HY448 FM Radio

Lantzos Stergios

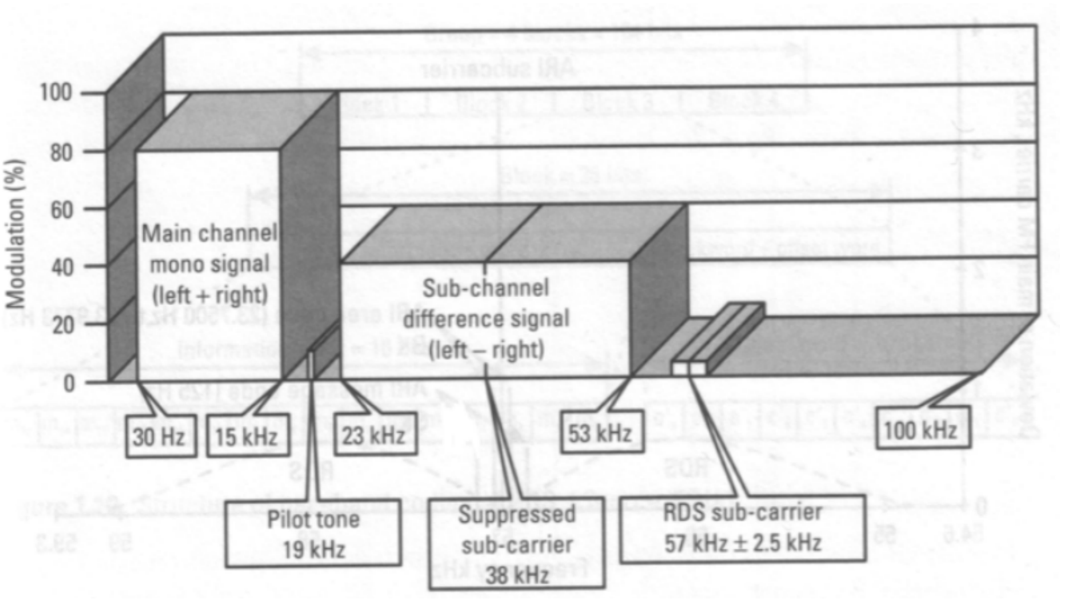
Department of Electrical and Computer Engineering
University Of Thessaly

February 10, 2023

Contents

1 Introduction

FM-Radio is a broadcast technology that uses frequency modulation to transmit sound over radio waves. It provides high-fidelity audio transmission and is commonly used for music and talk radio programs. FM radio operates in the frequency range of 87.5 to 108 MHz and is widely available in many countries around the world. It is popular due to its ease of use, portability and the availability of FM receivers in many devices, including cars and smartphones.A typical spectrum of composite base-band FM signal is shown in Figure 1:



Modulation (%)
100
80
60
40
20
0
Main channel mono signal (left + right)
Sub-channel difference signal (left – right)
30 Hz
15 kHz
23 kHz
53 kHz
100 kHz
Pilot tone 19 kHz
Suppressed sub-carrier 38 kHz
RDS sub-carrier 57 kHz ± 2.5 kHz

Figure 1: FM Spectrum

As we can observe, the FM spectrum consists of:

- L + R signal (Left + Right Channel), Bandwidth: 30 Hz - 15 kHz, and its called Mono Audio.
- A tone located at 19 kHz, which called Pilot Tone.
- L - R signal (Left - Right Channel), Bandwidth: 23 kHz - 53 kHz, centered at 38kHz, called Stereo Audio.
- RDS Signal(RDS stands for Radio Data System), Bandwidth: 54.6 kHz - 59.4 kHz, centred at 57 kHz.

In this project, we are going to implement an FM - Radio and some of its functionalities. using Gnu-Radio Companion(GRC), which is a Software Defined Radio platform that enables experimentation and education in wireless communications, along with Adalm-Pluto(SDR) Analog Device. More specifically, we will focus on:

- Creating an FM station, using our laptop's microphone as input
- Creating an FM receiver, using our laptop's speakers as output
- Receiving an FM station which broadcasts stereo
- Decoding FM-RDS packets
- Encoding FM-RDS packets

2 Requirements

2.1 Gnu-Radio Companion

Gnu-Radio Companion(GRC) installed in Ubuntu 22.04.1 LTS(64-bit), by typing the command:

sudo apt-get install gnuradio

The version is 3.10.1.1 .

2.2 Installing PlutoSDR Drivers

Three libraries required:

- **libiio**, Analog Device's "cross-platform" library for interfacing hardware
- **libad9361-iio**, AD9361 is the specific RF chip inside the PlutoSDR
- **pyadi-iio**, the Pluto's Python API

2.2.1 libiio

Libiio is a library that has been developed by Analog Devices to ease the development of software interfacing Linux Industrial I/O (IIO) devices. The library abstracts the low-level details of the hardware, and provides a simple yet complete programming interface that can be used for advanced projects. The installation commands:

- sudo apt-get install build-essential git libxml2-dev bison flex libcdk5-dev cmake python3-pip libusb-1.0-0-dev libavahi-client-dev libavahi-common-dev libaio-dev
- cd (to home directory)
- git clone –branch v0.23 https://github.com/analogdevicesinc/libiio.git
- cd libiio
- mkdir build
- cd build
- cmake -DPYTHON_BINDINGS=ON ..
- make -j$(nproc)
- sudo make install
- sudo ldconfig

2.2.2 libad9361-iio

A library for filter design/handling and multi-chip sync. The installation commands:

- cd (to home directory)
- git clone https://github.com/analogdevicesinc/libad9361-iio.git
- cd libad9361-iio
- mkdir build
- cd build
- cmake ..
- make -j$(nproc)
- sudo make install
- sudo ldconfig

2.2.3 pyadi-iio

The pyadi-iio library is used simplify the use of different devices a python package was created interface with the different IIO drivers. The module pyadi-iio, provides device-specific APIs built on top of the current libIIO python bindings. The installation commands:

- cd (to home directory)
- git clone –branch v0.0.14 https://github.com/analogdevicesinc/pyadi-iio.git
- cd pyadi-iio
- pip3 install –upgrade pip
- pip3 install -r requirements.txt
- sudo python3 setup.py install

2.3 "Hacking" Adalm-Pluto SDR to increase RF Range

When we plugged in Adalm-Pluto to our PC, a "config.txt" file appears in his folder. Here, we can define its IP(we set it to "192.168.2.1").
Open a terminal and:

- ssh root@192.168.2.1
- password: analog

Now, we can check our Adalm-Pluto's Firmware version. For version 0.32 and higher, type:

- fw_setenv compatible ad9364
- reboot

Now, we are able to tune up to 6 GHz and down to 70 MHz.

2.4 Installing and building gr-rds blocks from Source-GitHub

Since Gnu-Radio Companion does not provide us any blocks for RDS Decoding-Encoding, we need to install and build them. Specifically, a pretty good repository that contain these blocks is this. The installation and build of this repository, requires also the installation of the Boost.Locale library, by typing:

```
sudo apt-get install libboost-all-dev
```

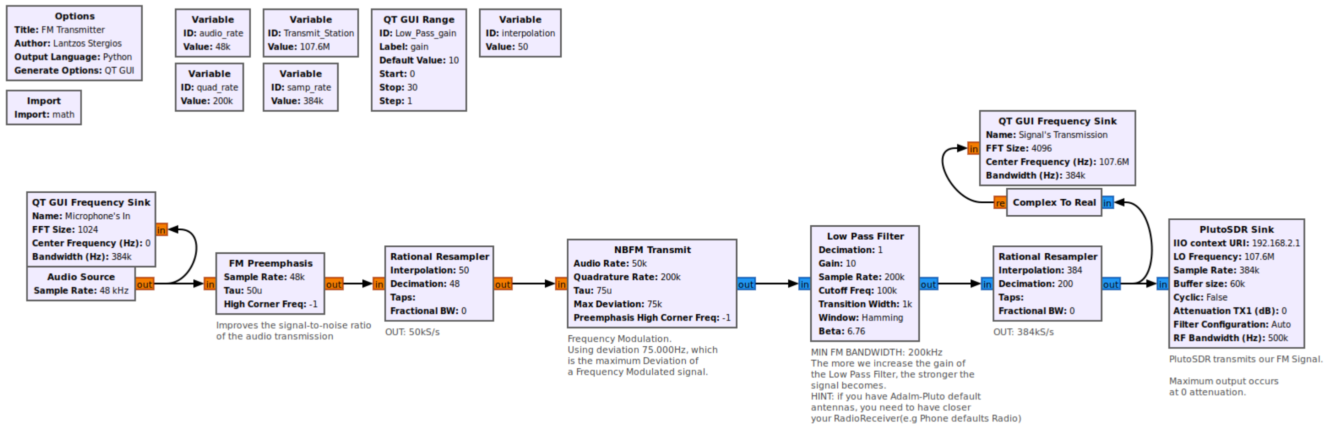
The installation commands for installing and building gr-rds blocks(GnuRadio Version: 3.10):

- cd (to our home directory)
- git clone –branch=maint-3.10 https://github.com/bastibl/gr-rds.git
- cd gr-rds
- mkdir build
- cd build
- -cmake .. (that requires Boost.Locale library to produce the Makefile)
- make
- sudo make install
- sudo ldconfig

Now, we are ready to begin our implementations.

3 FM Station

To build an FM-Station in GnuRadio-Companion, using Adalm-Pluto SDR, first we need to find frequencies that are not used. In my case, I'm transmitting in 107.6 MHz. The GRC Flow Graph of an FM-Station is shown in Figure 2. The variable **audio_rate** represents the sampling rate of the block **Audio Source**. This specific



Options
Title: FM Transmitter
Author: Lantzos Stergios
Output Language: Python
Generate Options: QT GUI

Import
Import: math

Variable
ID: audio_rate
Value: 48k

Variable
ID: quad_rate
Value: 200k

Variable
ID: Transmit_Station
Value: 107.6M

Variable
ID: samp_rate
Value: 384k

QT GUI Range
ID: Low_Pass_gain
Label: gain
Default Value: 10
Start: 0
Stop: 30
Step: 1

Variable
ID: interpolation
Value: 50

QT GUI Frequency Sink
Name: Microphone's In
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 384k

Audio Source
Sample Rate: 48 kHz

FM Preemphasis
Sample Rate: 48k
Tau: 50u
High Corner Freq: -1

Improves the signal-to-noise ratio of the audio transmission

Rational Resampler
Interpolation: 50
Decimation: 48
Taps:
Fractional BW: 0

OUT: 50kS/s

NBFM Transmit
Audio Rate: 50k
Quadrature Rate: 200k
Tau: 75u
Max Deviation: 75k
Preemphasis High Corner Freq: -1

Frequency Modulation.
Using deviation 75.000Hz, which is the maximum Deviation of a Frequency Modulated signal.

Low Pass Filter
Decimation: 1
Gain: 10
Sample Rate: 200k
Cutoff Freq: 100k
Transition Width: 1k
Window: Hamming
Beta: 6.76

MIN FM BANDWIDTH: 200kHz
The more we increase the gain of the Low Pass Filter, the stronger the signal becomes.
HINT: if you have Adalm-Pluto default antennas, you need to have closer your RadioReceiver(e.g Phone defaults Radio)

Rational Resampler
Interpolation: 384
Decimation: 200
Taps:
Fractional BW: 0

OUT: 384kS/s

QT GUI Frequency Sink
Name: Signal's Transmission
FFT Size: 4096
Center Frequency (Hz): 107.6M
Bandwidth (Hz): 384k

Complex To Real

PlutoSDR Sink
IIO context URI: 192.168.2.1
LO Frequency: 107.6M
Sample Rate: 384k
Buffer size: 60k
Cyclic: False
Attenuation TX1 (dB): 0
Filter Configuration: Auto
RF Bandwidth (Hz): 500k

PlutoSDR transmits our FM Signal.

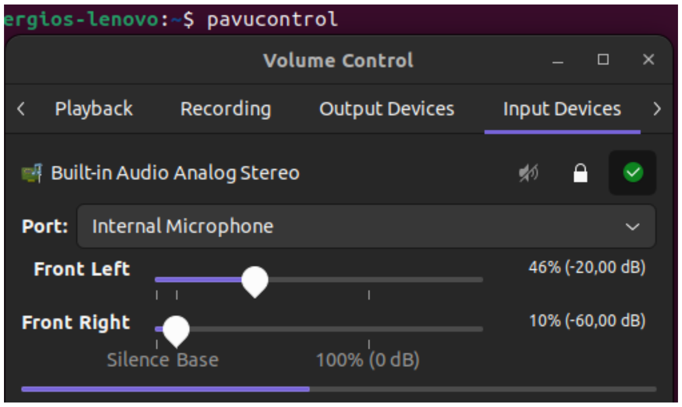
Maximum output occurs at 0 attenuation.

Figure 2: FM-Station Flow Graph

sampling rate is supported by the most common hardwares. If we tap in this block, we can see four fields:

- Sample Rate : set to 48kHz
- Device Name : left it blank (it detects our PC's microphone by itself). If its not, use: plughw:0,0 or hw:0,0
- OK to Block : select "NO"
- Num Outputs : set it to 1

Its important to mention that Audio Source with 1 Output, in my PC, it didn't work at first. I managed to make it work properly, Via pavucontrol(sudo apt-get install pavucontrol and then type "pavucontrol"), configuring my mic's settings like Figure 3:



```
ergios-lenovo:~$ pavucontrol
```

Built-in Audio Analog Stereo
Port: Internal Microphone
Front Left 46% (-20,00 dB)
Front Right 10% (-60,00 dB)

Figure 3: Pavucontrol Settings for my Mic

Basically, all i did, is to mute the Front Right(by moving it inside the Silence Base), and now Audio Source is working perfect! If i left it as default, Audio Source required at least two outputs.

So, Audio Source takes input from our microphone. **FM Preemphasis** follows, that is used for the transmission sessions. Its main functionality is to improve the signal-to-noise ratio(SNR) of the audio transmission(remember: SNR is defined as the ratio of signal power to the noise power). A **Rational Resampler** interpolates our sampling rate, giving us a new sampling rate of 50kS/s. Then, i used a **NBFM**(Narrow Band Frequency Modulation), that produces a single FM modulated complex base-band output, ready to be transmitted. At first, instead of NBFM block for the signal modulation, i used the WBFM(Wide Band Frequency Modulation). WBFM is more proper for high-quality music transmission, and NBFM works better for speech communications. Since i'm using my laptop's microphone, NBFM had a better performance. Its fields:

- Audio Rate: Input Sample Rate(for this scheme is 50kHz).
- Quadrature Rate: Sample Rate of the output stream. Needs to be multiple of Audio Rate. Because we want a typical Bandwidth of 200kHz, a Quadrature Rate of 200kS/s works fine.
- Tau: 75e-6 default value.
- Max Deviation: 75.000 Hz, which corresponds FM-Radios max allowable deviation.

A Low Pass Filter follows. Its main goal is to make stronger our signal and a little-bit more clear than before. Its fields:

- Decimation: 1
- Gain: is set by the variable Low_Pass_gain, so we can manually change it to find the proper gain required. Gain set to 10 as default.
- Sample Rate: the Quadrature Rate at 200kS/s.
- Cutoff Frequency: 100 kHz, keeping the 200 kHz bandwidth of FM standards, because this bandwidth is doubled, resulting the total FM-Bandwidth, which minimum value should be 200 kHz. Also, we are okay with the Nyquist-Sample Theorem, because we are sampling at 200kS/s and the highest frequencies that our signal passes from are 100kHz.
- Transition Width: 1000Hz.

Another **Rational Resampler** follows, that interpolates our sampling rate to match the Adalm-Pluto's one, at 384kS/s .
Finally, the last block is the **PlutoSDR Sink** which transmits our signal through TX Antenna, which most important fields are:

The plot of the FM-Station Flow Graph is represented in Figure 4.

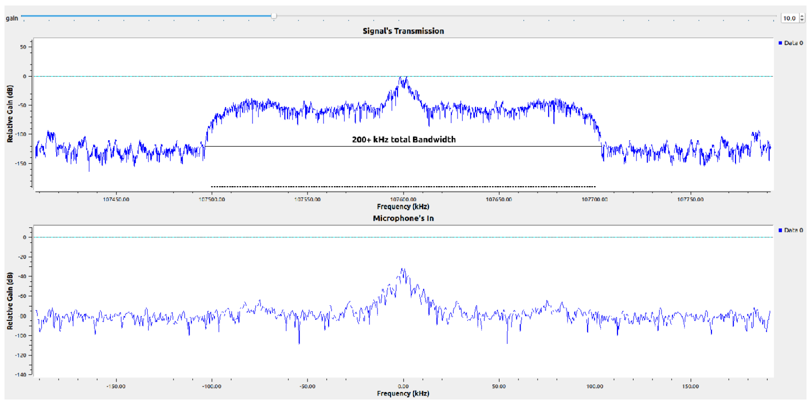
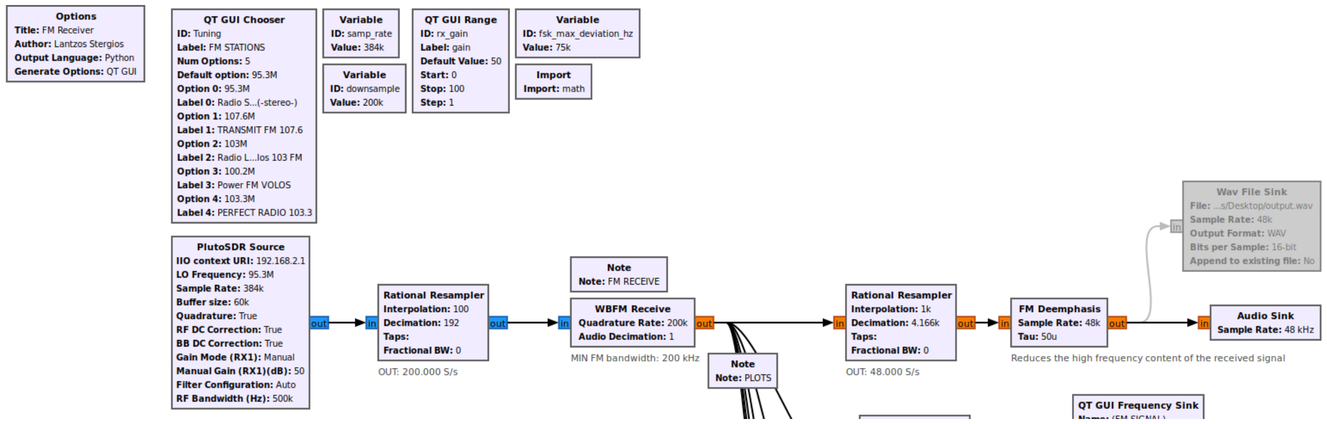


Figure 4: FM-Transmitted Signal at 107.6 MHz

As we can see, we have a total 200+ kHz signal bandwidth, that matches the minimum requirement bandwidth of FM. Also, the higher we set the gain of our Low Pass Filter, the more powerful the signal becomes. The FM-Station can be tested, by using our phone and lock into the frequency we transmit. It is important to mention that Adalm-Pluto's default antennas has a low TX Range, so if we want to hear what are we transmitting, we need to have our phone close to Adalm-Pluto, for achieving a better performance.

4 FM Receiver

Now we are ready to proceed with the FM-Receiver implementation. The Flow Graph of the FM-Receiver is illustrated in Figure 5. A **GT GUI Chooser** block, allows us to set frequencies of our desire(Like Radio Stations



```
Options
Title: FM Receiver
Author: Lantzos Stergios
Output Language: Python
Generate Options: QT GUI

QT GUI Chooser
ID: Tuning
Label: FM STATIONS
Num Options: 5
Default option: 95.3M
Option 0: 95.3M
Label 0: Radio S...(-stereo-)
Option 1: 107.6M
Label 1: TRANSMIT FM 107.6
Option 2: 103M
Label 2: Radio L...los 103 FM
Option 3: 100.2M
Label 3: Power FM VOLOS
Option 4: 103.3M
Label 4: PERFECT RADIO 103.3

Variable
ID: samp_rate
Value: 384k

Variable
ID: downsample
Value: 200k

QT GUI Range
ID: rx_gain
Label: gain
Default Value: 50
Start: 0
Stop: 100
Step: 1

Variable
ID: fsk_max_deviation_hz
Value: 75k

Import
Import: math

PlutoSDR Source
IIO context URI: 192.168.2.1
LO Frequency: 95.3M
Sample Rate: 384k
Buffer size: 60k
Quadrature: True
RF DC Correction: True
BB DC Correction: True
Gain Mode (RX1): Manual
Manual Gain (RX1)(dB): 50
Filter Configuration: Auto
RF Bandwidth (Hz): 500k

Rational Resampler
Interpolation: 100
Decimation: 192
Taps:
Fractional BW: 0
OUT: 200.000 S/s

Note
Note: FM RECEIVE

WBFM Receive
Quadrature Rate: 200k
Audio Decimation: 1
MIN FM bandwidth: 200 kHz

Note
Note: PLOTS

Rational Resampler
Interpolation: 1k
Decimation: 4.166k
Taps:
Fractional BW: 0
OUT: 48.000 S/s

FM Deemphasis
Sample Rate: 48k
Tau: 50u
Reduces the high frequency content of the received signal

Wav File Sink
File: ...s/Desktop/output.wav
Sample Rate: 48k
Output Format: WAV
Bits per Sample: 16-bit
Append to existing file: No

Audio Sink
Sample Rate: 48 kHz

QT GUI Frequency Sink
```

Figure 5: FM-Receiver Flow Graph

Frequencies et.c), with purpose to lock into them. In City of Volos, i found four Radio-Stations that broadcast FM Signals, and entered their frequencies in the fields of QT GUI Chooser, for tuning.

Beginning from the start of the Flow Graph, the very first block we meet is **PlutoSDR Source**. This block's fields are quite similar to PlutoSDR Sink, which explained in the section of the FM-Station. PlutoSDR Source LO(Local Oscillator) value is set to QT GUI Chooser's ID("Tuning"). From this point, PlutoSDR is capable to lock into these frequencies that QT GUI Chooser contains. We begin with a sample rate of 384 kHz.

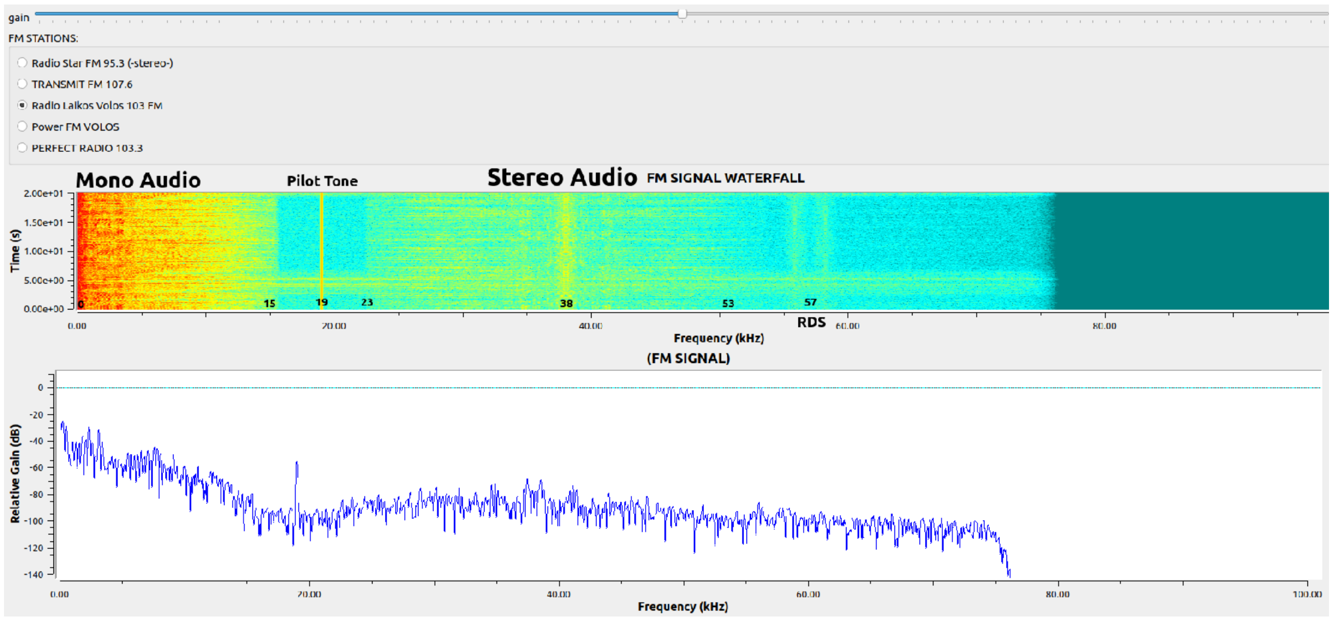
The second block we are faced of, is a **Rational Resampler**, which is decimating our sampling rate. In this case, we want to a sample rate of 200kHz(FM-Bandwidth standards), so we multiply 384kS/s with 100 and divide by 192, giving us a sample rate of 200kS/s.
Moving forward, a **WBFM Receive** appears to demodulate the signal we are receiving via PlutoSDR Source. Its main field:

- Quadrature Rate: Input sample rate of complex baseband input. In our case, is 200kHz.

This block outputs the demodulated audio, that we need to send to our speakers to hear what we are receiving.

At this stage, another **Rational Resampler** follows which decimates our sampling rate to exactly 48kHz, as much as my hardware can support. Last but not least, as we did with the FM-Station(adding FM-Preemphasis), now we add **FM-Deemphasis**, which reduces the high frequency content of the signal, too. Now, the demodulated audio is fed to our Speakers, through **Audio Sink** block. This block parameters are used exaclty as the Audio Source from the previous section.

For using both TX and RX and the same time, i suggest to right click on Audio Sink block and disable it, and enable the **Wav File Sink** block to record what you are receiving through PlutoSDR Source.
The Flow Graph continues downwards with some Plots, using LowPass and BandPass Filters just to isolate and plot the Mono Audio, Pilot Tone, Stereo Audio and finally the RDS. Executing the receiver and tuning at a Station, we get these plots:


gain

FM STATIONS:

- Radio Star FM 95.3 (-stereo-)
- TRANSMIT FM 107.6
- [✓ Radio Lalkos Volos 103 FM]
- Power FM VOLOS
- PERFECT RADIO 103.3

Figure 6: FM-Signal, FM-Waterfall

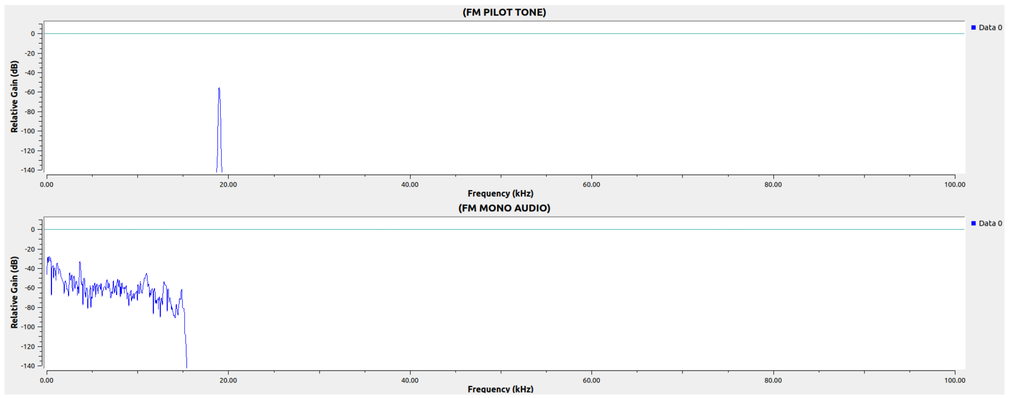


Figure 7: FM Pilot tone at 19kHz, FM Mono Audio at 0.03-15kHz

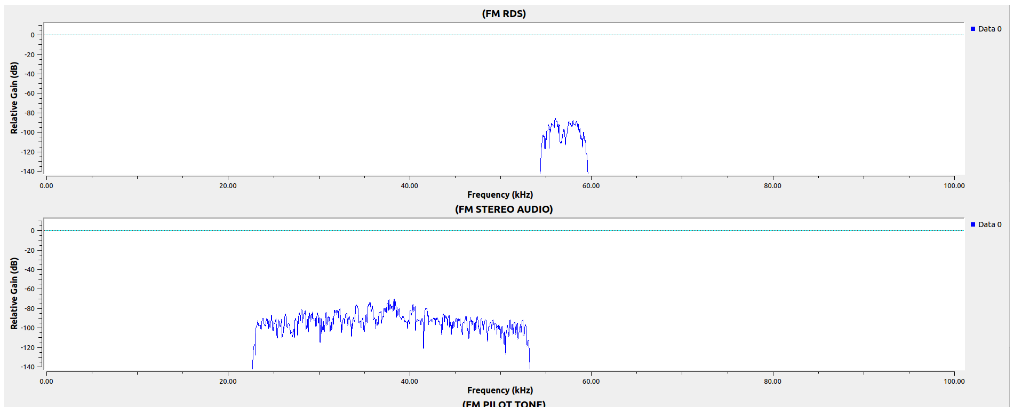


Figure 8: FM-Stereo Audio at 23-53kHz, FM-RDS at 54.6-59.4 kHz

As we expected(from the Introduction):

- The Mono Audio is located at: 0.03-15kHz
- The Pilot Tone at 19kHz
- The Stereo Audio at 23-53kHz, centered at 38kHz
- The RDS at 54.6-59.4kHz, centered at 57kHz

The Waterfall of the whole FM-Demodulated Signal, shows us how strong each signal is. The more hot is the color, the more powerful is the corresponding signal.

5 FM Stereo Receiver

Lets see how an FM-Stereo Decoder works and what are its functionalities. Our implementation is based in the scheme that is represented in the Figure 9:

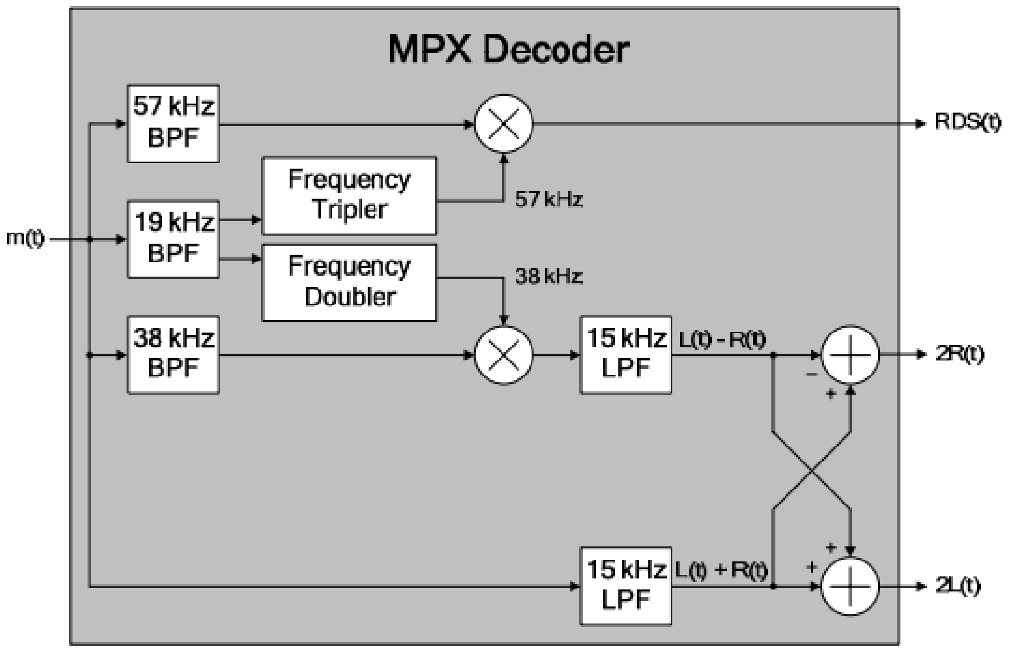
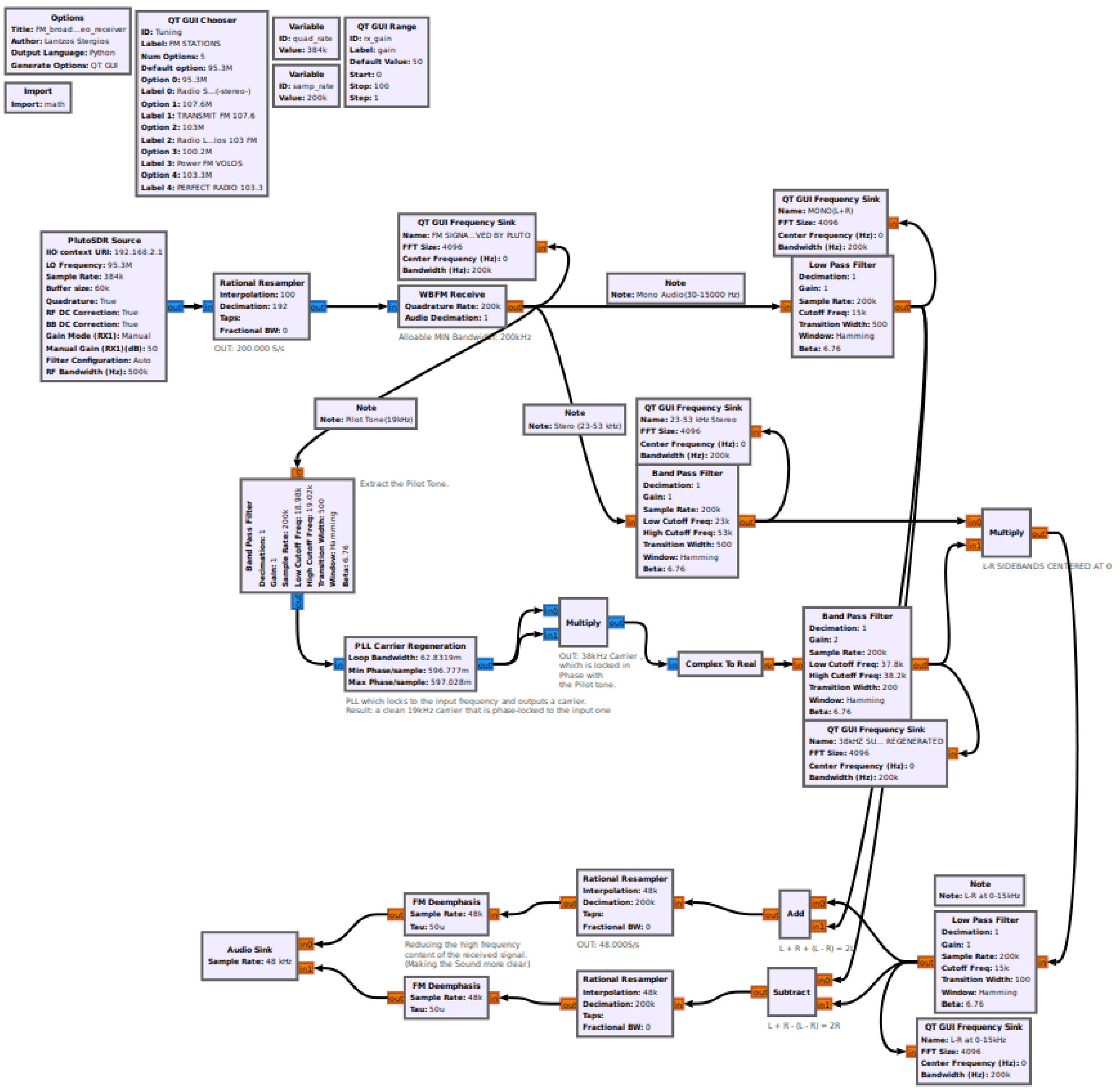


Figure 9: FM-Stereo Decoding Process

In this section, we will not deal with RDS, so the process is to:

- extract the L+R Signal from the Signal(Mono Audio), using LowPass Filter.
- generate a 38 kHz signal by frequency doubling the received 19 kHz Pilot. It will ensure that the 38 kHz signal is in phase with the 19 kHz signal(we will see how).
- demodulate the signals from 23 kHz to 53 kHz to get back the L-R channel(Stereo Audio), using a BandPass Filter.
- Build and use the L+R and the L-R signals to regenerate the original L and R signals.

The implementation of the FM-Stereo Receiver in GnuRadio:



Options
Title: FM_broad....eo_receiver
Author: Lantzos Stergios
Output Language: Python
Generate Options: QT GUI

Import
Import: math

QT GUI Chooser
ID: Tuning
Label: FM STATIONS
Num Options: 5
Default option: 95.3M
Option 0: 95.3M
Label 0: Radio S....(-stereo-)
Option 1: 107.6M
Label 1: TRANSMIT FM 107.6
Option 2: 103M
Label 2: Radio L....los 103 FM
Option 3: 100.2M
Label 3: Power FM VOLOS
Option 4: 103.3M
Label 4: PERFECT RADIO 103.3

Variable
ID: quad_rate
Value: 384k

Variable
ID: samp_rate
Value: 200k

QT GUI Range
ID: rx_gain
Label: gain
Default Value: 50
Start: 0
Stop: 100
Step: 1

PlutoSDR Source
IIO context URI: 192.168.2.1
LO Frequency: 95.3M
Sample Rate: 384k
Buffer size: 60k
Quadrature: True
RF DC Correction: True
BB DC Correction: True
Gain Mode (RX1): Manual
Manual Gain (RX1)(dB): 50
Filter Configuration: Auto
RF Bandwidth (Hz): 500k

Rational Resampler
Interpolation: 100
Decimation: 192
Taps:
Fractional BW: 0
OUT: 200.000 S/s

QT GUI Frequency Sink
Name: FM SIGNA....VED BY PLUTO
FFT Size: 4096
Center Frequency (Hz): 0
Bandwidth (Hz): 200k

WBFM Receive
Quadrature Rate: 200k
Audio Decimation: 1
Alloable MIN Bandwidth: 200kHz

Note
Note: Mono Audio(30-15000 Hz)

QT GUI Frequency Sink
Name: MONO(L+R)
FFT Size: 4096
Center Frequency (Hz): 0
Bandwidth (Hz): 200k

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Cutoff Freq: 15k
Transition Width: 500
Window: Hamming
Beta: 6.76

Note
Note: Pilot Tone(19kHz)

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Low Cutoff Freq: 18.98k
High Cutoff Freq: 19.02k
Transition Width: 500
Window: Hamming
Beta: 6.76
Extract the Pilot Tone.

Note
Note: Stero (23-53 kHz)

QT GUI Frequency Sink
Name: 23-53 kHz Stereo
FFT Size: 4096
Center Frequency (Hz): 0
Bandwidth (Hz): 200k

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Low Cutoff Freq: 23k
High Cutoff Freq: 53k
Transition Width: 500
Window: Hamming
Beta: 6.76

Multiply
L-R SIDEBANDS CENTERED AT 0

PLL Carrier Regeneration
Loop Bandwidth: 62.8319m
Min Phase/sample: 596.777m
Max Phase/sample: 597.028m
PLL which locks to the input frequency and outputs a carrier.
Result: a clean 19kHz carrier that is phase-locked to the input one

Multiply
OUT: 38kHz Carrier , which is locked in Phase with the Pilot tone.

Complex To Real

Band Pass Filter
Decimation: 1
Gain: 2
Sample Rate: 200k
Low Cutoff Freq: 37.8k
High Cutoff Freq: 38.2k
Transition Width: 200
Window: Hamming
Beta: 6.76

QT GUI Frequency Sink
Name: 38kHZ SU.... REGENERATED
FFT Size: 4096
Center Frequency (Hz): 0
Bandwidth (Hz): 200k

Note
Note: L-R at 0-15kHz

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Cutoff Freq: 15k
Transition Width: 100
Window: Hamming
Beta: 6.76

QT GUI Frequency Sink
Name: L-R at 0-15kHz
FFT Size: 4096
Center Frequency (Hz): 0
Bandwidth (Hz): 200k

Add
L + R + (L - R) = 2L

Subtract
L + R - (L - R) = 2R

Rational Resampler
Interpolation: 48k
Decimation: 200k
Taps:
Fractional BW: 0
OUT: 48.000S/s

Rational Resampler
Interpolation: 48k
Decimation: 200k
Taps:
Fractional BW: 0

FM Deemphasis
Sample Rate: 48k
Tau: 50u
Reducing the high frequency content of the received signal. (Making the Sound more clear)

FM Deemphasis
Sample Rate: 48k
Tau: 50u

Audio Sink
Sample Rate: 48 kHz

Figure 10: FM-Stereo Decoding Flow Graph in GnuRadio

Via **PlutoSDR Source**, we receive the Signal. A **Rational Resampler** is decimating our sampling rate to 200kS/s. The **WBFM Receive** demodulates our FM Signal, and now the process begins:

- **MONO AUDIO**: First of all, we extract the Mono Audio(Mono Audio locates at 0.03-15 kHz), using a **Low Pass Filter** with cutoff Frequency at 15 kHz. We direct it to the Add and Subtract Blocks, in purpose to build the L and R Channels.

- **STEREO AUDIO**: Now we need to do exactly the same process with the Stereo Audio. Extracting the Pilot Tone(which locates at 19kHz) using a **BandPass Filter**, with Low Cutoff Frequency at 18.90 kHz and High Cutoff Frequency at 19.02 kHz. We could now double it and regenerating the 38 kHz Carrier. But for better performance, we use the block **PLL Carrier Regeneration**. This block, locks into the

input frequency(19kHz Pilot tone in our case), and outputs a clean carrier, that is locked in-phase too. **Multiplying** by itself twice, we regenerate the 38 kHz carrier. Multiplying the Stereo Audio signal with the Pilot Tone ,allows us to extract the stereo information from the composite signal by removing the mono audio and recovering the original stereo audio. Meanwhile, we isolated the Stereo Audio from the Demodulated Signal(Stereo locates at 23-53kHz) using also a **BandPass Filter**, and we are **multiplying** it with the 38kHz carrier that we regenerated right now. The Stereo Audio is centered at 0 Hz now, and we are taking the Side-band(L-R) with a **Low Pass filter**, using cutoff Frequency at 15kHz.
Since we extracted successfully the L+R and the L-R side-bands, we need to build the channels(Right and Left). Using:

- **Add** Block: takes the L-R and adds it to L+R. The Result is the: L+R + L - R = 2L(Left channel)
- **Subtract** Block: Subtracts the L-R from the L+R. The Result is the: L+R - (L - R) = 2R(Right channel)

The last step, is to Downsample at 48kS/s and add the block **FM Deemphasis**. Left and Right channel are fed to **Audio Sink**, that plays the Stereo Audio in our Speakers.

6 RDS STANDARDS

FM RDS (Radio Data System) is a system for transmitting data over FM radio waves. It is widely used in many countries for providing listeners with additional information such as the name of the song or artist currently playing, station identification, and traffic reports. The technology allows stations to transmit text messages and digital signals, making it possible to provide a more interactive listening experience.

6.1 Data channel (Physical layer)

The RDS hardware first demodulates the 57 kHz RDS subcarrier signal to extract a differential Manchester encoded signal which contains both the bit clock and the differentially encoded bitstream. This allows the RDS decoder to tolerate phase inversion of its input.

6.2 Baseband coding (Data-link layer)

One RDS "group" (frame) is 104 bits long, comprised of 4 "blocks" of 26 bits each. A block includes the "information word" (16 bits) and the "checkword" (10 bits). There are 32 group types (0- 15, A or B) and the decoding of the infowords depends on the group type. This structure is illustrated in Figure 11:

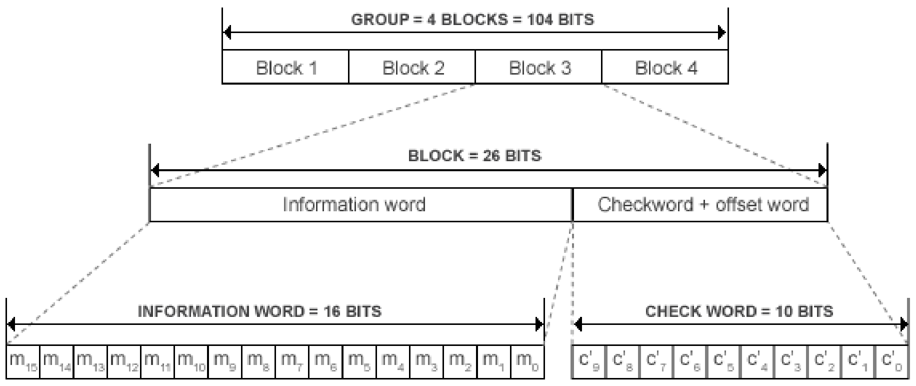

GROUP = 4 BLOCKS = 104 BITS
Block 1
Block 2
Block 3
Block 4
BLOCK = 26 BITS
Information word
Checkword + offset word
INFORMATION WORD = 16 BITS
CHECK WORD = 10 BITS
m15 m14 m13 m12 m11 m10 m9 m8 m7 m6 m5 m4 m3 m2 m1 m0
c'9 c'8 c'7 c'6 c'5 c'4 c'3 c'2 c'1 c'0

Figure 11: Structure of an RDS group

6.3 Method Of Modulation

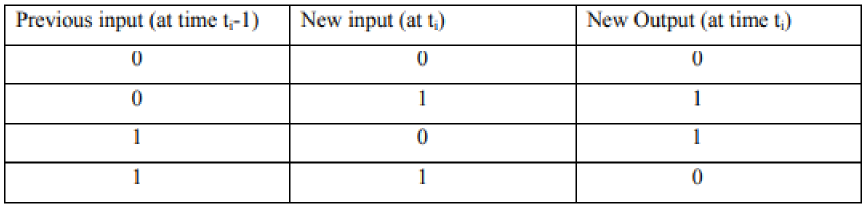
The RDS sub-carrier at 57kHz(with a total bandwidth of 4.8kHz) is amplitude-modulated(AM) by the shaped and Bi-Phase(or Manchester) coded signal. It is also suppressed. The method of modulation RDS uses is BPSK(Binary Phase-shift keying). So, when the digital information changes from 1 to -1, the modulated signal changes from cos(t) to cos(-t), which equals to cos(t+π) or a phase inversion.

6.4 Clock Frequency and Data Rate

The basic clock frequency in FM RDS is actually derived from the division of a higher reference frequency, which is 57 kHz with 48. This division by 48 is used to generate the RDS subcarrier frequency because it provides a convenient frequency that falls within the acceptable range for a subcarrier in the FM band and can be used for efficient data transmission. The division of 57 kHz by 48 results in a subcarrier frequency of 1187.5 Hz. So, the basic data-rate of the system is 1187.5 +- 0.125 bits/second. Also, the 57kHz is the 3rd harmonic of the Pilot Tone(19kHz). Since the Pilot tone has a +- 2Hz deviation, the 57kHz subcarrier has a deviation of +- 6Hz, because as we tripling the frequency, the deviation is tripled too.

6.5 Differential Encoding

RDS systems uses Differential Encoding. In differential encoding, the data is transmitted as the difference between the current data symbol and the previous one, instead of absolute values. Supposing ”ti” is some arbitrary time, and ”ti-1” is the time one data-clock period earlier, where data-clock period is equal to 1/1187.5 sec. When the input is 0, the output remains unchanged from the previous output bit, and when an input 1 occurs, the new output bit is the complement of the previous output bit. In conclusion, modulo 2 SUM(or X-OR) is performed(Figure 12).



Previous input (at time t_i-1)	New input (at t_i)	New Output (at time t_i)
0	0	0
0	1	1
1	0	1
1	1	0

Figure 12: Differential Encoding

6.6 Manchester Encoding

Manchester encoding is a form of binary phase-shift keying (BPSK) that has gained wide acceptance as the modulation scheme for low-cost radio-frequency (RF) transmission of digital data. The encoding of digital data in Manchester format defines the binary states of ”1” and ”0” to be transitions rather than static values. So, instead of sending zeros and ones, the Manchester algorithm sends two symbols(they called ”two halves”) in one data-clock period. Replaces the logic “1” by a positive impulse at t = 0 and an opposite polarity pulse delayed by td/2, where td is the bit duration. For logic 0, the inverse method is used(Figure 13).

[1, -1] represents logic 1

[-1, 1] represents logic 0

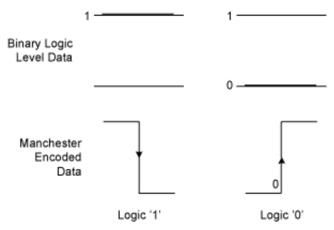


Figure 13: Manchester Encoding

Although the bit rate of the RDS is 1187.5 Bit/s, the Manchester encoding doubles this rate, resulting a new bit rate of 2375 Bit/s.

6.7 Matched Filter

A Pulse shaping filter to limit message spectrum and shape these impulses to smooth the signal, is added both in Transmitter and Receiver. This filter, we called "Root Raised Cosine Filter(RRC)", is used to improve the signal-to-noise ratio(SNR) of the received RDS signal and to reduce errors in decoding the data, providing also the required band limited spectrum. The combination of two RRC Filters is called a Raised Cosine Filter(RC), and is ideal for digital transmission and helps in reducing inter symbol interference (ISI). The Bi-Phase coded symbols are represented by the two below equations.
Logic 1 is represented by:

$$e(t) = d(t) - d(t - td/2) = 1 - (-1) = 2 > 0 \text{ (positive value)} \tag{1}$$

and a logic 0:

$$e(t) = -d(t) + d(t - td/2) = -1 - (+1) = -2 < 0 \text{ (negative value)} \tag{2}$$

where t is an arbitrary time value, and the d(t) and d(t - td/2) are the two halves of one Manchester Encoded bit. So, according to equations (1),(2), a Bi-Phase Symbol which has a negative value is represented as zero, and for a positive is represented as 1. We can see it clearly in the figure 14:

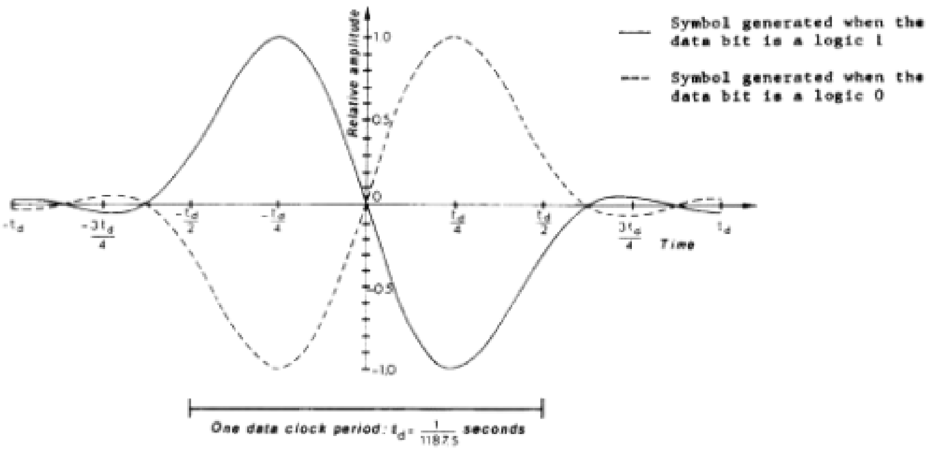


Figure 14: The RRC filter's response for one Manchester Symbol.

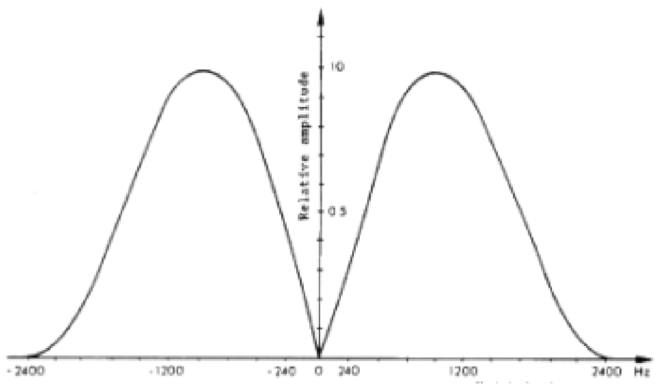
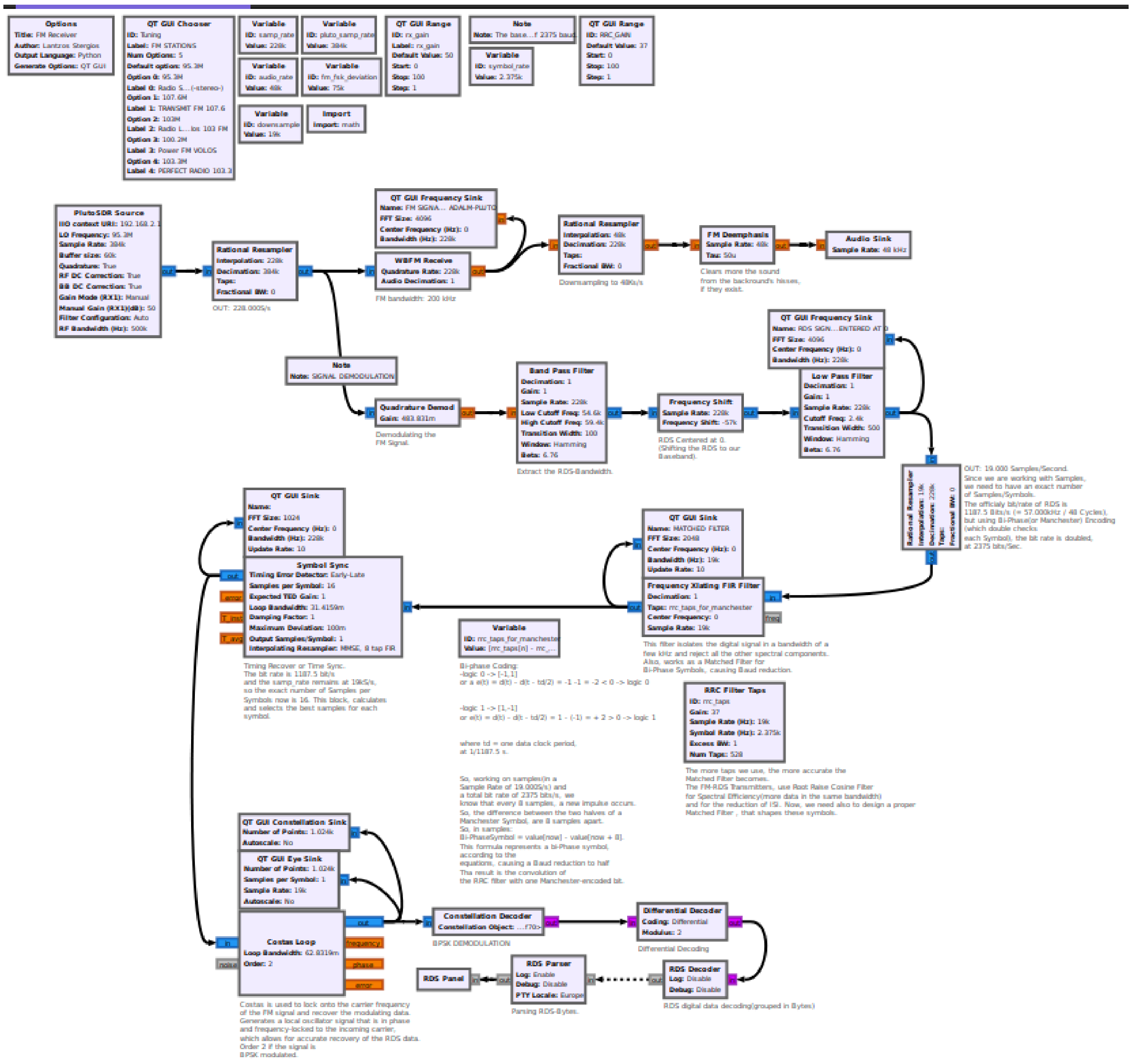


Figure 15: Spectrum of bi-phase coded radio-data signals. Notice the peak at 1187.5Hz !

7 FM-RDS Decoding

An FM-RDS Decoding process in GnuRadio Companion, is illustrated in Figure 16:



Options
Title: FM Receiver
Author: Lantzos Stergios
Output Language: Python
Generate Options: QT GUI

QT GUI Chooser
ID: Tuning
Label: FM STATIONS
Num Options: 5
Default option: 95.3M
Option 0: 95.3M
Label 0: Radio S...(-stereo-)
Option 1: 107.6M
Label 1: TRANSMIT FM 107.6
Option 2: 103M
Label 2: Radio L...los 103 FM
Option 3: 100.2M
Label 3: Power FM VOLOS
Option 4: 103.3M
Label 4: PERFECT RADIO 103.3

Variable
ID: samp_rate
Value: 228k

Variable
ID: pluto_samp_rate
Value: 384k

Variable
ID: audio_rate
Value: 48k

Variable
ID: fm_fsk_deviation
Value: 75k

Variable
ID: downsample
Value: 19k

Import
Import: math

QT GUI Range
ID: rx_gain
Label: rx_gain
Default Value: 50
Start: 0
Stop: 100
Step: 1

Note
Note: The base...f 2375 baud

Variable
ID: symbol_rate
Value: 2.375k

QT GUI Range
ID: RRC_GAIN
Default Value: 37
Start: 0
Stop: 100
Step: 1

PlutoSDR Source
IIO context URI: 192.168.2.1
LO Frequency: 95.3M
Sample Rate: 384k
Buffer size: 60k
Quadrature: True
RF DC Correction: True
BB DC Correction: True
Gain Mode (RX1): Manual
Manual Gain (RX1)(dB): 50
Filter Configuration: Auto
RF Bandwidth (Hz): 500k

Rational Resampler
Interpolation: 228k
Decimation: 384k
Taps:
Fractional BW: 0

OUT: 228.000S/s

QT GUI Frequency Sink
Name: FM SIGNA... ADALM-PLUTO
FFT Size: 4096
Center Frequency (Hz): 0
Bandwidth (Hz): 228k

WBFM Receive
Quadrature Rate: 228k
Audio Decimation: 1

FM bandwidth: 200 kHz

Rational Resampler
Interpolation: 48k
Decimation: 228k
Taps:
Fractional BW: 0

Downsampling to 48Ks/s

FM Deemphasis
Sample Rate: 48k
Tau: 50u

Clears more the sound
from the backround's hisses,
if they exist.

Audio Sink
Sample Rate: 48 kHz

Note
Note: SIGNAL DEMODULATION

Quadrature Demod
Gain: 483.831m

Demodulating the
FM Signal.

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 228k
Low Cutoff Freq: 54.6k
High Cutoff Freq: 59.4k
Transition Width: 100
Window: Hamming
Beta: 6.76

Extract the RDS-Bandwidth.

Frequency Shift
Sample Rate: 228k
Frequency Shift: -57k

RDS Centered at 0.
(Shifting the RDS to our
Baseband).

QT GUI Frequency Sink
Name: RDS SIGN...ENTERED AT 0
FFT Size: 4096
Center Frequency (Hz): 0
Bandwidth (Hz): 228k

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 228k
Cutoff Freq: 2.4k
Transition Width: 500
Window: Hamming
Beta: 6.76

Rational Resampler
Interpolation: 19k
Decimation: 228k
Taps:
Fractional BW: 0

OUT: 19.000 Samples/Second.
Since we are working with Samples,
we need to have an exact number
of Samples/Symbols.
The officialy bit/rate of RDS is
1187.5 Bits/s (= 57.000kHz / 48 Cycles),
but using Bi-Phase(or Manchester) Encoding
(which double checks
each Symbol), the bit rate is doubled,
at 2375 bits/Sec.

QT GUI Sink
Name: MATCHED FILTER
FFT Size: 2048
Center Frequency (Hz): 0
Bandwidth (Hz): 19k
Update Rate: 10

Frequency Xlating FIR Filter
Decimation: 1
Taps: rrc_taps_for_manchester
Center Frequency: 0
Sample Rate: 19k

This filter isolates the digital signal in a bandwidth of a
few kHz and reject all the other spectral components.
Also, works as a Matched Filter for
Bi-Phase Symbols, causing Baud reduction.

QT GUI Sink
Name:
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 228k
Update Rate: 10

Symbol Sync
Timing Error Detector: Early-Late
Samples per Symbol: 16
Expected TED Gain: 1
Loop Bandwidth: 31.4159m
Damping Factor: 1
Maximum Deviation: 100m
Output Samples/Symbol: 1
Interpolating Resampler: MMSE, 8 tap FIR

Timing Recover or Time Sync.
The bit rate is 1187.5 bit/s
and the samp_rate remains at 19kS/s,
so the exact number of Samples per
Symbols now is 16. This block, calculates
and selects the best samples for each
symbol.

Variable
ID: rrc_taps_for_manchester
Value: [rrc_taps[n] - rrc_...

Bi-phase Coding:
-logic 0 -> [-1,1]
or a e(t) = d(t) - d(t - td/2) = -1 -1 = -2 < 0 -> logic 0

-logic 1 -> [1,-1]
or e(t) = d(t) - d(t - td/2) = 1 - (-1) = + 2 > 0 -> logic 1

where td = one data clock period,
at 1/1187.5 s.

So, working on samples(in a
Sample Rate of 19.000S/s) and
a total bit rate of 2375 bits/s, we
know that every 8 samples, a new impulse occurs.
So, the difference between the two halves of a
Manchester Symbol, are 8 samples apart.
So, in samples:
Bi-PhaseSymbol = value[now] - value[now + 8].
This formula represents a bi-Phase symbol,
according to the
equations, causing a Baud reduction to half
This result is the convolution of
the RRC filter with one Manchester-encoded bit.

RRC Filter Taps
ID: rrc_taps
Gain: 37
Sample Rate (Hz): 19k
Symbol Rate (Hz): 2.375k
Excess BW: 1
Num Taps: 528

The more taps we use, the more accurate the
Matched Filter becomes.
The FM-RDS Transmitters, use Root Raise Cosine Filter
for Spectral Efficiency(more data in the same bandwidth)
and for the reduction of ISI. Now, we need also to design a proper
Matched Filter , that shapes these symbols.

QT GUI Constellation Sink
Number of Points: 1.024k
Autoscale: No

QT GUI Eye Sink
Number of Points: 1.024k
Samples per Symbol: 1
Sample Rate: 19k
Autoscale: No

Costas Loop
Loop Bandwidth: 62.8319m
Order: 2

Costas is used to lock onto the carrier frequency
of the FM signal and recover the modulating data.
Generates a local oscillator signal that is in phase
and frequency-locked to the incoming carrier,
which allows for accurate recovery of the RDS data.
Order 2 if the signal is
BPSK modulated.

Constellation Decoder
Constellation Object: ...f70>

BPSK DEMODULATION

Differential Decoder
Coding: Differential
Modulus: 2

Differential Decoding

RDS Decoder
Log: Disable
Debug: Disable

RDS digital data decoding(grouped in Bytes)

RDS Parser
Log: Enable
Debug: Disable
PTY Locale: Europe

Parsing RDS-Bytes.

RDS Panel

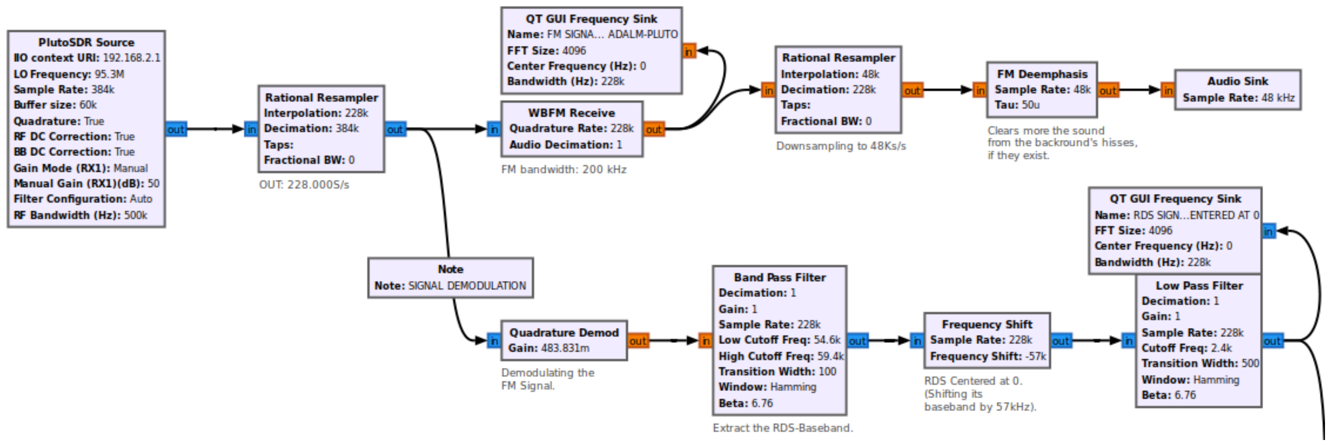
Figure 16: FM-RDS Decoder Flow Graph

Since the Flow Graph seems complicated, we are splitting it into three stages:

1. Signal Demodulation and RDS Bandwidth Isolation, centered at 0
2. Matched Filter and Symbol-Timing Synchronization, Baud Reduction
3. Carrier Recovery, BPSK Demodulation, Differential and RDS Data Decoding

7.1 Signal Demodulation and RDS Bandwidth Isolation

The first stage, demands the demodulation of our frequency modulated signal, playing the sound in our Speakers and isolating just the RDS Bandwidth. A Flow Graph that implements a scenario like this is illustrated in the Figure 17:



PlutoSDR Source
IIO context URI: 192.168.2.1
LO Frequency: 95.3M
Sample Rate: 384k
Buffer size: 60k
Quadrature: True
RF DC Correction: True
BB DC Correction: True
Gain Mode (RX1): Manual
Manual Gain (RX1)(dB): 50
Filter Configuration: Auto
RF Bandwidth (Hz): 500k

Rational Resampler
Interpolation: 228k
Decimation: 384k
Taps:
Fractional BW: 0
OUT: 228.000S/s

QT GUI Frequency Sink
Name: FM SIGNA... ADALM-PLUTO
FFT Size: 4096
Center Frequency (Hz): 0
Bandwidth (Hz): 228k

WBFM Receive
Quadrature Rate: 228k
Audio Decimation: 1
FM bandwidth: 200 kHz

Rational Resampler
Interpolation: 48k
Decimation: 228k
Taps:
Fractional BW: 0
Downsampling to 48Ks/s

FM Deemphasis
Sample Rate: 48k
Tau: 50u
Clears more the sound from the backround's hisses, if they exist.

Audio Sink
Sample Rate: 48 kHz

Note
Note: SIGNAL DEMODULATION

Quadrature Demod
Gain: 483.831m
Demodulating the FM Signal.

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 228k
Low Cutoff Freq: 54.6k
High Cutoff Freq: 59.4k
Transition Width: 100
Window: Hamming
Beta: 6.76
Extract the RDS-Baseband.

Frequency Shift
Sample Rate: 228k
Frequency Shift: -57k
RDS Centered at 0. (Shifting its baseband by 57kHz).

QT GUI Frequency Sink
Name: RDS SIGN...ENTERED AT 0
FFT Size: 4096
Center Frequency (Hz): 0
Bandwidth (Hz): 228k

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 228k
Cutoff Freq: 2.4k
Transition Width: 500
Window: Hamming
Beta: 6.76

Figure 17: 1st Stage of the RDS Decoding Process

Receiving the Frequency Modulated Signal via **PlutoSDR Source** Block, which is tuning in the frequencies of our desire. A **Rational Resampler** follows to reduce our sampling rate at 228kS/s. Since FM-Signal Bandwidth requires at least 200kHz Bandwidth, this sampling rate meets these specifications. **WBFM Receive** demodulates our signal, and feeds it to another **Rational Resampler**, just to set the new sampling rate that our Sound Card's Hardware Supports(48kS/s).
On the other pattern, we used a **Quadrature Demod** that works similar to WBFM Receiver, but it takes a complex baseband as input, and outputs a signal frequency in relation to the sample rate, multiplied with the gain. The gain can be calculated by this formula:

$$\frac{\text{Sampling Rate}}{2 \times \pi \times \text{FM Deviation}}, \tag{3}$$

where FM maximum allowable Deviation is 75.000Hz. A **BandPassFilter** isolates the RDS signal from the whole demodulated signal. RDS Bandwidth ranges from (54.6-59.4)kHz and it is centered at 57kHz. Shifting the RDS Signal to our base-band(0 Hz), for furthermore processing. A **Low Pass Filter** is used for the better performance of our signal, with cutoff Frequency at 2.4 kHz, since the whole RDS bandwidth is 4.8kHz.
The stage 1 is completed, and we can see it clearly in the Figure 18:

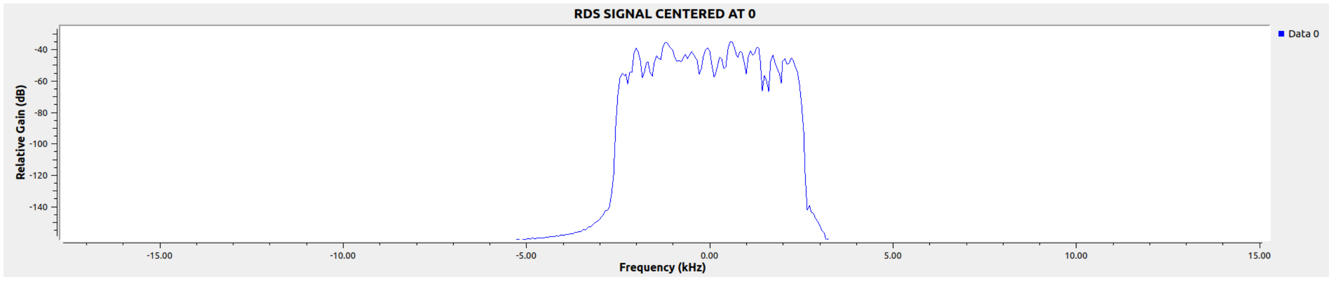
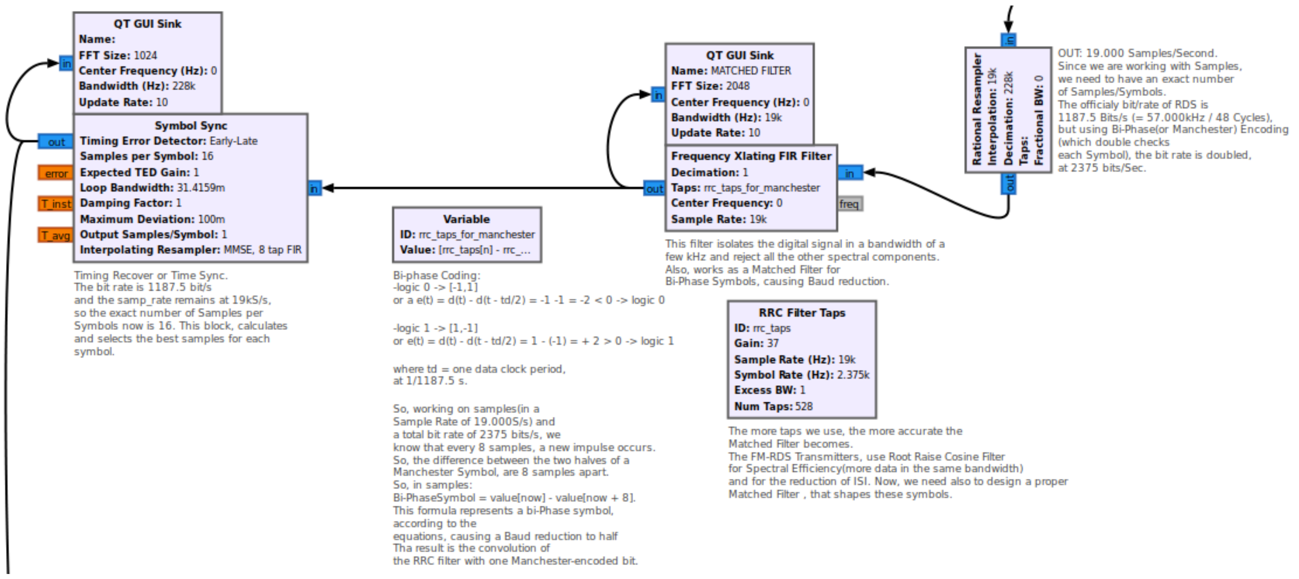


Figure 18: RDS Signal

7.2 Matched Filter and Symbol-Timing Synchronization

Now begins the stage 2(Figure 19), which is the most complicated part of the whole scenario. The RDS transmits



QT GUI Sink
Name:
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 228k
Update Rate: 10

Symbol Sync
Timing Error Detector: Early-Late
Samples per Symbol: 16
Expected TED Gain: 1
Loop Bandwidth: 31.4159m
Damping Factor: 1
Maximum Deviation: 100m
Output Samples/Symbol: 1
Interpolating Resampler: MMSE, 8 tap FIR

Timing Recover or Time Sync.
The bit rate is 1187.5 bit/s
and the samp_rate remains at 19kS/s,
so the exact number of Samples per
Symbols now is 16. This block, calculates
and selects the best samples for each
symbol.

Variable
ID: rrc_taps_for_manchester
Value: [rrc_taps[n] - rrc_...

Bi-phase Coding:
-logic 0 -> [-1,1]
or a e(t) = d(t) - d(t - td/2) = -1 -1 = -2 < 0 -> logic 0

-logic 1 -> [1,-1]
or e(t) = d(t) - d(t - td/2) = 1 - (-1) = + 2 > 0 -> logic 1

where td = one data clock period,
at 1/1187.5 s.

So, working on samples(in a
Sample Rate of 19.000S/s) and
a total bit rate of 2375 bits/s, we
know that every 8 samples, a new impulse occurs.
So, the difference between the two halves of a
Manchester Symbol, are 8 samples apart.
So, in samples:
Bi-PhaseSymbol = value[now] - value[now + 8].
This formula represents a bi-Phase symbol,
according to the
equations, causing a Baud reduction to half
Tha result is the convolution of
the RRC filter with one Manchester-encoded bit.

QT GUI Sink
Name: MATCHED FILTER
FFT Size: 2048
Center Frequency (Hz): 0
Bandwidth (Hz): 19k
Update Rate: 10

Frequency Xlating FIR Filter
Decimation: 1
Taps: rrc_taps_for_manchester
Center Frequency: 0
Sample Rate: 19k

This filter isolates the digital signal in a bandwidth of a
few kHz and reject all the other spectral components.
Also, works as a Matched Filter for
Bi-Phase Symbols, causing Baud reduction.

RRC Filter Taps
ID: rrc_taps
Gain: 37
Sample Rate (Hz): 19k
Symbol Rate (Hz): 2.375k
Excess BW: 1
Num Taps: 528

The more taps we use, the more accurate the
Matched Filter becomes.
The FM-RDS Transmitters, use Root Raise Cosine Filter
for Spectral Efficiency(more data in the same bandwidth)
and for the reduction of ISI. Now, we need also to design a proper
Matched Filter , that shapes these symbols.

Rational Resampler
Interpolation: 19k
Decimation: 228k
Taps:
Fractional BW: 0

OUT: 19.000 Samples/Second.
Since we are working with Samples,
we need to have an exact number
of Samples/Symbols.
The officialy bit/rate of RDS is
1187.5 Bits/s (= 57.000kHz / 48 Cycles),
but using Bi-Phase(or Manchester) Encoding
(which double checks
each Symbol), the bit rate is doubled,
at 2375 bits/Sec.

Figure 19: 2nd Stage of the RDS Decoding Process

the data, using a rate of 1187.5 Bits/sec. Also, the spectral efficiency of BPSK modulation described from this formula:

$$\text{BPSK Spectral Efficiency} = \log_2(\text{M}),$$

where M = 2. So the Spectral Efficiency is 1, and the transmission rate is:

$$\text{Tx Rate} = \text{Symbol Rate} \times \log_2(\text{M}),$$

which results a rate of 1187.5 symbols per second. Note that this number is the one-sixteenth of the Pilot Tone. So, a **Rational Resampler** reduces our Sampling Rate from 228kS/s, to 19kS/s. This allows us to have an exact number of Samples/Symbol, that makes the Symbol Sync more efficient, working on samples.
A **Frequency Xlating FIR Filter** follows, which removes the unwanted components, making more clear and strong the signal that is located in its centered Frequency. Also, this filter is a Matched filter for the Manchester's Encoded Symbols. As we said before, a Bi-Phase Symbol can be represented as:

$$e(t) = d(t) - d(t - td/2) \tag{4}$$

by means, subtracting the second half of the Manchester Symbol, from its first half. In order to make the Matched Filter, we need to transfer the equation (4), in samples. Our Sampling Rate is 19kS/s, and the Symbol Rate is 2375 bits/s. To make it more understandable:

$$\frac{\text{Sampling Rate}}{\text{Symbol Rate}} = \frac{19000\text{Samples per Seconds}}{2375\text{Symbols per Seconds}} = 16\text{Samples per Symbol}$$

That means that every 16 Samples a new Manchester Symbol is sent(or two halves are sent). So, splitting the 16 SpS, into two halves, we are getting 8 samples for the first half(the first bit[it could be 1 or -1]) and another 8 samples for the second half[the second bit[could be 1 or -1 too]]. So, it leads us to the conclusion that the two halves of a Manchester Symbols, in this scenario, are 8 samples apart(Using always 19.000S/s for our sampling rate).
So, the equation (4), in samples can be described by:

$$Bi - Phase - Symbol = fist\_half\ - second\_half = value[now] - value[now + 8samples] \tag{5}$$

In common words, the value of the equation (5) could be either positive(represented as one), or negative (represented as zero). We know that before the Frequency Xlating FIR Filter, the incoming Manchester Symbols require 16 Samples each, and their two halves of them were 8 samples apart. So, the variable **rrc_taps_for_manchester**(which value is the equation(5)), in combination with **RRC Filter Taps** block, gives us the convolution of the RRC Filter for one Manchester Encoded bit, subtracting its second half(which is 8 samples apart), from the first one, resulting a new value for our Bi-Phase Symbols:

- Logic 1: [1 - (-1) = 2]
- Logic 0: [-1 - (+1) = -2]

In common words, now negative means zero and positive means one.
Before this Matched Filter, our Symbols were represented with two consecutive opposite-impulses, and from now on, each one of them is represented by one. This effect also causes a Baud reduction from 2375 impulses/sec, to 1187.5 impulses/second.

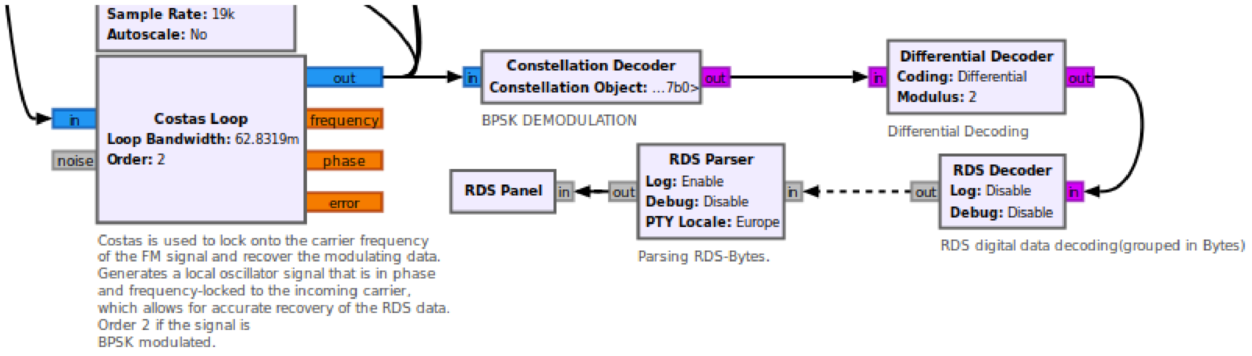
In the block RRC Filter Taps, we have added in the field "Gain", our Qt GUI Range, called RRC_GAIN, so we can manually changing the gain of the Matched Filter, for the better perfomance of BPSK, varying these values from [-1, 1].
The final part in this stage, is the **Symbol Sync**, and its goal is to perform a Timing Recovery algorithm, by selecting the best samples that can perform each impulse/symbol. Its most important fields:

- Samples per Symbol: 16
- Output Samples/Symbol: 1
- Timing Error Detector: Early-Late

This block is performing the timing synchronization needed so that the signal is sampled at exactly the right moment in time, which is when each Bi-Phase's symbol/impulse is at its max value. Early-Late algorithm for Timing Error Detector, seems working good!

7.3 Carrier Recovery, BPSK Demodulation, Differential and RDS Data Decoding

Now, things are not complicated as before. After timing sync, now its time for the carrier recovery. The stage 3,which is the final, is illustrated in Figure 20. A **Costas Loop** Block is used to remove any residual frequency


Sample Rate: 19k
Autoscale: No

Costas Loop
Loop Bandwidth: 62.8319m
Order: 2

Costas is used to lock onto the carrier frequency of the FM signal and recover the modulating data. Generates a local oscillator signal that is in phase and frequency-locked to the incoming carrier, which allows for accurate recovery of the RDS data. Order 2 if the signal is BPSK modulated.

Constellation Decoder
Constellation Object: ...7b0>

BPSK DEMODULATION

Differential Decoder
Coding: Differential
Modulus: 2

Differential Decoding

RDS Decoder
Log: Disable
Debug: Disable

RDS digital data decoding(grouped in Bytes)

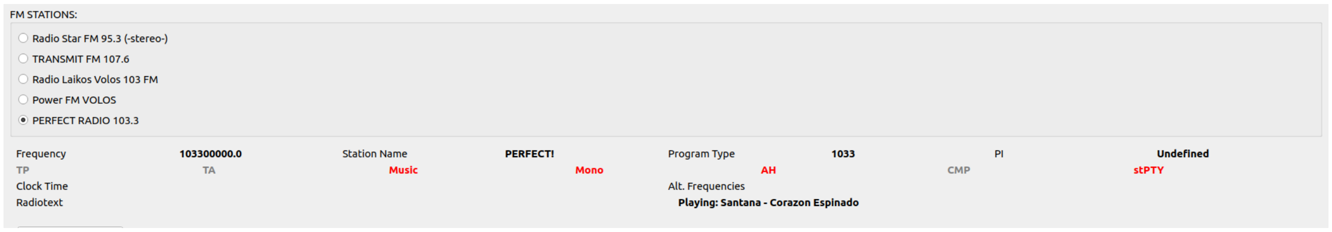
RDS Parser
Log: Enable
Debug: Disable
PTY Locale: Europe

Parsing RDS-Bytes.

RDS Panel

Figure 20: Final Stage of the RDS Decoding Process

offset. While the signal was previously base-banded, there may still exist a small frequency and phase offset that prohibits demodulating the BPSK signal. So Costas Loop, is a PLL circuit, that locks into the carrier frequency, by generating a LO that is locked in phase too. This allows us to continuously track the phase and frequency of the incoming FM signal, even in the presence of phase and frequency variations. **A Constellation Decoder** with "digital.constellation_bpsk().base()" in its only field, is used to perform the BPSK Demodulation in our signal. Feed its output to **Differential Decoder** with modulus 2, to perform the X-OR operation, as we discussed in the Section of Differential Encoding. The **RDS Decoder** follows, which performs the RDS digital data decoding, receiving raw bits and sending completed RDS frames(grouped in Bytes), and the **RDS Parser**, who is receiving these RDS frames and sends the Station/Song names to the **RDS Panel**. So, moving on with the plots:


FM STATIONS:

- ○ Radio Star FM 95.3 (-stereo-)
- ○ TRANSMIT FM 107.6
- ○ Radio Laikos Volos 103 FM
- ○ Power FM VOLOS
- ● PERFECT RADIO 103.3

Frequency	103300000.0	Station Name	PERFECT!	Program Type	1033	PI	Undefined
TP	TA	Music	Mono	AH	CMP	stPTY	
Clock Time				Alt. Frequencies			
Radiotext				Playing: Santana - Corazon Espinado			

Figure 21: RDS Messages in our Panel

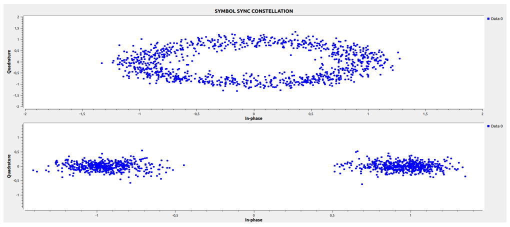


Figure 22: This Figure shows the actual constellations before and after carrier recovery. The received signal with a carrier frequency offset, causes a rotating or spinning constellation. Once Costas Loop lock's in the carrier's frequency and phase, this spinning effect is removed, since the small frequency offsets of our received carrier cause this cyclic-spinning move.

8 FM-RDS Encoding

The implementation of the FM-RDS Encoder, is illustrated in the Figure 23:

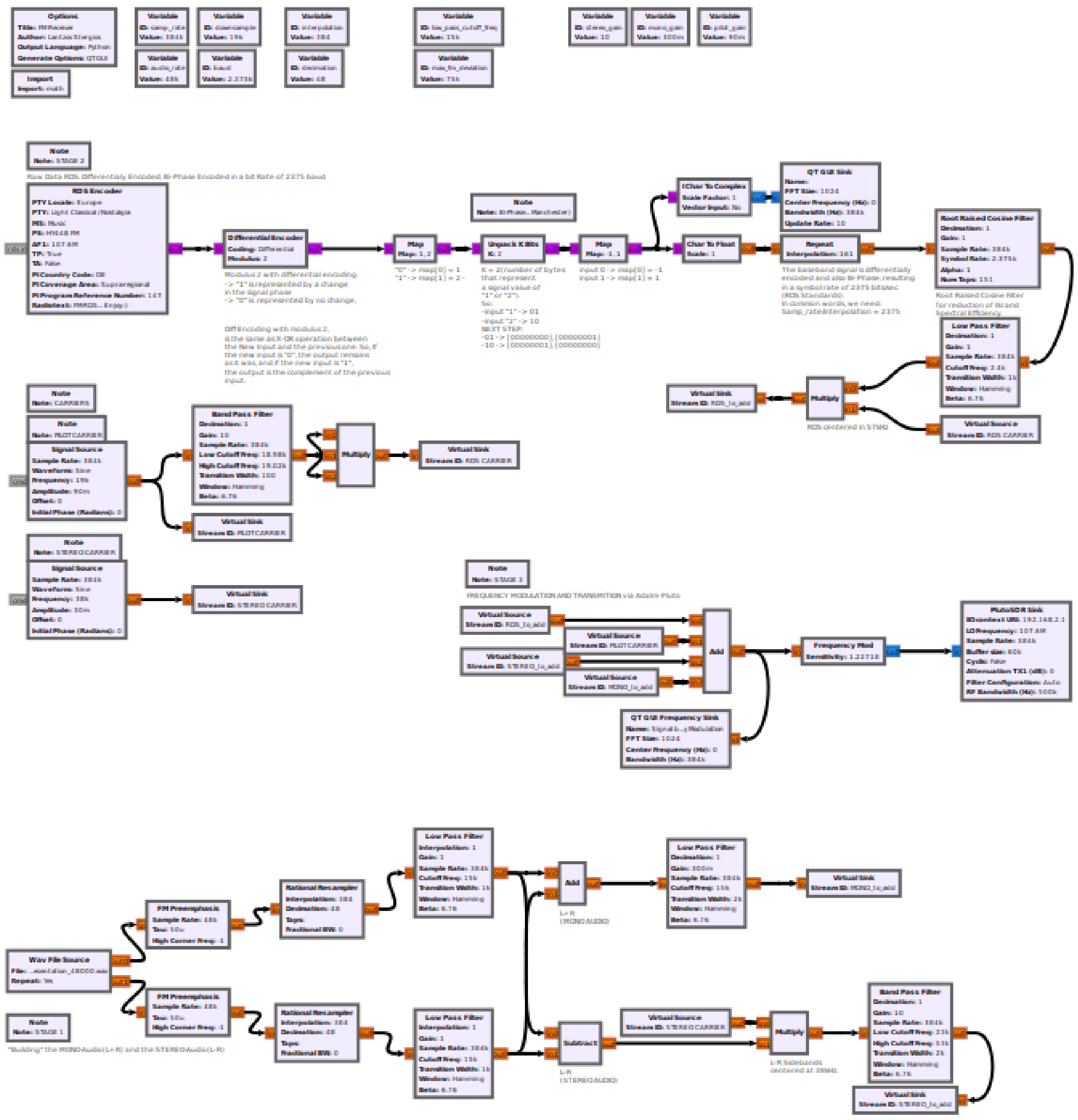


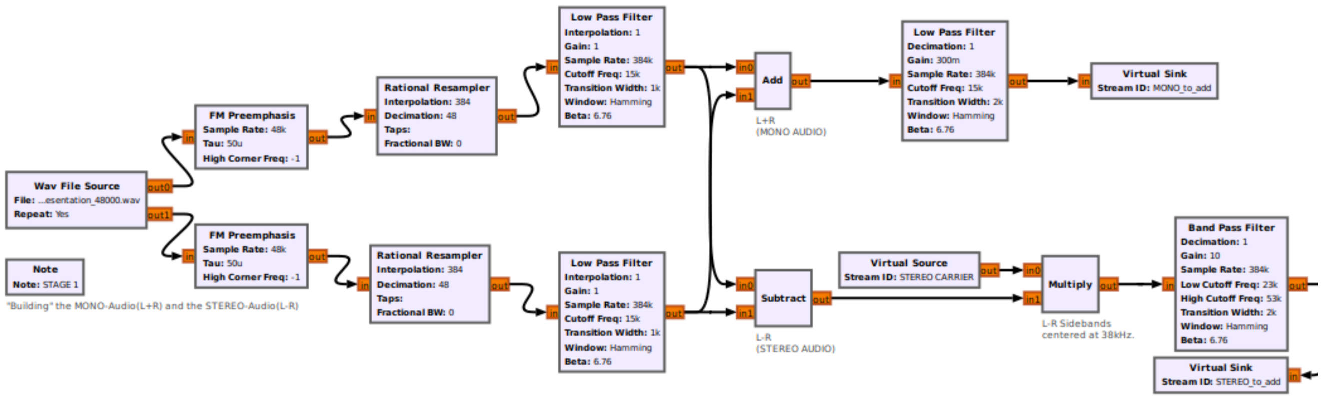
Figure 23: FM-RDS Encoder Flow Graph

For better understanding, we divided this implementation in 3 stages too:

- Building the Mono-Audio and Stereo-Audio
- RDS Encoding, Differential Encoded, Shaped Filter
- Frequency Modulation of our Signal and Transmit via PlutoSDR

8.1 Building the Mono-Audio and Stereo Audio

The first step, as you can see from the Figure 24, is to build the Mono and Stereo Audio.


Wav File Source
File: ...esentation_48000.wav
Repeat: Yes

Note
Note: STAGE 1

"Building" the MONO-Audio(L+R) and the STEREO-Audio(L-R)

FM Preemphasis
Sample Rate: 48k
Tau: 50u
High Corner Freq: -1

Rational Resampler
Interpolation: 384
Decimation: 48
Taps:
Fractional BW: 0

Low Pass Filter
Interpolation: 1
Gain: 1
Sample Rate: 384k
Cutoff Freq: 15k
Transition Width: 1k
Window: Hamming
Beta: 6.76

Add

L+R
(MONO AUDIO)

Low Pass Filter
Decimation: 1
Gain: 300m
Sample Rate: 384k
Cutoff Freq: 15k
Transition Width: 2k
Window: Hamming
Beta: 6.76

Virtual Sink
Stream ID: MONO_to_add

Subtract

L-R
(STEREO AUDIO)

Virtual Source
Stream ID: STEREO CARRIER

Multiply

L-R Sidebands
centered at 38kHz.

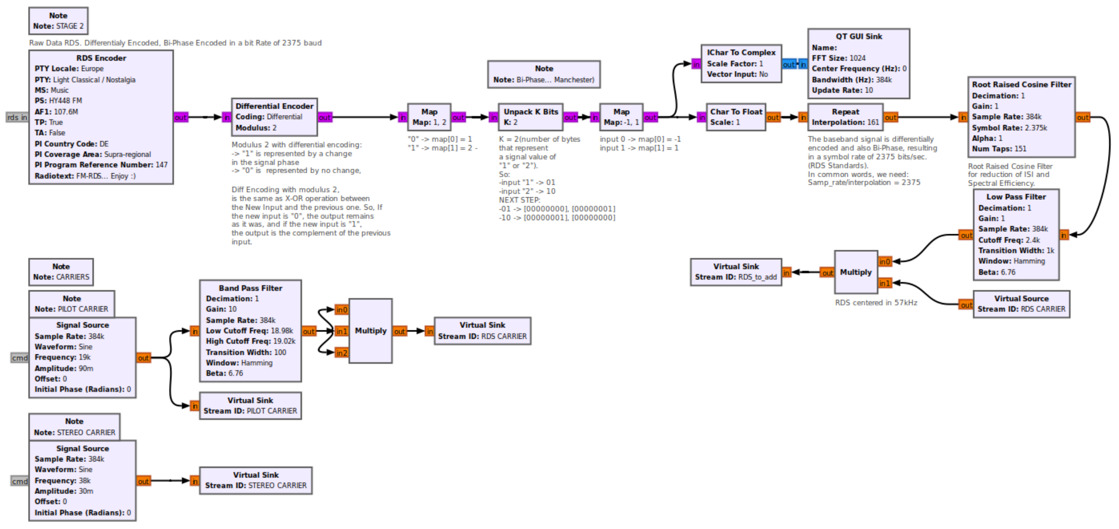
Band Pass Filter
Decimation: 1
Gain: 10
Sample Rate: 384k
Low Cutoff Freq: 23k
High Cutoff Freq: 53k
Transition Width: 2k
Window: Hamming
Beta: 6.76

Virtual Sink
Stream ID: STEREO_to_add

Figure 24: Mono and Stereo Audio

Used a **Wav File Source,** that's taking a .wav file and outputs a float stream. **FM Preemphasis** added for both Mono and Stereo Audio, since its reduces the high frequency components, making the sound little bit clear. **Both Rational Resamplers**, increases our sampling rate from 48kS/s to 384kS/S, which will be our sampling rate in the entire flow-graph. Also, the same **Low Pass Filters** added, with cutoff Frequencies at 15kHz(Mono Audio, and Stereo's each Side-band, have a bandwidth of 15 kHz). The **Add** block, outputs the L+R(which is the Mono Audio), and feeds its output to a **Low Pass Filter**, with cutoff Frequency at 15 kHz. Now, the part of the Mono Audio is done.
The **Subtract** block, builds the L-R Stereo side-bands, and its **multiplied** with the **Stereo Carrier** , in purpose to be centered at 38kHz. Using a **BandPass Filter**, with Low cutoff Frequency at 23kHz and High cutoff Frequency at 53 kHz, the Stereo Audio with its two side-bands is also ready.

8.2 RDS Encoding, Differential Encoding, Manchester Encoding, Shaped Filter


Note
Note: STAGE 2

Raw Data RDS. Differentialy Encoded, Bi-Phase Encoded in a bit Rate of 2375 baud

RDS Encoder
PTY Locale: Europe
PTY: Light Classical / Nostalgia
MS: Music
PS: HY448 FM
AF1: 107.6M
TP: True
TA: False
PI Country Code: DE
PI Coverage Area: Supra-regional
PI Program Reference Number: 147
Radiotext: FM-RDS... Enjoy :)

Differential Encoder
Coding: Differential
Modulus: 2

Modulus 2 with differential encoding:
-> "1" is represented by a change
in the signal phase
-> "0" is represented by no change,

Diff Encoding with modulus 2,
is the same as X-OR operation between
the New Input and the previous one. So, If
the new input is "0", the output remains
as it was, and if the new input is "1",
the output is the complement of the previous
input.

Map
Map: 1, 2

"0" -> map[0] = 1
"1" -> map[1] = 2 -

Note
Note: Bi-Phase... Manchester)

Unpack K Bits
K: 2

K = 2(number of bytes
that represent
a signal value of
"1" or "2").
So:
-input "1" -> 01
-input "2" -> 10
NEXT STEP:
-01 -> [00000000], [00000001]
-10 -> [00000001], [00000000]

Map
Map: -1, 1

input 0 -> map[0] = -1
input 1 -> map[1] = 1

IChar To Complex
Scale Factor: 1
Vector Input: No

QT GUI Sink
Name:
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 384k
Update Rate: 10

Char To Float
Scale: 1

Repeat
Interpolation: 161

The baseband signal is differentially
encoded and also Bi-Phase, resulting
in a symbol rate of 2375 bits/sec.
(RDS Standards).
In common words, we need:
Samp_rate/interpolation = 2375

Root Raised Cosine Filter
Decimation: 1
Gain: 1
Sample Rate: 384k
Symbol Rate: 2.375k
Alpha: 1
Num Taps: 151

Root Raised Cosine Filter
for reduction of ISI and
Spectral Efficiency.

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 384k
Cutoff Freq: 2.4k
Transition Width: 1k
Window: Hamming
Beta: 6.76

Multiply

RDS centered in 57kHz

Virtual Sink
Stream ID: RDS_to_add

Virtual Source
Stream ID: RDS CARRIER

Note
Note: CARRIERS

Note
Note: PILOT CARRIER

Signal Source
Sample Rate: 384k
Waveform: Sine
Frequency: 19k
Amplitude: 90m
Offset: 0
Initial Phase (Radians): 0

Band Pass Filter
Decimation: 1
Gain: 10
Sample Rate: 384k
Low Cutoff Freq: 18.98k
High Cutoff Freq: 19.02k
Transition Width: 100
Window: Hamming
Beta: 6.76

Multiply

Virtual Sink
Stream ID: RDS CARRIER

Virtual Sink
Stream ID: PILOT CARRIER

Note
Note: STEREO CARRIER

Signal Source
Sample Rate: 384k
Waveform: Sine
Frequency: 38k
Amplitude: 30m
Offset: 0
Initial Phase (Radians): 0

Virtual Sink
Stream ID: STEREO CARRIER

Figure 25: RDS Encoding, Differential and Manchester Encoding

The RDS Data for Transmission(RadioText, StationName et.c), are set in the **RDS Encoder** Block. **Differential Encoder** performs the differential Encoding, using modulus 2(X-OR). Now, the block **Map** is working like this:

- If the input is 0, it maps 0 at map[0], which is 1
- If the input is 1, it maps 1 at map[1], which is 2

Unpack K Bits , with K = 2, stands for the representation of the input bit value with 2 bytes. In common words:

- If input is 1, represents 1 as [00000000], [00000001]
- If input is 2, represents 2 as [00000001], [00000000]

The last **Map Block**, does the Bi-Phase(or Manchester) encoding. More specifically:

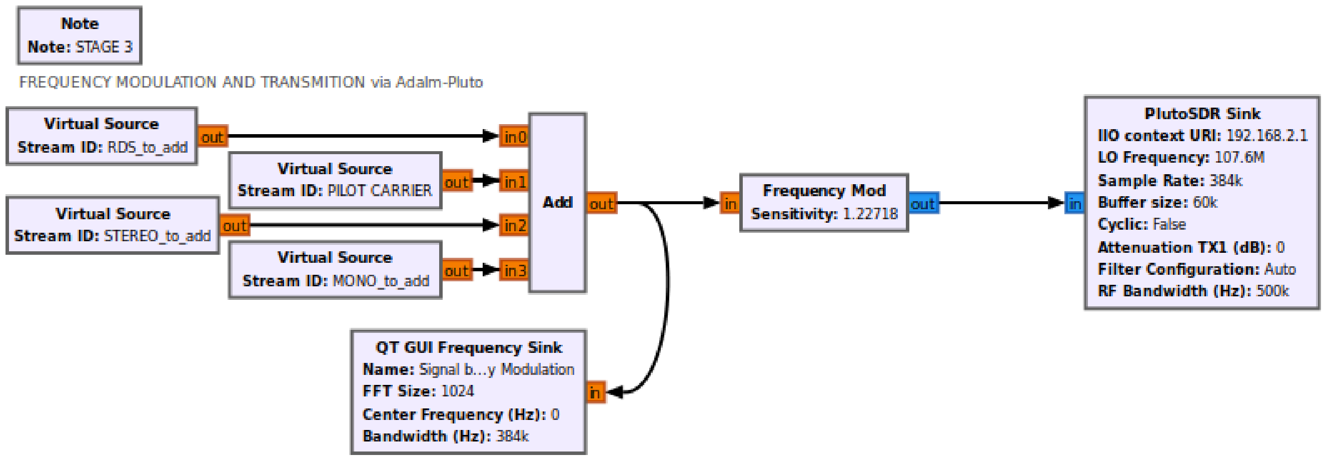
- If input-Byte-value is 0, represents 0 as map[0] = -1
- If input-Byte-value is 1, represents 1 as map[1] = 1

So, in conclusion, a binary value "0" is represented as [-1,1], and the binary value "1" is represented as [1,-1]. This is the Manchester Encoding algorithm. The **Repeat** interpolation, defines the ratio that these symbols are sent. RDS bit rate is 1187.5 bit/s, but the Manchester encoding, doubles this rate, since each symbols is represented by two bits,resulting a new rate of 2375 Bits/s. We need to find the proper value to interpolate. Since we are sampling at 384kS/s, and the bit rate is 2375 Bits/s, the number we are searching for is:

$$\frac{\text{Sampling Rate}}{\text{X}} = \text{Symbol Rate}, \tag{6}$$

,providing to us a result of 161. Next, we are using a **Root Raised Cosine Filter**, for minimizing the ISI, and its output is fed to a **Low Pass Filter**, with cutoff Frequency of 2.4kHz, since the whole bandwidth of RDS is 4.8kHz. **Multiplying** the result of the Low Pass, with the **57kHz carrier**(by tripling the Pilot Tone), centers our RDS to 57kHz. Now, the RDS is ready to be added into the whole signal.

8.3 Frequency Modulation of our Signal and Transmit via PlutoSDR



Note: STAGE 3

Virtual Source — Stream ID: RDS_to_add
Virtual Source — Stream ID: PILOT CARRIER
Virtual Source — Stream ID: STEREO_to_add
Virtual Source — Stream ID: MONO_to_add

Frequency Mod — Sensitivity: 1.22718

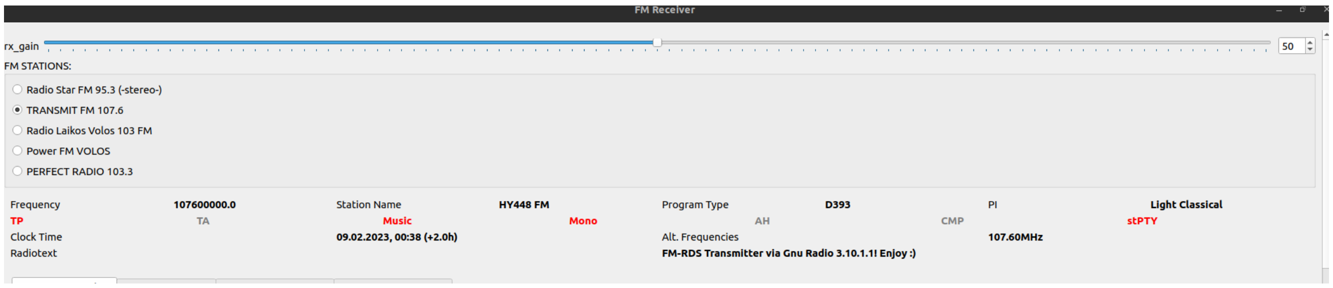
PlutoSDR Sink
IIO context URI: 192.168.2.1
LO Frequency: 107.6M
Sample Rate: 384k
Buffer size: 60k
Cyclic: False
Attenuation TX1 (dB): 0
Filter Configuration: Auto
RF Bandwidth (Hz): 500k

QT GUI Frequency Sink
Name: Signal b...y Modulation
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 384k

Figure 26: Frequency Modulation and Transmit

The last step, is to add all these frequency carriers, and modulates them into one, creating the FM-Signal. The **Frequency Mod** Block, modulates our signal in frequency, using the formula:

$$Sensitivity = \frac{2 \times \pi \times 75.000}{384.000}, \tag{7}$$



FM Receiver

rx_gain 50

FM STATIONS:

- Radio Star FM 95.3 (-stereo-)
- [✓ TRANSMIT FM 107.6]
- Radio Laikos Volos 103 FM
- Power FM VOLOS
- PERFECT RADIO 103.3

Frequency	107600000.0	Station Name	HY448 FM	Program Type	D393	PI	Light Classical
TP	TA	Music	Mono	AH		CMP	stPTY
Clock Time		09.02.2023, 00:38 (+2.0h)		Alt. Frequencies		107.60MHz	
Radiotext				FM-RDS Transmitter via Gnu Radio 3.10.1.1! Enjoy :)			

Figure 27: Text Messages in RDS Panel, received from the RDS Decoder Flow Graph

where 75.000 is the maximum allowable deviation and 384.000 is our sampling rate. The FM Signal, is fed to PlutoSDR for transmission, in the frequency of 107.6 MHz. The next step, to test this, requires the executing of the RDS Decoder and tuning it, in the frequency of 107.6MHz. Then, we will execute the RDS Encoder, and we will see what happens..

So, in the Figure 27, we can see that the data we set in the RDS Encoder Block, appear instantly in our panel. In the Figure 28, we can see its constellation, which responds to BPSK. Also, we can see the four possible transitions of the BPSK in the eye diagram: from +1 to +1, (ii) from +1 to -1, (iii) from -1 to +1, and (iv) from -1 to -1.

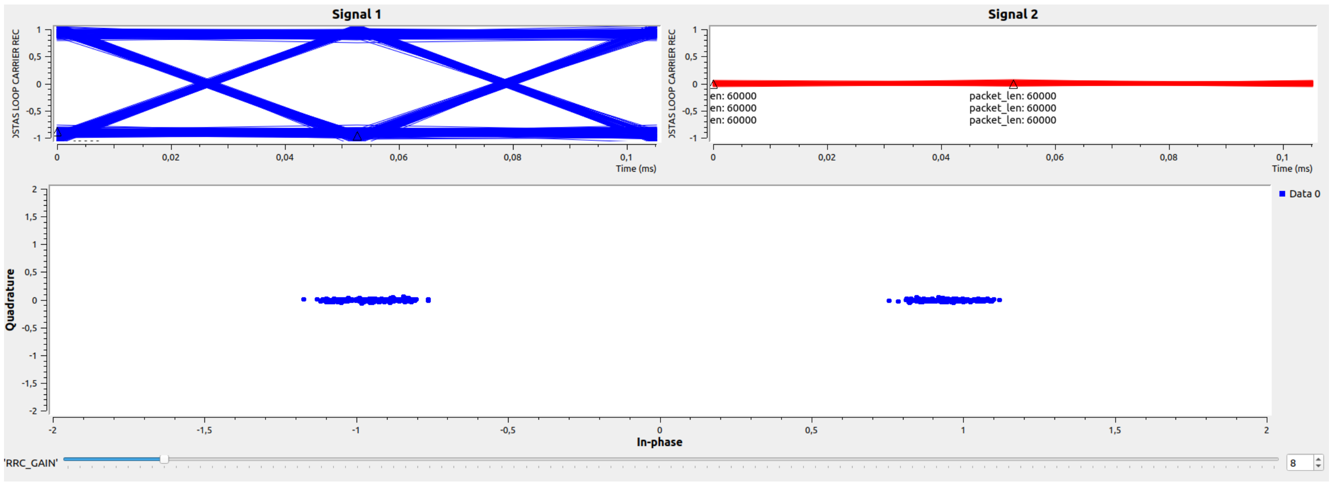


Figure 28: BPSK Constellation

References

[1] Dimitrios Symeonidis, *RDS-TMC Spoofing using GNU Radio.*

[2] Jonathan Ford, 2011, *Simultaneous Digital Demodulation and RDS Extraction of FM Radio Signals.*

[3] J.-M Friedt, April 2, 2017, *Radio Data System (RDS) – investigation of the digital channel used by commercial FM broadcast stations, introduction to error correcting codes.*

[4] Ge Verigy, Liang China *Introduction to FM-Stereo-RDS Modulation.*

[5] Roman Glazkov, *GNU Radio Based RDS Transceiver for Data Communication.*

[6] hi-Yuan Lin, Kuang-Hao Lin, and Shuenn-Yuh Lee' *Digital RDS demodulation in FM subcarrier systems.*

[7] Wikipedia, *Radio Data System.*

[8] Dr. Marc Lichtman, *PySDR: A Guide to SDR and DSP using Python.*

[9] C.S. KOUKOURLIS, *A NEW DIGITAL IMPLEMENTATION OF THE RDS IN THE FM STEREO.*

[10] Carsten Roppel, *AN FM/RDS (RADIO DATA SYSTEM) SOFTWARE RADIO.*