Project 2 Fall 2025: Channelizer for use in SDR

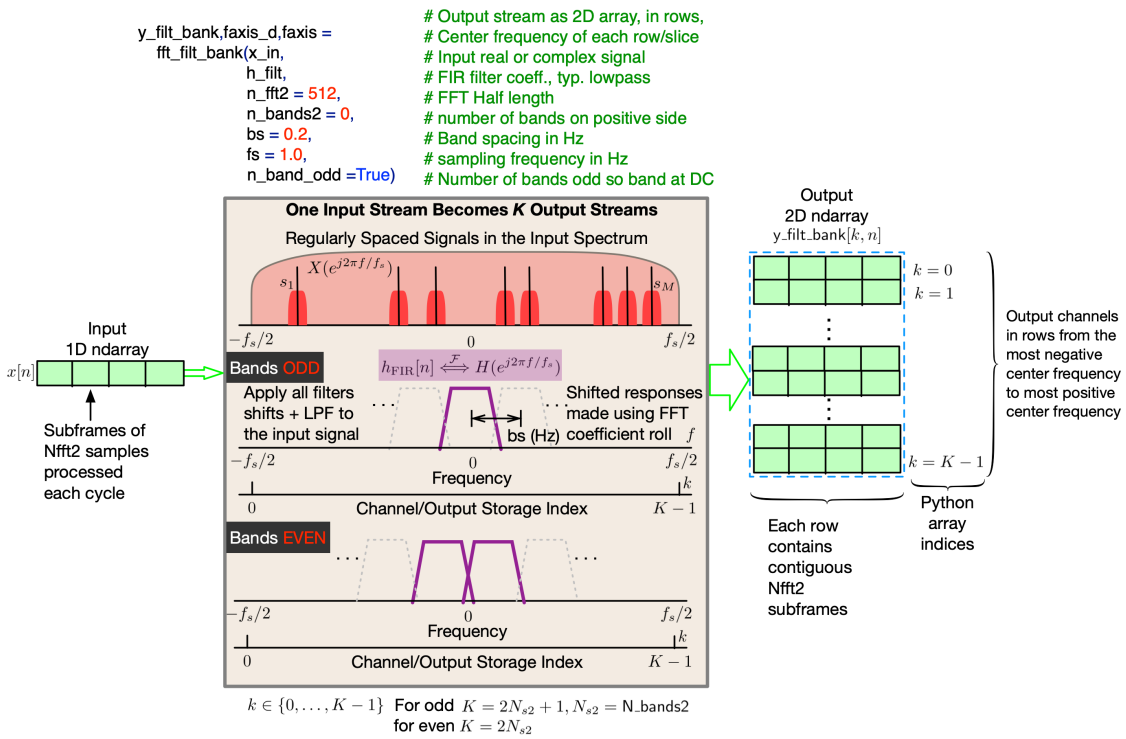
In [1]:
```
# %pylab inline
from numpy import *
from matplotlib.pyplot import *
%matplotlib inline
import sk_dsp_comm.sigsys as ss
import sk_dsp_comm.fir_design_helper as fir_h
import scipy.signal as signal
from IPython.display import Audio, display
from IPython.display import Image, SVG

%config InlineBackend.figure_formats=['svg'] # SVG inline viewing
```

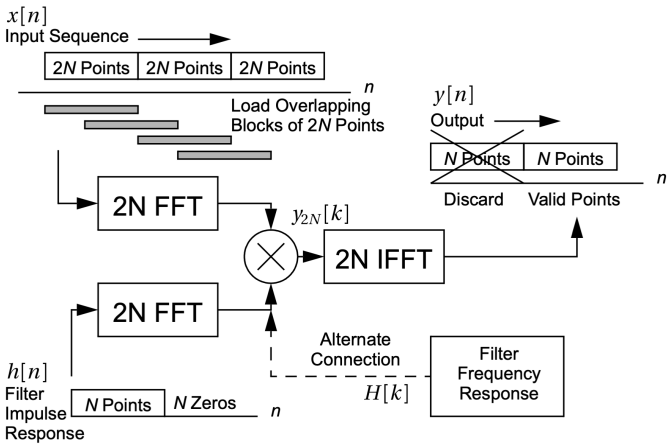
The FFT Filter Bank/Channelizer using Transform Domain Filtering

A block diagram of the function `fft_filt_bank()` is shown below. Two stages of filterng using this function will be implemented to break the signal into 10 kHz bandwidth channels that can then be demodulated to the underlying message signal. In this section of the document we explore a single filter bank using white noise testing.

Inside the function, which is contained in the module `sigsys_new.py`, found in the ZIP package, we repeatedly filter the input signal with frequency shifted versions of the input FIR filter, `h_filt`. The frequency shift values insure that a filter back is formed band center spacing of `BS_Hz`. The design of the filter itself controls the amount of the overlap between adjacent filter bank channels.



This function is based on transform domain filtering as described in notes Chapter 7. The relevant figure is shown below. Shifting the filter center frequency is easy once we are in the transform domain as all we need to is *roll* the coefficients left of right by a certain number of indices in the FFT of the filter.



Overlap and Save Filtering

To get started with the filter bank we first design a FIR lowpass filter to serve as the coefficients that are loaded into the parameter `h_filt` in the function block diagram. Later in this document FIR filter design specific to the two stage design will be synthesized. These filters are sufficient to complete the project tasks, but can modified if you so choose.

In [3]:
```
# Lowpass filter design for basic example
fs = 1000 # Hz
b_lpf_chan = fir_h.fir_remez_lpf(40,48,0.1,70,fs)
```

```
len(b_lpf_chan)
```

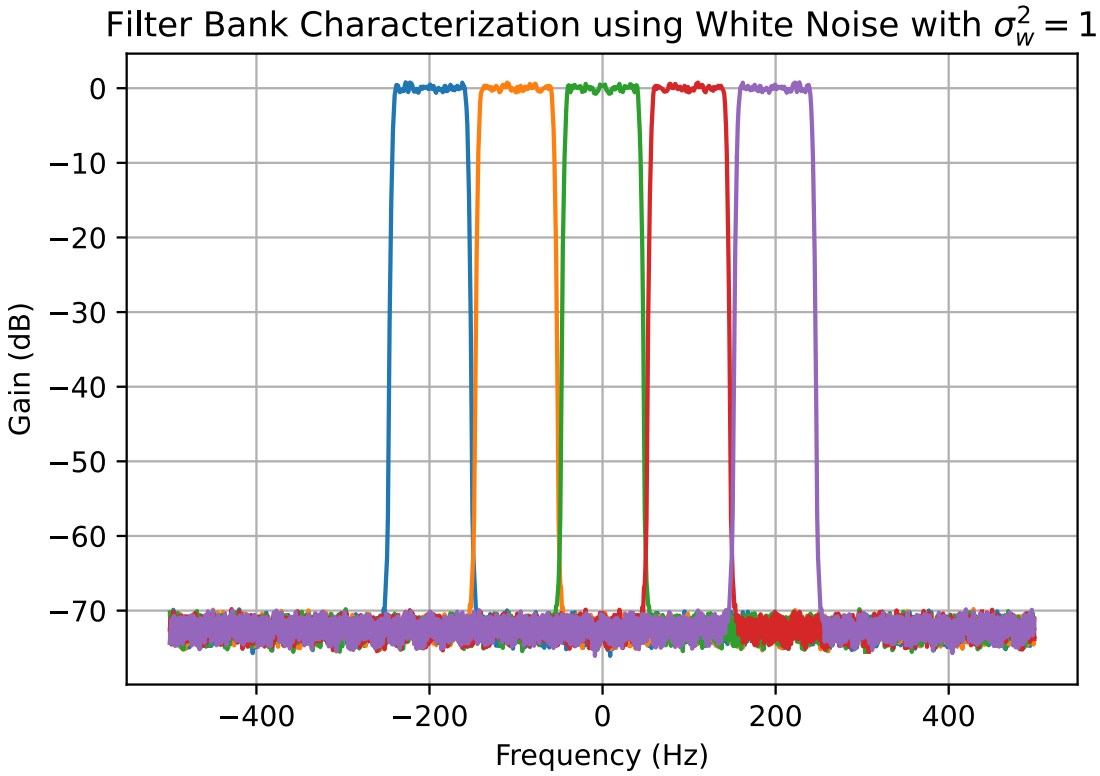
Out[3]: 355

Odd Band Design Noise Test

In [4]:
```
"""
New function interface notation
Nfft2 ==> n_fft2
N_bands2 ==> n_bands2
BS_Hz ==> bs
N_band_odd ==> n_band_odd
"""

fs = 1000 # Hz
BS_Hz = 100 # Hz
Nfft2 = 512
N_bands2 = 2
N_bands_tot = 2*N_bands2 + 1
w = random.randn(100000)
w_bank,fax,fax_des = ss.fft_filt_bank(w,b_lpf_chan + 0j,n_fft2=Nfft2,n
                                      bs=BS_Hz,fs = fs,n_band_odd=True)
figure(figsize=(6,4))
for k in range(N_bands_tot):
    P_w,f_w = ss.psd(w_bank[k,:],2**10,fs)
    plot(f_w,10*log10(P_w))
title(r'Filter Bank Characterization using White Noise with $\sigma_w^
ylabel(r'Gain (dB)')
xlabel(r'Frequency (Hz)')
grid();
# savefig('seven_band_example.pdf')
```

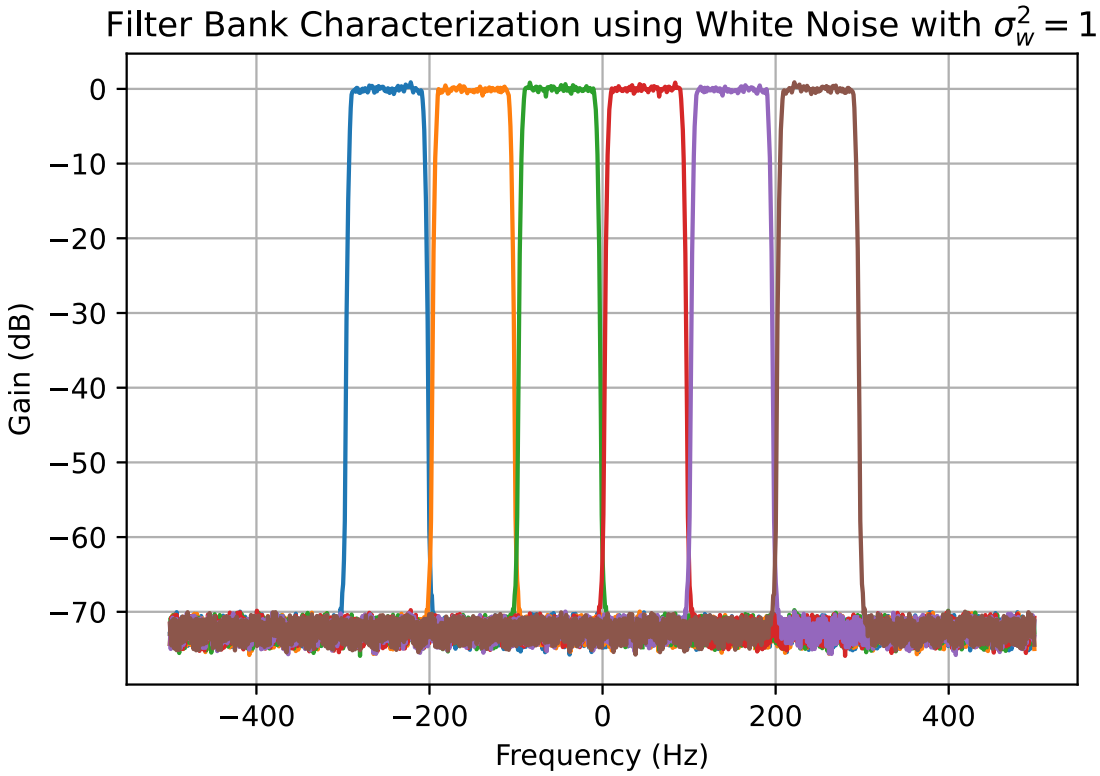
```
N_band_step = 102
N_band_step = 102 and BS_Hz_actual = 99.61 Hz
N_bands_tot = 5 and span_Hz = +/- 199.22 Hz
```



Even Band Design Noise Test

In [5]:
```
fs = 1000 # Hz
BS_Hz = 100 # Hz
Nfft2 = 512
N_bands2 = 3
N_bands_tot = 2*N_bands2 + 0
w = random.randn(100000)
w_bank,fax,fax_des = ss.fft_filt_bank(w,b_lpf_chan + 0j,n_fft2=Nfft2,n
                                      bs=BS_Hz,fs = fs,n_band_odd=Fals
figure(figsize=(6,4))
for k in range(N_bands_tot):
    P_w,f_w = ss.psd(w_bank[k,:],2**10,fs)
    plot(f_w,10*log10(P_w))
title(r'Filter Bank Characterization using White Noise with $\sigma_w^
ylabel(r'Gain (dB)')
xlabel(r'Frequency (Hz)')
grid();
# savefig('seven_band_example.pdf')
```

```
N_band_step = 51
N_band_step = 51 and BS_Hz_actual = 49.80 Hz
N_bands_tot = 6 and span_Hz = +/- 149.41 Hz
```



Frequency Accuracy and Rounding

Since we are using power of 2 FFT sizes and the center frequency step involves an integer, we cannot expect the desired filter center frequencies to be exact. Increasing N_{fft} will of course improve the accuracy. The function returns the *actual* band center frequencies and the *desired* values. From the desired band index frequencies we can also discern row index `k` that contains the input signal filtered with a center frequency of interest.

In [6]:
```
# Compare fax and rounded fax to nearest 100 Hz
print('fax_actual: ',rint(fax*1000)/1000)
print('fax_desired: ',fax_des)
```

```
fax_actual:  [-249.023 -149.414  -49.805   49.805  149.414  249.023]
fax_desired:  [-250. -150.  -50.   50.  150.  250.]
```

In [7]:
```
def freq2idx(fc,fax_des):
    """
    Helper for finding a list of frequency indices from a list
    desired center frequencies.
    """
    idx_list = []
    for k in range(len(fc)):
        idx_list.append(np.nonzero(np.ravel(fax_des == fc[k]))[0][0])
    return idx_list
```

In [8]:
```
freq2idx([-150,50],fax_des)
```

Out[8]: [np.int64(1), np.int64(3)]

Load Audio Test Vectors

This is not part of the project, but left some details in this document of how you create your own AM radio signals using speech test vectors. I opted to with real SDR capture as it is more realistic.

To create your own AM radio signals you could start with audio test vectors recorded at 8000 sps and then synthesize AM radio station signals for test purposes. In this project we do not do that.

In [9]:
```
fs, m1 = ss.from_wav('OSR_us_000_0018_8k.wav')
fs, m2 = ss.from_wav('OSR_us_000_0030_8k.wav')
fs, m3 = ss.from_wav('OSR_uk_000_0050_8k.wav')
```

In [10]:
```
Audio(m1,rate=fs)
```

Out[10]: 0:00 0:34

Load the SDR Capture File

These files were obtained using the `RTL-SDR` combined with a 125 MHz up converter designed for shifting high frequency (HF) signals covering 100 kHz - 65 MHz up in frequency to a range the SDR can receive.

Throughout this project we will make use of the following sampling frequency values:

In [11]:
```
# Sampling rate constants used in the system design.
fs_Hz = 1200000 # Hz
fs1_Hz = 240000 # Hz
fs2_Hz = 8000   # Hz
```

In [12]:
```
# Check the contents of the npz file and see what 'keys' are available
load('am3_fs1200_fc1070.npz')
```

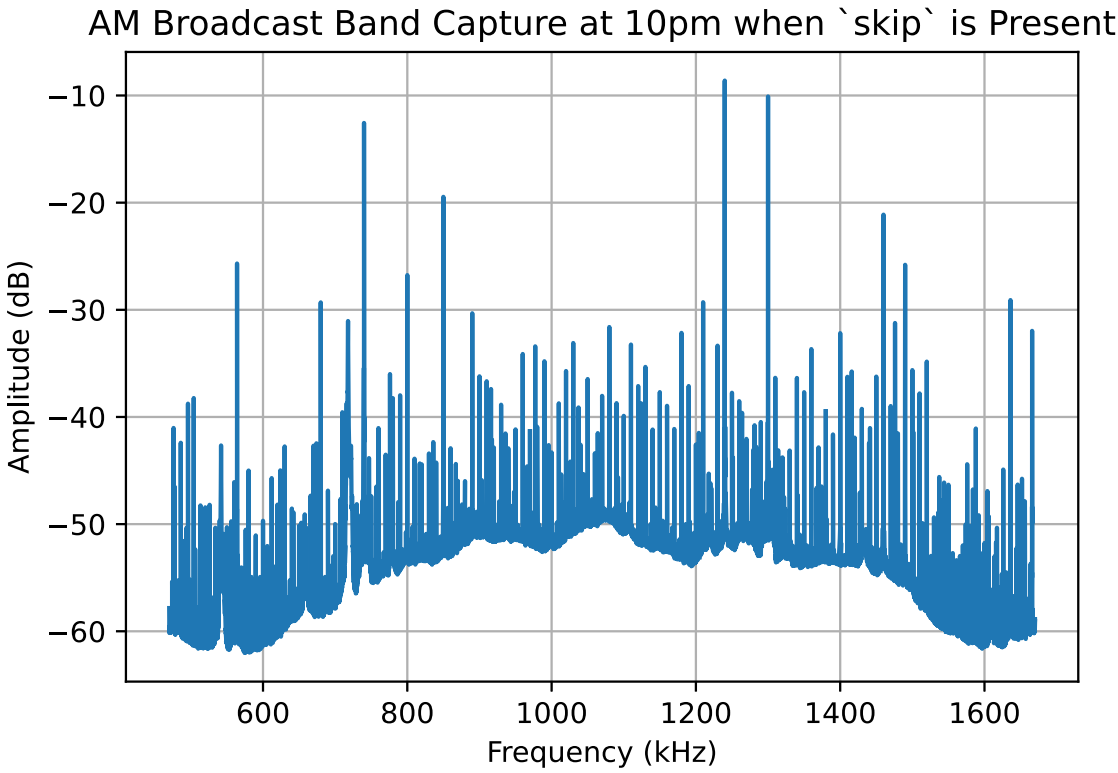
Out[12]: NpzFile 'am3_fs1200_fc1070.npz' with keys: am3_cpx

In [13]:
```
# Using the key 'am3_cpx' we copy the stream into an array
am_sdr_capture3 = load('am3_fs1200_fc1070.npz')
am_fs1200_fc1070 = am_sdr_capture3['am3_cpx']
```

```
# Check the sample length
len(am_fs1200_fc1070)
```

Out[13]: 35635198

In [14]:
```
Pam, fam = ss.psd(am_fs1200_fc1070,2**16,fs_Hz/1e3,scale_noise=False)
figure(figsize=(6,4))
plot(fam+1070,10*log10(Pam)) # shift the freq axis to match the captur
fcc = 1300 # kHz
# Comment the below to see the entire spectrum or zoom about fcc
# xlim(fcc-5,fcc+5)
title(r'AM Broadcast Band Capture at 10pm when `skip` is Present')
ylabel(r'Amplitude (dB)')
xlabel(r'Frequency (kHz)')
grid();
# savefig('full_capture_spectrum.pdf')
# savefig('zoom_capture_1300kHz.pdf')
```



Quick Test of the SDR Capture using $e^{j2\pi nfc/f_s}$ to Translate to DC

Here we verify that AM radio signals are present in the capture using complex frequency translation to DC followed by two stages of filtering and decimation (we do not yet employ the filter bank (channelizer) processing). We will use filter design that also work with the channelizer part of the project. First design the filters:

In [22]:
```
b_lpf_stage1 = fir_h.fir_remez_lpf(105,115,0.1,60,fs_Hz/1e3)
b_lpf_stage2 = fir_h.fir_remez_lpf(4.0,5.2,0.25,50,fs1_Hz/1e3,n_bump=5
print(f'stage1 taps: {len(b_lpf_stage1)}, stage2 taps: {len(b_lpf_stag
```

```
stage1 taps: 305, stage2 taps: 389
```

The Colorado Springs *AM Radio Market*

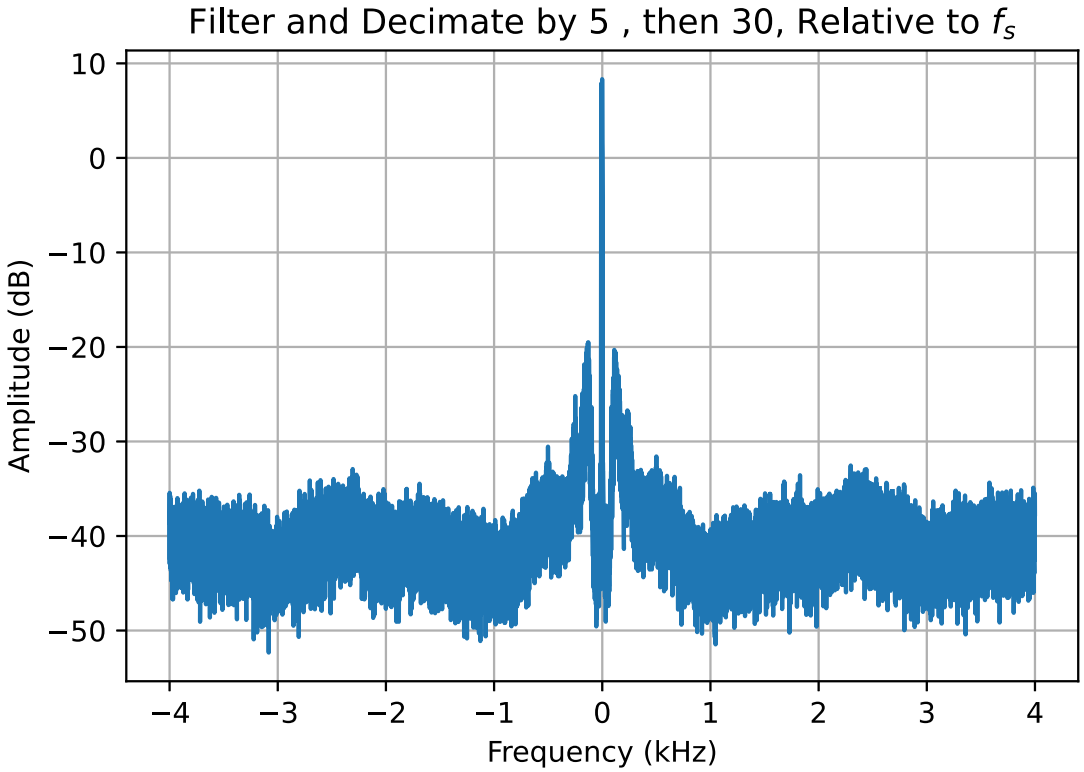
We have 3 three signal I was able to easily receive. However, after sundown the ionosphere moves closer the the surface of the earth making it possible for AM stations to skip from one end the continental US to the other. THe FCC mandates that most local AM stations reduce their transmit power to avoid interference with the strong **clear-channel** stations so they can be heard across the country (https://en.wikipedia.org/wiki/Clear-channel_station). The stations of interest from Colorado Springs in the daytime are:

- KVOR at 740 kHz
- KRDO at 1240 kHz
- KCSF at 1300 kHs
- Others are supposedly at 1460 (KZNT), 1530 (KOSC), and 1580 (KFCS), all frequencies in KHz

In [16]:
```
# Frequency shift KCSF which is at 1300 kHz
# we then decimate by 5 to get to fs1_Hz
n_shift = arange(len(am_fs1200_fc1070))
f_station = 1300 # kHz
am_sig1 = am_fs1200_fc1070 * exp(1j*2*pi*(1070-f_station)/(fs_Hz/1e3)*
am_sig2 = signal.lfilter(b_lpf_stage1,1,am_sig1)
am_sig2dn5 = ss.downsample(am_sig2,5)
```

In [17]:
```
# Filter and decimate by 30 to bring the rate down to fs2_Hz = 8000
# Gain the signal by 30 dB
am_sig3dn30 = ss.downsample(signal.lfilter(b_lpf_stage2,1,am_sig2dn5),
am_sig3dn30G30 = 10**(30/20) * am_sig3dn30
```

In [18]:
```
# Check the spectrum. An amplitude modulated signal has a strong carri
# at the center frequency.
Pam3, fam3 = ss.psd(am_sig3dn30G30,2**16,fs2_Hz,scale_noise=False)
figure(figsize=(6,4))
plot(fam3/1e3,10*log10(Pam3))
# xlim(-5,5)
title(r'Filter and Decimate by 5 , then 30, Relative to $f_s$')
ylabel(r'Amplitude (dB)')
xlabel(r'Frequency (kHz)')
grid();
```



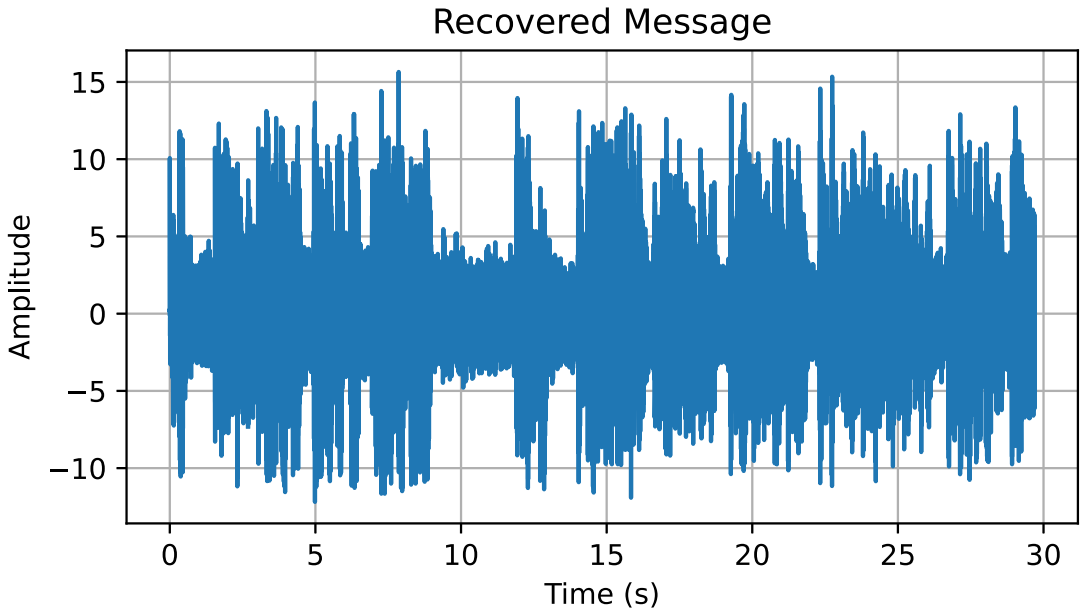
In [19]:
```
# Evelope detect using abs() and then DC block the signal
# Yes, this simple pair of operations is very trival in both analog ci
# AM radio has been around for more than 100 years.
x_sig3_env = abs(am_sig3dn30G30)
x_sig3_rec = signal.lfilter([1,-1],[1,-.95],x_sig3_env)
```

Take a Listen

In [20]:
```
Audio(x_sig3_rec,rate=fs2_Hz)
```

Out[20]: 0:00 0:29

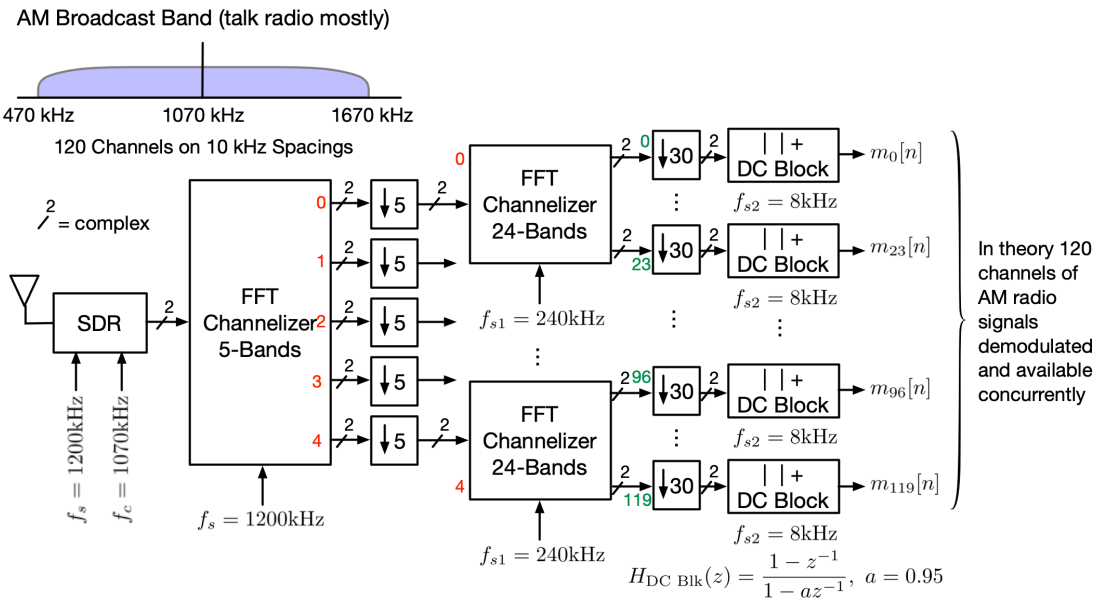
In [21]:
```
figure(figsize=(6,3)) # Set the figure size to 6" wide x 3" tall
t_s = arange(len(x_sig3_rec))/fs2_Hz
plot(t_s,x_sig3_rec)
title(r'Recovered Message')
ylabel(r'Amplitude')
xlabel(r'Time (s)')
grid();
```



Create a Channelizer Design Composed of Two Stages Using `ss.fft_filt_bank()`

Using the lowpass filters that are ready designed move to implement the two stage channelizer of the system block diagram. In the code below the output of stage 1 is the 2D `ndarray` variable `x_bank1`. Later we decimate by 5 a particular subband of interest and produce as output `x_bank2` from input `x_bank1[k,:]`, where `k` is the row index to the stage 1 channel of interest.

With the `x_bank2` output we again select a row index `k` of interest to finally decimate that signal by 30, envelope detect, and DC block the signal. At this point we have an 8 ksps waveform that we can listen to.



Stage 1: Create a 5-Band Channelizer

Since the current stage filter has length 305 we know that `Nfft2` > 512, assuming we use powers of 2 for the FFT. We may want to make `Nfft2` larger to provide more accurate frequency centers, which should be at multiples of 240 kHz.

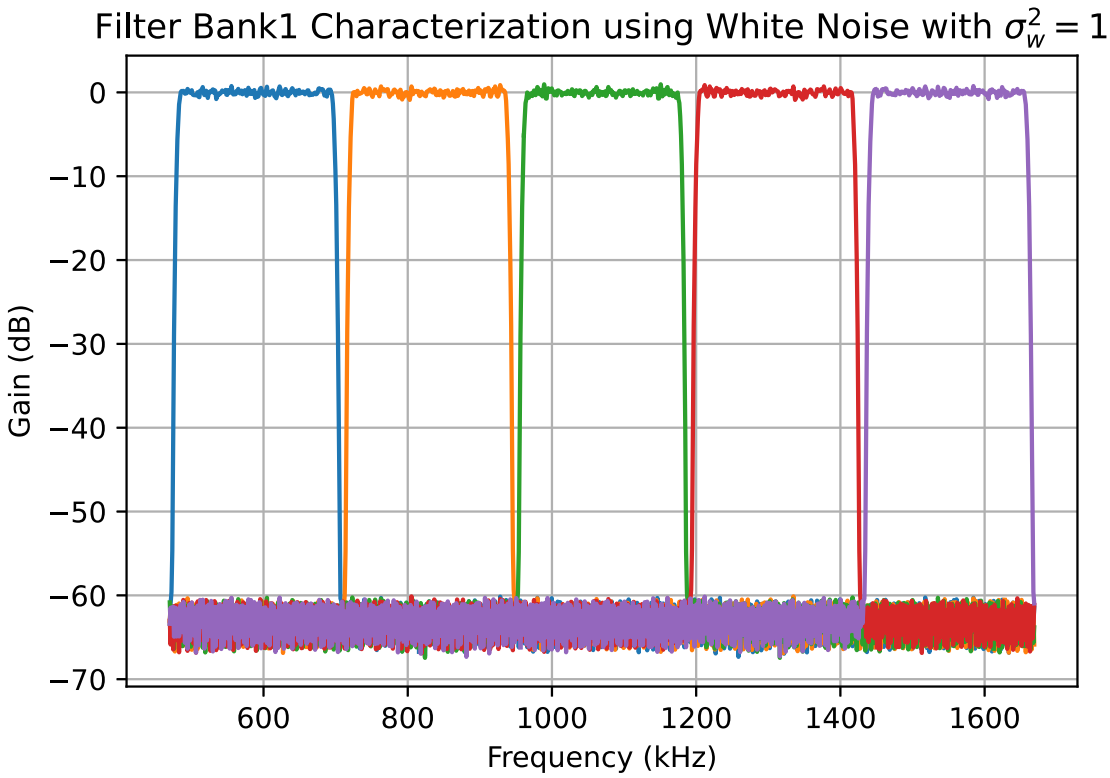
Noise Test Stage 1

Pass the SDR Signal Through

Begin by just filling the filter bank output array.

In [23]:
```
BS_Hz_1 = 240000
Nfft2 = 512
N_bands2_1 = 2
N_bands_tot = 2*N_bands2_1+ 1
# Select input source: noise test or SDR signal
w = random.randn(100000)
w_bank1,fax1,fax_des1  = ss.fft_filt_bank(w,b_lpf_stage1,n_fft2=Nfft2,
                                          n_bands2=N_bands2_1,bs=BS_Hz
                                          fs = fs_Hz,n_band_odd=True)
figure(figsize=(6,4))
for k in range(N_bands_tot):
    # Set scale_noise = True for noise and False for SDR signal
    P_x_in,f_x_in = ss.psd(w_bank1[k,:],2**10,fs_Hz/1e3,scale_noise=Tr
    plot(f_x_in + 1070,10*log10(P_x_in)) # Shift the frequency to repr
# title(r'Filter Bank Characterization using White Noise with $\sigma_
title(r'Filter Bank1 Characterization using White Noise with $\sigma_w
ylabel(r'Gain (dB)')
xlabel(r'Frequency (kHz)')
# ylim(-70,0)
grid();
# savefig('stage1_channelizer.pdf')
```

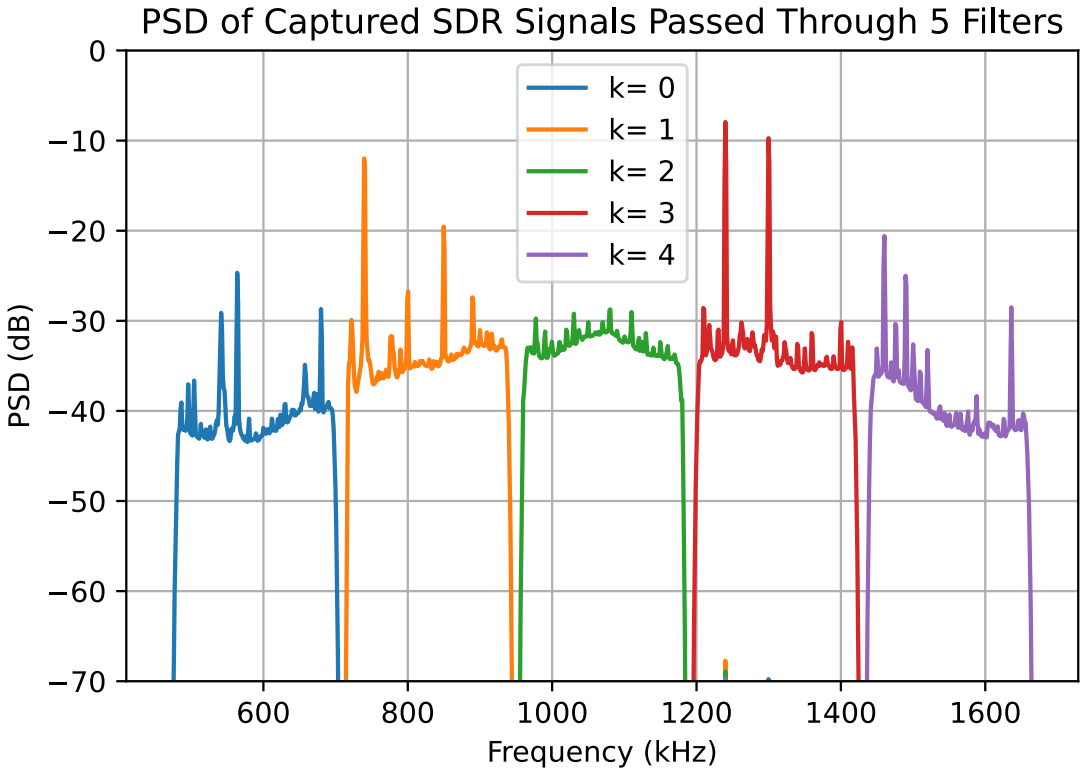
```
N_band_step = 205
N_band_step = 205 and BS_Hz_actual = 240234.38 Hz
N_bands_tot = 5 and span_Hz = +/- 480468.75 Hz
```



In [24]:
```
# %%timeit
BS_Hz_1 = 240000
Nfft2 = 512
N_bands2_1 = 2
N_bands_tot = 2*N_bands2_1+ 1
# Select input source: noise test or SDR signal
# x_in = random.randn(100000)
x_in = am_fs1200_fc1070
x_bank1,fax1,fax_des1  = ss.fft_filt_bank(x_in,b_lpf_stage1,n_fft2=Nff
                                          n_bands2=N_bands2_1,bs=BS_Hz
                                          fs = fs_Hz,n_band_odd=True)
```

```
N_band_step = 205
N_band_step = 205 and BS_Hz_actual = 240234.38 Hz
N_bands_tot = 5 and span_Hz = +/- 480468.75 Hz
```

In [25]:
```
figure(figsize=(6,4))
for k in range(N_bands_tot):
    # Set scale_noise = True for noise and False for SDR signal
    P_x_in,f_x_in = ss.psd(x_bank1[k,:],2**10,fs_Hz/1e3,scale_noise=Fa
    plot(f_x_in + 1070,10*log10(P_x_in),label='k= %d' % (k,))
# title(r'Filter Bank Characterization using White Noise with $\sigma_
title(r'PSD of Captured SDR Signals Passed Through 5 Filters')
ylabel(r'PSD (dB)')
xlabel(r'Frequency (kHz)')
legend()
ylim(-70,0)
grid();
# savefig('stage1_channelizer.pdf')
```



From the above we see that subbands `k=0`, `k=1`, `k=3`, and `k=4` all contain signals that may contain speech.

Stage 2: Create Channelizers of ~24-Bands Each Spaced by 10 kHz

Since the current stage filter has length 389 we know that `Nfft2` > 512, assuming we use powers of 2 for the FFT. We may want to make `Nfft2` larger to provide more accurate frequency centers, which should be at multiples of 10 kHz.

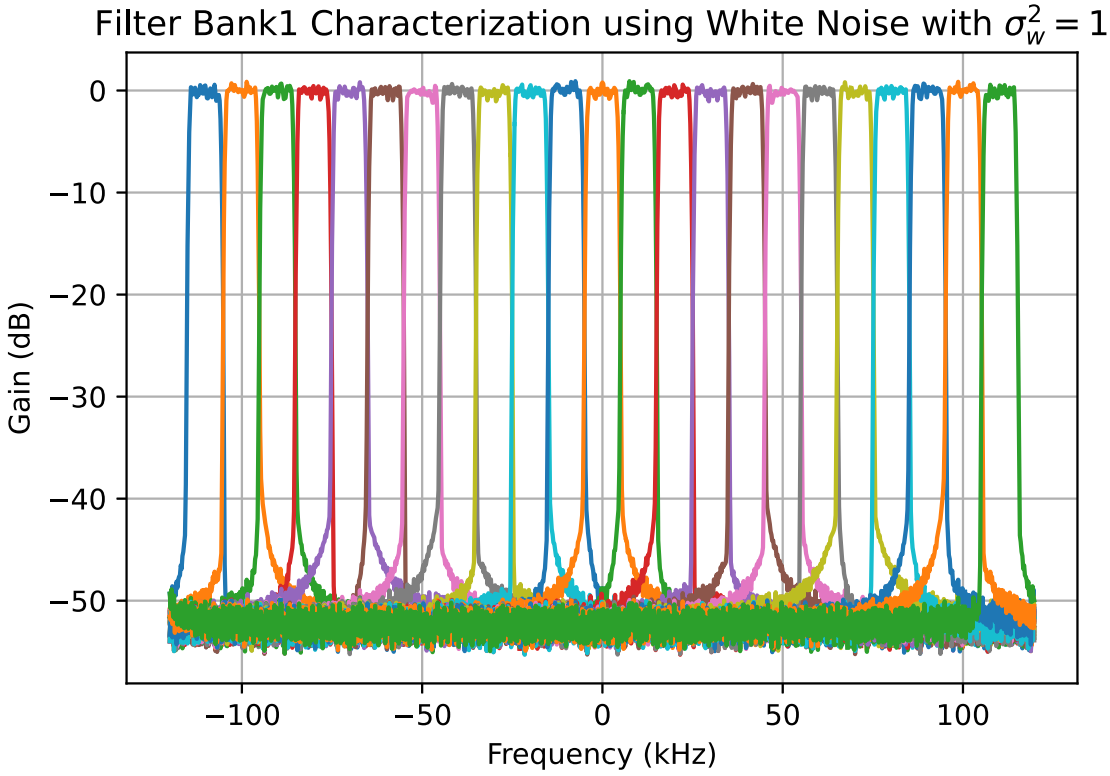
Noise Test A Single Stage 2 Filter Bank

In [26]:
```
BS_Hz_2 = 10000 # Hz
Nfft2 = 2048
N_bands2_2 = 11
N_bands_tot = 2*N_bands2_2 + 1
# Select input source: noise test or SDR signal
w = random.randn(100000)
w_bank2,fax2,fax_des2  = ss.fft_filt_bank(w,b_lpf_stage2,n_fft2=Nfft2,
                                          n_bands2=N_bands2_2,bs=BS_Hz
                                          fs = fs1_Hz,n_band_odd=True)

figure(figsize=(6,4))
for k in range(N_bands_tot):
    # Set scale_noise = True for noise and False for SDR signal
    P_x_in,f_x_in = ss.psd(w_bank2[k,:],2**10,fs1_Hz/1e3,scale_noise=T
```

```
    plot(f_x_in,10*log10(P_x_in))
title(r'Filter Bank1 Characterization using White Noise with $\sigma_w
ylabel(r'Gain (dB)')
xlabel(r'Frequency (kHz)')
grid();
# savefig('seven_band_example.pdf')
```

```
N_band_step = 171
N_band_step = 171 and BS_Hz_actual = 10019.53 Hz
N_bands_tot = 23 and span_Hz = +/- 110214.84 Hz
```



Pass the SDR Signal Through from Subband k=3 Decimated by 5

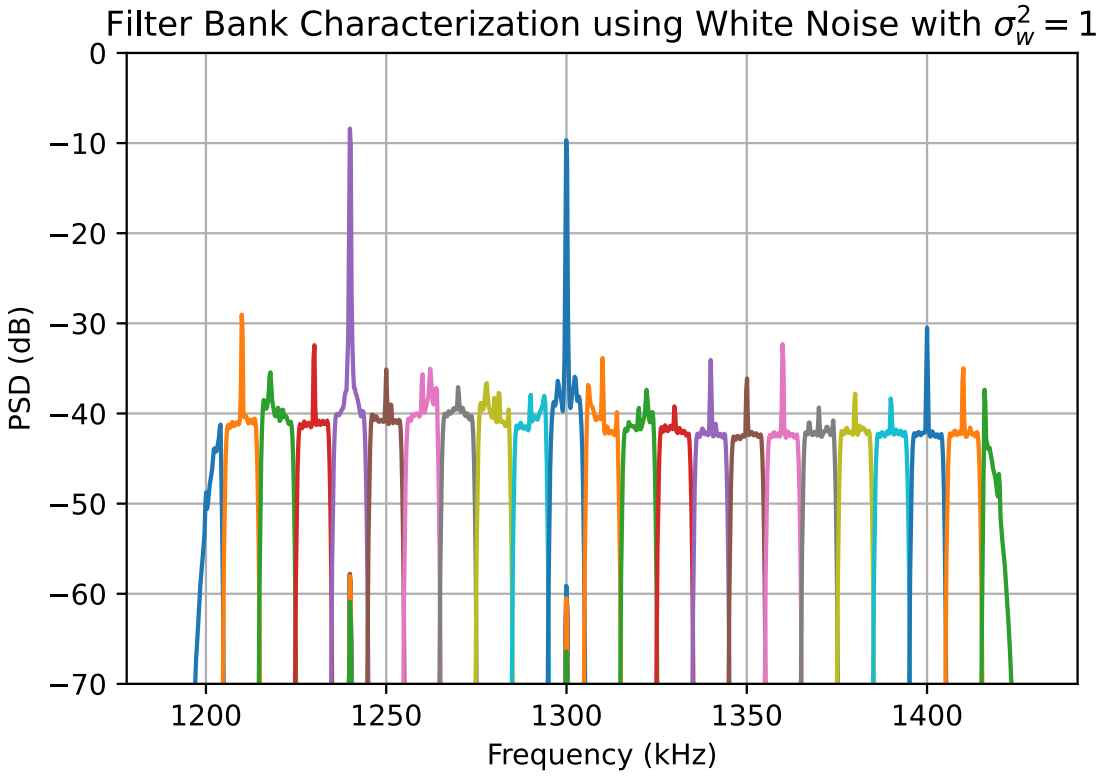
This signal is of special interest as Colorado Springs station KCSF at 1300 KHz is in this band. By passing this signal through the Bank2 channelizer with spacing 10 kHz we can isolate the 1300 kHz signal.

In [28]:
```
BS_Hz_2 = 10000 # Hz
Nfft2 = 2048
N_bands2_2 = 11
N_bands_tot = 2*N_bands2_2 + 1

x_in2 = ss.downsample(x_bank1[3,:],5) # Process the k = 3 band decimat
x_bank2,fax2,fax_des2  = ss.fft_filt_bank(x_in2,b_lpf_stage2,n_fft2=Nf
                                          n_bands2=N_bands2_2,bs=BS_Hz
                                          fs = fs1_Hz,n_band_odd=True)
```

```
N_band_step = 171
N_band_step = 171 and BS_Hz_actual = 10019.53 Hz
N_bands_tot = 23 and span_Hz = +/- 110214.84 Hz
```

In [29]:
```
figure(figsize=(6,4))
for k in range(N_bands_tot):
    # Set scale_noise = True for noise and False for SDR signal
    P_x_in,f_x_in = ss.psd(x_bank2[k,:],2**10,fs1_Hz/1e3,scale_noise=F
    plot(f_x_in + 1070 + 240,10*log10(P_x_in))
title(r'Filter Bank Characterization using White Noise with $\sigma_w^
ylabel(r'PSD (dB)')
xlabel(r'Frequency (kHz)')
ylim(-70,0)
grid();
# savefig('seven_band_example.pdf')
```



Comments The two strong signals from bank1, `k=3`, have now been isolated. The row index in `x_bank2` is needed so we can further process this channel. We do a reverse index lookup using the function `freq2idx` defined earlier in this notebook. We will work with the `fax_des_kHz` frequencies, are first scaled to kHz from Hz units and then frequency translated using capture centerfrequency of 1070 kHz and the subband `k=3` which is shifted up by 240 kHz. The calculations below detail how this is done. As an alternative you can just count the subband index start from the far left of the PSD plot knowing that the first subband is `k=0`.

In [30]:
```
# Convert to kHz
```

```
fax2_kHz = fax2/1e3
fax_des2_kHz = fax_des2/1e3
print('fax_actual kHz: ',rint(fax2_kHz*10)/10 + 1070 + 1*240)
print('fax_desired kHz: ',fax_des2_kHz + 1070 + 1*240)
```

```
fax_actual kHz:  [1199.8 1209.8 1219.8 1229.8 1239.9 1249.9 1259.9 126
9.9 1279.9 1290.
 1300.  1310.  1320.  1330.  1340.1 1350.1 1360.1 1370.1 1380.1 1390.2
 1400.2 1410.2 1420.2]
fax_desired kHz:  [1200. 1210. 1220. 1230. 1240. 1250. 1260. 1270. 128
0. 1290. 1300. 1310.
 1320. 1330. 1340. 1350. 1360. 1370. 1380. 1390. 1400. 1410. 1420.]
```

- Find indices for AM radio signals at 1210, 1240, and 1300 kHz:

In [31]:
```
freq2idx([1210,1240,1300],fax_des2_kHz + 1070 + 240) # Shift the frequ
```

Out[31]:
```
[np.int64(1), np.int64(4), np.int64(10)]
```

- Introduce the final decimation by 30 directly on say the `k=10` signal, then gain scale it by 30 dB, envelope detect, and filter with the DC block as described in the system block diagram

In [35]:
```
# Evelope detect using abs() and then DC block the signal
# Yes, this simple pair of operations is very trival in both analog ci
# AM radio has been around for more than 100 years.
x_1300 = ss.downsample(x_bank2[10,:],30)
x_1300_env = abs(x_1300*10**(30/20)) # Gain by 30 dB
x_1300_rec = signal.lfilter([1,-1],[1,-.95],x_1300_env)
### Take a Listen
Audio(x_1300_rec,rate=fs2_Hz)
```

Out[35]:

0:00 0:29

- Listen to the above and recall the class demo
- All things considered it sounds great!