0. Overview

In this visualization we use our MATLAB based communication simulation system. The Binary \( \pi \) Phase Shift Keying (BPSK) modulation scheme is implemented with the simulation and the student is asked to maximize the number of message bits without receiving an error. Baseline parameter values are given and the modulation and demodulation technique is given mathematically.

We simulate Binary Phase Shift Keying (BPSK) with phase modulation and mixer based demodulation. The modulated signal is synthesized by using an upsampled random unipolar bit
stream, used to modulate a carrier wave phase value to either 0 or $\pi$. The resulting signal is sent through a lowpass channel and corrupted by Additive White Gaussian Noise (AWGN). Assuming no phase error, the modulated signal is demodulated using a mixer configuration. Both input and output signals are analyzed for bandwidth and noise distribution. The goal is to reproduce the figures and processing presented in this document using MATLAB. The signal length is $N$; the number of data sequences is $N_{\text{seq}}=1$; number of bits is $N_{\text{bit}}$; Standard Deviation is $\text{STD}$ and carrier frequency is $k_c$.

The student is required to implement the mathematics of the modulator and demodulator to MATLAB code. In this visualization, there is no written report due.

The students are also required to formulate a group name, send the instructor their modified createBsize.m, modulator.m and demodulator.m files along with their group member names.

1. **SYSTEM ARCHITECTURE**

The system architecture includes programs that the student modifies and programs that the instructor uses to evaluate their system with. The student has access to the instructor programs so that they can pre-evaluate their systems performance. Given the student programs, the students are required to comply with the File and Data formatting specifications. The student programs have the I/O functionality already built in and specify sections where the students can insert their code.

Referring to Figs. 1-1 and 1-2, the procedure for running the system is as follows:

1. (student) Edit in the group name and number of bits ($N_{\text{bit}}$) into name_createBsize.m and change name to name_createBsize.m. For this visualization we set $N_{\text{seq}}=1$.
2. (student) Run name_createBsize.m. This stores $N_{\text{bit}}$ and groupname.
3. (instructor) Edit in group name into Bgenxx.m and run program. This stores two files, one with the random bit sequence of $N_{\text{bit}}$ bits and the other file contains the active group being processed.
4. (student) Run name_modulator.m. The input is the Message signal is the random bit sequence. Using a kronecker product operation, the binary message sequence is upsampled to length $N$. This in turn is used to modulate the phase of a BPSK system at a carrier frequency, $k_c$, specified by the student. The modulator outputs two files, the first with the modulated signal exactly $N$ real elements long and the other a special trinary signal where 1 indicates time location of a “1” bit and -1 indicates the location of a “0” bit and a zero value everywhere else. This trinary signal is NOT passed to the demodulator but ONLY ACCESSED by the bitcheckxx.m program.
5. (instructor) Run channelxx.m. The channel filters the spectrum of the modulated signal and adds Gaussian white noise to the signal based on the minimum and maximum values in the signal. The instructor determines the channel transfer function and the amount of
noise added. The filtered noisy output is stored in `name_r.mat` as the real vector “r” that is \( N \) elements in length. Channel will also test for errors in the input data format.

6. (student) Run `name_demodulator.m`. The student may use any parameters used in the modulator, including the number of bits, and will process the input “r” signal vector from the Channel. However, there is NO KNOWLEDGE of the random bit sequence allowed. The output of the demodulator, real vector Bs, is stored in `name_Bs.mat`.

7. (instructor) Run `bitcheckxx.m`. Bitcheck uses the Bcheck signal generated in the Modulator to test for 1 or 0 bits. It prints out any errors in format but most importantly how many false alarms and misses occur in the detection process. If there are errors in detection then a figure will be generated revealing the local signal characteristics that led to the error.

![Simulation Flow Chart with associated MATLAB programs and data variables and vectors.](image)

The flowchart in Fig. 1-1 shows the relationship of the MATLAB functions with respect to a standard communications system flowchart. Notice that “`name_modulator.m`” both encodes by upsampling the bit sequence and modulates. The function `Bitcheckxx.m` both decodes by thresholding the \( B \), output vector from the demodulator as well as tests for detection errors based on the \( B_{\text{check}} \) vector.

![Communications Simulation System Architecture.](image)

The simulation architecture is shown in Fig. 1-2. The left group corresponds to the student controlled MATLAB programs, the center group represents the data storage and the right group shows the instructor controlled MATLAB programs.
2. MESSAGE SIGNAL GENERATION

The student renames and uses the “name_createBsize.m” file to establish their groups’ name, number of sequences and number bits to be transmitted through the communications simulator. Once that is run, it stores these three parameters name_createBsize.mat. The next step is to run Bgenxx.m, given in Appendix: Bgen. But before running Bgenxx.m, the groupname variable needs to be manually changed to match which student group name that should be run. This way the instructor can choose which groups’ program set is to be run. If the students are using Bgen in development, then they just need to edit in their name one time.

2.1 Group Name and Number of Bits

Below is the source code for the name_createBsize.m program. For this example, “ee51215V5” has been entered as the group name and ??? need to be replaced by a number. For example, replace ??? with 2*2048 which means that 4096 bits will be transmitted through the communication system.

```matlab
% generate groupname_Bsize.mat
clear all;
% INSERT GROUP NAME AND NUMBER OF BITS
groupname='ee51215V5'
Nbit=???
Nseq=1
% END OF INSERT
filename=sprintf('%s_Bsize.mat',groupname)
save(filename); % stores groupname, Nbit, Nseq in ee51215V5_Bsize.mat
% load filename.mat to retrieve
```

2.2 Bit Sequence Generation

From Appendix: Bgen, the only line of code that would need to be changed is

```matlab
% generate groupname_Bsize.mat
clear all;
% INSERT GROUP NAME AND NUMBER OF BITS
groupname='ee51215V5' % instructor enters this name to select student project
Nbit=???
Nseq=1
% END OF INSERT
filename=sprintf('%s_Bsize.mat',groupname)
save(filename); % stores groupname, Nbit, Nseq in ee51215V5_Bsize.mat
% load filename.mat to retrieve
```

3. BPSK MODULATION

The modulator inputs the bit sequence and knows the full number of samples $N$ to be used in the system. It first determines the number of samples, $N_{sample}$, per bit. Then using the kronecker product to upsample the bit sequence to the message sequence plus any padding necessary to reach length $N$. The mathematical representation for this operation is:
\[ \mathbf{s}_m = \mathbf{b}_m \otimes \mathbf{u}_{N_s}, \]

where \( \mathbf{s}_m \) is the signal vector, \( \mathbf{b}_m \) is the binary message sequence, \( \otimes \) is the Kronecker product and \( \mathbf{u}_{N_s} \) is a unit vector \( N_s = N_{\text{sample}} \) long.

% generate bit matrix based on groupname_Bsize.mat
clear;
Nshowbits=16; % shows the first 16 bits of data
load 'ee51215V5_B.mat';
load 'ee51215V5_Bsize.mat';
% generate a real vector s, N=131072*8 or let N be less for debug process
N=131072*8 % N is set by instructor and cannot be changed
Nbit
% CREATE THE MESSAGE SIGNAL
Nsample=floor(N/Nbit)
% form pulse shape
pulshap=ones(1,Nsample);
% modulate sequence to either +1 and -1 values
b1(1:Nbit)=B(1,1:Nbit);
stem1=kron(b1,pulshap); % form continuous time approximation of message
sm1=zeros(1,N);
if N > (Nsample*Nbit)
    sm1(1:(Nsample*Nbit))=stem1(1:(Nsample*Nbit));
else
    sm1=stem1;
end;
size(sm1) % verify shape
% plot message signal or a section of the message signal
figure(1);
if Nbit<(Nshowbits+1)
    n=1:N;
    plot(n,sm1);
    axis([1,N,-0.1,1.1]);
    xlabel('Message Signal 1');
else
    n=1:(Nsample*Nshowbits);
    plot(n,sm1(1:(Nsample*Nshowbits)));
    axis([1,(Nsample*Nshowbits),-0.1,1.1]);
    xlabel('Sample section of Message Signal 1');
end;
print -djjpeg Modulator_figure1
% FT of message waveform
Sml=abs(fft(sml));
figure(2);
k=0:(N-1);
k=k-N/2;
% remove dc value
Sml(1)=0;
plot(k,fftshift(Sml));
xlabel('DFT spectrum of Message Signal 1');
print -djepg Modulator_figure2

Figure 3-1: Sample section of Message sequence after upsampling.

Figure 3-2: Message signal spectra.
The actual modulator code is placed between the commented sections below. The carrier frequency, $k_c$, is in cycles per $N$ samples. Its value should be as high as possible without suffering too much attenuation from the Channel response. If it is attenuated by the channel then the SNR will decrease because the channel noise is proportional to the max-min value before the signal is filtered by the channel. The cutoff for the channel is $f_c = N/8$. So a good first guess of a $k_c$ value might be $k_c = N/8$ which is half the channel cutoff. Given the carrier frequency, the AM modulation would be

$$s(t) = \cos \left( \frac{2\pi k_c t}{N} + s_m(t) \pi \right)$$  \hspace{1cm} (3.2)

The phase angle is dependent on the binary values such that a “0” is zero phase and a “1” is $\pi$ phase.

% INSERT MODULATION EQUATION:
% INSERT MODULATION EQUATION:
% INSERT MODULATION EQUATION: Inputs sm vector, kc, t and N
% create BPSK modulation signal s
% END OF MODULATION INSERT
% END OF MODULATION INSERT
% END OF MODULATION INSERT
% plot BPSK signal
figure(3);
if Nbit<(Nshowbits+1)
    plot(s);
    axis([1,N,-2.1,2.1]);
    xlabel('BPSK Signal');
else
    Ntemp=Nsample*Nshowbits;
    plot(s(1:Ntemp));
    axis([1,Ntemp,-2.1,2.1]);
    xlabel('Sample section of BPSK Signal');
end;
print -djpeg Modulator_figure3

Figure 3-3: Sample section of BPSK modulated signal.
%% FT of modulated waveform
S=abs(fftshift(fft(s))); figure(4); k=0:(N-1); k=k-N/2; plot(k,S); xlabel('Spectrum of BPSK Signal'); print -djjpeg Modulator_figure4

Figure 3-4: Spectrum of BPSK modulated Signal.

% create the bit check matrix to only be used by the Bcheckxx.m file
% YOU CANNOT PASS THIS INFORMATION TO YOUR DEMODULATOR!!
samplepulse=zeros(1,Nsample); samplepulse(floor(Nsample/2))=1; Bcheck=zeros(Nseq,N);
% modulate first sequence to either +1 and -1 values
bcheck(:,1:Nbit)=2*B(:,1:Nbit)-1; bchecktemp=kron(bcheck,samplepulse); Bcheck=zeros(2,N);
if N > (Nsample*Nbit) Bcheck(:,1:(Nsample*Nbit))=bchecktemp(:,1:(Nsample*Nbit));
else Bcheck=bchecktemp;
end;
figure(5);
if Nbit<(Nshowbits+1) n=1:N; plot(n,sm1,n,Bcheck(1,:)); axis([1,N,-1.1,1.1]); xlabel('Bit Check Signal');
else Ntemp=Nsample*Nshowbits; n=1:Ntemp; plot(n,sm1(1:Ntemp),n,0.9*Bcheck(1,1:Ntemp)); axis([1,Ntemp,-1.1,1.1]); xlabel('Sample Section of Bit Check Signal 1'); end;
save 'ee51215V5_signal' s;
save 'ee51215V5_Bcheck' Bcheck;

Figure 3-5: Sample section of the bit check signal.

There are two signals shown in Fig. 3-5, the unipolar binary signal in blue and the bitcheck signal in green. The bitcheck has been slightly attenuated from its true +/- 1 value to disconnect from the binary signal curve but it really has 3 values +1, 0 and -1. A “0” indicates no bit value in that time location, a “1” indicates there should be a high bit value and -1 indicates a low bit value for that time location. The student is not allowed to pass this information to the demodulator so there is no prior knowledge of the random bit sequence being used in the demodulator. However, bitcheckxx.m uses this information to test the demodulator output.

4. CHANNEL MODEL

See Appendix Channel code. The channel filters and adds noise to the input signal. The mathematical representation of the channel output is

\[ r(t) = s(t) * h(t) + \tilde{\omega}(t) \]  

(4.1)

where \( s(t) \) is the input signal, \( h(t) \) is the channel impulse response and \( \tilde{\omega}(t) \) is a white Gaussian noise process.
Fig. 4-1: Log Magnitude Spectrum of input signal.

Fig. 4-1 is obtained by taking the log of the input spectrum such that

$$\log(\|S(f)\| + 0.1)$$

The channel uses a butterworth frequency response as shown in Fig. 4-2.
Figure 4-3: Input response after filtering.

Figure 4-4: Sample sequence of original modulated input signal

Figure 4-5: Input signal after channel filtering but no noise has been added.
Fig. 4-6 shows the final channel response after filtering and additive noise corruption.

5. BPSK DEMODULATION

The demodulation simulates mixing the received signal with a reference carrier signal and then lowpass filtering to leave just the baseband. The mathematics for these is indicated near the question marked lines in the source code below. One thing to remember in this case, is that since the reference signal is 0 phase, then it will yield a positive “1” value for 0 phase and “-1” for π phase. These values must be decoded to “0” and “1”, respectively.

% generate bit matrix based on groupname_Bsize.mat
clear all;
load 'ee51215V5_Bsize';  % get number of bits and sequences
load 'ee51215V5_r';
[M,N]=size(r)
figure(12)
Nsample=floor(N/Nbit)
if Nbit<41
    plot(r);  
    axis([1,N,-2.1,2.1]);
    xlabel('Received quadrature Signal');
else
    Ntemp=Nsample*40;
    plot(r(1:Ntemp));
    axis([1,Ntemp,-2.1,2.1]);
    xlabel('Sample section of Received quadrature Signal with Noise');
end;
print -djepg Demod_figure12
The first thing the demodulator needs to do is create a reference carrier signal that is in exact synch with the original carrier. Since this simulation does not delay or cause frequency distortion, we can simply generate a cosine waveform such that.

\[ s_{ref}(t) = \cos\left(\frac{2\pi k t}{N}\right) \]  

(5.1)

In turn, \( s_{ref}(t) \) is multiplied by the input signal \( r(t) \) and decoded with a \(-1\) such that

\[ r_i(t) = -r(t)s_{ref}(t) \]  

(5.2)
The carrier frequency $f_c$ is determined by trial and error. Once it is determined then the non-normalized output is simply.

$$r_c(t) = \text{real}(\mathcal{F}^{-1}\{\mathcal{F}\{r_c(t)\}H(f)\})$$

(5.3)

The last step in the demodulation is to scale the signal between 0 and 1 for the bitcheck process. The result is shown in Fig. 5-2.

![Image](image.png)

Figure 5-2: Section of final output of demodulation system.

6. PERFORMANCE VERIFICATION
The detection error is determined by bitcheckxx.m. See Appendix A.3 Bitcheck for source code. The bitcheck program checks for error in formatting and provides the false alarm and miss count. If a miss or false alarm occurs, then the program will create a figure of the local region where the error occurred.

7. REFERENCES


A. APPENDIX: INSTRUCTOR PROGRAMS

There are 3 instructor-controlled programs, Bgenxx.m, Channelxx.m, and bitcheckxx.m. The “xx” will be different for different years or visualizations or projects. These programs are in sections A.1, A.2, and A.3 respectively.

A.1 BGEN

% generate bit matrix based on groupname_Bsize.mat
clear all;
groupname='ee51215V5' % instructor enters this name to select student project
filename=sprintf('%s_Bsize.mat',groupname);
load (filename) % retrieve matrix size
filename
Nbit
Nseq
B=rand(Nseq,Nbit);
B=binarize(B);
size(B)
filename=sprintf('%s_B.mat',groupname);
save(filename);
% save the active groupname
save 'activegroup' groupname;

A.2 CHANNEL

% channel function
clear all;
noiseCoef=0.05;
% input active group
load 'activegroup' groupname;
groupname
% input groupname_signal.mat
filename=sprintf('%s_signal.mat',groupname);
load(filename);
% make sure s is real
signal=real(s);
[M,N]=size(signal)
if N== 1048576
'Incorrect vector length, should be 1048576'
end;
%
% form filter
fc=N/8;
Norder=10;
n=1:N;
K=1; % filter gain
% low pass filter
[f Hchannel]=lp_butterworth_oN_dft15(fc,K,N,Norder);
% filter signal through channel
S=fft(signal);
H=Hchannel;
R=S.*H;
Rn=real(ifft(R));
k=n;k=k-N/2;
figure(6);
plot(k,log(abs(fftshift(S))+.1));
xlabel('Log Magnitude Spectrum of Input Signal');
print -djpeg Channel_figure6
figure(7);
plot(k,abs(fftshift(H)));
axis([0 1000 1 1.1]);
xlabel('Spectrum of Channel');
print -djpeg Channel_figure7
figure(8);
plot(k,fftshift(log(abs(R)+.1)));
xlabel('Log Spectrum of Output Signal, No Noise');
print -djpeg Channel_figure8
% find noise deviation
sigma=noiseCoef*(max(signal)-min(signal))
% add noise
w=sigma*randn(1,N);
r=Rn+w;
% store result in groupname_r.mat
filename=sprintf('%s_r.mat',groupname);
save(filename,'r');
% PLOT spectrum and sample sections of the signal
figure(9);
Nsamplesection=2000;
Nsamples=floor(N/Nsamplesection);
if N<Nsamples
    plot(n,signal);
    axis([1,N,-2.1,2.1]);
    xlabel('Input Signal');
else
    plot(signal(1:Nsamples));
    axis([1,Nsamples,-2.1,2.1]);
    xlabel('Sample section of Input Signal');
end;
print -djpeg Channel_figure9
figure(10);
if N<Nsamples
    plot(n,Rn);
    axis([1,N,-1.1,1.1]);
    xlabel('Output Signal, No Noise');
else
    plot(signal(1:Nsamples));
    axis([1,Nsamples,-2.1,2.1]);
    xlabel('Sample section of Input Signal');
end;
print -djpeg Channel_figure10
plot(sn(1:Nsamples));
axis([1,Nsamples,-1.1,1.1]);
xlabel('Sample section of Output Signal, No Noise');
end;

print -djep Channel_figure10
figure(11);
if N<Nsamples
    plot(n,r);
    axis([1,N,-1.1,1.1]);
xlabel('Output Signal');
else
    plot(r(1:Nsamples));
    axis([1,Nsamples,-1.1,1.1]);
xlabel('Sample section of Output Signal with Noise');
end;

print -djep Channel_figure11

A.3 BITCHECK

% bit check
clear all;
% input active group
load 'activegroup' groupname;

% input original size
filename=sprintf('%s_Bsize.mat',groupname);
load (filename) % retrieve matrix size
filename

Nbitsingle=Nbit
NbitALL=Nbit*Nseq

% input original bit matrix
filename=sprintf('%s_B.mat',groupname);
load (filename);
% load bitcheck
filename=sprintf('%s_Bcheck.mat',groupname);
load (filename);
% load received signal
filename=sprintf('%s_Bs.mat',groupname);
load (filename);
% check for consistancy
[Nseqnow,N]=size(Bs)
NALL=N*Nseq

if Nseqnow~=Nseq
    'ERROR:bitcheck matrices inconsistant'
else
    'OK: bitcheck matrices consistant'
end;
% lexicographically create one sequence
bcheckALL=zeros(1,NALL);
BsALL=zeros(1,NALL);
for q=1:Nseq
    bcheckALL((1+(q-1)*N):(N+(q-1)*N))=Bcheck(q,1:N);
    BsALL((1+(q-1)*N):(N+(q-1)*N))=Bs(q,1:N);
end;
bcheck1=bcheckALL;
Nwindow=128; % select first few samples in window
% plot
k=1:NALL;
f0=15;
f=f0;
figure(f+1);
k1=1:Nwindow;
bs1temp(1:Nwindow)=BsALL(1:Nwindow);
bcheck1temp(1:Nwindow)=bcheckALL(1:Nwindow);
plot(k1,bs1temp,k1,bcheck1temp);
axis([1 Nwindow -0.1 1.1]);
xlabel('first few samples of signal');
% loop through bits
Btest=zeros(1,NbitALL);
miss=0;
false=0;
Nerror=0;
nbreceived=0;
nb=1;
for n=1:NALL
    if bcheck1(1,n) > 0.5 % "1" should be present in check signal
        if BsALL(1,n)>0.5
            Btest(1,nb)=1;
        else
            'ERROR, missing 1'
            'Bcheck'
bcheck1(1,n)
            'Bs'
            BsALL(1,n)
n
        if Nerror<10
            figure(2+f0+Nerror)
            istart=floor(n-(2*NALL/NbitALL));
            istop=floor(n+(2*NALL/NbitALL));
            if istart<1
                istart=1
            end;
            if istop>NALL
                istop=NALL
            end;
clear x;
x=1:(1+istop-istart);
btemp=x;
bchecktemp=x;
btemp(1:(1+istop-istart))=BsALL(1,istart:istop);
bchecktemp(1:(1+istop-istart))=bcheckALL(1,istart:istop);
plot(x,btemp-.5,x,bchecktemp);
%clear x,btemp,bchecktemp;
        end; % if Nerror
        miss=miss+1;
        Nerror=Nerror+1;
    end; % if BsALL
    nb=nb+1;
nbreceived=nbreceived+1;
end; % if bcheck1(1,n)
if bcheckALL(n) < -0.5 % "-1" should be present in check signal
if BsALL(n) < 0.5 % "0" is present demodulated/binarized signal
    Btest(nb)=0;
else
    'ERROR, missing 0'
    'Bcheck'
    bcheckALL(n)
    'Bs'
    BsALL (n)
    n
    nb
    if Nerror<10
        figure(2+f0+Nerror);
        istart=floor(n-(2*NALL/NbitALL));
        istop=floor(n+(2*NALL/NbitALL));
        if istart<1
            istart=1
        end;
        if istop>NALL
            istop=NALL
        end;
        clear x;
        x=1:(1+istop-istart);
        btemp=x;
        bchecktemp=btemp;
        btemp(1:(1+istop-istart))=BsALL(istart:istop);
        bchecktemp(1:(1+istop-istart))=bcheckALL(istart:istop);
        istart
        istop
        size(x)
        size(btemp)
        size(bchecktemp)
        plot(x,btemp-.5,x,bchecktemp);
    end; % if Nerror
    false=false+1;
    Nerror=Nerror+1;
end; % if BsALL(n)
nb=nb+1;
nbreceived=nbreceived+1;
end; % if bcheckALL(n)
end; % for n=1:NALL
nbsent=NbitALL
nbreceived
miss
false
Nerror
if nbsent~=nbreceived
    'Error between sent and recieved'
    'Number of ones and zeros sent'
    Nones=sum(sum(B))
    Nzeros=nbsent-Nones
end;
% STATISTICAL ANALYSIS OF BINARY SIGNAL
Bit1index=find(bcheckALL>0.5);
Bit0index=find(bcheckALL<0.5);
[MBit1 NBit1]=size(Bit1index);
[MBit0 NBit0]=size(Bit0index);
Bits1=zeros(1,NBit1);
Bits0=zeros(1,NBit0);
Bits1=BsALL(Bit1index);
Bits0=BsALL(Bit0index);
mu1=mean(Bits1,2)
mu0=mean(Bits0,2)
var1=var(Bits1);
var0=var(Bits0);
STD1=sqrt(var1)
STD0=sqrt(var0)
SQRTofSNR1=mu1/STD1
SQRTofSNR0=mu0/STD0
Discriminate=abs(mu1-mu0)/sqrt(var0+var1)

% HISTOGRAM OF BITS
W=100;
w=1:W;
maxhist=1.5;
minhist=-.5;
% find coefficients to map from the received values to the histogram index
% W=a*maxhist+b, 1=a*minhist+b, a=(W-1)/(maxhist-minhist) b=1-a*minhist
acoef=(W-1)/(maxhist-minhist); bcoef=1-acoef*minhist;
h1=zeros(1,W);
for n=1:NBit1
    m=floor(acoef*Bits1(n)+bcoef);
    if m>0
        if m<(W+1)
            h1(m)=h1(m)+1;
        end;
    end; % if m>0
end; % for n
h1=h1/NBit1;
h0=zeros(1,W);
for n=1:NBit0
    m=floor(acoef*Bits0(n)+bcoef);
    if m>0
        if m<(W+1)
            h0(m)=h0(m)+1;
        end;
    end; % if m>0
end; % for n
h0=h0/NBit0;
maxhisto=max(h0);
if maxhisto<max(h1)
    maxhisto=max(h1);
end;
figure(15);
v=(w-bcoef)/acoef; % make horizontal axis be minhist to maxhist units
plot(v,h0,v,h1);
xlabel('Received Bit value');
ylabel('pdf Estimate');
axis([minhist maxhist 0 maxhisto]);
legend('f(bit0)', 'f(bit1)');
print -djpeg Bitcheck_figure15