wx_custom_receiver_sample

April 28, 2022

1 Introduction to SDR Signal Processing

In this special Lab 6 problem you finalize the design of digital lowpass filters used the receiver block diagram used to the message from the COS National Weather Service station. The software is written in Python making extensive use of the package `scikit-dsp-comm`. The signal flow Python code is provided.

[1]:
```
from numpy import *
from matplotlib.pyplot import *
%matplotlib inline
import sk_dsp_comm.sigsys as ss
import sk_dsp_comm.sdr_helper as sdrh
import sk_dsp_comm.fir_design_helper as fir_d
import sk_dsp_comm.digitalcom as dc
import scipy.signal as signal
from IPython.display import Audio, display
from IPython.display import Image, SVG

%config InlineBackend.figure_formats=['svg'] # SVG inline viewing
#%config InlineBackend.figure_formats=['pdf'] # render pdf figs for LaTeX
```

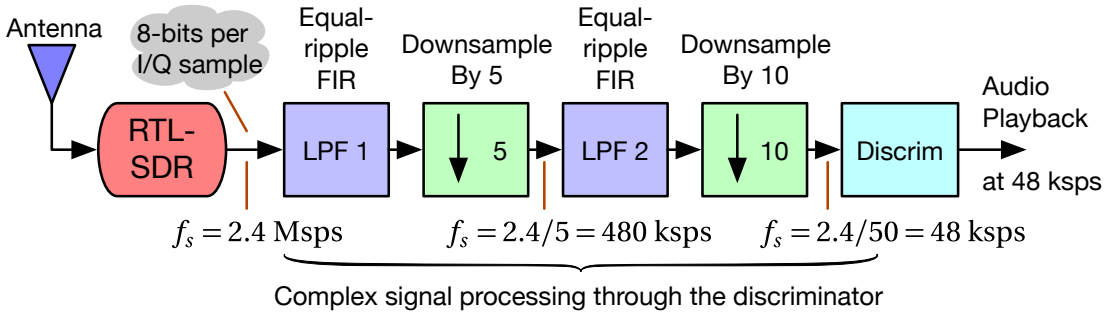
[2]:
```
# package to allow interfacing directly to the RTLSDR radio hardware
import rtlsdr as rtlsdr1
```

[3]:
```
def capture(Tc,fo=88.7e6,fs=2.4e6,gain=40,device_index=0):
    """
    Capture an array of complex radio samples:

    capture(Tc,fo=88.7e6,fs=2.4e6,gain=40,device_index=0)
    """
    # Setup SDR
    sdr1 = rtlsdr1.RtlSdr(device_index) #create a RtlSdr object
    #sdr.get_tuner_type()
    sdr1.sample_rate = fs
    sdr1.center_freq = fo
    #sdr.gain = 'auto'
    sdr1.gain = gain
```

```
    # Capture samples
    Nc = np.ceil(Tc*fs)
    x = sdr1.read_samples(Nc)
    sdr1.close()
    return x
```

1.1 A Custom Weather Channel Receiver



NBFM Receiver System Block Diagram

The signal flow makes use of *multirate* signal processing techniques, in particular *decimation*. From your understanding of sampling theory aliasing can be avoided so long as the sampling rate is greater than twice the highest frequency in the signal being sampled. When you lowpass filter a signal that is already in the discrete time domain, the bandwidth reduction may mean that the effective sampling is greater than needed. The downsamping block (arrow pointing down followed by an integer factor) means keep every Mth sample and discard the rest. The combination of the lowpass filter follwed by the downsampler forms a decimator. As described here, decimation by M. If the input sampling rate is f_s the output sampling rate becomes f_s/M.

1.1.1 Make a Live Signal Capture

Here we capture the National Weather Service (NWS) weather broadcast for COS at 162.475 MHz.

- We choose to sample the RF spectrum using the RTLSDR at 2.4 Msps
- We wish to demodulate the narrowband FM signal using a complex baseband discriminator, `x_disc = sdrh.discrim(z)`, operating at 48 ksps
- Two stages of lowpass filtering and decimation are to be implemented
- For lowpass filtering we use FIR digital filters design using `b_coeff = sk_dsp_comm.fir_design_helper.fir_remez_lpf(f_pass,f_stop,Ap_dB,As_dB,fs)`
- For decimation we use `x_dn = sk_dsp_comm.sigsys.downsample(x,M)`
- Two stages of decimation ease the processing burden and give flexibility
  - Starting at 2.4 Msps allows the low 8-bit resolution to increase as a result of the filtering and down sampling action
  - Try to design the filters to have length no more than 200 coefficients each

[8]:
```
Tc = 10 # capture time
# Tc, fo=88700000.0, fs=2400000.0, gain=40, device_index=0
x_wx = capture(Tc, fo=162475000.0, fs=2400000.0, gain=40)
```

```
Found Rafael Micro R820T tuner
[R82XX] PLL not locked!
```

Save a Backup Copy of the Capture

[27]:
```
sdrh.complex2wav('wx_IQ_capture.wav',2400000,x_wx)
```

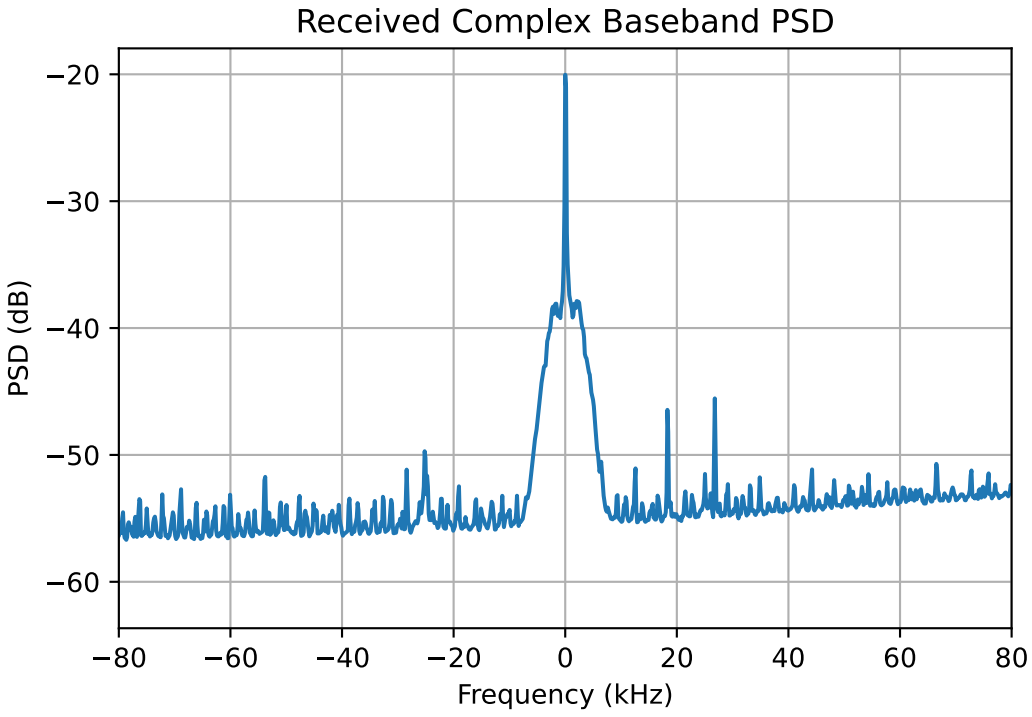
```
Saved as binary wav file with (I,Q)<=>(L,R)
```

Restore the Backup Copy of the Capture

[28]:
```
fs_IQ, x_wx_bak = sdrh.wav2complex('wx_IQ_capture.wav')
```

Check the Received Signal PSD

[32]:
```
P_x_wx, f_in = ss.my_psd(x_wx,2**14,2400);
plot(f_in,10*log10(P_x_wx))
xlim(-80,80);
title(r'Received Complex Baseband PSD')
ylabel(r'PSD (dB)')
xlabel('Frequency (kHz)');
grid();
```



Design the Lowpass Filters

[10]:
```
f_LPF1_pass = ?? # kHz; with fs 2400 kHz choose well below 1200 kHz
f_LPF1_stop = ?? # kHz; choose above the passband but again well below 1200 kHz
b1 = fir_d.fir_remez_lpf(f_LPF1_pass,f_LPF1_stop,0.1,60,2400) # lowpass at 2.4␣
 ↪Msps
f_LPF2_pass = ?? # kHz; about the narrowband FM lowpass bandwidth (~5 kHz)
f_LPF2_stop = ?? # kHz; choose above the passband control noise input to discrim.
b2 = fir_d.fir_remez_lpf(f_LPF2_pass,f_LPF2_stop,0.1,60,480) # lowpass at 480␣
 ↪ksps
(len(b1),len(b2)) # keep lengths under 200
```

[10]:
```
(125, 125)
```

Plot the Filter Magnitude Responses

[ ]:
```
fir_d.freqz_resp_list([b1],[1],'dB',2400,4096,fsize=(6,2)) # At fs = 2.400 Msps
ylim([-70,1])
# xlim([0,500])
title(r'First Lowpass: %d Coefficients' % len(b1))
xlabel(r'Frequency (kHz)')
grid();
```

[ ]:
```
fir_d.freqz_resp_list([b2],[1],'dB',480,fsize=(6,2)) # At fs = 480 ksps
ylim([-70,1])
title(r'Second Lowpass: %d Coefficients' % len(b2))
xlabel(r'Frequency (kHz)')
grid();
```

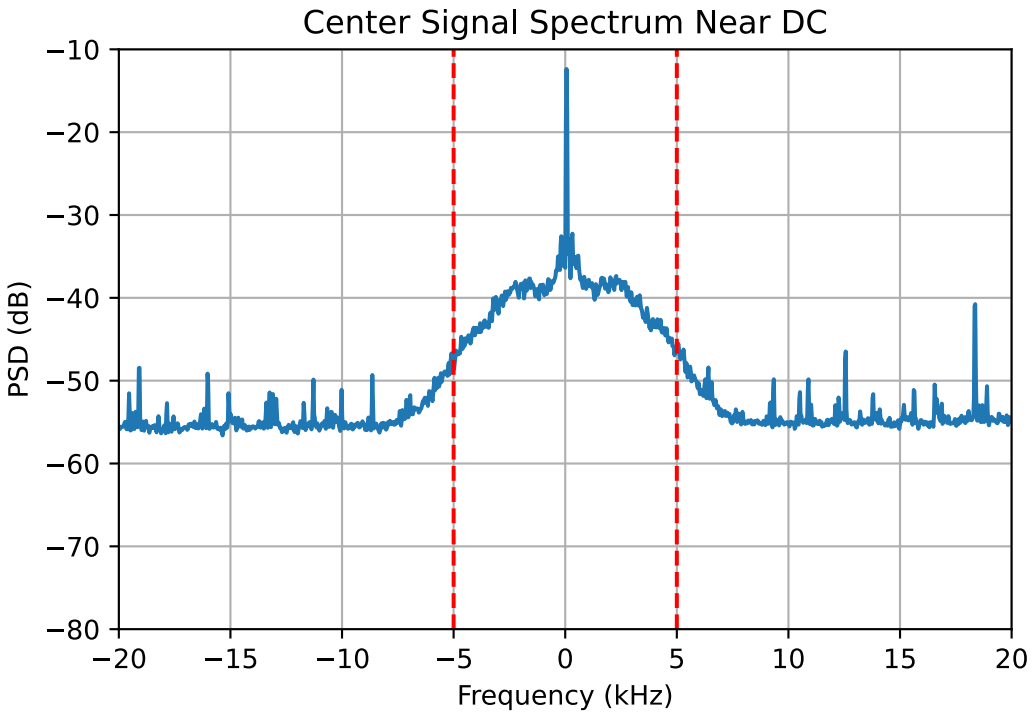
1.1.2 Implement the Block Diagram in a Sequential Signal Flow

[13]:
```
x_wx1 = signal.lfilter(b1,1,x_wx) # lowpass filter 1
x_wx1d = ss.downsample(x_wx1,5) # to get to 480 ksps
# center the signal at DC if needed
f_lo = 0 # Hz
x_wx1dc = x_wx1d * exp(1j*2*pi*f_lo/480e3*arange(len(x_wx1d))) # freq. shift
x_wx2 = signal.lfilter(b1,1,x_wx1dc) # lowpass filter 2
x_wx2d = ss.downsample(x_wx2,10) # to get to 480 ksps
x_wx_disc = sdrh.discrim(x_wx2d)
```

Check to See if the Signal Is Centered Near DC At the input to the discriminator function, `sdrh.discrim()` check to see that the spectrum is properly centered. Use the variable `f_lo` to complex frequency shift the signal up or down in frequency. The sample capture here was made using a higher performance RTLSDR which has higher quality local oscillator.

[14]:
```
Px_wx1dc, f_wx1dc = ss.my_psd(x_wx1dc,2**14,480);
plot(f_wx1dc,10*log10(Px_wx1dc))
plot([-5,-5],[-80,-10],'r--') # plot band edge lines
```

```
plot([5,5],[-80,-10],'r--')
ylim([-80,-10])
xlim([-20,20]);
title(r'Center Signal Spectrum Near DC')
ylabel(r'PSD (dB)')
xlabel('Frequency (kHz)')
grid();
```



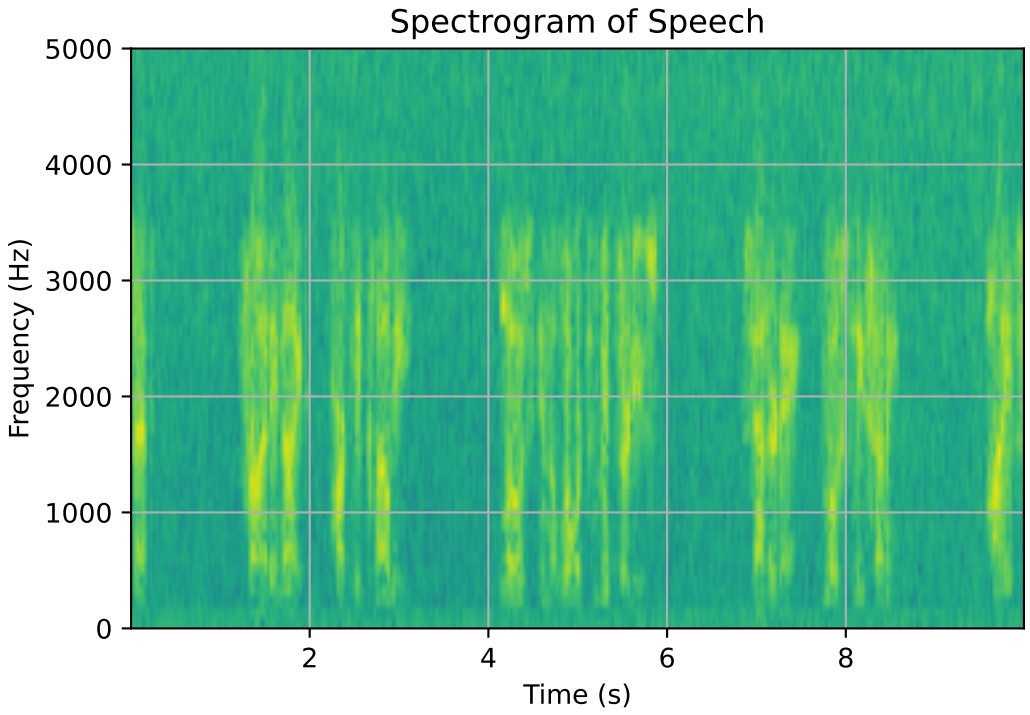
Listen to the Recovered Message Signal Turn in a `*.wav` file with the Lab 6 report that I can listen to when grading your report.

[34]:
```
Audio(x_wx_disc/max(abs(x_wx_disc)),rate=48000)
```

Display a Spectrogram of the Recovered Message to Verify Speech is Present

[25]:
```
specgram(x_wx_disc/max(abs(x_wx_disc)),NFFT=512,Fs=48000);
ylim(0,5000)
title(r'Spectrogram of Speech')
ylabel(r'Frequency (Hz)')
xlabel(r'Time (s)')
```

```
grid();
```


Spectrogram of Speech
5000
4000
3000
2000
1000
0
Frequency (Hz)
2
4
6
8
Time (s)