RF Signal Generator User Guide

This RF signal generator is written in python and provides a simple way to generate signals for research and development purposes

There are two ways to interface with the generator:

1. Use the “signal_gen.py” interface to generate signal files through a menu system
2. Write your own program to call the library functions with the desired arguments

This user guide will cover both methods.

Method 1: using the “signal_gen.py” interface

[placeholder – program not written yet]

Method 2: using the individual library functions

You can call individual library functions contained in the modulation, audio, and spread spectrum utility libraries. A detailed breakdown of the libraries and functions follows:

mod_utils.py

This library contains functions to conduct basic modulation on data, typically entered as a byte array. It has the following callable functions:

exit_code, array = tone_gen (freq, N, samp_rate)

generates a single complex tone

inputs:

freq:	[int] the desired tone frequency in Hz
N:	[int] length, in samples, of the returned tone array
samp_rate:	[int] sample rate of the returned complex array, samples-per-second

outputs:

exit_code:	[int] function return code. 0 for success, 1+ for error
array:	[np array, c64] returned modulated array. Numpy data type is complex64 (gunradio complex32)

exit_code, array = fsk_mod_2 (raw_data, samp_rate, baud_rate, freq_div, center_freq)

generates a 2FSK waveform for digital TX. Non-coherent phase

inputs:

raw_data:	[bytearray] raw data to be modulated
samp_rate:	[int] sample rate of the returned complex array, samples-per-second
baud_rate:	[int] sample rate of the returned complex array, symbols-per-second. Used with the sample rate to derive the samples-per-symbol
freq_div:	[int] frequency spacing between high (1) and low (0) frequency in Hz. This is the signal bandwidth in the spectrum
center_freq	[int] baseband offset frequency in Hz. For example, 0 is at baseband, -10000 would be 10kHz below baseband, and 25000 would be 25kHz above baseband. This is not the center frequency of transmission

outputs:

exit_code:	[int] function return code. 0 for success, 1+ for error
array:	[np array, c64] returned modulated array. Numpy data type is complex64 (gunradio complex32)

exit_code, array = fsk_mod_4 (raw_data, samp_rate, baud_rate, freq_div, center_freq)

generates a 4FSK waveform for digital TX. Non-coherent phase

inputs:

raw_data:	[bytearray] raw data to be modulated
samp_rate:	[int] sample rate of the returned complex array, samples-per-second
baud_rate:	[int] sample rate of the returned complex array, symbols-per-second. Used with the sample rate to derive the samples-per-symbol
freq_div:	[int] frequency spacing between high (11) and low (00) frequency in Hz. This is the signal bandwidth in the spectrum
center_freq	[int] baseband offset frequency in Hz. For example, 0 is at baseband, -10000 would be 10kHz below baseband, and 25000 would be 25kHz above baseband. This is not the center frequency of transmission

outputs:

exit_code:	[int] function return code. 0 for success, 1+ for error
array:	[np array, c64] returned modulated array. Numpy data type is complex64 (gunradio complex32)

exit_code, array = gfsk_mod_2 (raw_data, samp_rate, baud_rate, freq_div, center_freq, window_len)

generates a Gaussian 2FSK waveform for digital TX. Has a coherent phase due to Gaussian window

inputs:

raw_data:	[bytearray] raw data to be modulated
samp_rate:	[int] sample rate of the returned complex array, samples-per-second
baud_rate:	[int] sample rate of the returned complex array, symbols-per-second. Used with the sample rate to derive the samples-per-symbol
freq_div:	[int] frequency spacing between high (1) and low (0) frequency in Hz. This is the signal bandwidth in the spectrum
center_freq	[int] baseband offset frequency in Hz. For example, 0 is at baseband, -10000 would be 10kHz below baseband, and 25000 would be 25kHz above baseband. This is not the center frequency of transmission
window_len	[int] Gaussian window length in samples. Larger window lengths will result in lower spectral sidelobes. Odd values preferred. Ideal window lengths vary based on TX waveform, see below image for details

outputs:

exit_code:	[int] function return code. 0 for success, 1+ for error
array:	[np array, c64] returned modulated array. Numpy data type is complex64 (gunradio complex32)

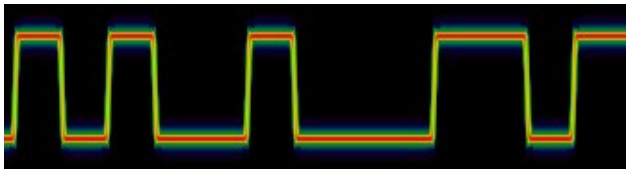

Figure: low window length (0.15 sps)

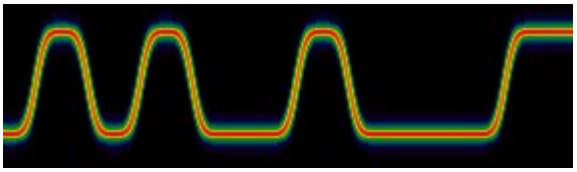

Figure: high window length (0.85 sps)

exit_code, array = gfsk_mod_4 (raw_data, samp_rate, baud_rate, freq_div, center_freq, window_len)

generates a Gaussian 4FSK waveform for digital TX. Has a coherent phase due to Gaussian window

inputs:

raw_data:	[bytearray] raw data to be modulated
samp_rate:	[int] sample rate of the returned complex array, samples-per-second
baud_rate:	[int] sample rate of the returned complex array, symbols-per-second. Used with the sample rate to derive the samples-per-symbol
freq_div:	[int] frequency spacing between high (11) and low (00) frequency in Hz. This is the signal bandwidth in the spectrum

center_freq	[int] baseband offset frequency in Hz. For example, 0 is at baseband, -10000 would be 10kHz below baseband, and 25000 would be 25kHz above baseband. This is not the center frequency of transmission
window_len	[int] Gaussian window length in samples. Larger window lengths will result in lower spectral sidelobes. Odd values preferred. Ideal window lengths vary based on TX waveform, see below image for details

outputs:

exit_code:	[int] function return code. 0 for success, 1+ for error
array:	[np array, c64] returned modulated array. Numpy data type is complex64 (gunradio complex32)

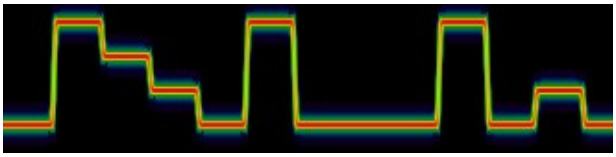

Figure: low window length (0.15 sps)

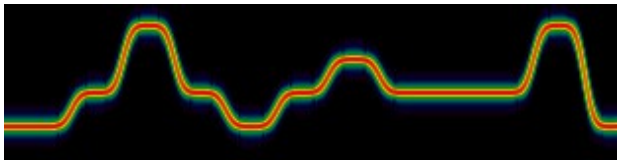

Figure: high window length (0.85 sps)

exit_code, array = bfsk_mod (raw_data, samp_rate, baud_rate, center_freq)

generates a BPSK waveform for digital TX. Has a non-coherent phase

inputs:

raw_data:	[bytearray] raw data to be modulated
samp_rate:	[int] sample rate of the returned complex array, samples-per-second
baud_rate:	[int] sample rate of the returned complex array, symbols-per-second. Used with the sample rate to derive the samples-per-symbol
center_freq	[int] baseband offset frequency in Hz. For example, 0 is at baseband, -10000 would be 10kHz below baseband, and 25000 would be 25kHz above baseband. This is not the final center frequency of transmission, only the baseband center frequency

outputs:

exit_code:	[int] function return code. 0 for success, 1+ for error
array:	[np array, c64] returned modulated array. Numpy data type is complex64 (gunradio complex32)

exit_code, array = qfsk_mod (raw_data, samp_rate, baud_rate, center_freq)

generates a QPSK waveform for digital TX. Has a non-coherent phase

inputs:

raw_data:	[bytearray] raw data to be modulated
samp_rate:	[int] sample rate of the returned complex array, samples-per-second
baud_rate:	[int] sample rate of the returned complex array, symbols-per-second. Used with the sample rate to derive the samples-per-symbol
center_freq	[int] baseband offset frequency in Hz. For example, 0 is at baseband, -10000 would be 10kHz below baseband, and 25000 would be 25kHz above baseband. This is not the final center frequency of transmission, only the baseband center frequency

outputs:

exit_code:	[int] function return code. 0 for success, 1+ for error
array:	[np array, c64] returned modulated array. Numpy data type is complex64 (gunradio complex32)

exit_code, array = gauss_window_gen (window_len)

generates a PSD from the standard normal distribution

inputs:

window_len:	[int] length of the Gaussian window in samples

outputs:

exit_code:	[int] function return code. 0 for success, 1+ for error
array:	[np array, f32] returned window array. Numpy data type is float32