Lab 6 Sample Notebook: RTL-SDR

As Lab 6 moves into the section *Writing Your Own Demodulator Algorithms* you will need to add two Python packages beyond the installation of the RTL-SDR drivers and making use of module `lab6.py` included in the ZIP package. The lab reader discusses this but as of May 2026 there have been some changes in the packages.

The two Python packages that need to be suinstalled are:

- `sdr_helper` which was originally written for this lab; to install from the terminal in your `dsp-comm313` or similar virtual environment use

  ```
  (dsp-comm313) PS C:\Users\mwick> pip install sdr-helper
  ```

- `rtlsdr` a package that can directly control the RTL-SDR dongle; to install from the terminal in your `dsp-comm313` or similar virtual environment use

  ```
  (dsp-comm313) PS C:\Users\mwick> pip install pyrtlsdr[lib]
  ```

In [1]:
```
# %pylab inline
from numpy import * # better in import numpy as np
from matplotlib.pyplot import * # better to import matplotlib.pyplot as plt
import sk_dsp_comm.sigsys as ss
# As a result of: pip install pip install sdr-helper
import sk_dsp_comm.sdr_helper as sdr
# This package also requires: pip install pyrtlsdr[lib]
import lab6
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
```

Capture 5 Seconds of I/Q Samples at 88.7 MHz (KCME, Colorado Springs)

In [2]:
```
Tc = 5
# Tc, fo=88700000.0, fs=2400000.0, gain=40, device_index=0
x = lab6.capture(Tc, fo=88700000.0, fs=2400000.0, gain=40)
```

Archive the capture in a wave file

In [3]:
```
lab6.complex2wav('KCME_IQ.wav',2400000,x)
```

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

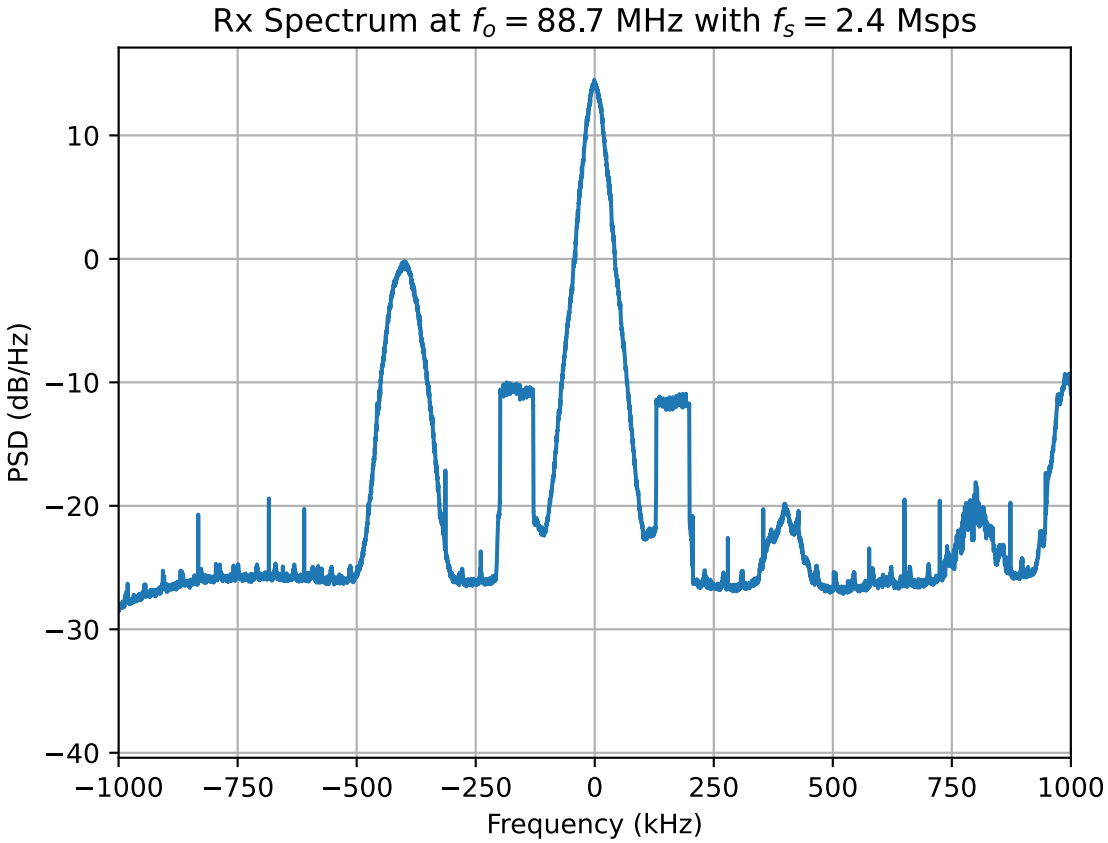
Restore the IQ Samples into a Complex Baseband Array

If need be, restore an archived capture back into the workspace. The sampling rate will be 2.4 Msps as that is how it was originall captured and save.

In [4]:
```
fs, x = lab6.wav2complex('KCME_IQ.wav')
```

Look at the Complex Baseband Spectrum from the RTL-SDR

In [11]:
```
Px, fx = ss.psd(x,2**14,fs=2400,scale_noise=True);
plot(fx,10*log10(Px))
xlim(-1000,1000);
title(r'Rx Spectrum at $f_o = 88.7$ MHz with $f_s = 2.4$ Msps')
xlabel(r'Frequency (kHz)')
ylabel(r'PSD (dB/Hz)')
grid();
```



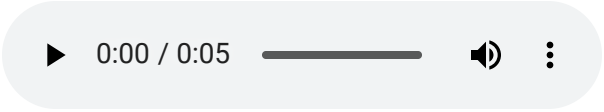
Process Samples Through a Mono FM Demodulator

In [7]:
```
z_bb, z_out = sdr.mono_fm(x,fs=2.4e6,file_name='KCME.wav')
```

```
Done!
```

In [8]:
```
Audio('KCME.wav')  # From file
```

Out[8]:

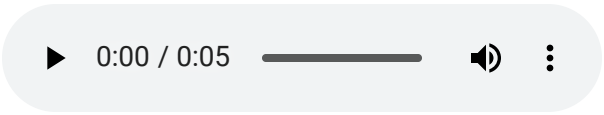


In [33]:
```
z_bb, theta, y_lpr, y_lmr, z_out = \
        sdr.stereo_fm(x,fs=2.4e6,file_name='KCME_stereo.wav')
```

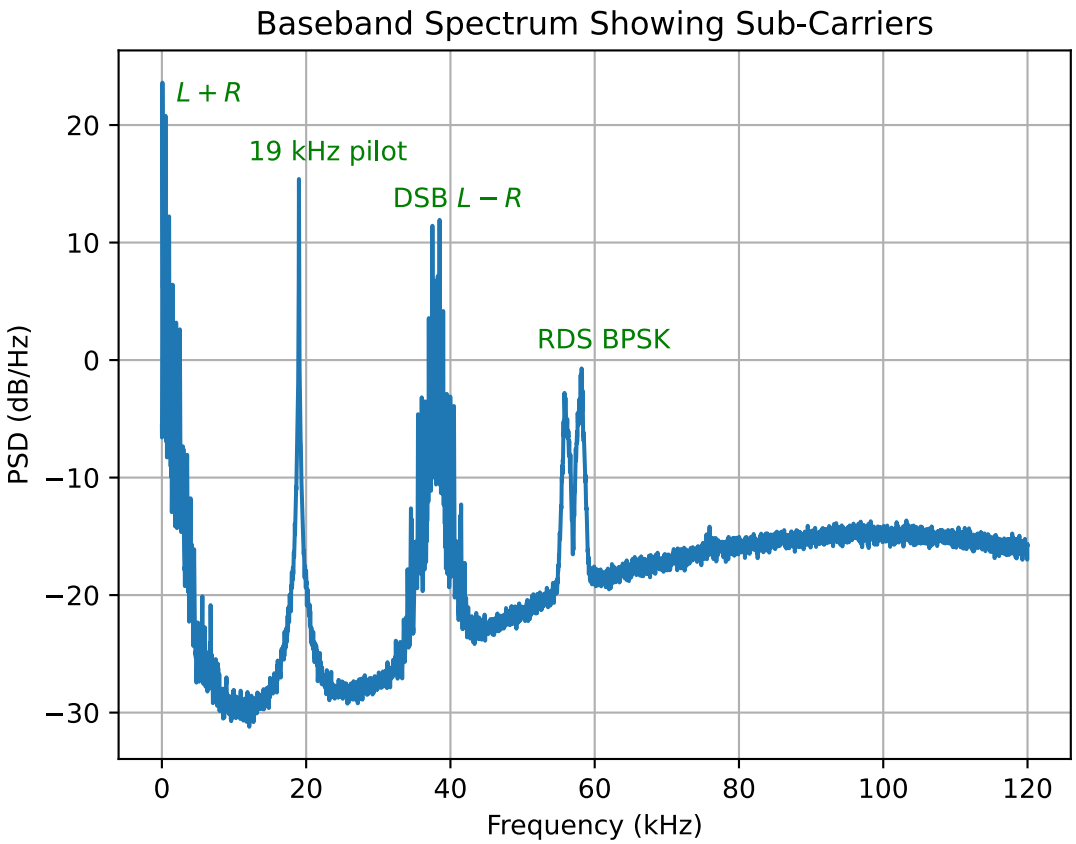
```
Done!
```

In [34]:
```
Audio('KCME_stereo.wav')  # From file
```

Out[34]:



In [40]:
```
P_z_bb, f_x_bb = ss.psd(z_bb,2**14,fs=2400/10.,scale_noise=True)
plot(f_x_bb,10*log10(P_z_bb))
title(r'Baseband Spectrum Showing Sub-Carriers')
text(2.0, 22.0, r'$L+R$',c='green')
text(12.0, 17.0, r'19 kHz pilot',c='green')
text(32.0, 13.0, r'DSB $L-R$',c='green')
text(52.0,1.0, r'RDS BPSK',c='green')
xlabel(r'Frequency (kHz)')
ylabel(r'PSD (dB/Hz)')
grid();
```



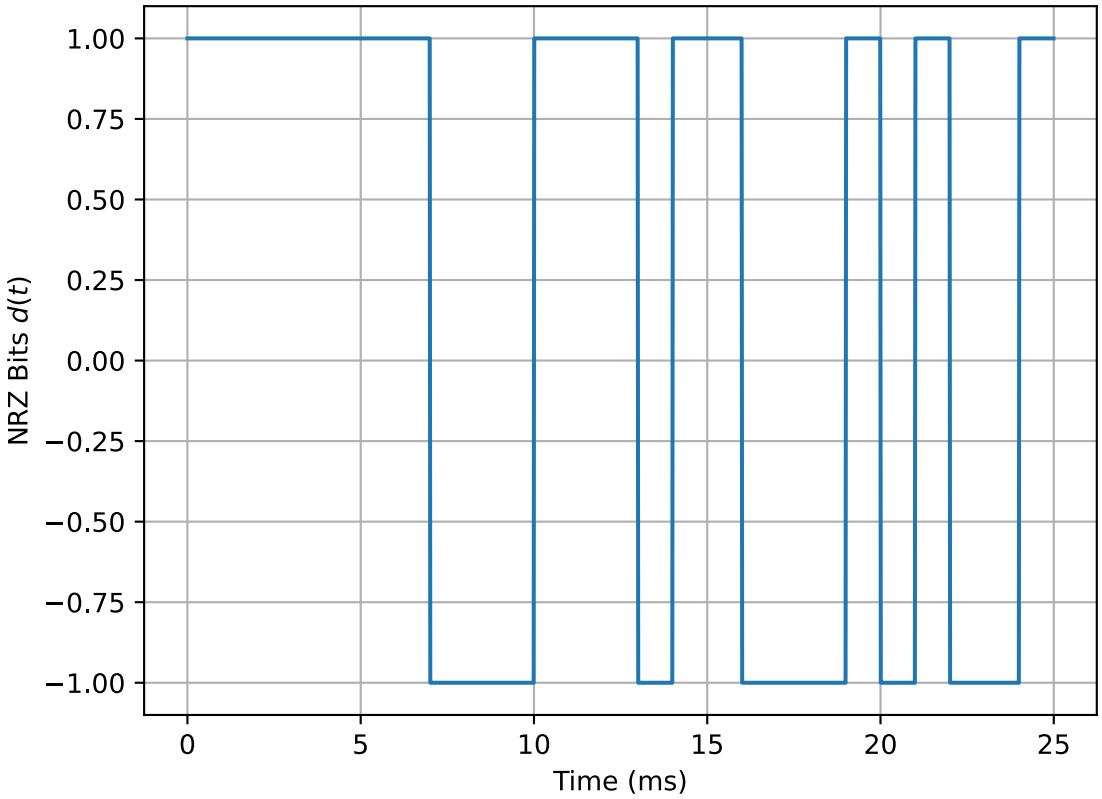
- Note the subcarrier label `RDS BPSK` is a binary phase-shift keyed subcarrier responsible for the *radio data service* that produces information such as the current sone being played by the radio station; https://pysdr.org/content/rds.html
- Coherent demodulation from the 57 kHz subcarrier is possible using three times the 19 pilot

FSK Simulation

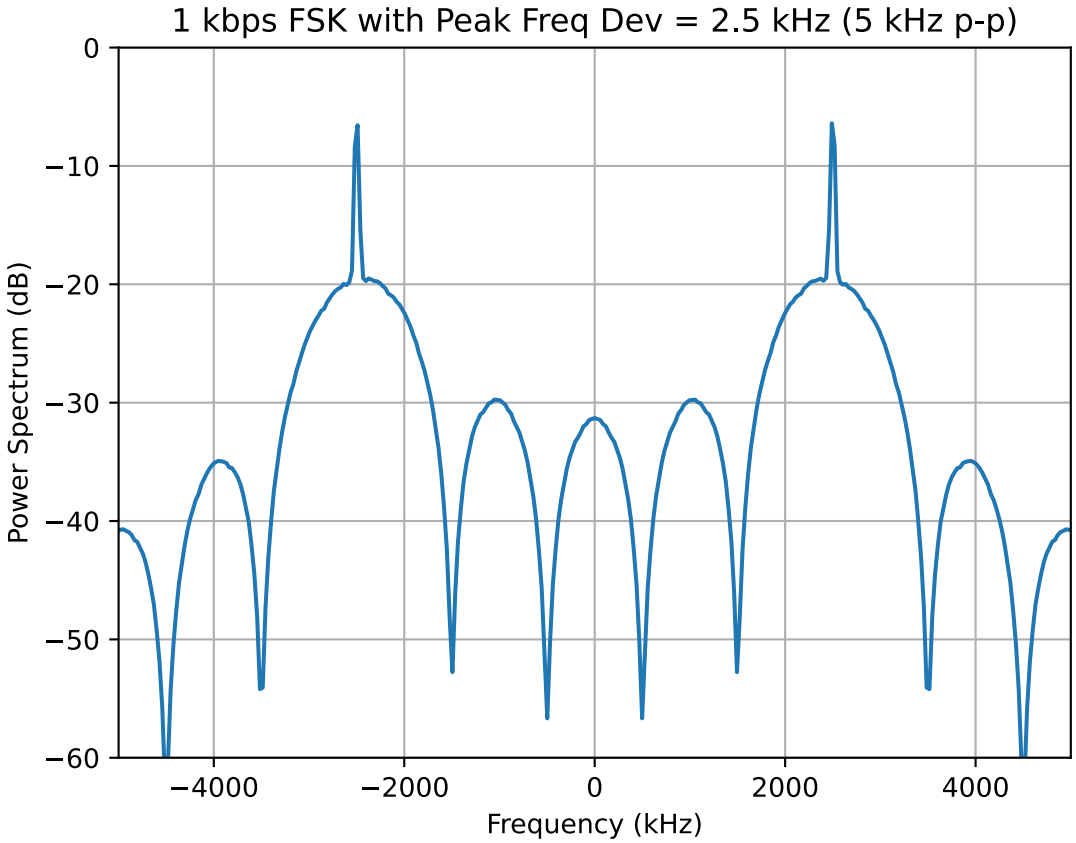
The simulation code shown below can be used to understand the the performance of an FSK receiver using a real RF transmnit signal generated by the Keysight 33600A and then received by the RTLSDR.

In [42]:
```
Ns = 120; Nbits = 1000; Rb = 1000; Df = 2500.0; fs = Rb*Ns;
data = ss.pn_gen(Nbits,7)
#data = ss.pn_gen(Nbits,7)
waveform = signal.lfilter(ones(Ns),1,ss.upsample(data,Ns))
x, b = ss.nrz_bits2(data,Ns,'rect')
n = arange(Ns*Nbits)
xc = exp(1j*2*pi*Df*x/fs*n);
```

In [43]:
```
plot(n[:3000]/fs*1000,x[:3000])
xlabel(r'Time (ms)')
ylabel(r'NRZ Bits $d(t)$')
grid();
```



In [46]:
```
Pxc,fxc = ss.psd(xc,2**12,fs,scale_noise=False);
plot(fxc,10*log10(Pxc))
xlim([-5000,5000])
ylim([-60,0])
title(r'1 kbps FSK with Peak Freq Dev = 2.5 kHz (5 kHz p-p)')
xlabel(r'Frequency (kHz)')
ylabel(r'Power Spectrum (dB)')
grid();
savefig('FSK1kbps2_5.pdf')
```



FSK Capture $\Delta f = 2.5$ kHz

The details here require first configuring the Keysight 33600A to produce a low power FSK signal at 70 MHz.

In [333…
```
Tc = 20 # or 10s
# Tc, fo=88700000.0, fs=2400000.0, gain=40, device_index=0
x_FSK2_5 = sdr.capture(Tc, fo=70000000.0, fs=2400000.0, gain=40)
```

In [ ]:
```
psd(x_FSK2_5,2**14,2400);
xlim(-50,50);
xlabel('Frequency (kHz)');
```

Save and Restore IQ Data

In [113…
```
sdr.complex2wav('FSK_70M_IQ2_5.wav',2400000,x_FSK2_5)
```

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

In [22]:
```
fs, x_FSK2_5 = sdr.wav2complex('FSK_70M_IQ2_5.wav')
```

Processing the Capture

$N_2 = 5$ Final Decimation Stage Lowpass Filter

In [27]:
```
b_FSK = fir_d.fir_remez_lpf(1000,4000,.1,60,2.4e6/10)
```

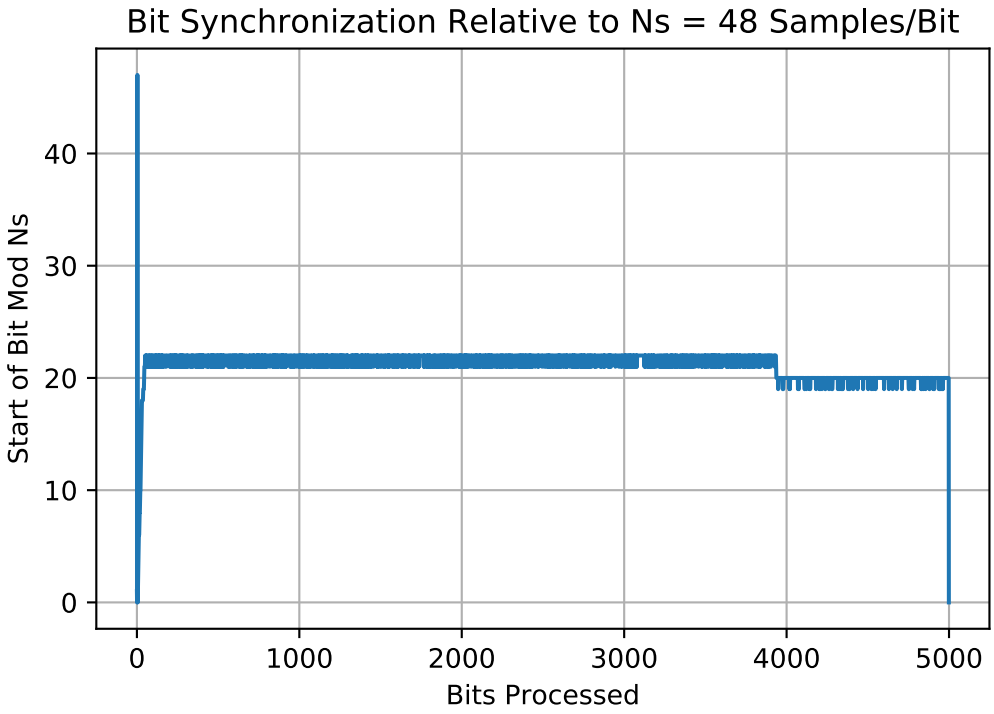
```
Remez filter taps = 205.
```

In [ ]:
```
# FSK demod code
```

Bit Synch to Received Data Bits

In [31]:
```
data_hat, clk, track = sdr.sccs_bit_sync(data_hat_FSK2_5,48)
```

In [63]:
```
plot(track)
title(r'Bit Synchronization Relative to Ns = 48 Samples/Bit')
ylabel(r'Start of Bit Mod Ns')
xlabel(r'Bits Processed')
grid();
```



Prepare an Equal Length Stream Transmitted PN7 Bits

In [56]:
```
def PN7_33600A(Nbits):
    """
    Create an Nbits stream equivalent of the Keysight 33600A PN7 PBRS
```

```
    Mark Wickert April 2019
    """
    m = 7
    PN7_oct_taps = [0o211,0o217,0o235,0o367,0o277,0o325,0o203,0o313,0o345]
    taps = np.array([1, 0, 0, 0, 0, 0, 1, 1])
    sr = np.ones(m)
    # M-squence length is:
    Q = 2**m - 1
    c = np.zeros(Q)
    for n in range(Q):
        tap_xor = 0
        c[n] = sr[-1]
        for k in range(1,m):
            if taps[k] == 1:
                tap_xor = np.bitwise_xor(tap_xor,np.bitwise_xor(int(sr[-1]),int(sr[
        sr[1:] = sr[:-1]
        sr[0] = tap_xor
    PN7 = c[::-1] # reverse the sequence
    PN7_stream = zeros(Nbits)
    for n in range(Nbits):
        PN7_stream[n] = PN7[mod(n,127)]
    return PN7_stream
```

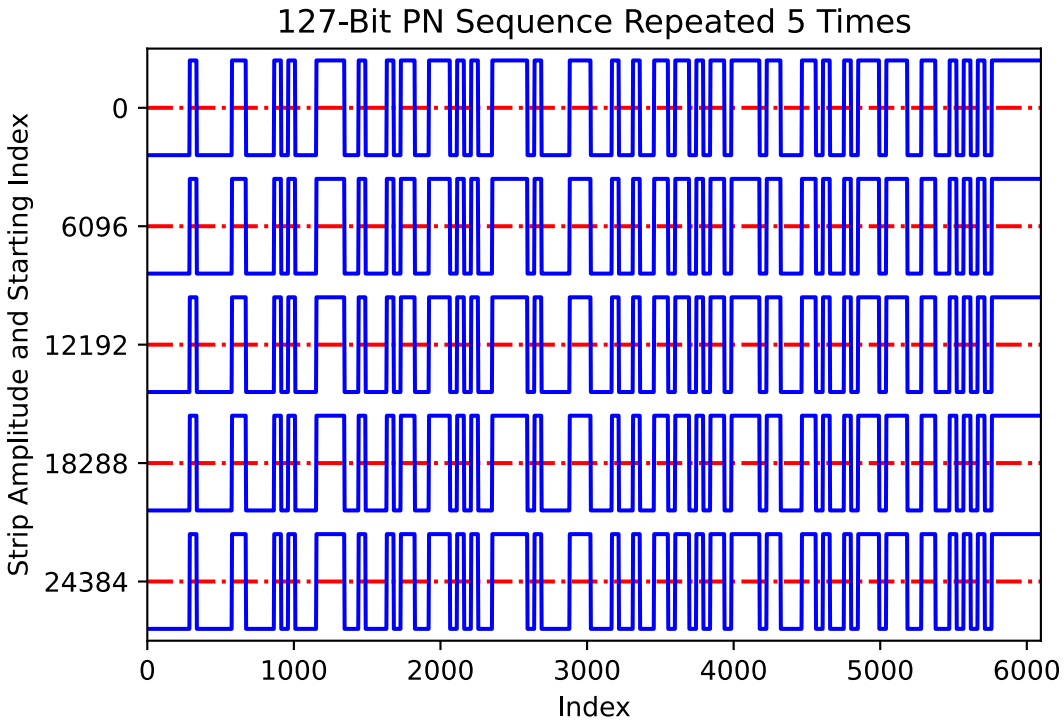
Generate Tx Reference Bits and Error Detect

In [ ]:
```
tx_bits = PN7_33600A(len(data_hat))
N_bits, N_errors = dc.bit_errors(tx_bits,int16((data_hat[5:]+1)/2))
print('N_bits = %d, N_errors = %d, BEP = %1.2e' % (N_bits,N_errors, N_errors/N_bits
```

Other Support Code

Using `strips()` for Plotting a Long Waveform

In [57]:
```
# data = ss.pn_gen(127*5,7)
# Make 5 periods of the 33600A PN7
data = PN7_33600A(5*127) # create 5 periods of a 127 bit sequence
x, b = ss.nrz_bits2(data,48,'rect')
dc.strips(x,48*127);
title(r'127-Bit PN Sequence Repeated 5 Times')
savefig('strips_example.svg')
```



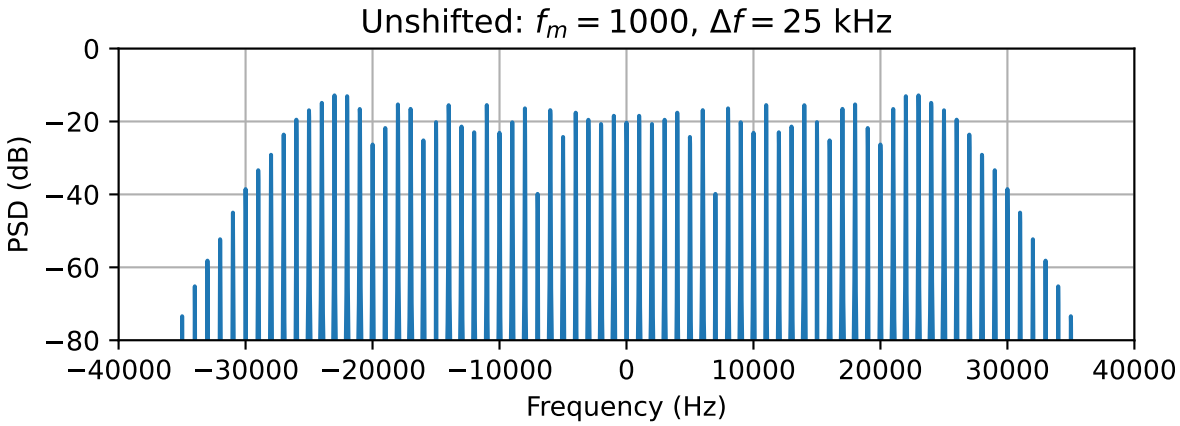
Sinusoidal FM Transmitter

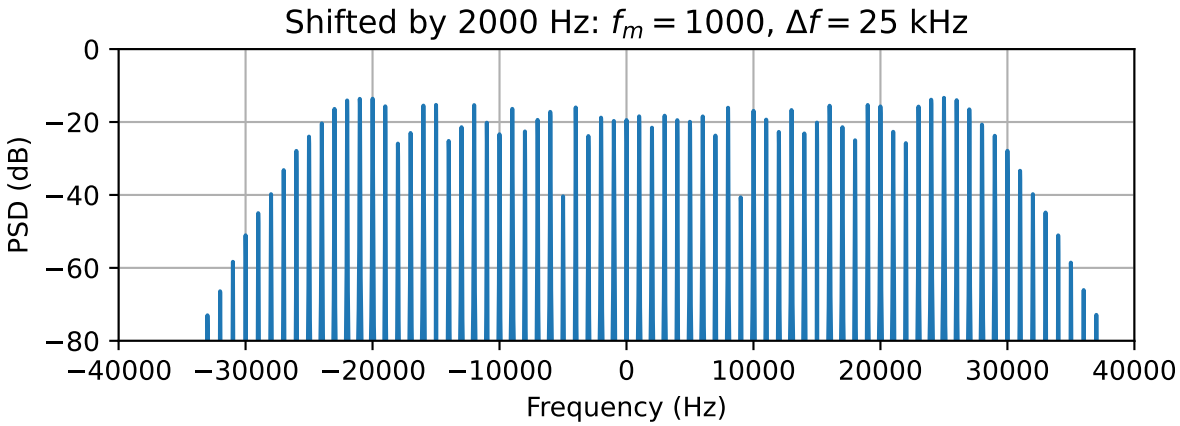
In [50]:
```
def bb_fm_tx(m,fd,fs):
    '''
    A simple complex baseband FM modulator

    Mark Wickert April 2019
    '''
    return exp(1j*2*pi*fd/float(fs)*cumsum(m))
```

In [75]:
```
# Here m is a 1kHz tone with fs = 240kHz, fd = 25kHz
fm = 1000.; fs = 240e3; fd = 25e3;
n = arange(0,100000)
x_tx = bb_fm_tx(cos(2*pi*fm/fs*n),fd,fs)
subplot(211)
P_x_tx, f_x_tx = ss.psd(x_tx,2**14,fs,scale_noise=False)
plot(f_x_tx,10*log10(P_x_tx))
title(r'Unshifted: $f_m = 1000$, $\Delta f = 25$ kHz')
ylabel(r'PSD (dB)')
xlabel(r'Frequency (Hz)')
xlim([-40e3, 40e3])
ylim([-80, 0])
grid();
subplot(212)
# Include a tuning frequency error of +2000 Hz
x_tx *= exp(1j*2*pi*2000./fs*n)
P_x_tx, f_x_tx = ss.psd(x_tx,2**14,fs, scale_noise=False)
plot(f_x_tx,10*log10(P_x_tx))
title(r'Shifted by 2000 Hz: $f_m = 1000$, $\Delta f = 25$ kHz')
ylabel(r'PSD (dB)')
xlabel(r'Frequency (Hz)')
```

```
xlim([-40e3, 40e3])
ylim([-80, 0])
grid();
tight_layout()
# savefig('FM_tx_Python.svg')
```





In [ ]: