AECENAR
Association for Economical and Technological Cooperation in the Euro-Asian and North-African Region
www.aecenar.com

AS-COMSAT Communication System

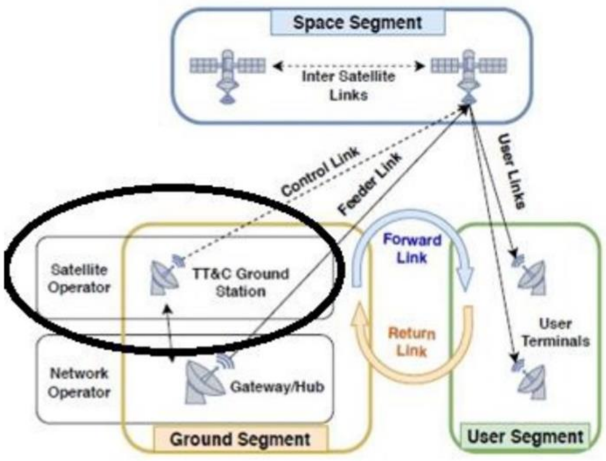
Author: Raja Murad

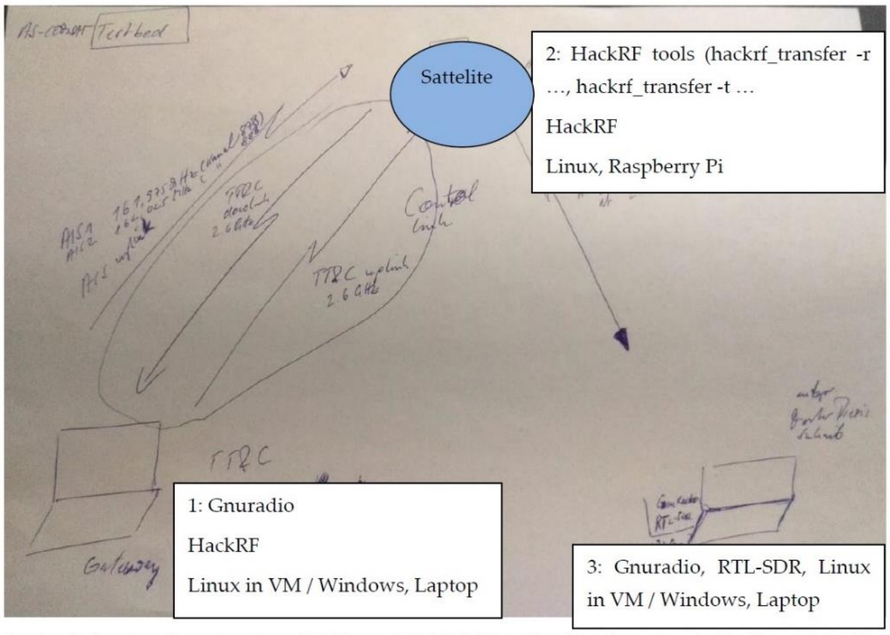
Last Update: 17.03.2022 11:21

CONTENT

1 THE PLAN

This section is a complimentary part of the AS-COMSAT project. The aim of this section is to establish a communication channel between the Cubesat satellite and a ground station. The station shall transmit the control commands for the satellite as well as receiving different information from it. The figure below shows the basic communication architecture between the Telemetry, Tracking, and Control (TT&C) ground station and the satellite system. For primary prototyping and testing, the communication frequencies in the figure below may vary and other frequencies might be used.





Payload: Sending from 1 to 2 an AIS file on 161.975 MHz – Sending from 2 to 3 this file on 2.6 GHz

Telemetry, Tracking & Control (TT&C): Sending from 1 to 2 a control command file on 2.6 GHz, sending from 2 to 1 a file with sensor information on 2.6 GHz

The basic components of this architecture are the ground station controller (A regular Laptop), a raspberry pi which is the onboard computer, and a second computer at the receiving end. The communication devices are HackRf One which offers a state-of-art Software Defined Radio (SDR) and **GNUradio** for rapid-implementing all digital signal processing.

2 SETTING UP THE EXPERIMENT REQUIREMENTS

To achieve the goal of this project, I used 2 computers that act as sending and receiving PCs respectively. Each PC has Ubuntu OS which is much simpler and robust in such applications. In this section, I will show the setup made and demonstrate the installation procedure of HackRf and GNUradio modules on Ubuntu 20.04 LTS.

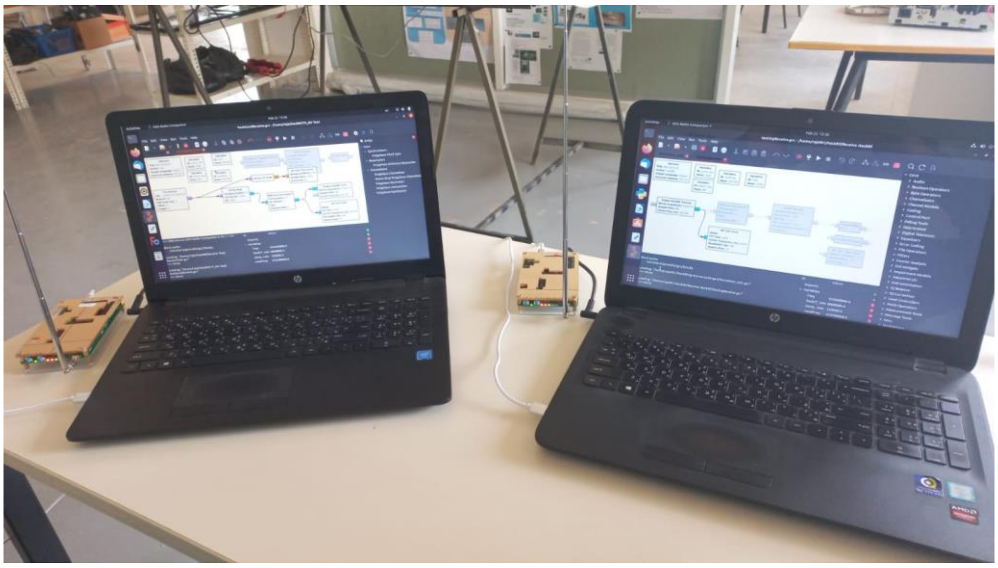


Figure 2-1. Experiment Setup. Computer A to the left and Computer B to the right

The setup, as shown in Figure 2-1 above, includes 2 PCs with the following specs:

- Computer A: HP Laptop equipped with Ubuntu 21.10 OS, 8 GB RAM, Intel Celeron CPU N3060 @1.60 GHz x 2, and a 512 GB HDD.
- Computer B: HP Laptop equipped with Ubuntu 20.04 via Virtual Machine, 8 GB RAM, Intel Core i5-6200U CPU @ 2.3 GHz 2.4 GHz, and a 200 GB HDD for the virtual machine.

The setup is used to boost the testing and make sure every device is working well before mounting to the onboard computer. The reason why Ubuntu is chosen because it is much simpler than windows to install already existing Debian packages by *sudo* commands through terminal window.

2.1 INSTALLING AND TESTING HACKRF PACKAGES

Open a terminal window and insert the following commands.

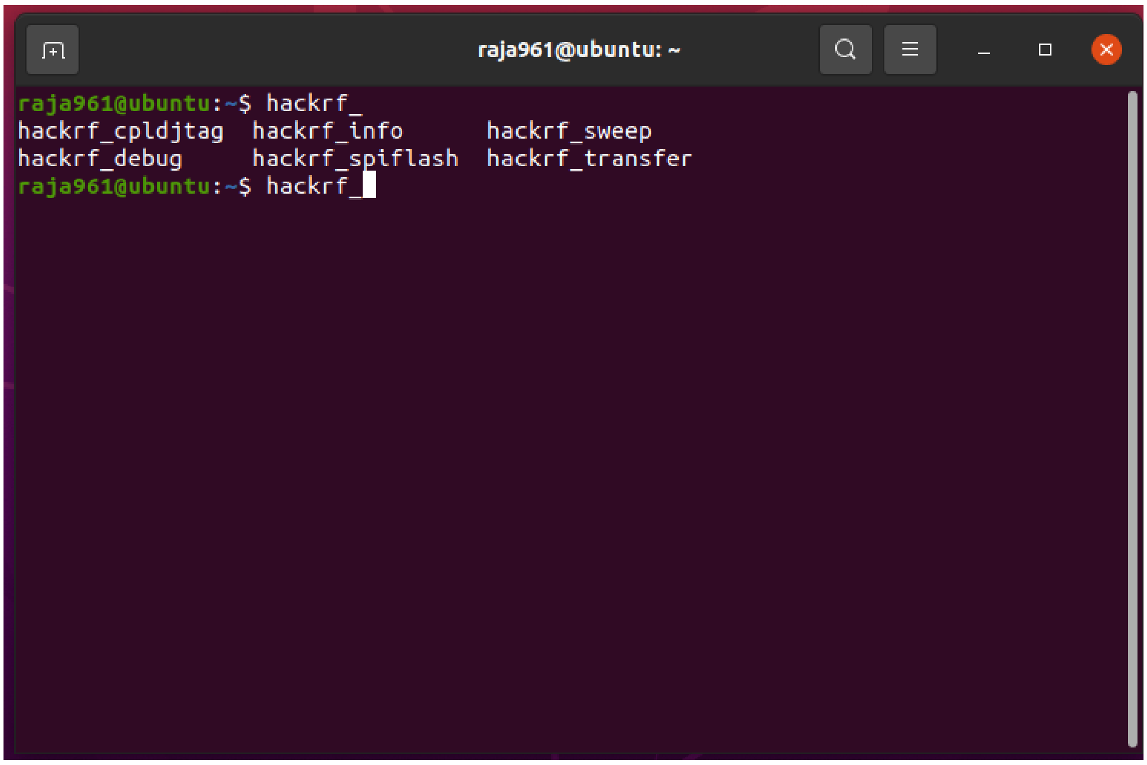
1- Run the update command to update package repositories and get latest package information.

```
sudo apt-get update -y
```

2- Run the install command with -y flag to quickly install the packages and dependencies.

```
sudo apt-get install -y hackrf
```

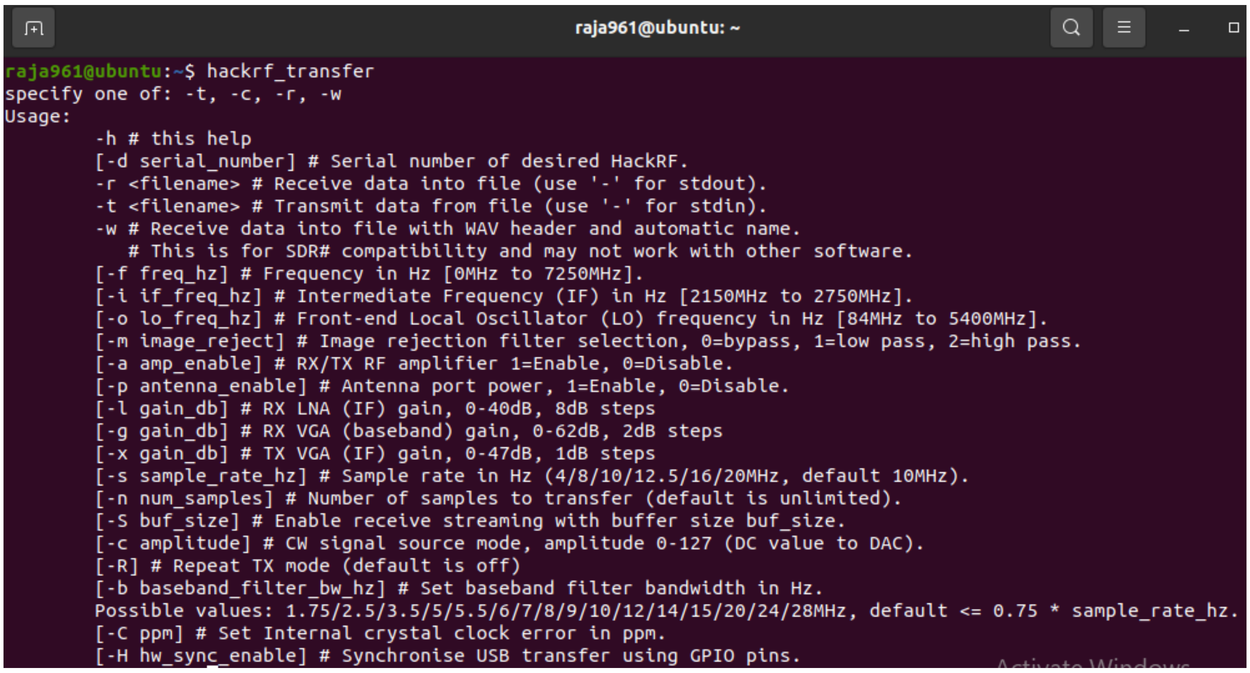
The HackRf now is installed. To check the hackrf available commands, open a terminal window and type hackrf_ and press the "Tab" button twice to list all available commands as shown below.



```
raja961@ubuntu:~$ hackrf_
hackrf_cpldjtag  hackrf_info      hackrf_sweep
hackrf_debug     hackrf_spiflash  hackrf_transfer
raja961@ubuntu:~$ hackrf_
```

Figure 2-2. HackRf available commands

The important commands for this application are "*hackrf_info*" which can be executed to check the information of the connected HackRf and "*hackrf_transfer*" which is used to send or receive IQ-type or binary-type files. It has the following documentation:



```
raja961@ubuntu:~$ hackrf_transfer
specify one of: -t, -c, -r, -w
Usage:
        -h # this help
        [-d serial_number] # Serial number of desired HackRF.
        -r <filename> # Receive data into file (use '-' for stdout).
        -t <filename> # Transmit data from file (use '-' for stdin).
        -w # Receive data into file with WAV header and automatic name.
           # This is for SDR# compatibility and may not work with other software.
        [-f freq_hz] # Frequency in Hz [0MHz to 7250MHz].
        [-i if_freq_hz] # Intermediate Frequency (IF) in Hz [2150MHz to 2750MHz].
        [-o lo_freq_hz] # Front-end Local Oscillator (LO) frequency in Hz [84MHz to 5400MHz].
        [-m image_reject] # Image rejection filter selection, 0=bypass, 1=low pass, 2=high pass.
        [-a amp_enable] # RX/TX RF amplifier 1=Enable, 0=Disable.
        [-p antenna_enable] # Antenna port power, 1=Enable, 0=Disable.
        [-l gain_db] # RX LNA (IF) gain, 0-40dB, 8dB steps
        [-g gain_db] # RX VGA (baseband) gain, 0-62dB, 2dB steps
        [-x gain_db] # TX VGA (IF) gain, 0-47dB, 1dB steps
        [-s sample_rate_hz] # Sample rate in Hz (4/8/10/12.5/16/20MHz, default 10MHz).
        [-n num_samples] # Number of samples to transfer (default is unlimited).
        [-S buf_size] # Enable receive streaming with buffer size buf_size.
        [-c amplitude] # CW signal source mode, amplitude 0-127 (DC value to DAC).
        [-R] # Repeat TX mode (default is off)
        [-b baseband_filter_bw_hz] # Set baseband filter bandwidth in Hz.
        Possible values: 1.75/2.5/3.5/5/5.5/6/7/8/9/10/12/14/15/20/24/28MHz, default <= 0.75 * sample_rate_hz.
        [-C ppm] # Set Internal crystal clock error in ppm.
        [-H hw_sync_enable] # Synchronise USB transfer using GPIO pins.
```

Figure 2-3. Hackrf_transfer command documentation

2.2 INSTALLING GNURADIO COMPANION

The GNU Radio is an open-source development toolkit that provides signal processing blocks to implement software radios. It can be used with plenty pf low to high-cost RF hardware to create software-defined radios (SDRs, or without hardware in a simulation environment. Installing GNU Radio is easy, just open a terminal window and type the following commands:

1- To access the current released version (3.10), we shall add the GNU Radio ppa and remove all existing versions:

```
$ sudo add-apt-repository ppa:gnuradio/gnuradio-releases
```

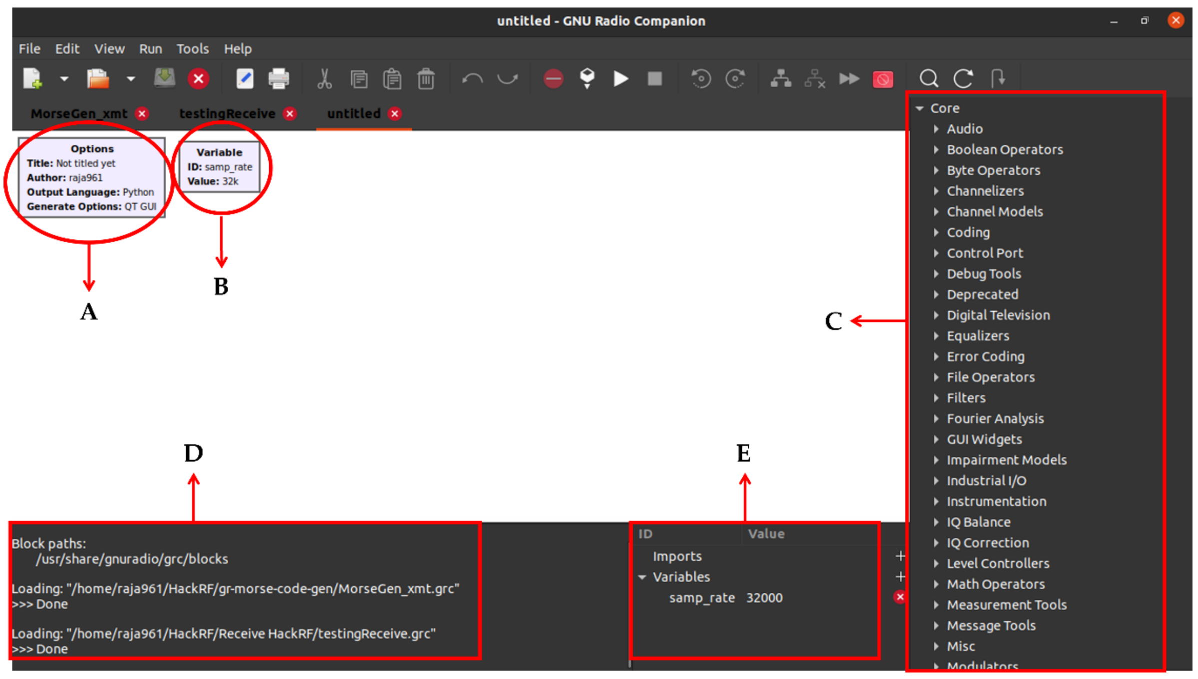
2- Update the apt sources, and install gnuradio

```
$ sudo apt-get update
$ sudo apt install gnuradio
```

3- Install python module 'packaging' using pip which may also need to be installed:

```
$ sudo apt install python3-pip
$ pip install packaging
```

Once the GNU Radio is installed, open it either by clicking on its icon in the applications menu or by typing "*gnuradio-companion*" in a terminal window. The GNU Radio will open a new script as shown below.



Options
Title: Not titled yet
Author: raja961
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

```
Block paths:
    /usr/share/gnuradio/grc/blocks

Loading: "/home/raja961/HackRF/gr-morse-code-gen/MorseGen_xmt.grc"
>>> Done

Loading: "/home/raja961/HackRF/Receive HackRF/testingReceive.grc"
>>> Done
```

```
ID              Value
  Imports
▾ Variables
    samp_rate   32000
```

Figure 2-4. GNU Radio main page

The GNU Radio's new script has the following parts:

- **A:** The option block, it sets special parameters for the flow graph. Only one option block is allowed per flow graph. The generate options controls the type of code generated. Non-graphical flow graphs should avoid using graphical sinks or graphical variable controls. In a graphical application, run can be controlled by a variable to start and stop the flowgraph at runtime. The id of this block determines the name of the generated file and the name of the class. For example, an id of my_block will generate the file my_block.py and class my_block(gr...).
- **B:** The variable block, it maps a value to a unique variable. This variable block has no graphical representation.
- **C:** Functions list, different function blocks can be selected from this menu by dragging the desired block to the workspace.
- **D:** Output window, here, you can track the flowgraph's status and used for debugging.
- **E:** The variables window, this window shows all the variables placed inside the workspace in which you can track them easily and change their values.
-

2.3 Installing GQRX

GQRX is a SDR receiver application. It supports many SDR hardware including HackRf. It will be used to check the presence of signals in a selected frequency band. To install it, open a terminal window and type the following commands:

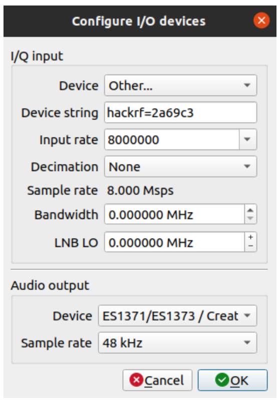
1- Run update command to update package repositories and get latest package information

```
$ sudo apt-get update -y
```

2- Run the install command with -y flag to quickly install the packages and dependencies.

```
$ sudo apt-get install -y gqrx-sdr
```

Once it is installed, open a terminal window and type *"gqrx"* to open it. The software will ask you to configure the hardware device as in the figure below:


Configure I/O devices

I/Q input

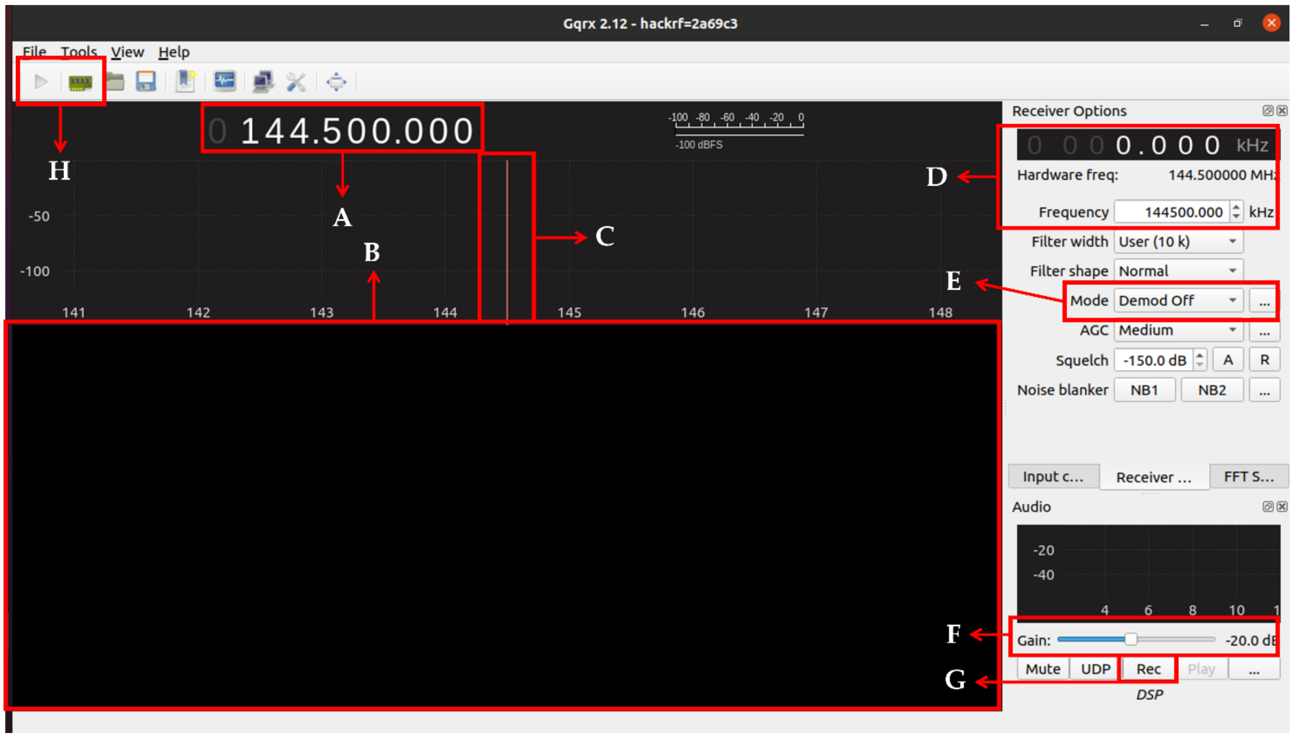
Device: Other...
Device string: hackrf=2a69c3
Input rate: 8000000
Decimation: None
Sample rate: 8.000 Msps
Bandwidth: 0.000000 MHz
LNB LO: 0.000000 MHz

Audio output

Device: ES1371/ES1373 / Creat
Sample rate: 48 kHz

Figure 2-5. GQRX Configuration menu

In the device tab, choose the HackRf device (make sure the device string tab updates with the selected HackRf device). The input rate or in other words the sample rate must be chosen between 4 and 8 MHz to not heavily-load the CPU and good results can be achieved. In the Device tab, choose the speaker hardware (built-in hardware) and press OK. If everything was configured well, the following main menu will be displayed:


Gqrx 2.12 - hackrf=2a69c3

Receiver Options

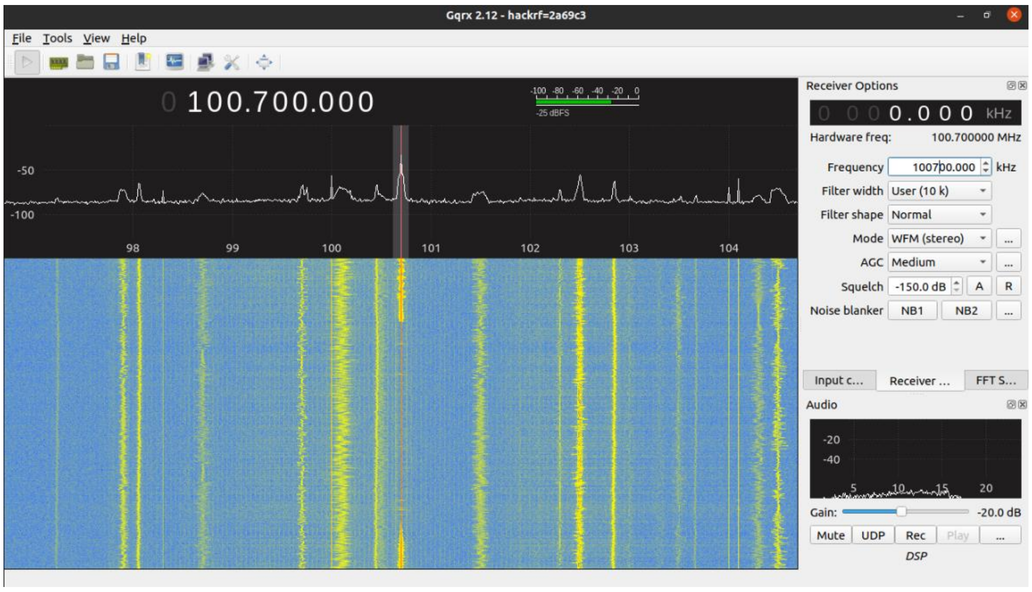
Hardware freq: 144.500000 MHz
Frequency: 144500.000 kHz
Filter width: User (10 k)
Filter shape: Normal
Mode: Demod Off
AGC: Medium
Squelch: -150.0 dB
Gain: -20.0 dB

Figure 2-6. GQRX main menu

- **A:** The selected baseband frequency in MHz.
- **B:** Waterfall display of the frequency bandwidth. This is a good representation of where are the strong signals in the frequency spectrum,
- **C:** Tuning knob. This can be used to change the baseband frequency.
- **D:** This menu here can change the frequency in different decimals quickly (10s, 100s, or 1000s of KHz).
- **E:** The output mode. If you are receiving a voice signal, choose WFM(stereo) option to activate the internal amplifiers of your PC speakers.

- **F:** Change the receiving amplifier power.
- **G:** This button can be used to record the signal and save it in IQ or binary format.
- **H:** The button to the right can be used to edit the hardware configuration and the button to the left starts receiving. Once everything is configured correctly, press the start button.

Once the start button is pressed, the GUI's menu becomes:


Gqrx 2.12 - hackrf=2a69c3

0 100.700.000

Receiver Options

Hardware freq: 100.700000 MHz

Frequency 100700.000 kHz

Filter width User (10 k)

Filter shape Normal

Mode WFM (stereo)

AGC Medium

Squelch -150.0 dB

Gain: -20.0 dB

Figure 2-7. GQRX receiving FM radio station at 100.7 MHz

As can be seen from Figure 2-7 above, the spectrum spans about 8 MHz and the central frequency is tuned at 100.7 MHz. The amplitude of pulse represents the power of the channel in DB. Using GQRX makes it easier to ensure the reception of a specific frequency.

3 EXPERIMENTS

This section handles all the experiments done (including failed and succeeded ones) in order to achieve a well-functioning communication system between 2 HackRfs. The main aim of the experiments is to be able to send a custom text from one source to a receiver.

3.1 RECEIVING/TRANSMITTING RAW IQ FILES

At first, I tried to receive raw IQ files through hackrf by using terminal commands. Figure 2-3 above shows the commands that can be used to transmit or receive available signals at the determined frequency. I started the experiment by receiving click buttons from a Proton Persona key which transmits data information with a carrier frequency of 433.92 MHz and then tried to re-transmit them again to the other HackRf. I used the following terminal command to receive key signal:

```
$ hackrf_transfer -r fileReceived.iq -f 433920000 -a 1 -p 1 -s 8000000
```

The command above will receive an *"iq"* file name *"fileReceived"* and save it in the current repository. The center frequency is 433.92 MHz and I ensured that both the antenna and power amplification are enabled. The hackRf's sampling frequency was set to 8 MHz.

I tried pushing the key button 3 or 4 times to make sure it was clearly detected and saved. I kept the receiving process running for 8 seconds. Then I tried to transmit the file using the first hackRf that I received by entering the following command:

```
$ hackrf_transfer -t fileReceived.iq -f 433920000 -a 1 -p 1 -s 8000000 -x 47
```

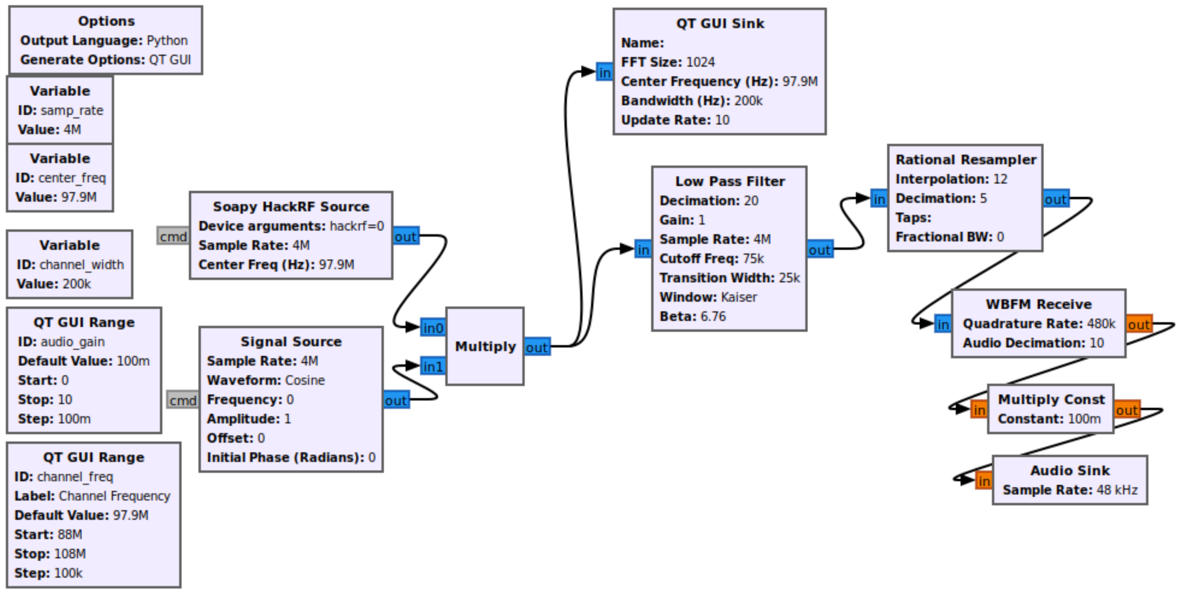
The only difference is that I used the letter "t" for transmission instead of "r" for receiving and I enabled the maximum transmission amplification of 47 db. On the other hackRf, I reused the receiving command to store the data coming from the first hackRf:

```
$ hackrf_transfer -r fileReceived.iq -f 433920000 -a 1 -p 1 -s 8000000
```

And as expected, I received the exact same signal coming from car's key at the second hackRf.

3.2 Receiving Basic FM Stations Through GNU Radio

After making sure the HackRfs are operating correctly and can send/receive data, I made a simple "hello world" SDR application that detects FM radio stations by using GNU Radio Companion. The below figure shows the receiving flow graph.



Options
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 4M

Variable
ID: center_freq
Value: 97.9M

Variable
ID: channel_width
Value: 200k

QT GUI Range
ID: audio_gain
Default Value: 100m
Start: 0
Stop: 10
Step: 100m

QT GUI Range
ID: channel_freq
Label: Channel Frequency
Default Value: 97.9M
Start: 88M
Stop: 108M
Step: 100k

Soapy HackRF Source
Device arguments: hackrf=0
Sample Rate: 4M
Center Freq (Hz): 97.9M

Signal Source
Sample Rate: 4M
Waveform: Cosine
Frequency: 0
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Multiply

QT GUI Sink
Name:
FFT Size: 1024
Center Frequency (Hz): 97.9M
Bandwidth (Hz): 200k
Update Rate: 10

Low Pass Filter
Decimation: 20
Gain: 1
Sample Rate: 4M
Cutoff Freq: 75k
Transition Width: 25k
Window: Kaiser
Beta: 6.76

Rational Resampler
Interpolation: 12
Decimation: 5
Taps:
Fractional BW: 0

WBFM Receive
Quadrature Rate: 480k
Audio Decimation: 10

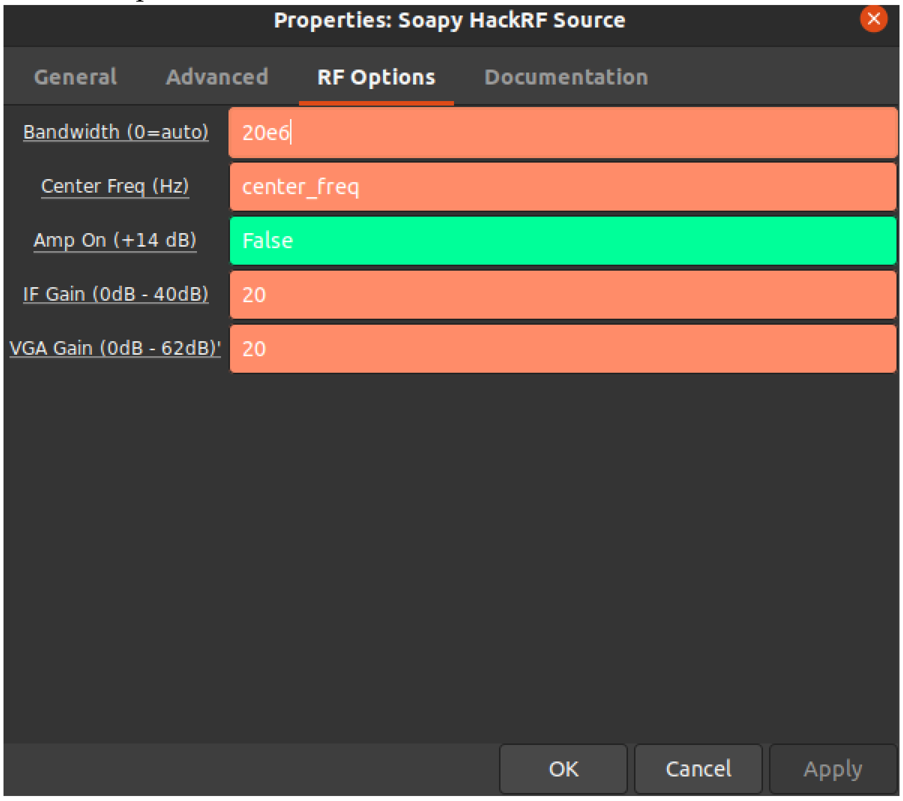
Multiply Const
Constant: 100m

Audio Sink
Sample Rate: 48 kHz

Figure 3-1. Flow graph of receiving FM stations via GNU Radio

The flow graph has embedded python code for each block. There are mainly 2 sensitive parameters that must be configured precisely in order for the flow graph to operate smoothly; the receiving central frequency and the sample rate. The central frequency can be tuned until a FM station is detected. However, sample rate should be treated differently. The output of the FM station is sound, hence, a 48 KHz sampling frequency should be present at the output to drive the speakers efficiently. But, the sampling rate of the HackRf (which is the input) should be something between 4 and 8 MHz. Thus, the sample rate must be narrowed down to match the speaker's rate at the output. The flow graph, with each block's sample rate, works as follow:

1- **Soapy HackRF Source:** This block receives the signal detected by the hackRf's antenna at the specified center frequency and amplifies by a factor of *"IF Gain"* and *"VGA Gain"* determined. The bandwidth could be specified so that it regulates for some frequency errors. The sample rate of the HackRf was set to 4 MHz.



Bandwidth (0=auto)	20e6
Center Freq (Hz)	center_freq
Amp On (+14 dB)	False
IF Gain (0dB - 40dB)	20
VGA Gain (0dB - 62dB)'	20

Figure 3-2. Soapy HackRF Configuration

2- **Signal Source:** this block is used to generate a cosine signal that is capable of shifting the central frequency when multiplied with the received signal. The frequency of this block is determined by how far is the center frequency from the selected (tuned) frequency. Hence, it is selected as (center frequency – selected frequency). This block does not affect the sample rate.

3- **Low Pass Filter:** From its name, a low pass filter allows the passage of frequencies below its cutoff frequency and prevents the passage of frequencies above it. It is used to eliminate the carrier frequency and keeps the original message. It affects the sample rate thus, a decimation of 20 is set so the sample rate at the block's output will be equal to the sample rate of its input divided by 20 (4 MHz / 20 = **200 KHz**).

4- **Rational Resampler:** This block affects the sample rate by multiplying the input's rate by the *"interpolation"* and divides it by *"decimation"* values. Thus, the sample rate at the output of this block is (200 KHz x 12) / 5 = **480 KHz**.

5- **WBFM Receive:** This block does the demodulation of FM radio stations. The sample rate of the input to this block must be 480 KHz and its output sample rate is divided by the *"decimation"* value. Thus, the output sample rate of this block is (480 KHz / 10 = **48 KHz**).
6- **Multiply Constant:** This block multiplies the input's magnitude by the value specified. It is used to control the volume.
7- **Audio Sink:** This block outputs the input signal directly to the speakers.
8- **QT GUI Sink:** This is a GUI that shows the received signals in frequency and time domain. The GUI page when running the above flow graph appears as the below figure. The "Channel Frequency" and "audio_gain" are 2 variables set by the "QT GUI Range" block where the min and max values of these variables can be adjusted by configuring the QT GUI Range block.

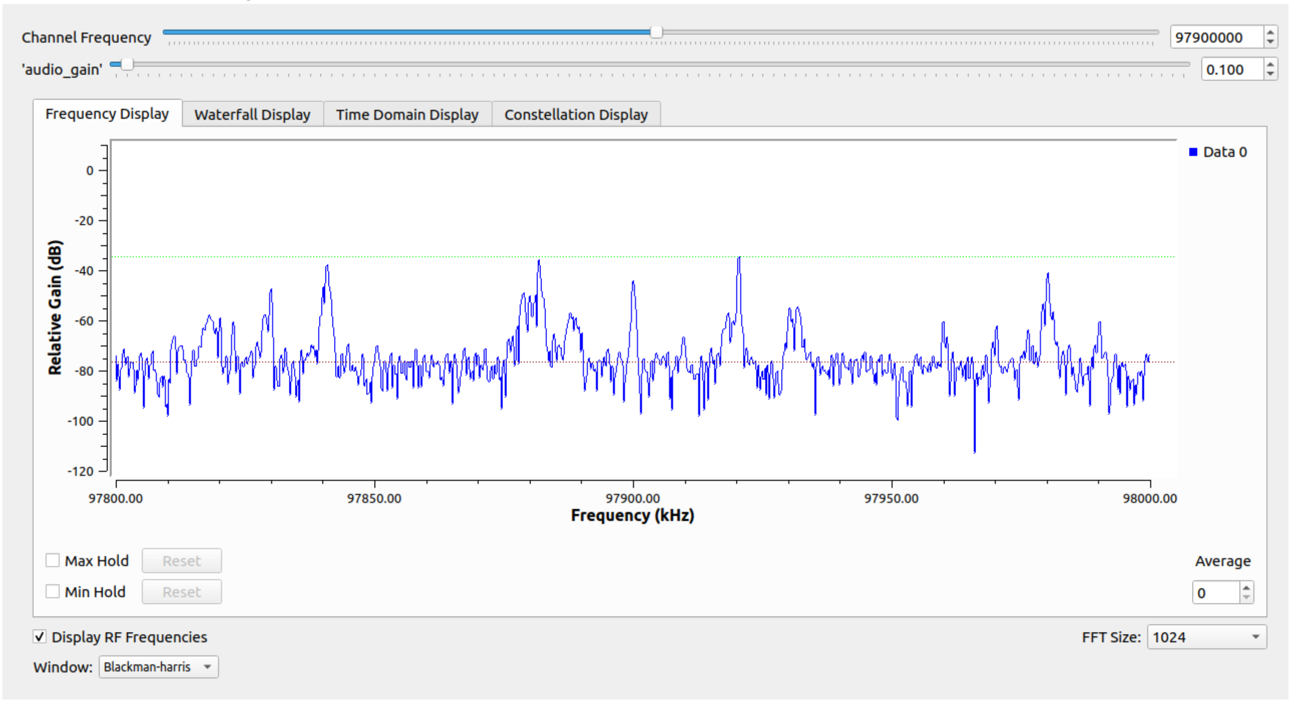


Figure 3-3. QT GUI Sink

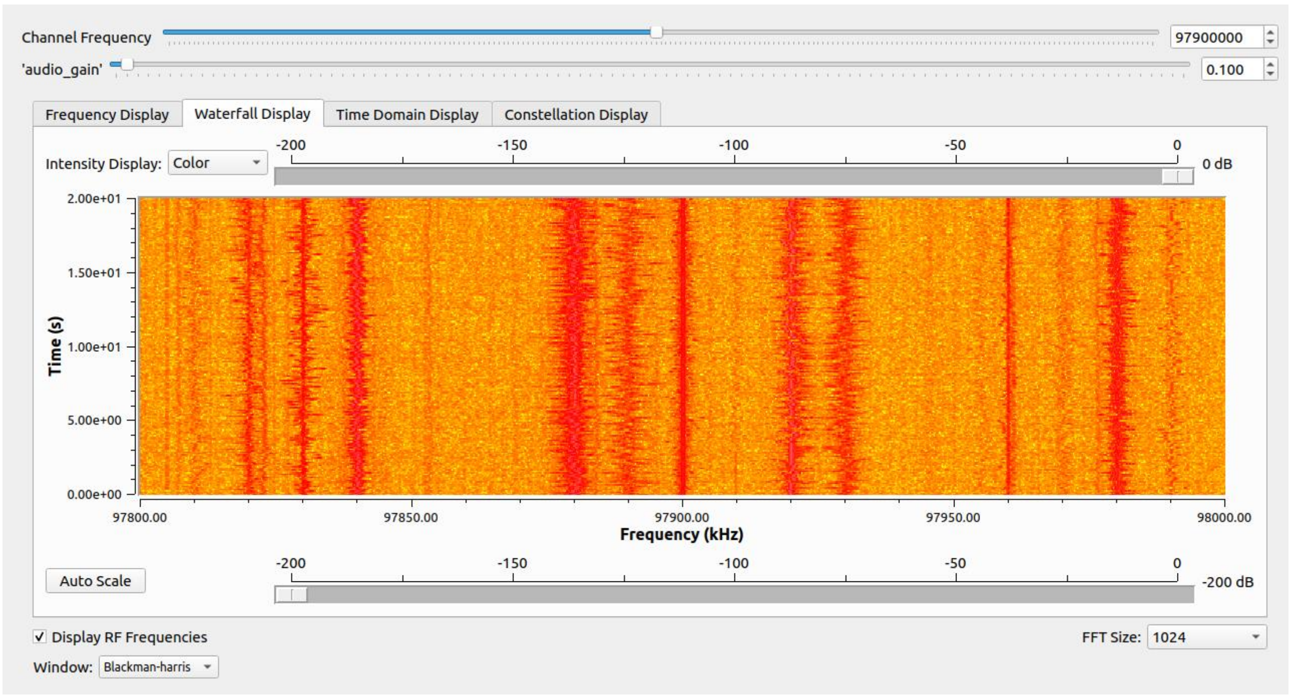


Figure 3-4. QT GUI Sink Waterfall Representation

3.3 FSK Transmission

In this experiment, I connected 2 HackRfs to a 1 computer in order to form a communication loopback system where 1 hackrf transmits and the other receives on the same PC. The modulation scheme I am trying to send my data over is the Frequency Shift Keying (FSK) which transforms the digital message into a varying frequency (high frequency for 1 and low frequency for 0). It is one of the most useful transmission techniques requiring low receiver complexity.

Binary FSK (BFSK) is the simplest form of FSK where the two bits 0 and 1 correspond to two distinct carrier frequencies F_0 and F_1 to be sent over the air. The bits can be translated into symbols through the relations:

$$0 \rightarrow -1$$

$$1 \rightarrow 1$$

So, the frequencies can be written as:

$$F_i = F_c + (-1)^{i+1} = F_C \pm \Delta F$$

Where F_c is the nominal carrier frequency and ΔF is the frequency deviation from this carrier. Thus, the signal waveform can be written as:

$$\mathrm{s(t) = Acos(2\pi Fit + \phi) = Acos[2\pi(Fc \pm \Delta F)t + \phi]}$$

where $0 \leq t \leq T_b$ and ϕ is an arbitrary phase. The message signal is taken to be a random stream of bits. Figure 3-5 below displays a BFSK waveform for a random stream of data at a rate of $Rb=1/T_b$. Note that we are not distinguishing between a bit period and a symbol period because both are the same for a binary modulation technique.

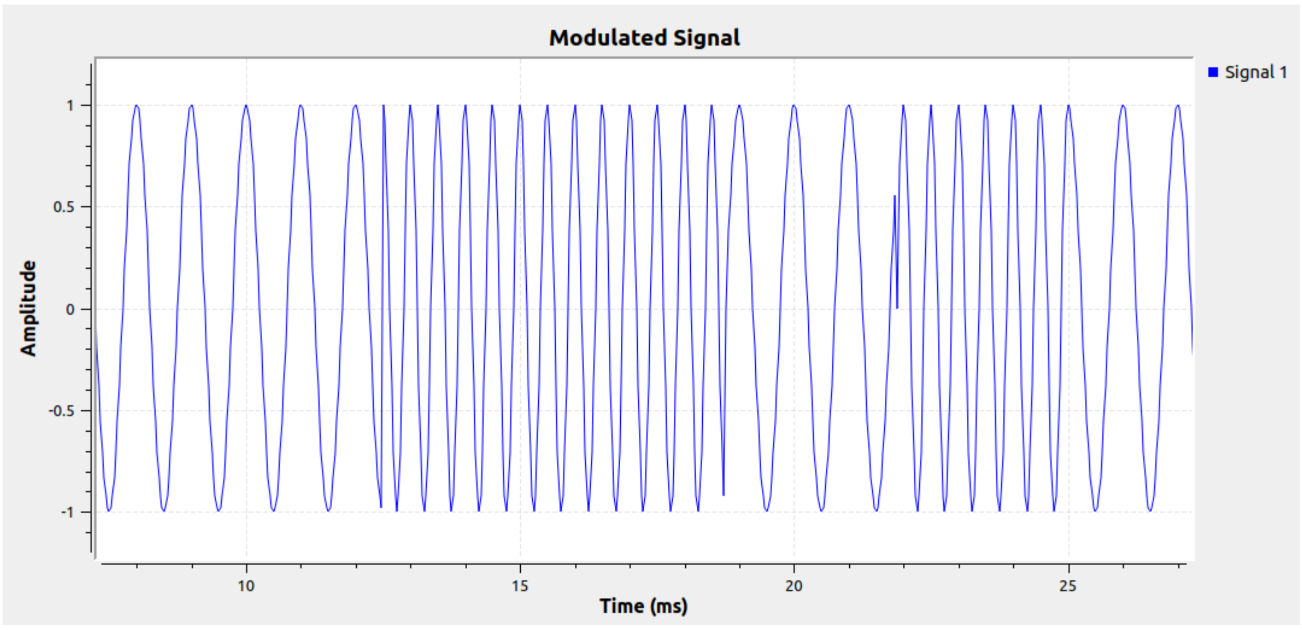
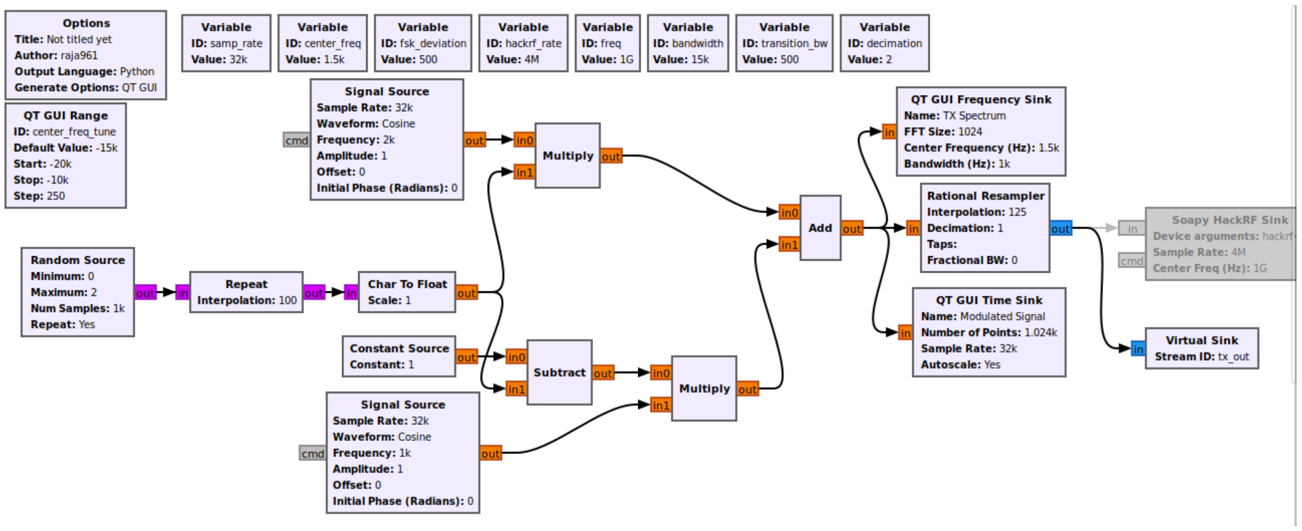


Figure 3-5. FSK Modulated Signal

As can be seen, for a digital “1”, the frequency of the modulated signal goes higher and when the message bit is “0”, the frequency goes lower.

The very basic scheme of an FSK modulation can be generated using the flowgraph below. An FSK signal is generated with a center frequency of 1.5 KHz and a frequency deviation of 500 Hz. This result in 2 carrier frequencies, one at 1 KHz (for 0), and one at 2 KHz (for 1).



Options
Title: Not titled yet
Author: raja961
Output Language: Python
Generate Options: QT GUI

Variable **ID:** samp_rate **Value:** 32k
Variable **ID:** center_freq **Value:** 1.5k
Variable **ID:** fsk_deviation **Value:** 500
Variable **ID:** hackrf_rate **Value:** 4M
Variable **ID:** freq **Value:** 1G
Variable **ID:** bandwidth **Value:** 15k
Variable **ID:** transition_bw **Value:** 500
Variable **ID:** decimation **Value:** 2

QT GUI Range
ID: center_freq_tune
Default Value: -15k
Start: -20k
Stop: -10k
Step: 250

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 2k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Random Source
Minimum: 0
Maximum: 2
Num Samples: 1k
Repeat: Yes

Repeat
Interpolation: 100

Char To Float
Scale: 1

Constant Source
Constant: 1

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

QT GUI Frequency Sink
Name: TX Spectrum
FFT Size: 1024
Center Frequency (Hz): 1.5k
Bandwidth (Hz): 1k

Rational Resampler
Interpolation: 125
Decimation: 1
Taps:
Fractional BW: 0

Soapy HackRF Sink
Device arguments: hackrf
Sample Rate: 4M
Center Freq (Hz): 1G

QT GUI Time Sink
Name: Modulated Signal
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: Yes

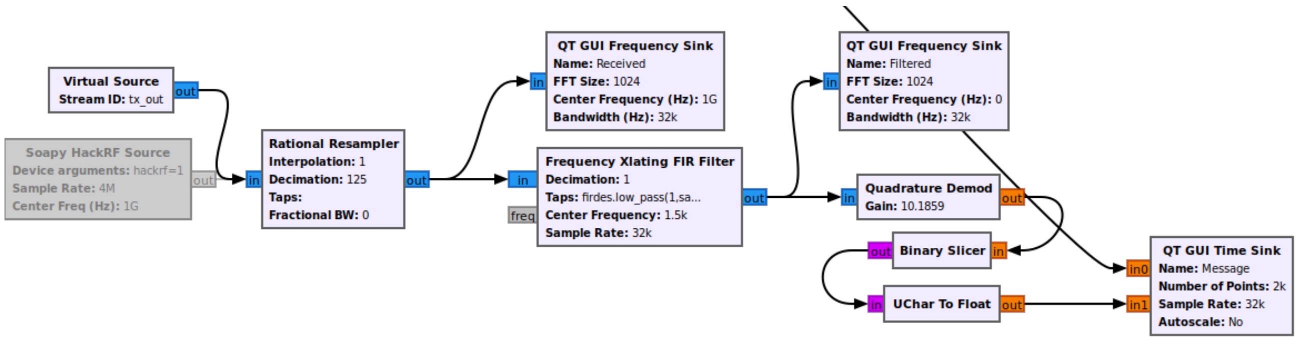
Virtual Sink
Stream ID: tx_out

Figure 3-6. FSK Modulation Flowgraph

The above flowgraph works as follows:

- Multiply the bits, e.g., 0,1,1,0,0,1,…, with the higher frequency wave at 2 kHz that produces a 2 kHz wave for 1 bits and a blank space of zeros for 0 bits.
- Multiply one minus the bits, e.g., 1- (0,1,1,0,0,1,…), with the lower frequency wave at 1 kHz that produces a 1 kHz wave for 0 bits and a blank space of zeros for 1 bits.
- Add the two waves together thus generating a BFSK waveform.

For the receiving part, a commonly used technique in the gnuradio is the quadrature demodulation block. The below flowgraph shows the receiver diagram.


Virtual Source
Stream ID: tx_out

Soapy HackRF Source
Device arguments: hackrf=1
Sample Rate: 4M
Center Freq (Hz): 1G

Rational Resampler
Interpolation: 1
Decimation: 125
Taps:
Fractional BW: 0

QT GUI Frequency Sink
Name: Received
FFT Size: 1024
Center Frequency (Hz): 1G
Bandwidth (Hz): 32k

Frequency Xlating FIR Filter
Decimation: 1
Taps: firdes.low_pass(1,sa...
Center Frequency: 1.5k
Sample Rate: 32k

QT GUI Frequency Sink
Name: Filtered
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 32k

Quadrature Demod
Gain: 10.1859

Binary Slicer

UChar To Float

QT GUI Time Sink
Name: Message
Number of Points: 2k
Sample Rate: 32k
Autoscale: No

Figure 3-7. FSK Demodulation Flowgraph

Such a structure can be used to demodulate several frequency modulation schemes such as FM, FSK and GMSK. The input to the block is the complex baseband waveform. Within the block, a product of the one-sample delayed input and the conjugate original signal is computed, the output of which is a complex number.

A "Frequency Xlating FIR Filter" with a center frequency of 1.5 KHz is employed. According to its documentation, this block performs a frequency translation on the signal, as well as down-samples the signal by running a decimating FIR filter on it. This operation places the modulated signal at baseband and hence the two possible frequencies are now located at ±500 Hz. This is shown by two impulses at ±500 Hz below that is the output of the flowgraph described above.

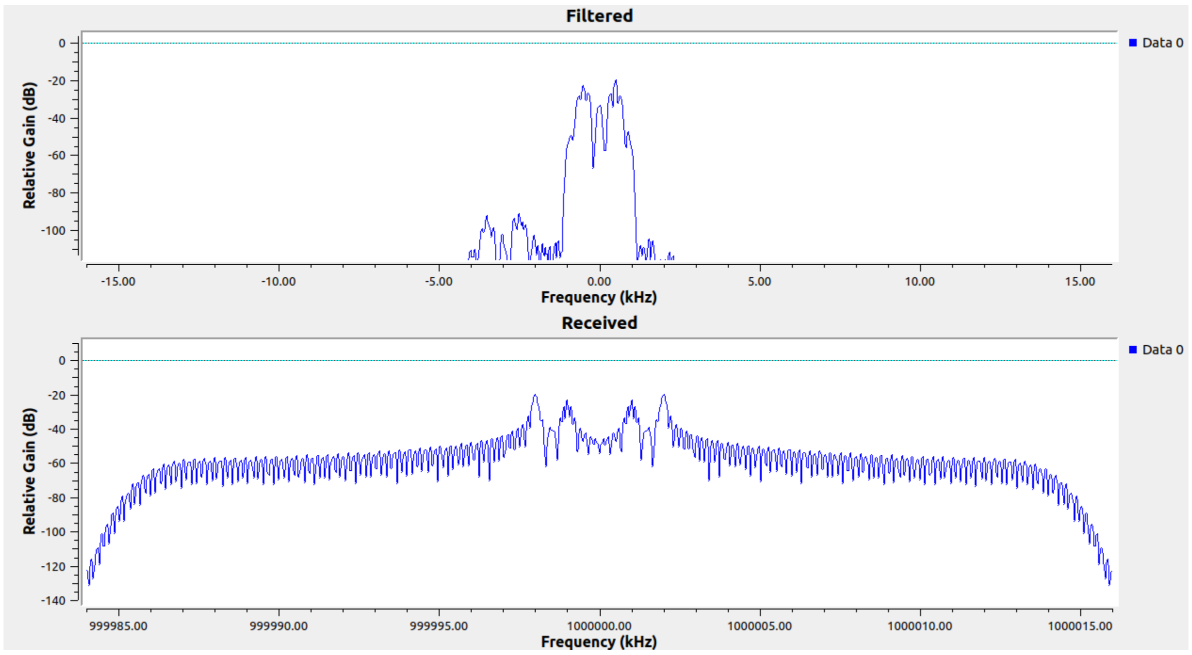


Figure 3-8. The transmitted signal and the filtered one (after the FIR filter)

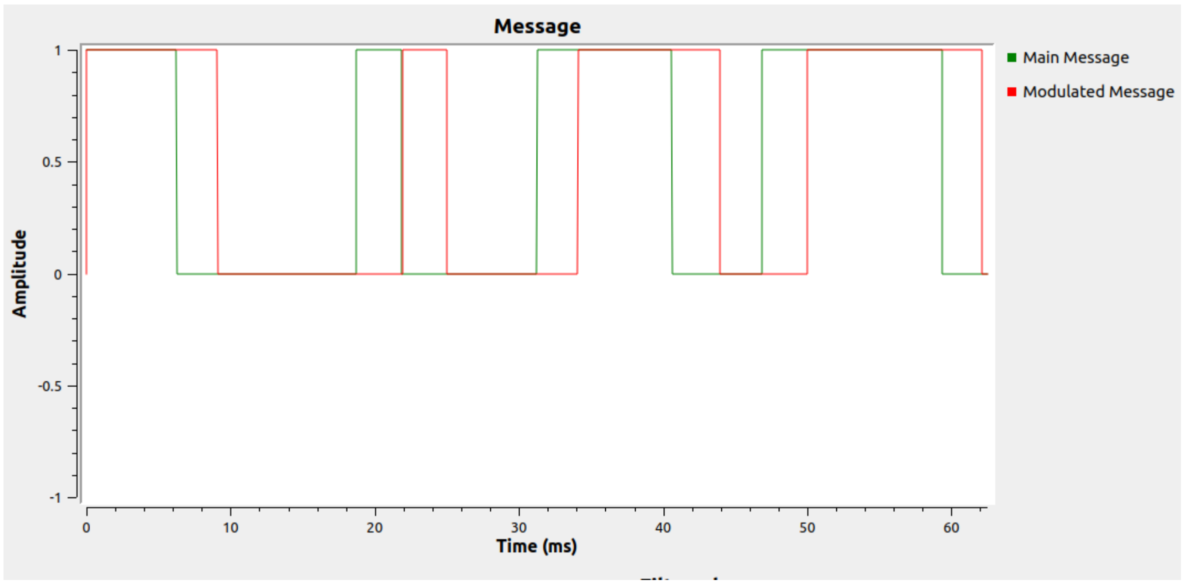


Figure 3-9. Main Message (Green) and the demodulated Message (Red) at the reception

The command firdes.low_pass(1.0, samp_rate, 900,300) uses a lowpass filter with unity gain, a sample rate of 32 kHz, a cutoff frequency of 900 Hz and a transition bandwidth of 300 Hz. According to **GNU Radio documentation**, the cutoff frequency is meant to be at the center of transition band in firdes.low_pass function. This implies that the edge of passband lies at 900-300/2=750 Hz, well beyond the impulse location of 500 Hz.

Simulating the FSK modulation gave satisfactory results. However, when coming to real hardware, the blocks "Soapy HackRf sink" and "Soapy HackRf source" block have been enabled. The received signal along with the filtered signal (after the Xlating FIR filter) are shown below.

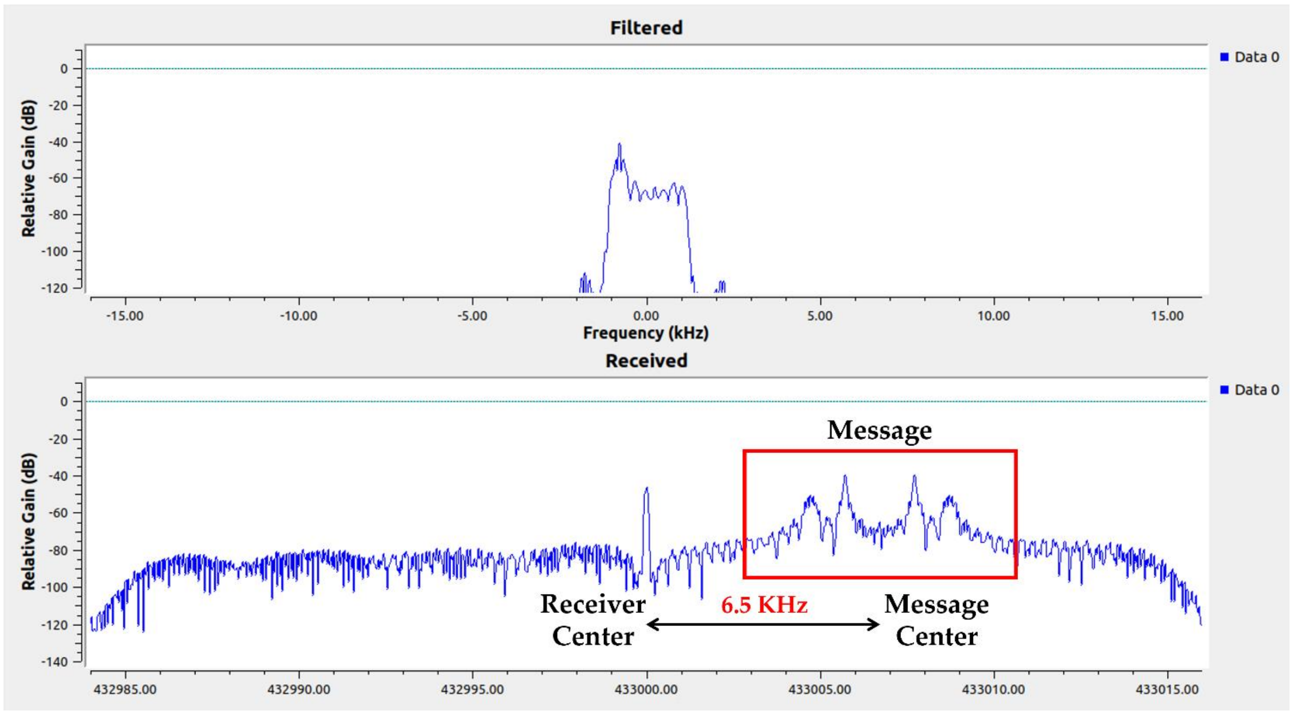


Figure 3-10 Received Signal and filter signal

Here, the XIR filter center frequency, which will shift and filter the entire spectrum, is chosen based on the frequency deviation between the receiver (hackRf transmission frequency which is chosen 433 MHz) and the message center which is located 6.5 KHz away from the center. The filter could almost perfectly filter out everything but the message but message distortions appeared as in the figure below.

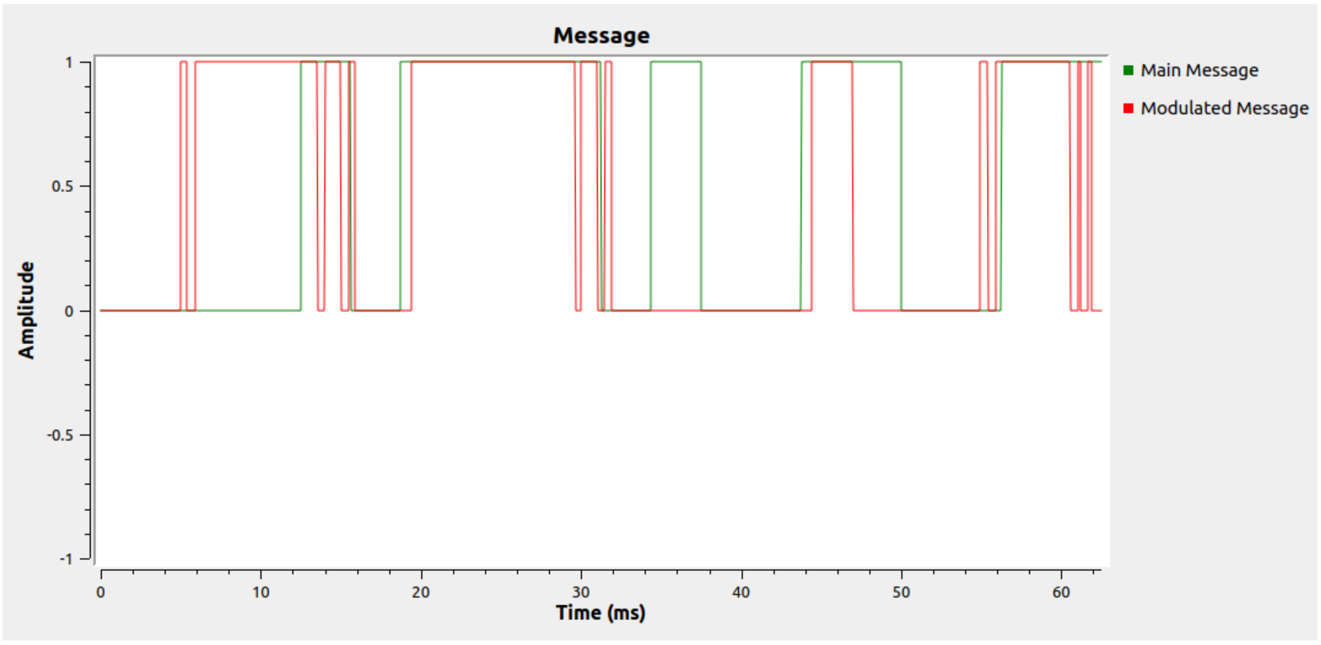
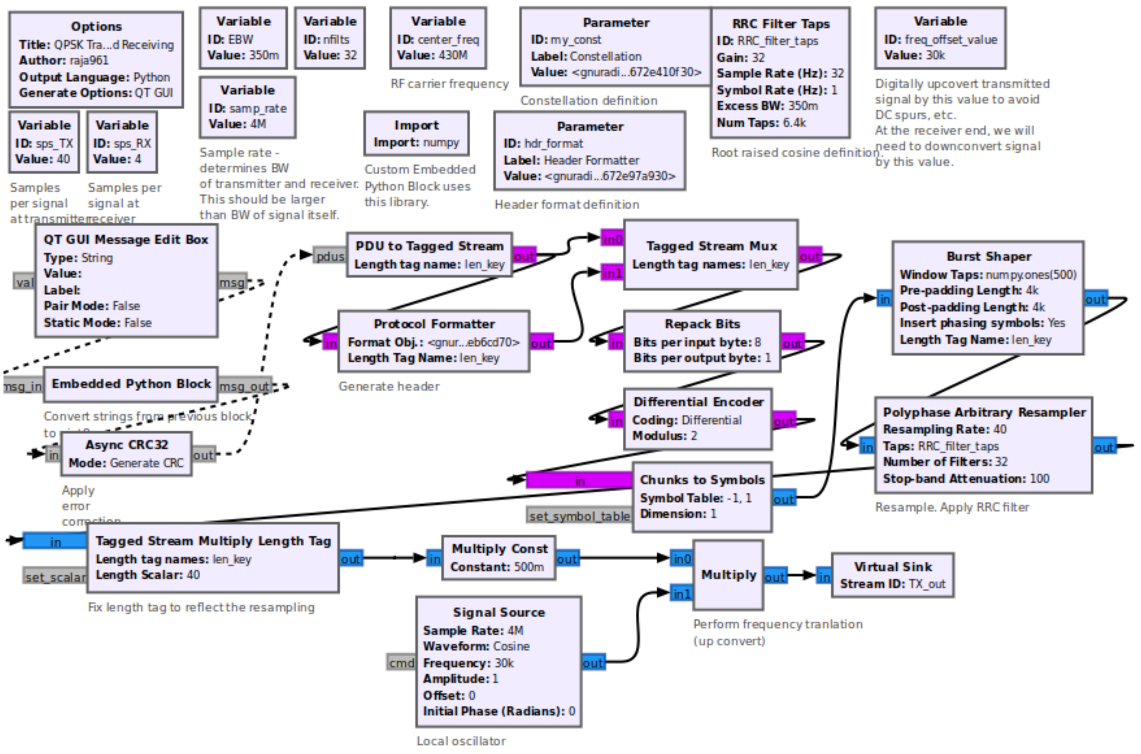


Figure 3-11. Main message and the received message in real hardware application

The XIR filter requires a bit more tuning. The sample frequency also plays a major role in message formation. Increasing or decreasing the sample rate significantly will lead to message formation errors. I found that for FSK modulation, 32 KHz sample rate up-sampled to 4 MHz (for the hackrf) is the best choice.

3.4 QPSK Modulation Method

Transmitter:


Options
Title: QPSK Tra...d Receiving
Author: raja961
Output Language: Python
Generate Options: QT GUI

Variable
ID: EBW
Value: 350m

Variable
ID: nfilts
Value: 32

Variable
ID: center_freq
Value: 430M
RF carrier frequency

Parameter
ID: my_const
Label: Constellation
Value: <gnuradi...672e410f30>
Constellation definition

RRC Filter Taps
ID: RRC_filter_taps
Gain: 32
Sample Rate (Hz): 32
Symbol Rate (Hz): 1
Excess BW: 350m
Num Taps: 6.4k
Root raised cosine definition

Variable
ID: freq_offset_value
Value: 30k
Digitally upconvert transmitted signal by this value to avoid DC spurs, etc.
At the receiver end, we will need to downconvert signal by this value.

Variable
ID: sps_TX
Value: 40
Samples per signal at transmitter

Variable
ID: sps_RX
Value: 4
Samples per signal at receiver

Variable
ID: samp_rate
Value: 4M
Sample rate - determines BW of transmitter and receiver. This should be larger than BW of signal itself.

Import
Import: numpy
Custom Embedded Python Block uses this library.

Parameter
ID: hdr_format
Label: Header Formatter
Value: <gnuradi...672e97a930>
Header format definition

QT GUI Message Edit Box
Type: String
Value:
Label:
Pair Mode: False
Static Mode: False

PDU to Tagged Stream
Length tag name: len_key

Tagged Stream Mux
Length tag names: len_key

Burst Shaper
Window Taps: numpy.ones(500)
Pre-padding Length: 4k
Post-padding Length: 4k
Insert phasing symbols: Yes
Length Tag Name: len_key

Protocol Formatter
Format Obj.: <gnur...eb6cd70>
Length Tag Name: len_key
Generate header

Repack Bits
Bits per input byte: 8
Bits per output byte: 1

Embedded Python Block
Convert strings from previous block to uint8

Differential Encoder
Coding: Differential
Modulus: 2

Polyphase Arbitrary Resampler
Resampling Rate: 40
Taps: RRC_filter_taps
Number of Filters: 32
Stop-band Attenuation: 100
Resample. Apply RRC filter

Async CRC32
Mode: Generate CRC
Apply error correction

Chunks to Symbols
Symbol Table: -1, 1
Dimension: 1

Tagged Stream Multiply Length Tag
Length tag names: len_key
Length Scalar: 40
Fix length tag to reflect the resampling

Multiply Const
Constant: 500m

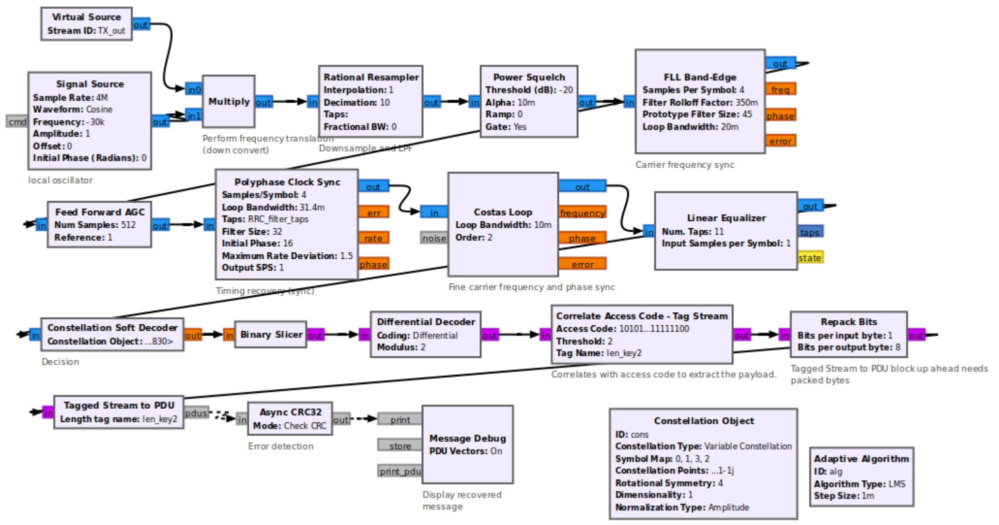
Multiply
Perform frequency tranlation (up convert)

Virtual Sink
Stream ID: TX_out

Signal Source
Sample Rate: 4M
Waveform: Cosine
Frequency: 30k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0
Local oscillator

Figure 3-12. QPSK Transmitter Flowgraph

Receiver:


Virtual Source
Stream ID: TX_out

Signal Source
Sample Rate: 4M
Waveform: Cosine
Frequency: -30k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0
local oscillator

Multiply
Perform frequency translation (down convert)

Rational Resampler
Interpolation: 1
Decimation: 10
Taps:
Fractional BW: 0
Downsample and LPF

Power Squelch
Threshold (dB): -20
Alpha: 10m
Ramp: 0
Gate: Yes

FLL Band-Edge
Samples Per Symbol: 4
Filter Rolloff Factor: 350m
Prototype Filter Size: 45
Loop Bandwidth: 20m
Carrier frequency sync

Feed Forward AGC
Num Samples: 512
Reference: 1

Polyphase Clock Sync
Samples/Symbol: 4
Loop Bandwidth: 31.4m
Taps: RRC_filter_taps
Filter Size: 32
Initial Phase: 16
Maximum Rate Deviation: 1.5
Output SPS: 1
Timing recovery (sync)

Costas Loop
Loop Bandwidth: 10m
Order: 2
Fine carrier frequency and phase sync

Linear Equalizer
Num. Taps: 11
Input Samples per Symbol: 1

Constellation Soft Decoder
Constellation Object: ...830>
Decision

Binary Slicer

Differential Decoder
Coding: Differential
Modulus: 2

Correlate Access Code - Tag Stream
Access Code: 10101...11111100
Threshold: 2
Tag Name: len_key2
Correlates with access code to extract the payload.

Repack Bits
Bits per input byte: 1
Bits per output byte: 8
Tagged Stream to PDU block up ahead needs packed bytes

Tagged Stream to PDU
Length tag name: len_key2

Async CRC32
Mode: Check CRC
Error detection

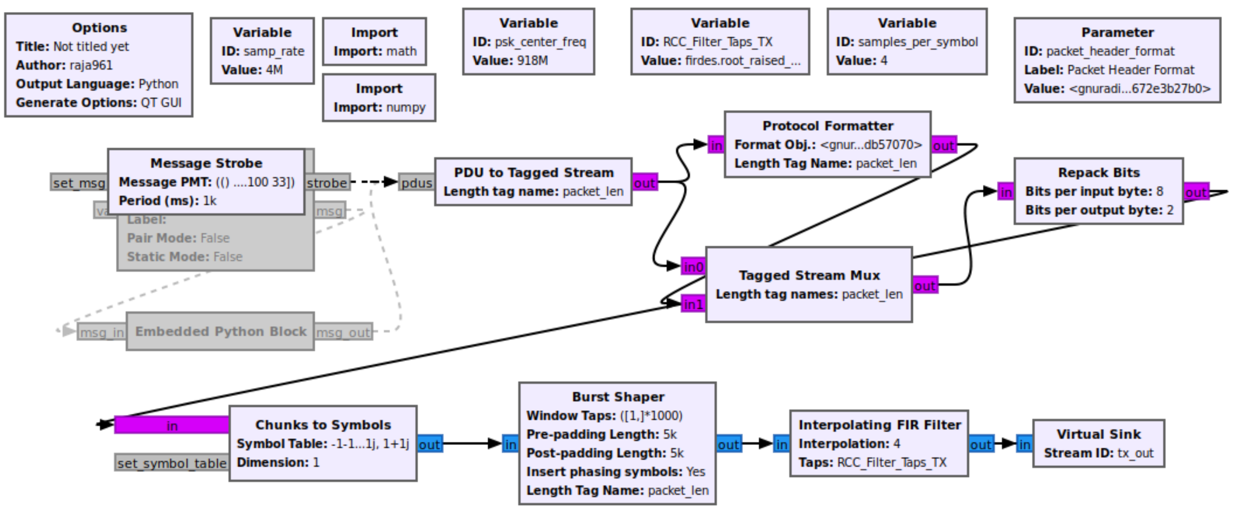
Message Debug
PDU Vectors: On
Display recovered message

Constellation Object
ID: cons
Constellation Type: Variable Constellation
Symbol Map: 0, 1, 3, 2
Constellation Points: ...1-1j
Rotational Symmetry: 4
Dimensionality: 1
Normalization Type: Amplitude

Adaptive Algorithm
ID: alg
Algorithm Type: LMS
Step Size: 1m

Figure 3-13. QPSK Receiver

This resulted in an error “Burst shaper skipped 112 samples”. So I tried another technique:



Options
Title: Not titled yet
Author: raja961
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 4M

Import
Import: math

Import
Import: numpy

Variable
ID: psk_center_freq
Value: 918M

Variable
ID: RCC_Filter_Taps_TX
Value: firdes.root_raised_...

Variable
ID: samples_per_symbol
Value: 4

Parameter
ID: packet_header_format
Label: Packet Header Format
Value: <gnuradi...672e3b27b0>

Message Strobe
Message PMT: (() ....100 33])
Period (ms): 1k
Label:
Pair Mode: False
Static Mode: False

Embedded Python Block

PDU to Tagged Stream
Length tag name: packet_len

Protocol Formatter
Format Obj.: <gnur...db57070>
Length Tag Name: packet_len

Repack Bits
Bits per input byte: 8
Bits per output byte: 2

Tagged Stream Mux
Length tag names: packet_len

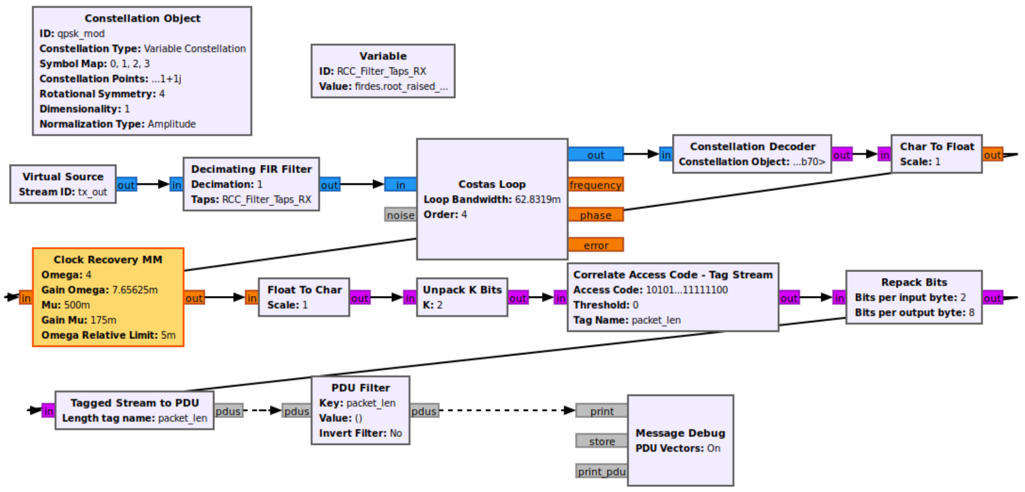
Chunks to Symbols
Symbol Table: -1-1...1j, 1+1j
Dimension: 1

Burst Shaper
Window Taps: ([1,]*1000)
Pre-padding Length: 5k
Post-padding Length: 5k
Insert phasing symbols: Yes
Length Tag Name: packet_len

Interpolating FIR Filter
Interpolation: 4
Taps: RCC_Filter_Taps_TX

Virtual Sink
Stream ID: tx_out

Figure 3-14. QPSK second trial transmitter flowgraph



Constellation Object
ID: qpsk_mod
Constellation Type: Variable Constellation
Symbol Map: 0, 1, 2, 3
Constellation Points: ...1+1j
Rotational Symmetry: 4
Dimensionality: 1
Normalization Type: Amplitude

Variable
ID: RCC_Filter_Taps_RX
Value: firdes.root_raised_...

Virtual Source
Stream ID: tx_out

Decimating FIR Filter
Decimation: 1
Taps: RCC_Filter_Taps_RX

Costas Loop
Loop Bandwidth: 62.8319m
Order: 4

Constellation Decoder
Constellation Object: ...b70>

Char To Float
Scale: 1

Clock Recovery MM
Omega: 4
Gain Omega: 7.65625m
Mu: 500m
Gain Mu: 175m
Omega Relative Limit: 5m

Float To Char
Scale: 1

Unpack K Bits
K: 2

Correlate Access Code - Tag Stream
Access Code: 10101...11111100
Threshold: 0
Tag Name: packet_len

Repack Bits
Bits per input byte: 2
Bits per output byte: 8

Tagged Stream to PDU
Length tag name: packet_len

PDU Filter
Key: packet_len
Value: ()
Invert Filter: No

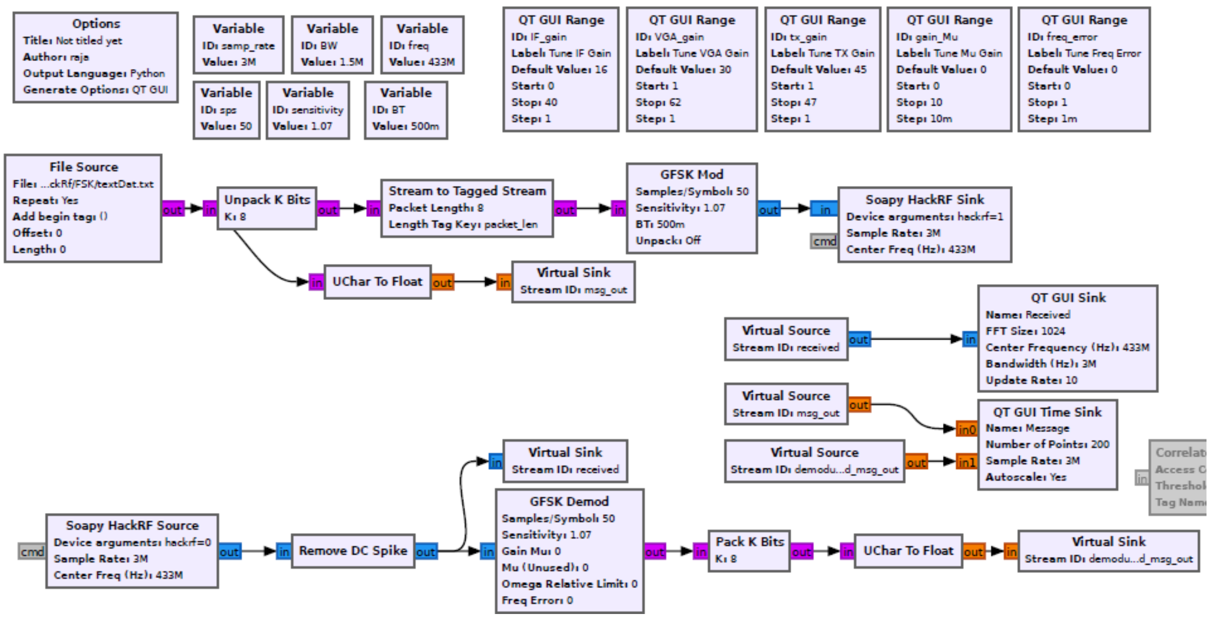
Message Debug
PDU Vectors: On

Figure 3-15. QPSK second trial receiver flowgraph

3.5 GFSK Modulation/Demodulation

GFSK stands for Gaussian Frequency Shift keying. It is an improved version of the typical FSK. It improves the frequency response by narrowing down the bandwidth.

In this experiment, I tried GFSK modulation and demodulation of a text file including the letter "H ". Letter "H" has a decimal representation of 72 and the space has a decimal representation of 32. The text file automatically includes an "end line" terminator which has a decimal representation of 10. Hence, I am expecting my output message to be in form of 72 32 32 32 32 32 10. The figures below show the transmitter side (upper side of the flowgraph) and the receiver side (lower part of the flowgraph). It can be clearly seen that the message has been successfully transmitted (blue is the message and red is the received message).


Options
Title: Not titled yet
Author: raja
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 3M

Variable
ID: BW
Value: 1.5M

Variable
ID: freq
Value: 433M

Variable
ID: sps
Value: 50

Variable
ID: sensitivity
Value: 1.07

Variable
ID: BT
Value: 500m

QT GUI Range
ID: IF_gain
Label: Tune IF Gain
Default Value: 16
Start: 0
Stop: 40
Step: 1

QT GUI Range
ID: VGA_gain
Label: Tune VGA Gain
Default Value: 30
Start: 1
Stop: 62
Step: 1

QT GUI Range
ID: tx_gain
Label: Tune TX Gain
Default Value: 45
Start: 1
Stop: 47
Step: 1

QT GUI Range
ID: gain_Mu
Label: Tune Mu Gain
Default Value: 0
Start: 0
Stop: 10
Step: 10m

QT GUI Range
ID: freq_error
Label: Tune Freq Error
Default Value: 0
Start: 0
Stop: 1
Step: 1m

File Source
File: ...ckRf/FSK/textDat.txt
Repeat: Yes
Add begin tag: ()
Offset: 0
Length: 0

Unpack K Bits
K: 8

Stream to Tagged Stream
Packet Length: 8
Length Tag Key: packet_len

GFSK Mod
Samples/Symbol: 50
Sensitivity: 1.07
BT: 500m
Unpack: Off

Soapy HackRF Sink
Device arguments: hackrf=1
Sample Rate: 3M
Center Freq (Hz): 433M

UChar To Float

Virtual Sink
Stream ID: msg_out

Virtual Source
Stream ID: received

QT GUI Sink
Name: Received
FFT Size: 1024
Center Frequency (Hz): 433M
Bandwidth (Hz): 3M
Update Rate: 10

Virtual Source
Stream ID: msg_out

Virtual Source
Stream ID: demodu...d_msg_out

QT GUI Time Sink
Name: Message
Number of Points: 200
Sample Rate: 3M
Autoscale: Yes

Virtual Sink
Stream ID: received

Soapy HackRF Source
Device arguments: hackrf=0
Sample Rate: 3M
Center Freq (Hz): 433M

Remove DC Spike

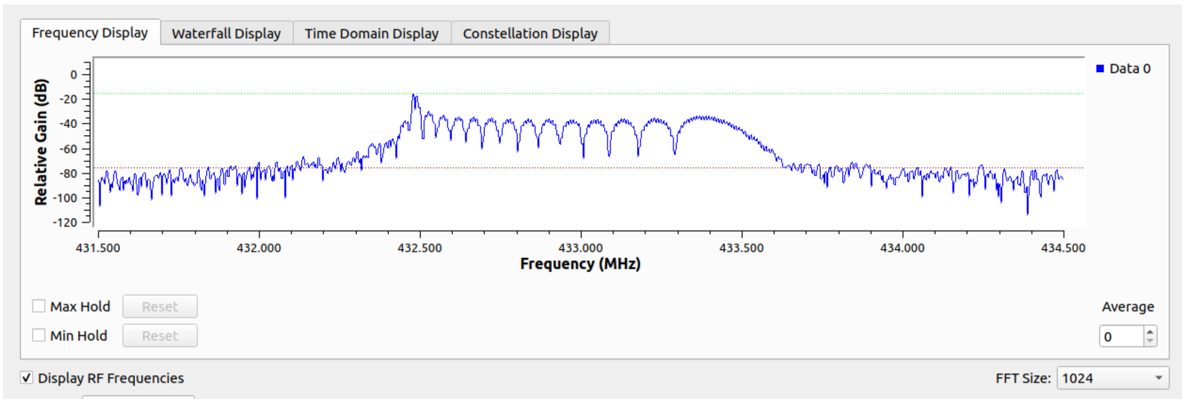
GFSK Demod
Samples/Symbol: 50
Sensitivity: 1.07
Gain Mu: 0
Mu (Unused): 0
Omega Relative Limit: 0
Freq Error: 0

Pack K Bits
K: 8

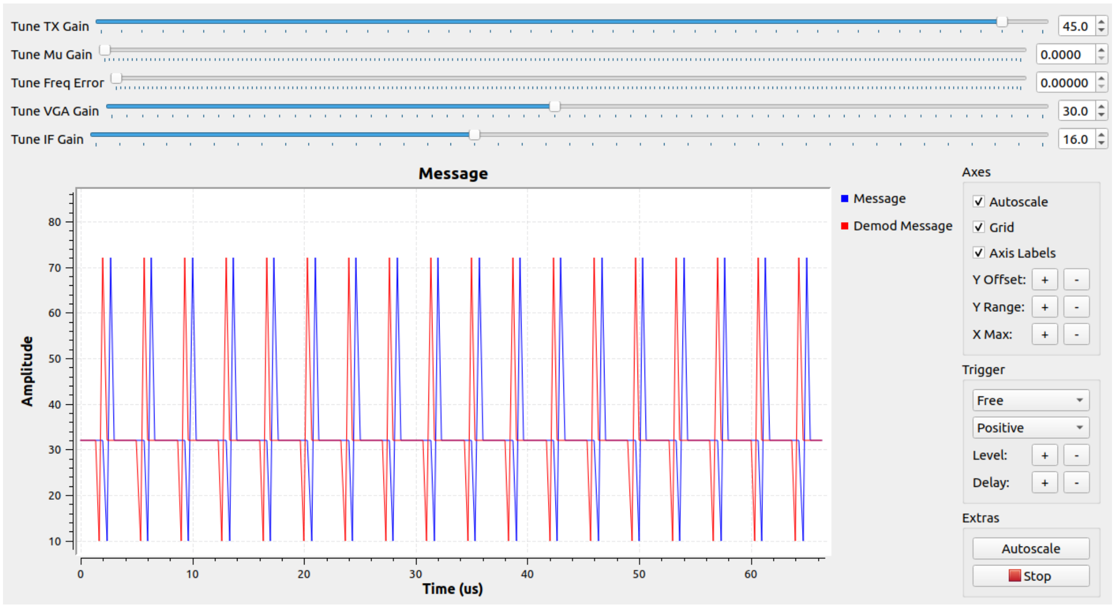
UChar To Float

Virtual Sink
Stream ID: demodu...d_msg_out

3-16. GFSK mod/demod. Top part: TX, low part: RX



3-17. Received Frequency Spectrum (centered at 2.4 GHz)


Tune TX Gain 45.0
Tune Mu Gain 0.0000
Tune Freq Error 0.00000
Tune VGA Gain 30.0
Tune IF Gain 16.0

3-18. Source message in blue and received message in red

Although I received the message successfully however, there might be times where the message gets corrupted and the transmitting power must be minimized then maximized in order to get the correct shape of the message.

3.6 Conclusion Up to 3.17.2022

I got good results using GFSK modulation and demodulation blocks. However, sometimes, the transmitted signal gets corrupted at the receiving end. One possible reason is that the source message and the received message has to be synchronize via a packet code which I will be working on during the next days.