# SDR Assignment number 1 (FM Radio)

**Author: Serafim Ciobanu**

**Date: 31.12.2024**

**Course: Emergent Security Techniques**

**Disclaimer: I will try to explain the way this decoding work, the way I understand, and it might not be the best academical or scientific explanation.**

## Part 1

Description: Design a flow graph that will allow to change the frequency in real-time and listen to a radio station. Additionally, add some visualizations that shows how the radio works.

A diagram of a computer program

Description automatically generated with medium confidence

Above you can see a working example that allows me to have the FM radio as input and decode it correctly that then can be heard as a normal radio player. In this particular case I added the bare minimum that allows me to properly tune to various frequencies, and even change the volume of the outcoming sound.

Here is an image of current configuration

A screenshot of a computer

Description automatically generated

Tuned frequency slider allows me to change to a different frequency and try and “catch” a station, because the FM radio works in a range of “motion” or easier said frequency. The Volume for track is a slider that allows to change the sound level for at least one track, which is currently present. Track in this case is the same as a radio station because we have the tuned radio frequency volume.

Next step is to add visualization to this flow graph.

A diagram of a computer flowchart

Description automatically generated

Here you can see the visualizations I added to each step. I mostly focused on adding frequencies, due to the nature of FM radios, since it is mostly dependent on the frequency synchronization, or the way the frequency varies.

Here is how it looks like:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

I consider that this is enough of a visualization how the values change/appear. The most important one is Time Sink, which for me represents how synchronous the signals are and how exactly they blend in together.

After some more looking into the visualization I might admit that the after last resampling Time Sink graph might look obsolete, however I will leave it in the state it is right now.

The explanation of the blocks and further information will be done after I add the second “channel” for the next part.

## Part 2

Description: Expand the network from part 1 in such a way that you can tune into and listen to two different FM radio stations at the same time. Combine the audio output from both radio stations into a single audio output, in which you can separately set the volume of each radio station.

To accomplish this, my way will be to make an exact copy of the same flow as in the first case, except I might leave the visualizations alone, since in that case they are pretty much similar and the values can be seen in case of one band if needed, and the extra is obsolete.

A diagram of a computer flowchart

Description automatically generated

Here is the flow graph that allows me to listen to 2 radio stations at the same time and tune the frequency and volume for both of them, separately.



Here is an example of values that will work.

## Block explanation

The hardest part is explaining the necessity for all the blocks that are present in this graph, and I will try to explain it as good as I can, because radio topic is not something easy for myself.

A white box with black text

Description automatically generated

First blocks, common for everything is the **Variable** and **Options. Variable** allows me to set up a well-known variable like in programming for the whole graph, which then can be used as a placeholder, by using the value “center\_freq” in the **Throttle** block afterwards.

**Options** block is most things regarding the file itself, which create the Python compiled executable, and the rest of the fields is for specifying who is the author of the file and graph.

A screenshot of a computer

Description automatically generated

A diagram of a computer

Description automatically generated

Another common part of the flows is the **File Source, Throttle, Signal Source** and **QT GUI Range.**

**File source** allows me to select a file that is used as a signal (imagine it would be not a file in .iq format, but a signal receiver that is used as input), and allows me to set up various settings for this audio file.

**Throttle** is meant to limit how much “data” or signal overall is being sent for processing to my CPU, so that it does not get overflooded with data.

A close-up of a card

Description automatically generated

The amount of data used is set to the sample rate of 10 MHz

**Signal Source** is viewed (for me) as a frequency, or signal generator. In order to listen or catch the difference between frequency differences, we need a carrier wave and the FM signal wave itself. That way, we generate a signal that we can operate on and catch various frequencies.

**QT GUI Range** is meant to represent another “variable” for the tuning process. In this case, it represents the “center\_freq” that has to be variable, while used inside the **Signal Source.** Thanks to it, the tuning process takes place and we can think of it as turning the knob on a radio player to catch a frequency where the radio station works.

A screenshot of a computer

Description automatically generated

The default value was set (manually) to the “center\_freq”. The start and stop value is the placeholders for the extremities that I will not go further while looking for a radio station. The step is how much of the frequency is stepped each time I change it in the GUI. I found out that 10kHz is optimal for a step because it allows some fine-tuning of the radio stations and get some better sound quality.

This variable, while being changed, is used in the **Signal Source** block, that is responsible for providing us with actual frequencies.

A screenshot of a computer

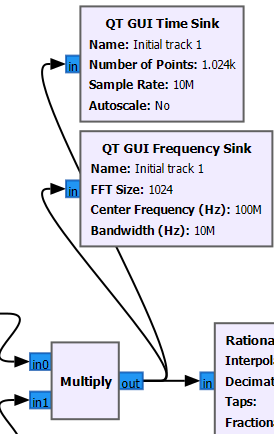
Description automatically generated

Next block is **Multiply** which I can not say a lot about, except for the fact that it blends together the generated FM signal and the source file data and creates a likely symbiosis of carrier wave and FM wave.

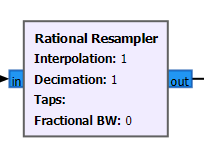
A diagram of a multiply

Description automatically generated

From this block I also have my **frequency** and **time sink** that will show me the initial frequency that I work with.



Next, I have a **Rational Resampler**, which fairly enough in this case might be obsolete, as it does not really have any particular role, since it does not have to “reduce” any sound or frequency to any value.



However, in case of other use cases this can be necessary, because we want to “prepare” our signal before working further on it.

A very interesting and necessary block is the next **Low Pass Filter** which is meant to get rid of unwanted noise, and cut off the high frequencies and hence clearing the FM signal which can be processed further.

A white rectangular sign with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

Values set here are meant to change how clear the sound becomes (in the end), however the value of Cutoff Freq and Transition Width can be played around with. I consider that the values set in my case are more or less adequate and provide clear enough sound, but I mind you that I am not a professional in this topic.

The visualization added afterwards for frequency shows how much values are cut away.

A screenshot of a computer

Description automatically generated

A very important block is **WBFM Receive** which is meant to transform the filtered sounds into a wider frequency difference. But more importantly it is that it takes the sample rate and decimates it to get to the value of the output, for the Audio Sink. It minimizes the general value to a smaller value that can be played/represented.

A close-up of a card

Description automatically generated

It additionally transforms the complex sound to float, and further we work with floats.

The values set are meant to come down to a “playable” sound level, which in this case 10MHz = 10000000 / 208 = 48076 = ~48kHz.

I set a visualization for the frequency after it to see how it changes overall. Instead of WBFM Receive also could be used the **FM Demod**, however it would need more calculations from my side.

A screenshot of a computer

Description automatically generated

A close-up of a diagram

Description automatically generated

As I can see, it spreads the frequency relatively equal along a wider level.

Next, another **Rational Resampler** which is meant usually to regulate the sound, or better said signal to a normal level. It is supposed to work like a gear ratio, because it gets higher values and transforms it into lower values. It is meant to make the demodulated audio signal’s sample rate into something more compatible for audio hardware.

A white rectangular sign with black text

Description automatically generated

However, in this case I left the values default, because it works as is.

A screenshot of a computer screen

Description automatically generated

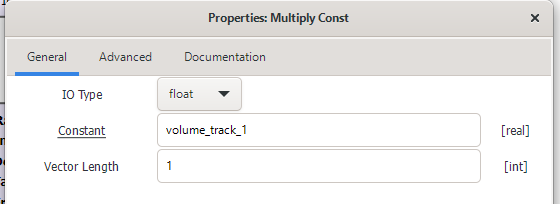
A close-up of a diagram

Description automatically generated

One of the last blocks, responsible for an additional function, of changing the volume of the outcoming sound, is **Multiply Const**.

A screenshot of a computer

Description automatically generated



In this particular case it is added at the end so that the final sound is being manipulated, which is already supposed to be clear enough.

The **QT GUI Range** in this case is the same as the frequency tuning, as it is meant to change the values of how much we amplify the volume.

A screenshot of a computer

Description automatically generated

Stop at 5 is obsolete, as it is too big of a value. The optimal can be 2, or 2.5 to really hear a difference.

The last 2 blocks are **Add** and **Audio Sink**. **Add** block is meant to represent a combination of 2 audio streams that we get as a result.

The **Audio Sink** is the latest step that actually plays the sound via the hardware, and in this case can output 2 or more streams that we get at the end.

This seems to be everything related to this assignment, and I tried to do the most of my knowledge to get this to work.