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

This program is a more “user friendly” way to interface with the signal generator

Call the program by typing the command

python3 sig_gen.py

The following menu will be displayed:

```
ubuntu@ubuntu:~/rf_signal_gen$ python3 sig_gen.py
Signal Generator Tool
version: 0.0.3

===== basic waveforms =====
0:   single tone
1:   swept tone
2:   analog AM
3:   analog FM
4:   on-off-keyed (OOK)
5:   2-level Frequency Shift Keyed (2FSK)
6:   4-level Frequency Shift Keyed (4FSK)
7:   2-level Gaussian Frequency Shift Keyed (2GFSK)
8:   4-level Gaussian Frequency Shift Keyed (4GFSK)
9:   Binary Phase Shift Keyed (BPSK)
10:  Quadrature Phase Shift Keyed (QPSK)

===== advanced waveforms =====
11:  Frequency Hopping Spread Spectrum (FHSS)
12:  Direct Sequence Spread Spectrum (DSSS)
13:  Orthogonal Frequency Division Multiplexing (OFDM)

select a signal to generate:
```

input numbers for the signal you would like to generate. As an example, 7 will generate a 2-level GFSK waveform

```
select a signal to generate: 7

2-level gaussian frequency keying selected

waveform data (full filename path unless in local directory): /home/ubuntu/test.txt
opening file: /home/ubuntu/test.txt
waveform frequency bandwidth (Hz): 50000
baseband center frequency (Hz): 0
signal baud rate (symbols-per-second): 9600
sample rate (sps): 250000
gaussian window percentage (0.0 through 1.0): .35

generator function successful
size in memory: 38272/38.272/0.038272 B/KB/MB

output the complex 64-bit array to a file
NOTE - file name should include the file type for the use case.
.fc32, .cf32, and .iq are all common file types for complex data

output file name: out.fc32

enter output file path, leave blank to save in the local directory

output file path: /home/ubuntu/
file write successful

signal generation complete - exiting
ubuntu@ubuntu:~/rf_signal_gen$
```

Follow the prompts to input waveform parameters. For example, the 2-level GFSK waveform takes:

file name: the file name of data to use for the waveform. Make sure to include the path if the file does not exist in the local directory

frequency bandwidth: separation frequency between the high and low frequency components. In this case, the example was 50000 Hz (50 kHz) which would be +25 kHz for a “1” and -25 kHz for a “0”

baseband center frequency: offset from baseband in Hz for the center carrier frequency. A value of 0 has no offset. A value of 100000 (100 kHz) would mix the signal to a center frequency of 100 kHz

baud rate: the symbols-per-second of the waveform

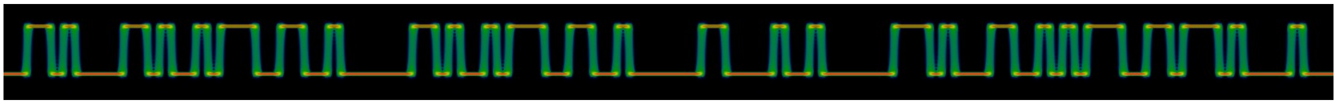
sample rate: the sample rate of the generated waveform IQ file

gaussian window percentage: beta value to indicate Gaussian window length. Must be between 0.0 an 1.0, this value determines the “spreading” of the energy between the two frequencies. Only used in GFSK waveforms

output file name: the output name of the IQ file. Make sure to include the desired file type (.fc32, c64, .iq, ect)

output file path: path where the output file is saved. Leave blank to save in the sig_gen directory.

The following screenshot shows the sample IQ file generated from the above example:



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)

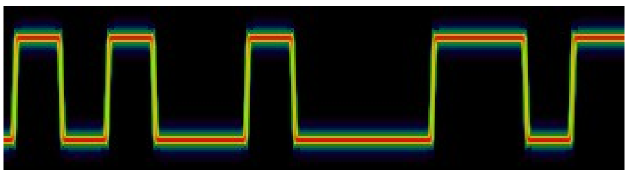


Figure: low window length (0.15 sps)

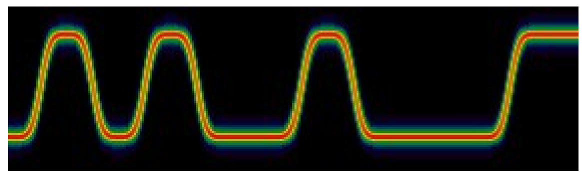


Figure: high window length (0.85 sps)

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)

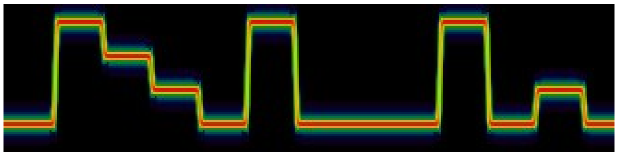


Figure: low window length (0.15 sps)

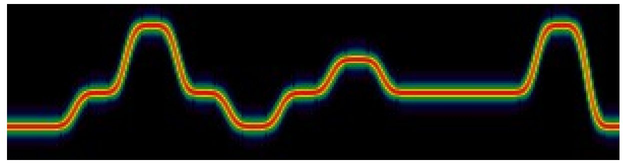


Figure: high window length (0.85 sps)

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