MIDDLE EAST TECHNICAL UNIVERSITY

EE430 | Digital Signal Processing

Term Project 2014

"DSP on computer in MATLAB"

Hasan Semih Genç

1949866 – Section 3

21.1.2015

1. Project Description

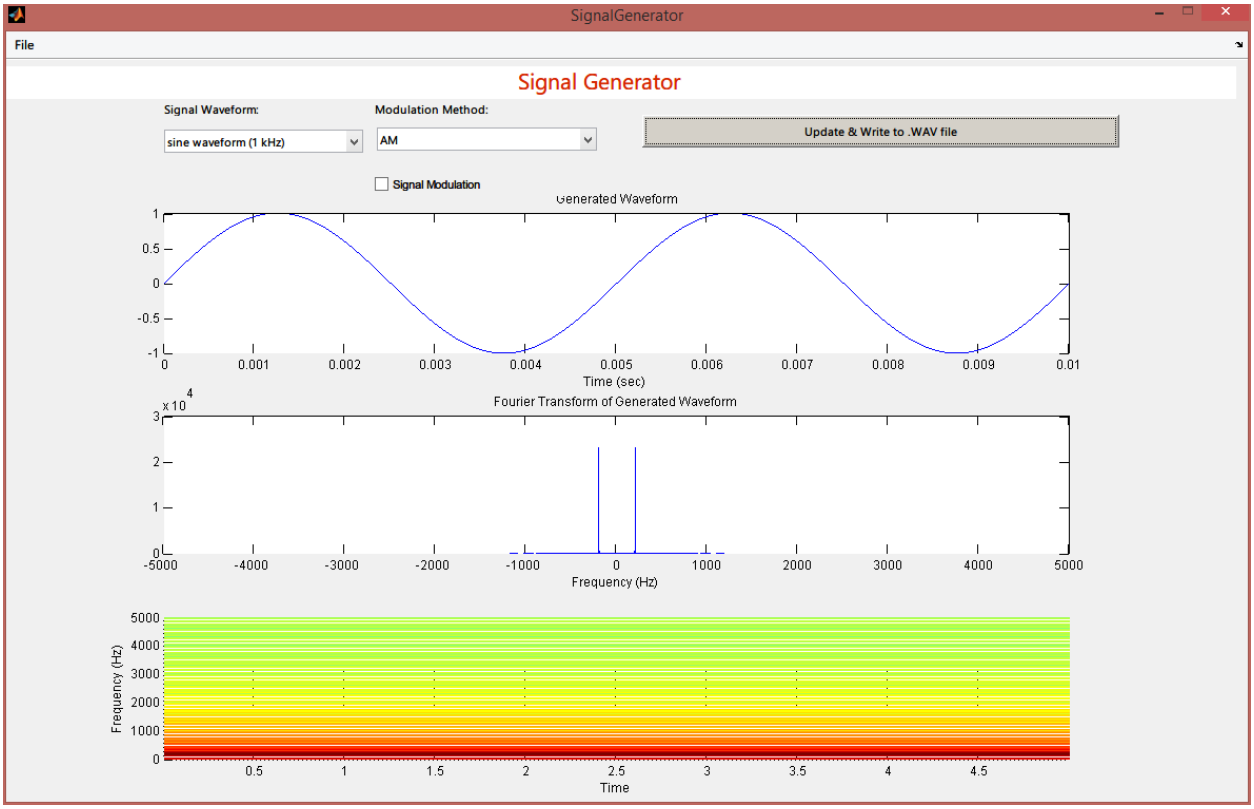
Designing a signal generator module and signal analysis module in MATLAB. Signal generator module is used to generate a desired signal in MATLAB on a computer. This generated signal is then played as a wav file at a selected frequency by using the D/A converter in the sound card on the same computer. At the same time, the Signal Analysis module works in another MATLAB window and it receives the input signal from the same sound card by sampling the analog signal at a selected sampling frequency. Signal analysis module has several subroutines for sampling, filtering, processing, etc. The desired implementation does not require any hardware, except a computer and a special audio cable which you can build it yourself by following the procedure described below (or purchase it from the market).

2. Implemented Modules

The project consists of two MATLAB modules, namely, “Signal Generation” and “Signal Analysis”.

Signal Generation Module

In the first module, various signal waveforms will be produced and modulated by using different modulation techniques. The generated signal then is written to a “.wav” file. This output file can be played at any time for analysis. The following figure shows the GUI (Graphical User Interface) of “Signal Generation” module. The signal waveform can be selected. The modulation technique is given as an option. The resultant waveform can be seen in the time domain plot. Additionally, the Fourier transform and spectrogram of the signal is plotted on the GUI. Lastly, the button placed on the GUI helps us to write the signal to a “.wav” file.



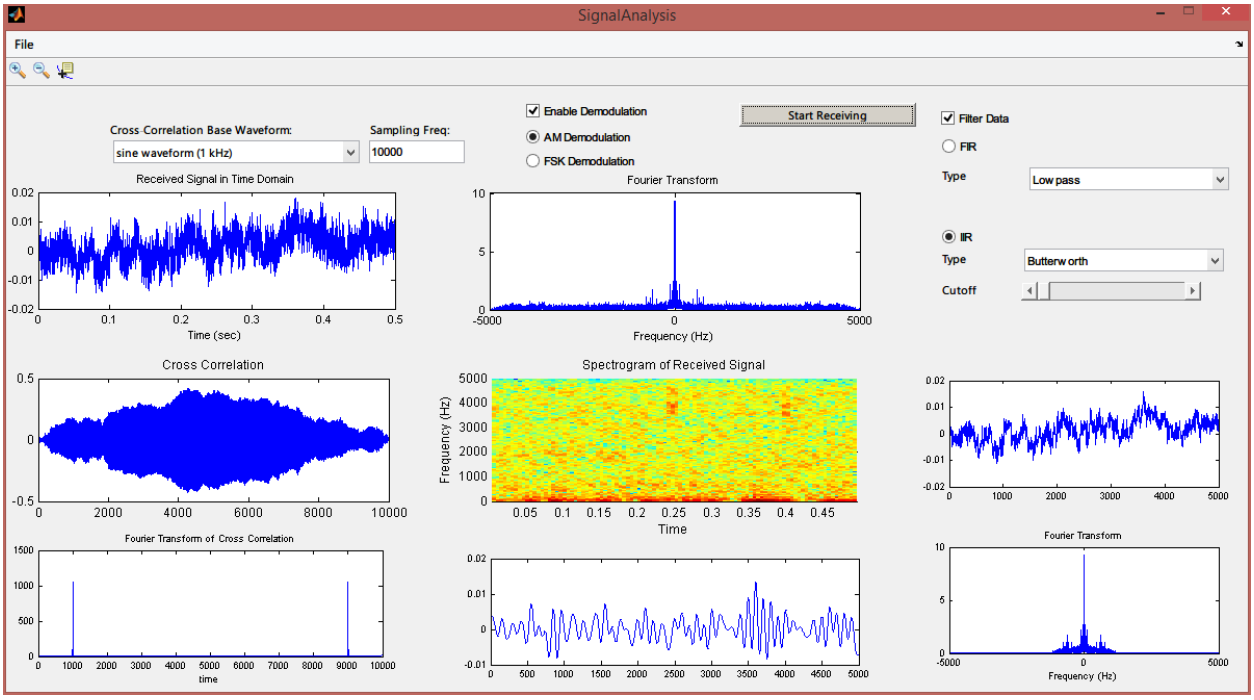
Signal Analysis Module

In this module, the signal is captured through input of the sound card. The signal is collected for a specified duration and then the examination takes place. After examination, the signal is again captured and this process moves so on. The following plots will be drawn on the GUI of "Signal Analysis Module":

- Received signal in time domain
- Fourier transform of the received signal
- Spectrogram of received signal
- Cross-correlation of received signal with a predefined and generated signal
- Fourier transform of cross correlation
- Received signal after filtering in time domain
- Fourier transform of filtered signal

- Demodulated signal in time domain

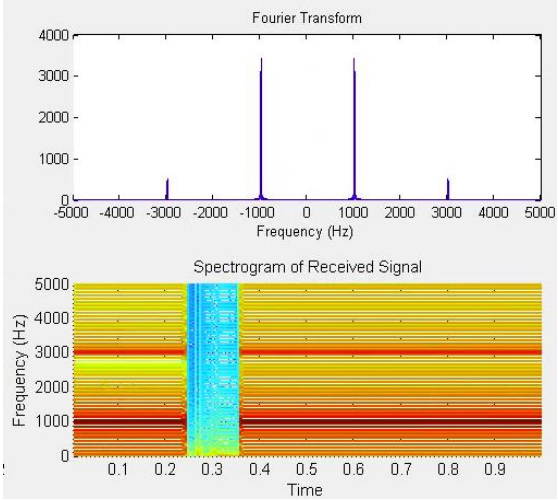
In total, there will be 8 different plots for examination on the module. The demodulation and filtering is optional. The button on the GUI makes enable turning capture on/off. The following figure is a screenshot of the GUI module.

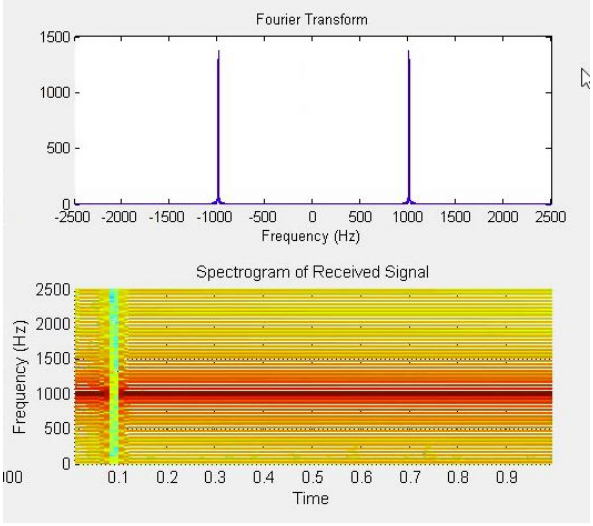


Sampling frequency can be selected with help of a button. Filter types can be selected if filtering data is enabled.

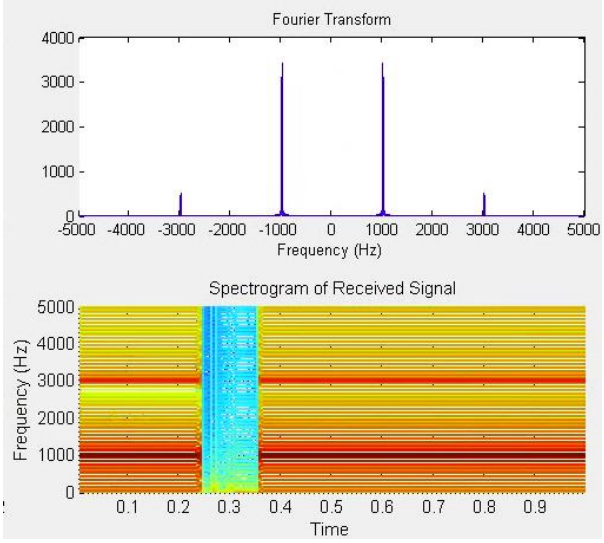
3. Results and Discussions

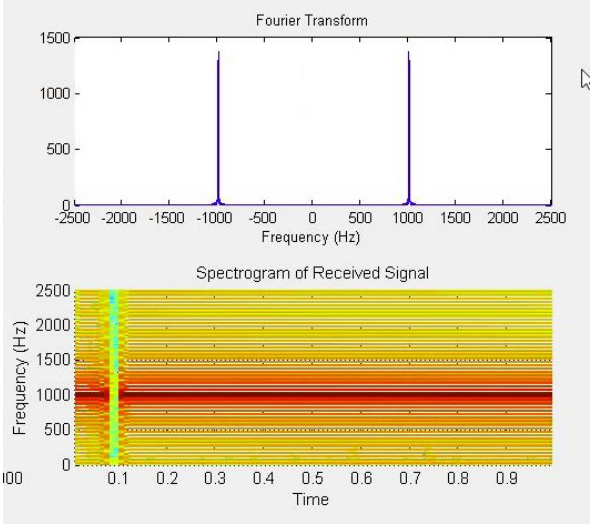
For the chirp waveform, when generating, the sampling frequency is 10 kHz. The largest frequency of the waveform is 4 kHz. So the sampling frequency is decreased to 5 kHz to see some aliasing. For the sinusoid signal, the sampling frequency could not be decreased because MATLAB "analoginput" function does not support low sampling frequencies.





The figure on the left is taken from signal analysis module when the same computer is used for generation and analysis of the signal. The figure on the right is taken from signal analysis module when different computers are used for generation and analysis of the signal.

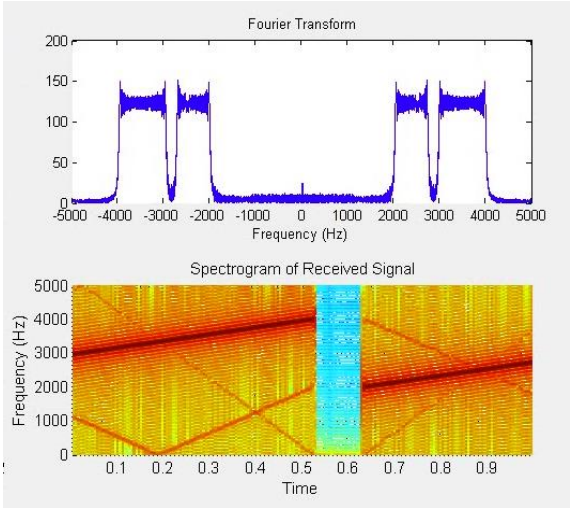


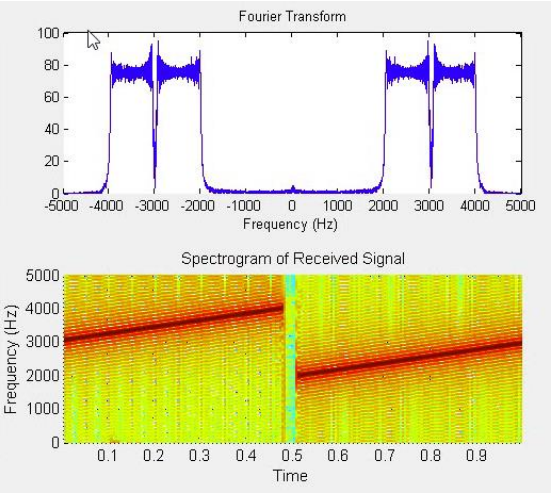


As you can see, when the same computer is used, analysis module receives frequency parts other than 1 kHz. The source of this additional part is unknown. Also, when the plots are examined, there are more noisy parts on the plot from the same computer case.

Additionally, there is more delay in the same computer case. The blue regions on the spectrograms indicate the duration in which there is no signal. On the same computer, we have more delay (wider blue region).

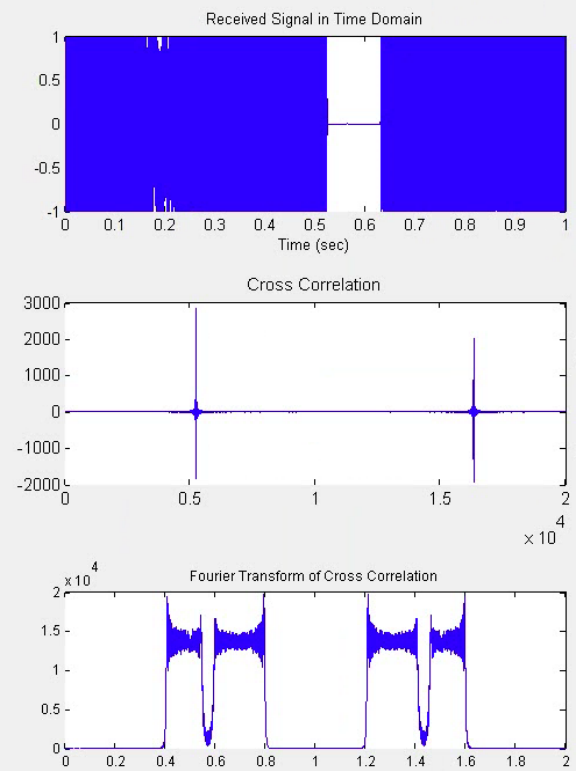
The same plots for the chirp waveform case is shown below.

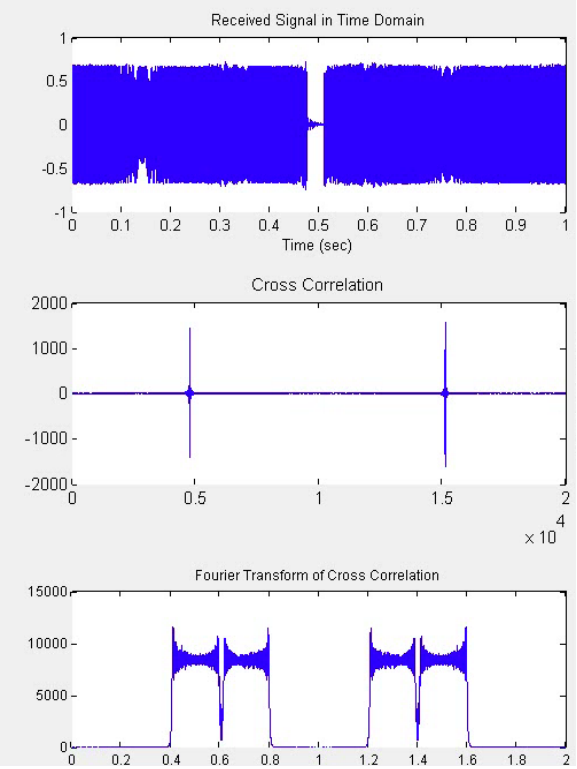




For the chirp waveform case, when we correlate the received signal with the original ideal signal we obtain the following cross correlation plots. The vertical lines on this plot indicate the starting of the signal. At those points, the received signal and the ideal signal with the same parameters overlap exactly and give a large output.

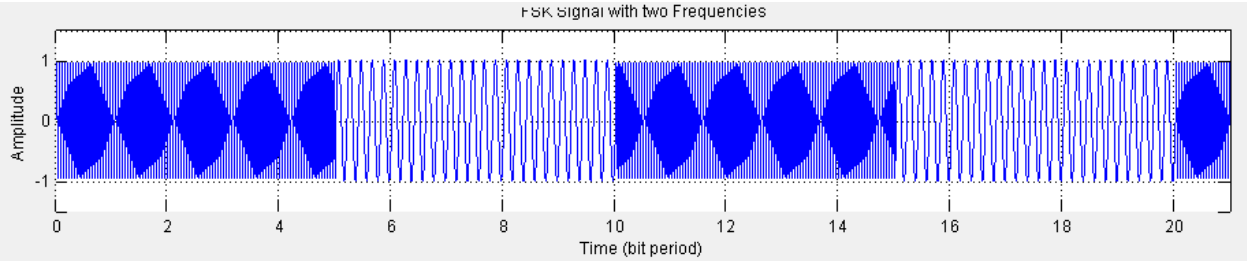
For the sinusoidal waveform case, the cross correlation is a little different. There is more than one point where the ideal signal and the received signal overlap exactly or partially. This results in a plot that we have rectangular polygon shapes.





Modulation

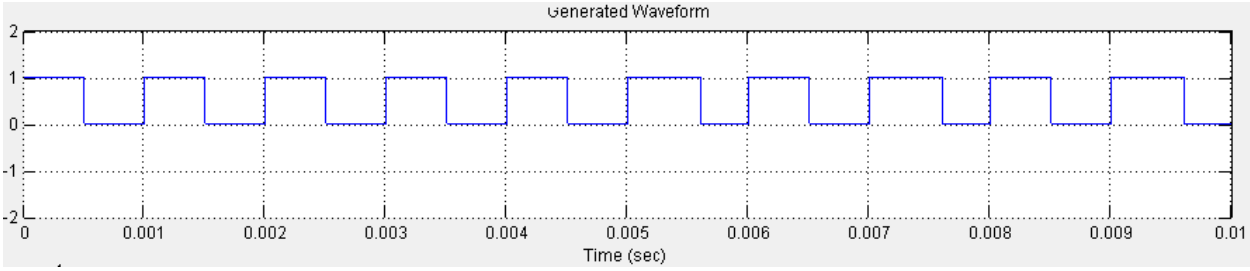
After generating signal waveform, there will be three modulation options can be applied to the signal. These are AM, AM (SSB) and FSK (Binary) modulation techniques. For AM modulation, there exists a MATLAB library function. For FSK (Frequency Shift Keying), a small program is written. Since FSK modulation requires integer valued sequence, only square waveform can be modulated by FSK. A zero "0" will be converted to a predefined duration of a sinusoid whose frequency 500 Hz and for one "1" the sinusoid frequency is 2000 Hz. The following figure shows AM modulated signal of sequence [1 0 1 0 1....] .



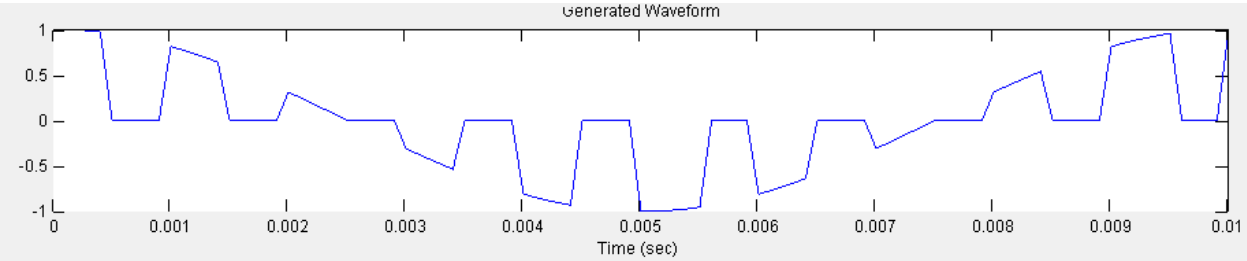
AM vs. FSK

Both of AM and FSK modulation are used in modules. For different waveforms, their performance and applicability differ. For example, FSK can only be used with integer valued digital signals. However, AM can modulate both digital and analog signals. Instead of FSK, similar FM modulation can be used for analog signals but it is out of scope of this project.

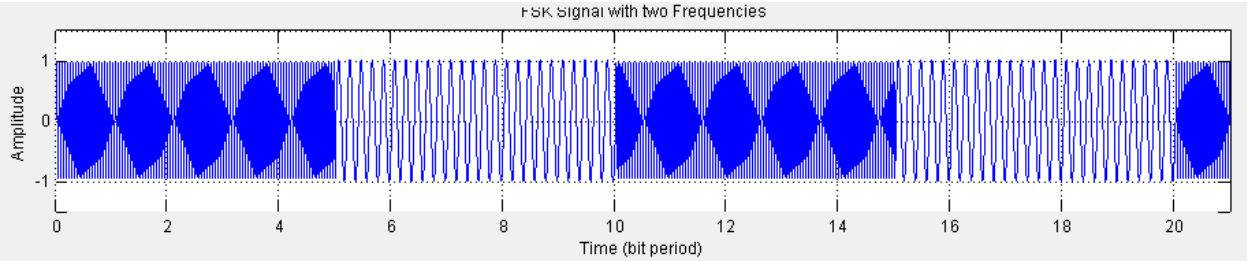
Let's compare FSK and AM modulation techniques. The following square waveform will be modulated with these two modulators.



After AM modulation

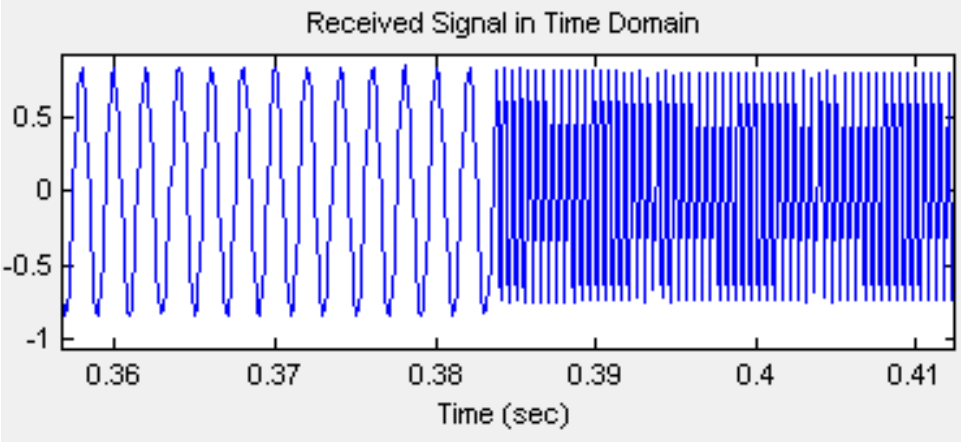


After FSK modulation (0.5 kHz and 2 kHz)

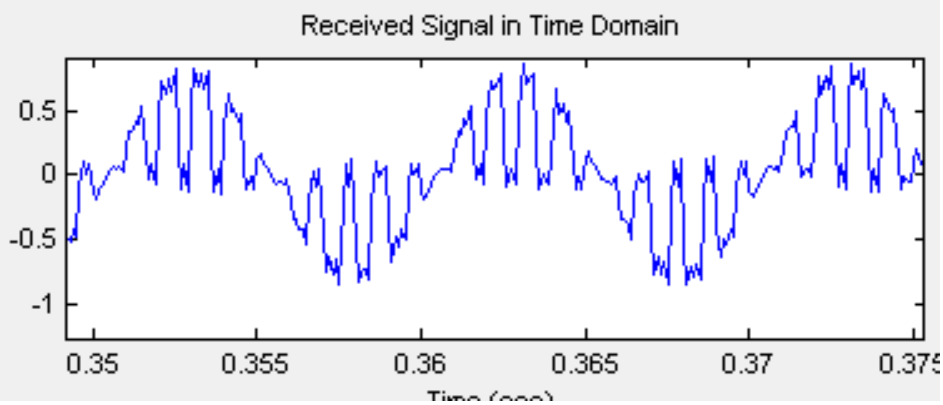


These two signals are sent to signal analysis module. The following figures show what's captured in the time domain.

FSK modulated signal (zoomed)

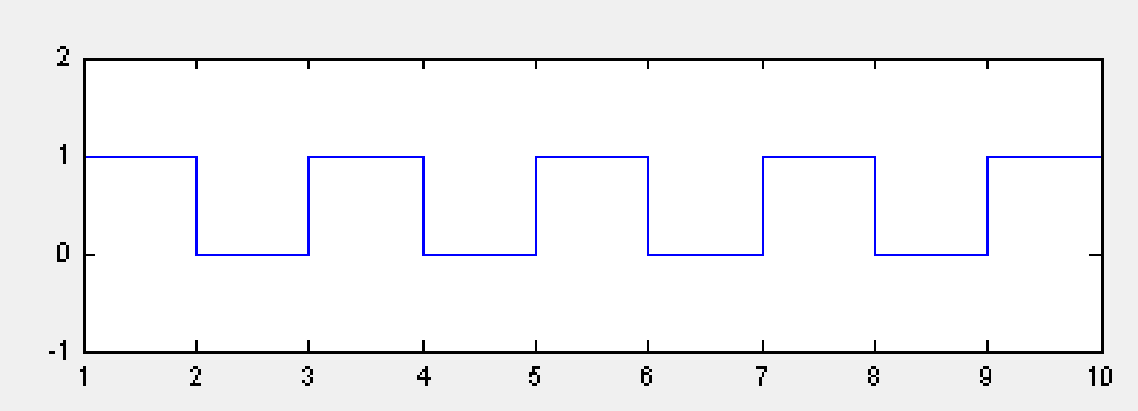


AM modulated signal (zoomed)

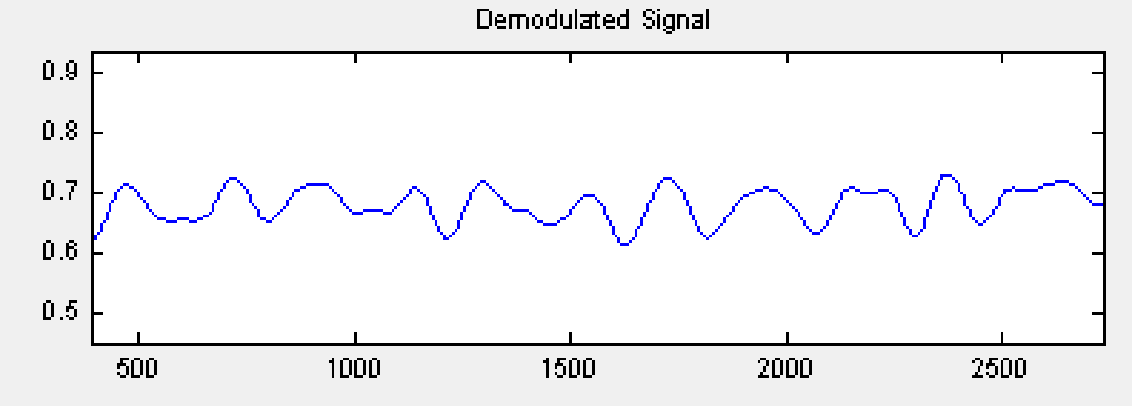


After demodulation of these received signals,

Square wave obtained from FSK demodulation:



Square wave obtained from AM demodulation



I wasn't expecting this result obtained from AM demodulation. By changing the parameters of AM modulation and demodulation we can expect better results. However, the signal will still carry a lot of noise and gibbs effect will be observed on the received modulated signal due to the loss of some parts of signal. With these results, we can say FSK modulation is better at doing this job.

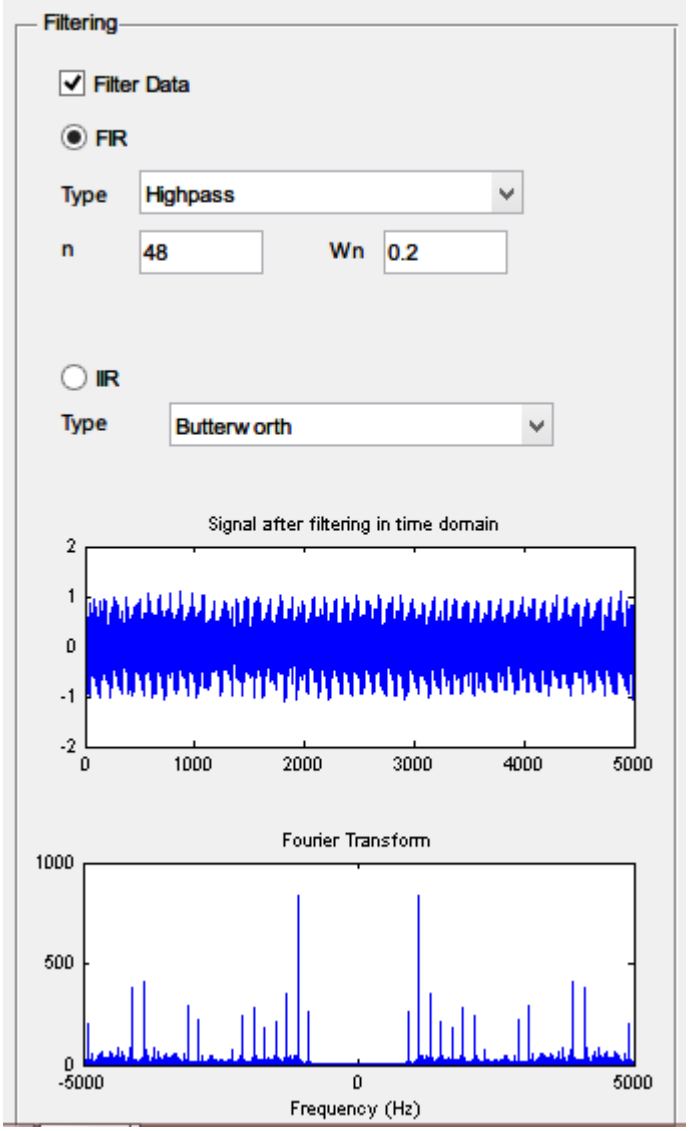
Filters

In the signal analysis module, two types of filter can be used: IIR and FIR.

For FIR type filters, there are options to choose filter order (n) and there exists a switch for highpass/bandstop selection. Also, parameters of filters can be changed.

For IIR type filters, Butterworth, Chebychev and Elliptic filters are implemented with fixed parameters.

The following figure shows filtering part of GUI.



4. MATLAB Codes

Signal Generation Module

```
function varargout = SignalGenerator(varargin)
% SIGNALGENERATOR MATLAB code for SignalGenerator.fig
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @SignalGenerator_OpeningFcn, ...
                   'gui_OutputFcn',  @SignalGenerator_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before SignalGenerator is made visible.
function SignalGenerator_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for SignalGenerator
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using SignalGenerator.
if strcmp(get(hObject,'Visible'),'off')
 % SignalGenerator_OutputFcn(hObject, eventdata, handles);
end

% UIWAIT makes SignalGenerator wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = SignalGenerator_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
%% Parameters

% sampling: 10 kHz
global fs;
fs= 10000;

% For a second
global t;
t = 0:1/fs:1;

% Time vector for a small portion of the signal (to be able to see the signal plot)
global ts;
ts=0:1/fs:100/fs;

% Duty
global duty;
duty=50;

%% Signal Generation
global xcurr;
global xcurrs;
% sine function
x1 = sin(2*pi*200*t);
x1s= sin(2*pi*200*ts);

% chirp waveform
x2 = chirp(t,2000,1,4000,'linear');
x2s= chirp(ts,2000,1,4000,'linear');

% square waveform
x3 = (square(2*pi*1000*t, duty)+1)*0.5;
x3s= (square(2*pi*1000*ts,duty)+1)*0.5;

%% .WAV files
 % file path
global wavefile
wavefile ='signal.wav';

%% Plots
 global m;
m = length(x1);         % Window length
 global n;
n = pow2(nextpow2(m));  % Transform length
 global f;
f = (-n/2:n/2-1)*(fs/n);
```

```
popup_sel_index = get(handles.popupmenu1, 'Value');
switch popup_sel_index
    case 1
        xcurr=x1;
        xcurrs=x1s;
        axes(handles.axes1);
        cla;
        plot(ts,x1s);
        title('Generated Waveform');
        xlabel('Time (sec)');

        axes(handles.axes2);
        cla;
        fft_x1=abs(fft(x1,n));
        fftshft=fftshift(fft_x1);
        plot(f,abs(fftshft));
        title('Fourier Transform of Generated Waveform');
        xlabel('Frequency (Hz)');

        axes(handles.axes3);
        spectrogram(x1, rectwin(100), 50, 1000, fs,'yaxis');

        % write to file
        wavwrite(x1,fs,32,wavefile);
    case 2
        xcurr=x2;
        xcurrs=x2s;
        axes(handles.axes1);
        cla;
        plot(ts,x2s);
        title('Generated Waveform');
        xlabel('Time (sec)');

        axes(handles.axes2);
        fft_x2=abs(fft(x2,n));
        fftshft2=fftshift(fft_x2);
        plot(f,abs(fftshft2));
        title('Fourier Transform of Generated Waveform');
        xlabel('Frequency (Hz)');

        axes(handles.axes3);
        spectrogram(x2, rectwin(100), 50, 1000, fs,'yaxis');
        % write to file
        wavwrite(x2,fs,32,wavefile);
    case 3
        xcurr=x3;
        xcurrs=x3s;
        axes(handles.axes1);
        cla;
        stairs(ts,x3s);
        title('Generated Waveform');
        xlabel('Time (sec)');
        ylim([-2 2]);
```

```
        grid  on;

        axes(handles.axes2);
        fft_x3=abs(fft(x3,n));
        fftshft3=fftshift(fft_x3);
        plot(f,abs(fftshft3));
        title('Fourier Transform of Generated Waveform');
        xlabel('Frequency (Hz)');

        axes(handles.axes3);
        spectrogram(x3, rectwin(100), 50, 1000, fs,'yaxis');
        % write to file
        wavwrite(x3,fs,32,wavefile);
end

set(handles.IsModEnabled,'Value',0);

% --------------------------------------------------------------------
function OpenMenuItem_Callback(hObject, eventdata, handles)
file = uigetfile('*.fig');
if ~isequal(file, 0)
    open(file);
end

% --------------------------------------------------------------------
function PrintMenuItem_Callback(hObject, eventdata, handles)
printdlg(handles.figure1)

% --------------------------------------------------------------------
function CloseMenuItem_Callback(hObject, eventdata, handles)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
                     ['Close ' get(handles.figure1,'Name') '...'],...
                     'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

delete(handles.figure1)



% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)




% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end

set(hObject, 'String', {'sine waveform (1 kHz)', 'chirp waveform (2-4 kHz)','square
waveform (1 kHz)'});



% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



% --- Executes on selection change in modmenu.
function modmenu_Callback(hObject, eventdata, handles)



% --- Executes during object creation, after setting all properties.
function modmenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to modmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

set(hObject, 'String', {'AM', 'AM SSB','FSK'});



% --- Executes on button press in IsModEnabled.
function IsModEnabled_Callback(hObject, eventdata, handles)

%% Signal Modulation

global xcurr;
global xcurrs;
global fs;

global ts;
global f;
```

```
global n;
global wavefile;
%x3 = ssbmod(x3,Fc,fs);
Fc=100;
mod_sel_index = get(handles.modmenu, 'Value');

if get(hObject,'Value')==1

switch mod_sel_index
    case 1
        xcurr= ammod(xcurr,Fc,fs);
        xcurrs=ammod(xcurrs,Fc,fs);
        axes(handles.axes1);
        cla;
        plot(ts,xcurrs);
        title('Generated Waveform');
        xlabel('Time (sec)');

        axes(handles.axes2);
        cla;
        fft_xcurr=abs(fft(xcurr,n));
        fftshft=fftshift(fft_xcurr);
        plot(f,abs(fftshft));
        title('Fourier Transform of Generated Waveform');
        xlabel('Frequency (Hz)');

        axes(handles.axes3);
        spectrogram(xcurr, rectwin(100), 50, 1000, fs,'yaxis');

        % write to file
        wavwrite(xcurr,fs,32,wavefile);
    case 2
        xcurr=ssbmod(xcurr,Fc,fs);
        xcurrs=ssbmod(xcurrs,Fc,fs);
        axes(handles.axes1);
        cla;
        plot(ts,xcurrs);
        title('Generated Waveform');
        xlabel('Time (sec)');

        axes(handles.axes2);
        cla;
        fft_xcurr=abs(fft(xcurr,n));
        fftshft=fftshift(fft_xcurr);
        plot(f,abs(fftshft));
        title('Fourier Transform of Generated Waveform');
        xlabel('Frequency (Hz)');

        axes(handles.axes3);
        spectrogram(xcurr, rectwin(100), 50, 1000, fs,'yaxis');

        % write to file
```

```
    wavwrite(xcurr,fs,32,wavefile);

case 3

    f1 = 5;
    f2 = 20;

    fss=100;
    % Time for one bit
    t1 = 0: 1/fss : 1;

    % This time variable is just for plot
    time = [];

    FSK_signal = [];
    FSK_signals = [];

    for ind = 1: 1: length(xcurrs)

        % The FSK Signal
        FSK_signals = [FSK_signals (xcurrs(ind)==0)*sin(2*pi*f1*t1)+...
            (xcurrs(ind)==1)*sin(2*pi*f2*t1)];

        time = [time t1];
        t1 =  t1 + 1;

    end
    axes(handles.axes1);
    plot(time,FSK_signals);
    xlabel('Time (bit period)');
    ylabel('Amplitude');
    title('FSK Signal with two Frequencies');
    axis([0 time(end) -1.5 1.5]);
    grid  on;

    t1 = 0: 1/fss : 1;
    time = [];
    for ind = 1: 1: length(xcurr)

        % The FSK Signal
        FSK_signal = [FSK_signal (xcurr(ind)==0)*sin(2*pi*f1*t1)+...
            (xcurr(ind)==1)*sin(2*pi*f2*t1)];

        time = [time t1];
        t1 =  t1 + 1;

    end

    axes(handles.axes2);
    cla;
    fft_xcurr=abs(fft(FSK_signal,n));
```

```
        fftshft=fftshift(fft_xcurr);
        plot(f,abs(fftshft));
        xlim([0 1000]);
        title('Fourier Transform of Generated Waveform');
        xlabel('Frequency (Hz)');

        axes(handles.axes3);
        %spectrogram(FSK_signal, rectwin(100), 50, 1000, fs,'yaxis');

        % write to file
        wavwrite(FSK_signal,fs,32,wavefile);
end
else
    pushbutton1_Callback(handles.popupmenu1, eventdata, handles);
end
```

Signal Analysis Module

```
function varargout = SignalAnalysis(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @SignalAnalysis_OpeningFcn, ...
                   'gui_OutputFcn',  @SignalAnalysis_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before SignalAnalysis is made visible.
function SignalAnalysis_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to SignalAnalysis (see VARARGIN)


% Choose default command line output for SignalAnalysis
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using SignalAnalysis.
if strcmp(get(hObject,'Visible'),'off')
 % SignalAnalysis_OutputFcn(hObject, eventdata, handles);
end

%%

% UIWAIT makes SignalAnalysis wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% --- Outputs from this function are returned to the command line.
function varargout = SignalAnalysis_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
if(strcmp(get(handles.pushbutton1, 'String'),'Start Receiving'))
set(handles.pushbutton1, 'String','Stop Receiving');
else
set(handles.pushbutton1, 'String','Start Receiving');
end



%% Parameters
Fs = str2num(get(handles.samplingFreq,'String'));
duration = 0.5;

t=0:1/Fs:(duration)-1/Fs;
tcorr=0:1:(2*Fs.*duration)-2;
%tcorr=tcorr./2;
t1 = 0:1/Fs:1-1/Fs;
x1 = sin(2*pi*1000*t);

while (strcmp(get(handles.pushbutton1, 'String'),'Stop Receiving'))
AI = analoginput('winsound');
addchannel(AI, 1);

set (AI, 'SampleRate', Fs)
set(AI, 'SamplesPerTrigger', duration*Fs);
start(AI);
data = getdata(AI);
delete(AI)

m = length(data);       % Window length
```

```
n = pow2(nextpow2(m));  % Transform length
f = (-n/2:n/2-1)*(Fs/n);



%% Time Domain
axes(handles.axes1);
plot(t,data);
title('Received Signal in Time Domain');
xlabel('Time (sec)');

%% Fourier Transform
fft_x1=fft(data,n);
fft_x1=fftshift(fft_x1);
w=linspace(-Fs/2,Fs/2,length(data));
axes(handles.axes2);
plot(f,abs(fft_x1));
title('Fourier Transform');
xlabel('Frequency (Hz)');




%% Spectrogram
axes(handles.axes4);
spectrogram(data, rectwin(100), 50, 1000, Fs,'yaxis');
title('Spectrogram of Received Signal');



popup_sel_index = get(handles.popupmenu1, 'Value');
switch popup_sel_index
    case 1
        x1 = sin(2*pi*1000*t);
    case 2
        x1 = chirp(t1,2000,1,4000);
end

%% Correlation
axes(handles.axes3);
xcor=xcorr(data,x1);
plot(xcor);
title('Cross Correlation');

%% Fourier of Cross Correlation
axes(handles.axes5);
xcorfft=fft(xcor);
plot(tcorr,abs(xcorfft));
title('Fourier Transform of Cross Correlation');
xlabel('time');

%% Demodulation
IsDemod=get(handles.IsDemEnabled,'Value');
IsAMdem=get(handles.am_demod,'Value');
```

```
IsFSKdem=get(handles.fsk_demod,'Value');
Fc=100;

if IsDemod
    axes(handles.axes6);

    if(IsAMdem)
        plot(amdemod(data,Fc,Fs));
        title('Demodulated Signal');
    elseif(IsFSKdem)
        %FSK Detection
        F=[];
        t1 = 1/Fc: 1/Fc : 5;
        ss=length(t1);
        sf1=sin(2*pi*20*t1);
        for i=ss:ss:length(data)
            if sum(sf1'.*data(i-(ss-1):i))>0.5
                F=[F 1];
            else
                F=[F 0];
            end

        end
        stairs(F);
        ylim([-1 2]);
        title('Demodulated Signal');
    end
end

%% Filtering

FilteringOption = get(handles.IIRFilter, 'Value');

IsFiltering=get(handles.IsFiltering,'Value');
IsFIR=get(handles.IsFIR,'Value');
IsIIR=get(handles.IsIIR,'Value');
nn=str2num(get(handles.n,'String'));

FIRoption=get(handles.popupmenu3, 'Value');

if IsFiltering
    axes(handles.axes7);

    if(IsFIR)
        switch FIRoption
            case 1
                Wn=str2num( get(handles.Wn,'String'));
                b = fir1(nn,Wn,'high');
            case 2
                C=strsplit(get(handles.Wn,'String'),' ');
                Wn(1)=str2num(C{1});
                Wn(2)=str2num(C{2});
```

```
                b = fir1(nn,Wn,'stop');
        end
        dataOut = filter(b,1,data);
        plot(dataOut);
         title('Signal after filtering in time domain ');
        fft_x1=fft(dataOut,n);
        fft_x1=fftshift(fft_x1);
        w=linspace(-Fs/2,Fs/2,length(dataOut));
        axes(handles.axes8);
        plot(f, abs(fft_x1));
        title('Fourier Transform');
        xlabel('Frequency (Hz)');

    elseif (IsIIR)
        switch FilteringOption
            case 1  % Butterworth
                [b,a] = butter(6,0.2,'low');
                %freqz(b,a);
                dataOut = filter(b,a,data);
                plot(dataOut);
            case 2 % Chebychev
                [b,a] = cheby1(6,10,0.6,'high');
                dataOut = filter(b,a,data);
                plot(dataOut);
            case 3 % Elliptic
                [b,a] = ellip(6,5,40,0.6);
                dataOut = filter(b,a,data);
                plot(dataOut);
        end
         title('Signal after filtering in time domain ');
        fft_x1=fft(dataOut,n);
        fft_x1=fftshift(fft_x1);
        w=linspace(-Fs/2,Fs/2,length(dataOut));
        axes(handles.axes8);
        plot(f,abs(fft_x1));
        title('Fourier Transform');
        xlabel('Frequency (Hz)');

    end

end
end
guidata(hObject,handles);

function OpenMenuItem_Callback(hObject, eventdata, handles)
file = uigetfile('*.fig');
if ~isequal(file, 0)
    open(file);
end

% --------------------------------------------------------------------
function PrintMenuItem_Callback(hObject, eventdata, handles)
printdlg(handles.figure1)
```

```
% --------------------------------------------------------------------
function CloseMenuItem_Callback(hObject, eventdata, handles)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
                     ['Close ' get(handles.figure1,'Name') '...'],...
                     'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

delete(handles.figure1)




% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end

set(hObject, 'String', {'sine waveform (1 kHz)', 'chirp waveform (2-4 kHz)'});



% --- Executes during object creation, after setting all properties.
function samplingFreq_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function pushbutton1_CreateFcn(hObject, eventdata, handles)
```

```
% --- Executes on selection change in IIRFilter.
function IIRFilter_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function IIRFilter_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


set(hObject, 'String', {'Butterworth', 'Chebychev', 'Elliptic'});

% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu3 contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from popupmenu3


% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


set(hObject, 'String', {'Highpass', 'Bandstop'});


% --- Executes during object creation, after setting all properties.
function IsDemEnabled_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in IsDemEnabled.
function IsDemEnabled_Callback(hObject, eventdata, handles)
% --- Executes on slider movement.
function slider3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
function n_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function n_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Wn_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function Wn_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```