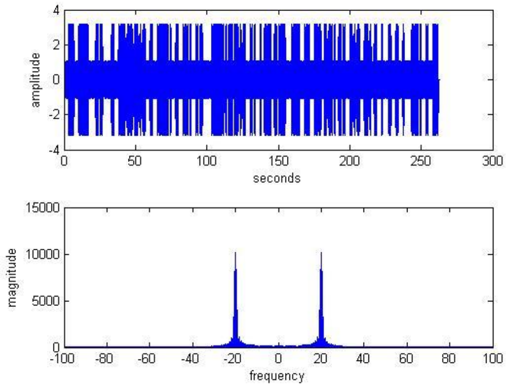
The Matlab code was written to simulate the nonideal transmission and decode back in the receiver to get the original text.

The transmitted text: "A0Oh well whatever Nevermindl". The simulation result reported below follows the process sequence described in the block diagram in Figure 1.

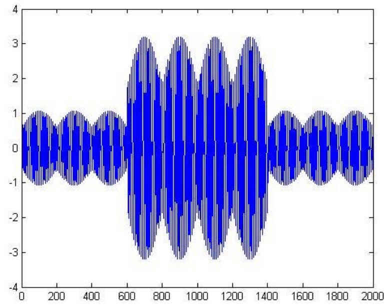
1. Encoding text into 4 PAM



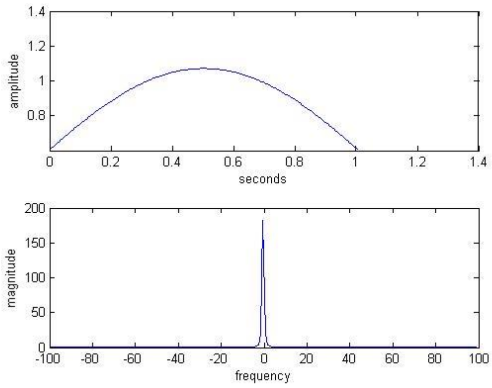
2. Modulated signal



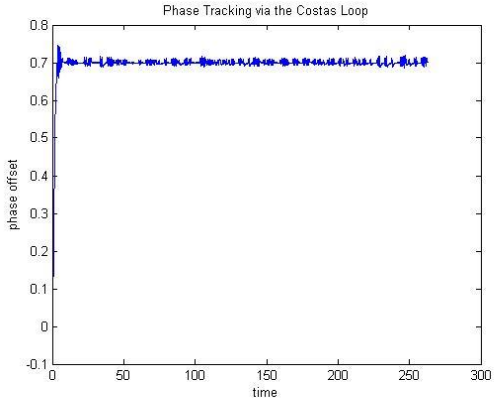
3. The first 10 symbols from the modulated signal



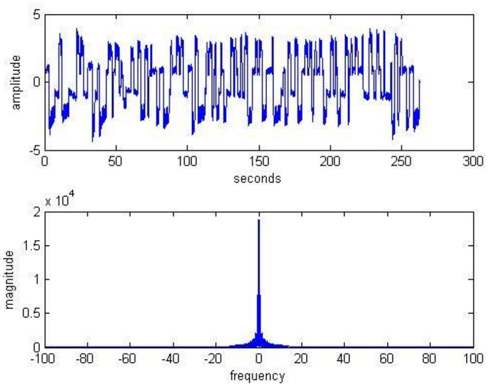
4. Pulse shaping



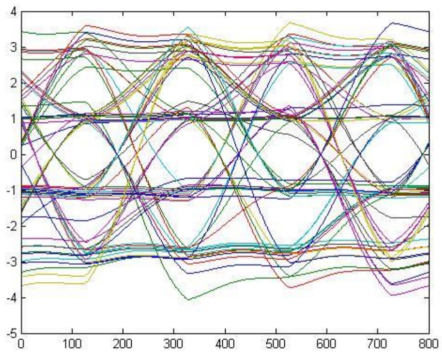
5. Phase tracking with Costas Loop



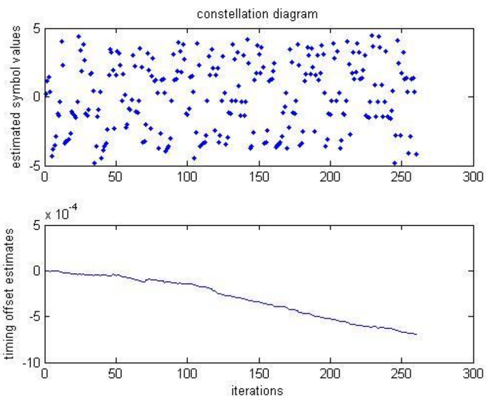
6. Demodulated signal



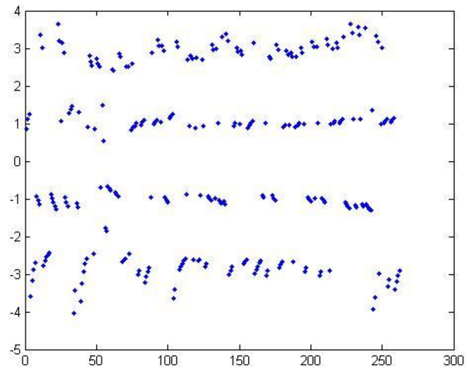
7. Eye diagram after matched filter



8. Clock recovery: frequency tracking



9. 4 PAM Data (after decoding)



This is the resulted text in the receiver:

ytext =

A0Oh wmll whatever Nevermindl

percentage_symbol_errors =

1.1450

It can be seen there was 1 wrong character. This is caused by the bad result in the frequency tracking. From the picture in the Clock recovery section, it can be seen the system could not estimate and track the frequency offset. It deviates far from zero. Such offset caused the decision device to misinterpret the amplitude. This is shown in the resulted 4 PAM signal in the diagram above (No. 9). There are some points which are located far from the correct 4 PAM values (+3, +1, -1, -3).

Other parts of the system work well. For example, the carrier recovery which successfully track the phase offset and the LPF that filters out the higher frequency signal.

IV. Summary

The system works perfectly under the condition close to ideal. If there were so many impairments, the clock recovery block could not track the frequency offset correctly. This caused bad 4 PAM signal i.e., data points were spread out about the correct 4 PAM values (+3, +1, -1, -3).

V. References

C.R. Johnson and W.A. Sethares, *Telecommunications Breakdown: Concepts of Communication Transmitted via Software-Defined Radio*, Prentice Hall, 2003.

Appendix

1. TRANSMITTER : CHANNEL : RECEIVER

```
clear all
clf

% specification of impairments

cng=input('channel noise gain: try 0, 0.6 or 2 :: ');
cdi=input('channel multipath: 0 for none, 1 for mild or 2 for harsh ::  ');
fo =input('tranmsitter mixer freq offset in %: try 0 or 0.01 ::  ');
po =input('tranmsitter mixer phase offset in rad: try 0, 0.7 or 0.9 ::  ');
toper=input('baud timing offset as % of symb period: try 0, 20 or 30 ::  ');
so=input('symbol period offset: try 0 or 1 ::  ');

%TRANSMITTER
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% encode text string as T-spaced PAM (+/-1, +/-3) sequence
str2='01234 I wish I were an Oscar Mayer wiener 56789 ';
str3 = 'A0Oh well whatever Nevermind1 ';
str4 = 'Awell ';
n=text2bin(str3);          % change text into 7 bit binary using text2bin
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% encode
coded_txt = blockcode52_encode(n);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% biner to 4-PAM
j=1; mpam=zeros(1,ceil(length(coded_txt)/2));
for i=1:2:length(coded_txt)-1                  % and then into 4-PAM
  if coded_txt(i:i+1)==[0,0], mpam(j)=-3; end
  if coded_txt(i:i+1)==[0,1], mpam(j)=-1; end
  if coded_txt(i:i+1)==[1,0], mpam(j)=1; end
  if coded_txt(i:i+1)==[1,1], mpam(j)=3; end
  j=j+1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N=length(mpam);                                 % 4-level signal of length N
% zero pad T-spaced symbol sequence to create upsampled T/M-spaced
%       sequence of scaled T-spaced pulses (with T = 1 time unit)
M=200-so; mup=zeros(1,N*M); mup(1:M:end)=mpam; % oversampling factor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SRRC pulse filter with T/M-spaced impulse response
L=0.5; p=(14.0)*srrc(L,0.3,M,0.4);             % blip pulse of width M
x=filter(p,1,mup);                    % convolve pulse shape with data
figure(1), plotspec(x,1/M)            % baseband signal spectrum
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% am modulation
t=1/M:1/M:length(x)/M;                % T/M-spaced time vector
fc=20;                                % carrier frequency
c=cos(2*pi*(fc*(1+0.01*fo))*t+po);   % carrier with offsets relative to rec osc
r=c.*x;                               % modulate message with carrier
figure(2),plot(r(1:10*M));
figure(3), plotspec(r,1/M);
figure(4), plotspec(p,1/M);
% OK!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%CHANNEL

%IMPAIRMENT : FADING
ds=pow(r);                         % desired average power of signal
lr=length(r);                      % length of transmitted signal vector
fp=[ones(1,floor(0.2*lr)),0.5*ones(1,lr-floor(0.2*lr))]; % flat fading profile
r=r.*fp;                           % apply profile to transmitted signal vector

if cdi < 0.5,                     % channel ISI
  mc=[1 0 0];                     % distortion-free channel
elseif cdi<1.5,
  mc=[1 zeros(1,M) 0.28 zeros(1,2.3*M) 0.11]; % mild multipath channel
else
  mc=[1 zeros(1,M) 0.28 zeros(1,1.8*M) 0.44]; % harsh multipath channel
end
mc=mc/(sqrt(mc*mc'));             % normalize channel power
dv=filter(mc,1,r);                % filter transmitted signal through channel
nv=dv+cng*(randn(size(dv)));      % add Gaussian channel noise
to=floor(0.01*toper*M);           % fractional period delay in  sampler
rnv=nv(1+to:end);                 % delay in on-symbol designation
rt=(1+to)/M:1/M:length(nv)/M;     % modified time vector with delayed message start
rM=M+so;                          % receiver sampler timing offset (delay)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%RECEIVER
```

```
% automatic gain control (AGC)
g=zeros(1,lr); g(1)=1;          % initialize gain
nr=zeros(1,lr);
mu=0.0003;                      % stepsize
for i=1:lr-1
 nr(i)=g(i)*r(i);               % AGC output
 g(i+1)=g(i)-mu*(nr(i)^2-ds);   % adapt gain
end
rnv=nr;                           % received signal is still called r

% pllconverge.m simulate costas loop
% input rsc from pulrecsig.m
rpll=rnv;                                  % rsc is from pulrecsig.m
fl=100; ff=[0 .01 .02 1]; fa=[1 1 0 0];
h=remez(fl,ff,fa);                        % LPF design
mu=.003;                                  % algorithm stepsize
fc=20;                                  % assumed freq. at receiver
theta=zeros(1,length(t)); theta(1)=0;     % initialize estimate vector
zs=zeros(1,fl+1); zc=zeros(1,fl+1);       % initialize buffers for LPFs
for k=1:length(t)-1                       % z's contain past fl+1 inputs
  zs=[zs(2:fl+1), 2*rpll(k)*sin(2*pi*fc*t(k)+theta(k))];
  zc=[zc(2:fl+1), 2*rpll(k)*cos(2*pi*fc*t(k)+theta(k))];
  lpfs=fliplr(h)*zs'; lpfc=fliplr(h)*zc'; % new output of filters
  theta(k+1)=theta(k)-mu*lpfs*lpfc;       % algorithm update
end

figure(5),plot(t,theta),
title('Phase Tracking via the Costas Loop')
xlabel('time'); ylabel('phase offset')


theta;
phoff = theta;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%am demodulation of received signal sequence r
ram = rnv;
c2=cos(2*pi*fc*t + phoff);                 % synchronized cosine for mixing
x2=ram.*c2;                              % demod received signal
% LPF design
fl=100;                                 % LPF length
fbe=[0 0.1 0.2 1]; damps=[1 1 0 0 ]; % design of LPF parameters
b=remez(fl,fbe,damps);               % create LPF impulse response
x3=2*filter(b,1,x2);                 % LPF and scale downconverted signal
figure(6),plotspec(x3,1/M)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % matchfilt.m: test of SNR maximization
%x3 = r; fl = 100;
recfilt=(15)*srrc(L,0.3,M,0.0);                 % receive filter H sub R
%recfilt=recfilt/sqrt(sum(recfilt.^2));  % normalize the pulse shape
v=1/180*filter(fliplr(recfilt),1,x3);        % matched filter with data
figure(7), ul=floor((length(v)-124)/(4*rM));
plot(reshape(v(125:ul*4*rM+124),4*rM,ul)) % plot the eye diagram
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%clock recovery algorithm
xcl = v; n = N; l = L;
tnow=l*M+1; tau=0; xs=zeros(1,n);         % initialize variables
tausave=zeros(1,n); tausave(1)=tau; i=0;
mu=0.003;                                 % algorithm stepsize
delta=0.1;                                % time for derivative
while tnow<length(xcl)-2*l*M                % run iteration
  i=i+1;
  xs(i)=interpsinc(xcl,tnow+tau,l);         % interpolated value at tnow+tau
  x_deltap=interpsinc(xcl,tnow+tau+delta,l); % get value to the right
  x_deltam=interpsinc(xcl,tnow+tau-delta,l); % get value to the left
  dx=x_deltap-x_deltam;                   % calculate numerical derivative
  qx=quantalph(xs(i),[-3,-1,1,3]);        % quantize xs to nearest 4-PAM symbol
  tau=tau+mu*dx*(qx-xs(i));               % alg update: DD
  tnow=tnow+M; tausave(i)=tau;            % save for plotting
end
figure(8), subplot(2,1,1), plot(xs(1:i-2),'b.')    % plot constellation diagram
title('constellation diagram');
ylabel('estimated symbol values')
subplot(2,1,2), plot(tausave(1:i-2))               % plot trajectory of tau
ylabel('timing offset estimates'), xlabel('iterations')
tausave;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% downsample to symbol rate
z=v(0.5*fl+rM:rM+so:end);  % set delay to first symbol-sample and increment by M
figure(9), plot([1:length(z)],z,'.')      % soft decisions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LSequalizer.m find a LS equalizer f for the channel b
n=3;      re = n;                     % length of equalizer - 1
delta=3;                        % use delay <=n*length(b)
p=length(re)-delta;
RE=toeplitz(re(n+1:p),re(n+1:-1:1)); % build matrix R
SE=re(n+1-delta:p-delta)';          % and vector SE
f=inv(RE'*RE)*RE'*SE                  % calculate equalizer f
```

```
Jmin=SE'*SE-SE'*RE*inv(RE'*RE)*RE'*SE     % Jmin for this f and delta
ye=filter(f,1,re);                  % equalizer is a filter
dec=sign(ye);                      % quantize and find errors
err=0.5*sum(abs(dec(delta+1:end)-re(1:end-delta)))
z = ye;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% decision device and symbol matching performance assessment
mprime=quantalph(z,[-3,-1,1,3])';         % quantize to +/-1 and +/-3 alphabet
cluster_variance=(mprime-z)*(mprime-z)'/length(mprime),     % cluster variance

j=1; y=zeros(1,2*length(mprime));
rq=quantalph(mprime,[-3,-1,1,3]);              % quantize back to 4-PAM
for i=1:length(mprime)                         % translate back into 0-1 binary
  if rq(i)==3, y(j:j+1)=[1,1]; end
  if rq(i)==1, y(j:j+1)=[1,0]; end
  if rq(i)==-1, y(j:j+1)=[0,1]; end
  if rq(i)==-3, y(j:j+1)=[0,0]; end
  j=j+2;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% decode
y = blockcode52_decode(y);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% binary to text
ytext=bin2text(y)
numerr=length(find(ytext~=text));

lmp=length(mprime);   % number of recovered symbol estimates
percentage_symbol_errors=100*sum(abs(sign(mprime-mpam(1:lmp))))/lmp % symb err
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2. Function block: blockcode52_encode

```
% blockcode52.m Part 1: Definition of (5,2) binary linear block code
% the generator and parity check matrices
function y=blockcode52_encode(dat);
g=[1 0 1 0 1;
   0 1 0 1 1];
h=[1 0 1 0 0;
   0 1 0 1 0;
   1 1 0 0 1];
% the four code words cw=x*g (mod 2)
x(1,:)=[0 0]; cw(1,:)=mod(x(1,:)*g,2);
x(2,:)=[0 1]; cw(2,:)=mod(x(2,:)*g,2);
x(3,:)=[1 0]; cw(3,:)=mod(x(3,:)*g,2);
x(4,:)=[1 1]; cw(4,:)=mod(x(4,:)*g,2);
% the syndrome table
syn=[0 0 0 0 0;
     0 0 0 0 1;
     0 0 0 1 0;
     0 1 0 0 0;
     0 0 1 0 0;
     1 0 0 0 0;
     1 1 0 0 0;
     1 0 0 1 0];
% blockcode52.m Part 2: encoding and decoding data
p=.0;                               % probability of bit flip
m=length(dat);                           % length of message
% m = 111;
% dat=0.5*(sign(rand(1,m)-0.5)+1);  % m random 0s and 1s

mod_dat = mod(m, 10);
nmod = 0;
if(mod_dat ~= 0)
    nmod = 10 - mod_dat;
end

for(j=1:nmod)
    dat(m+j) = 0;
end

m = length(dat);
k=1;
for i=1:2:m
  c=mod([dat(i) dat(i+1)]*g,2);  % build codeword
  for j=1:length(c)
    if rand<p, c(j)=-c(j)+1; end % flip bits with prob p
  end
  y(k)   = c(1);
  y(k+1) = c(2);
  y(k+2) = c(3);
  y(k+3) = c(4);
  y(k+4) = c(5);
  k = k + 5;
end

% k1=1;
% for i=1:2:m
%   y1(1) = y(k1);
%   y1(2) = y(k1+1);
%   y1(3) = y(k1+2);
%   y1(4) = y(k1+3);
%   y1(5) = y(k1+4);
%   k1 = k1+5;
%   eh=mod(y1*h',2);                    % multiply by parity check h'
%   ehind=eh(1)*4+eh(2)*2+eh(3)+1;  % turn syndrome into index
%   e=syn(ehind,:);                     % error from syndrome table
%   y1=mod(y1-e,2);                     % add e to correct errors
%   for j=1:max(size(x))                % recover message from codewords
%     if y1==cw(j,:), z(i:i+1)=x(j,:); end
%   end
% end


%err=sum(abs(z-dat))               % how many errors occurred
```

3. Function block: blockcode52_decode

```
% blockcode52.m Part 1: Definition of (5,2) binary linear block code% the generator and
parity check matrices

function z=blockcode52_decode(y);
% blockcode52.m Part 2: encoding and decoding data
%m=10000;                          % length of message
%dat=0.5*(sign(rand(1,m)-0.5)+1); % m random 0s and 1s
m = length(y);
mod_dat = mod(m, 10);
nmod = 0;
if(mod_dat ~= 0)
    nmod = 10 - mod_dat;
end

for(j=1:nmod)
    y(m+j) = 0;
end

p=.0;                             % probability of bit flip
m = length(y)/2.5;

%Decoding
k1=1;m
for i=1:2:m
  y1(1) = y(k1);
  y1(2) = y(k1+1);
  y1(3) = y(k1+2);
  y1(4) = y(k1+3);
  y1(5) = y(k1+4);
  k1 = k1+5;
  eh=mod(y1*h',2);                 % multiply by parity check h'
  ehind=eh(1)*4+eh(2)*2+eh(3)+1;   % turn syndrome into index
  e=syn(ehind,:);                  % error from syndrome table
  y1=mod(y1-e,2);                  % add e to correct errors
  for j=1:max(size(x))             % recover message from codewords
    if y1==cw(j,:), z(i:i+1)=x(j,:); end
  end
end
```

4. Block Diagram