Mendel University in Brno

Wireless Signal Processing in GNU Radio Environment
Study text

Remigiusz Olejnik
West Pomeranian University of Technology in Szczecin

Project: Innovative Open Source Courses for Computer Science Curriculum

24. 6. 2022

Co-funded by the Erasmus+ Programme of the European Union
West Pomeranian University of Technology Szczecin
UNIVERSITY OF ŽILINA
Mendel University in Brno

Reviewer: Ing. Peter Šarafín, PhD., University of Žilina, Slovakia
Project: Innovative Open Source Courses for Computer Science Curriculum

ISBN 978-80-7509-891-7 (online; pdf)
DOI https://doi.org/10.11118/978-80-7509-891-7

This material teaching was written as one of the outputs of the project "Innovative Open Source Courses for Computer Science Curriculum", funded by the Erasmus+ grant no. 2019-1-PL01-KA203-065564. The project is coordinated by West Pomeranian University of Technology in Szczecin (Poland) and is implemented in partnership with Mendel University in Brno (Czech Republic) and University of Žilina (Slovak Republic). The project implementation timeline is September 2019 to December 2022.

Project information

Project was implemented under the Erasmus+.
Project name: **"Innovative Open Source courses for Computer Science curriculum"**
Project nr: **2019-1-PL01-KA203-065564**
Key Action: **KA2 – Cooperation for innovation and the exchange of good practices**
Action Type: **KA203 – Strategic Partnerships for higher education**

Consortium
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE
MENDELOVA UNIVERZITA V BRNĚ
ŽILINSKÁ UNIVERZITA V ŽILINE

Preface

The book is a teaching resource produced as part of the project "Innovative Open Source Courses for Computer Science". It is dedicated to teachers, students and people interested in gaining or extending their knowledge in the use Wireless Signal Processing in GNU Radio Environment.

GNU Radio is a free and open source software development toolkit that provides signal processing blocks to implement Software Defined Radios (SDRs). It is a highly modular, "flowgraph"-oriented framework, that comes with a large set of existing blocks. GNU Radio can be used with readily-available low-cost external RF hardware (such as RTL-SDR or HackRF) to create software-defined radios. It is a great tool to be used at any university course related with wireless/radio signal processing. Presented examples could be easily built and run and form a solid base for further experimentation.

Acknowledgments

At this point, I would like to express my gratitude to Ing. Peter Šarafín, PhD., for valuable comments and suggestions on this text.

Contents

What is GNU Radio and Why it is Worth to Use it

GNU Radio [1] is a free, open source, universal software toolkit based on C++ and Python, that enables DSP applications to be created without knowledge of a programming language. GNU Radio provides just signal processing blocks, thus allowing to implement software-defined radios.

It supports numerous devices and external interfaces, so it can be used with low cost RF hardware (such as RTL-SDR receiver or HackRF transceiver) allowing to create Software Defined Radios, or even without any piece of hardware in a simulation-like manner. GNU Radio is very popular in wide range of applications, starting from academia and R&D through industry and government to hobbyist environments. It is easy to deploy in all the applications demanding on wireless communications research.

In traditional approach, the RF engineer developed radio communications devices by creating a specific circuits for detection of one RF signal class. It had to be implemented as a specific integrated circuit that would be able to make decoding and/or encoding process of that particular transmission possible and at the end to debug all these steps using costly equipment.

Software-Defined Radio (SDR) approach takes the analog signal processing and moves it, as far as it is physically and economically possible and feasible, to process the RF signals directly on a computer using specialized software algorithms instead of using costly hardware.

It is of course possible to utilize a radio device which is connected to the computer, in a program that is composed of numerous signal processing algorithms merged together. However it is a waste of time and energy to re-implement basic and well-known operations on radio signals like filtering or mixing. It is much more efficient to use highly optimized and peer-reviewed algorithms' implementations rather than writing them from scratch. Moreover the program is scalable on multi-core architecture and run on an energy-efficient embedded devices as well. And there is no need to create own GUIs. It is GNU Radio, a framework powering the world of RF signal processing world today.

1.1 A Flowgraph-Based Approach to Digital Signal Processing

GNU Radio offers a universal software library for different devices with easy ways to expand it. A „GNU Radio Companion" (GRC) is an IDE-like software environment that simplifies creating and running so-called **flowgraphs**, a complete graph of blocks. Fig. 1.1 shows an example of flowgraph.

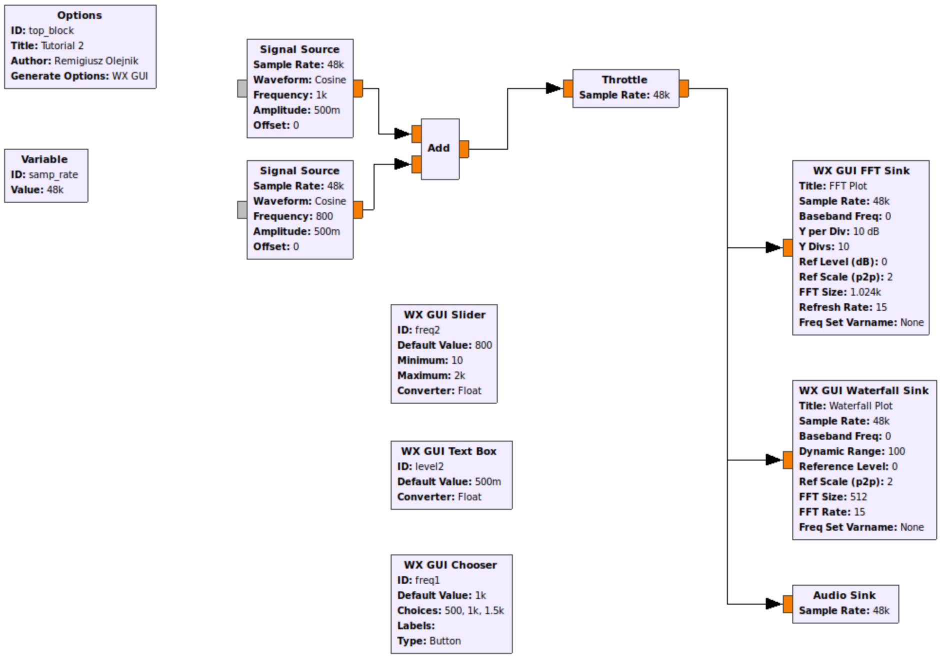

Figure 1.1: GNU Radio — seven blocks connected together form a flowgraph

However, GNU radio programs can be run with or without user interface and also standalone without GRC. General structure of a flowgraph is based on the flow of the signal from a ***Source*** to a ***Processing Block(s)*** and then to a ***Sink***. Sources and/or sinks can be SDR devices, files, audio devices and even network services such as TCP/UDP that allow to send signals over the networks. Two groups of the blocks (***Sources*** and ***Sinks***) building flowgraphs are presented in a subchapter 1.2.

GNU Radio framework allows developing these processing blocks and creating flowgraphs, which comprise radio processing applications. Existing blocks could be combined into a high-level flowgraph that does something as complex as receiving digitally modulated signals.

In GNU Radio framework individual processing stages such as filtering, correction, analysis, detection etc. are represented by processing blocks; these blocks are connected using simple flow-indicating arrows — see example in Fig. 1.2.

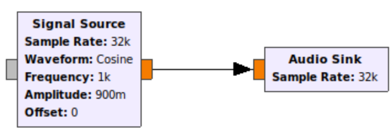

*Figure 1.2: GNU Radio — two blocks (**Signal Source** and **Audio Sink**) connected with an arrow showing flow of the signal data*

1.2 Most Popular GNU Radio Blocks

GNU Radio comes with a large set of existing blocks. Most popular ones are presented below and an index to all of them can be found in ***Block Docs***.

- Waveform Generators
  - Constant Source
  - Noise Source
  - Signal Source (e.g. Sine, Square, Saw Tooth)
- Modulators
  - AM Demod
  - Continuous Phase Modulation
  - PSK Mod / Demod
  - GFSK Mod / Demod
  - GMSK Mod / Demod
  - QAM Mod / Demod
  - WBFM Receive
  - NBFM Receive
- Instrumentation
  - Constellation Sink
  - Frequency Sink
  - Histogram Sink
  - Number Sink
  - Time Raster Sink
  - Time Sink
  - Waterfall Sink
- Math Operators
  - Abs
  - Add
  - Complex Conjugate
  - Divide
  - Integrate
  - Log10
  - Multiply
  - RMS
  - Subtract
- Channel Models
  - Channel Model
  - Fading Model
  - Dynamic Channel Model
  - Frequency Selective Fading Model
- Filters
  - Band Pass / Reject Filter
  - Low / High Pass Filter
  - IIR Filter
  - Generic Filterbank
  - Hilbert
  - Decimating FIR Filter
  - Root Raised Cosine Filter
  - FFT Filter
- Fourier Analysis
  - FFT
  - Log Power FFT
  - Goertzel (Resamplers)
  - Fractional Resampler
  - Polyphase Arbitrary Resampler
  - Rational Resampler (Synchronizers)
  - Clock Recovery MM
  - Correlate and Sync
  - Costas Loop
  - FLL Band-Edge
  - PLL Freq Det
  - PN Correlator
  - Polyphase Clock Sync

Using these blocks, many standard tasks, like signal normalization, synchronization, measurements, and visualization can be done by just connecting the appropriate block to your signal processing flow graph.

It is also possible to write out own blocks, that either combine existing blocks with some intelligence to provide new functionality together with some logic, or to combine operations on the input and output data. Thus, GNU Radio is mainly a framework for the development of signal processing blocks and their interaction. It comes with an extensive standard library of blocks, and there are a lot of systems available that a developer might build upon. However, GNU Radio itself is not a software that is ready to do something specific – it's the user's job to build something useful out of it, though it already comes with a lot of useful working examples. Think of it as a set of building blocks [2].

1.2.1 Default Blocks

The most important blocks that are automatically created in every GNU Radio project are: Variable **samp_rate** and Options **top_block**.

The Variable **samp_rate** block sets the global sampling rate for the whole project, the default here is 32000 samples/second, but the value can be adjusted to meet the needs of specific project. All new blocks that will be added to the project later will use this sampling rate as the default value, see Fig. 1.3.

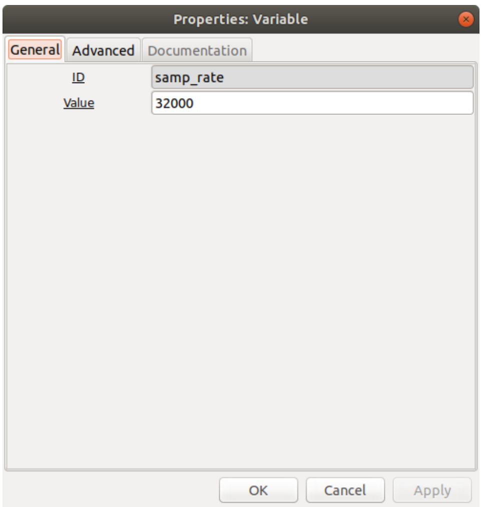

Figure 1.3: GNU Radio — Variable ***samp_rate*** *block*

In the Options **top_block** block, the values that are global for the project are specified: *Title*, *Author*, *Description*, *Canvas Size* (width and length of the workspace in pixels), *Generate Options* (QT GUI, WX GUI, No GUI (No GUI should be used), Hier Block (a hierarchical block without GUI, which can be included in other projects), Hier Block with QT GUI (hierarchical block with QT GUI, which can be included in other projects)), *Run* (Autostart or Off), *Realtime Scheduling* (On or Off), *QSS Theme* (path to a .qss theme file that defines how the project's GUI should look like), see Fig. 1.4.

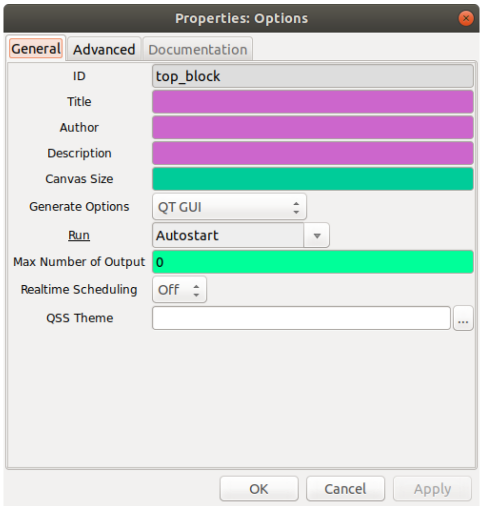

*Figure 1.4: GNU Radio — Options **top_block** block*

Variables: any variables can be created with global visibility for the current project. It is done similar to the **Variable samp_rate** block.

1.3 Signal Data Types in GNU Radio

Every signal processing block in GNU Radio has an input/output port(s) that are able to receive/send signal(s) of predefined data type(s). For each signal data type GNU Radio shows the ports colored in the predefined way. The data types can be found in GNU Radio Companion by clicking **Help** -> **Types**. The Fig. 1.5 shows all the signal data types along with the colors associated with them.

The most often used signal data types are blue **Complex Float 32** and orange **Float 32**. Common signals are also yellow **Integer 16** and purple **Integer 8**. Two ports of different blocks have to be compatible in sense of signal data types. That means, that only **Float 32** output port of one processing block can be connected to **Float 32** input port of another processing block. If the ports are incompatible, the arrow connecting two block will be red, indicating a data mismatch error. It could be resolved by changing the signal data types at one of the blocks.

1.4 Sources and Sinks in GNU Radio

As source blocks in GNU Radio we assume the blocks that provide data in various formats such as **Complex**, **Complex Float**, **Float**, **Integer** or **Byte**. The format in which the data is provided at the output can be selected in the options of the block and is indicated by the color of the small rectangle on the right side of the respective block. Only blocks that use the same data format can be connected to each other. If this is not the case, the arrows connecting the blocks to each other are displayed in red and the program cannot

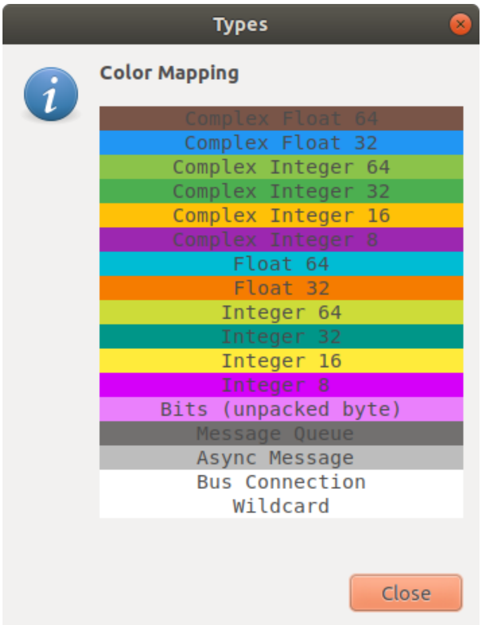

Figure 1.5: GNU Radio signal data types

be executed until the error is corrected. If blocks are to be connected to one another for which it is not possible to select the same data format in the options, the data format converting block has to be inserted. These can be found in the right panel of GRC under ***Type Converters***.

1.4.1 GNU Radio Sources

1. Null Source — Fig. 1.6
2. Noise Source — Fig. 1.7
3. Signal Source — Fig. 1.8
4. File Source — Fig. 1.9
5. TCP Source — Fig. 1.10
6. UDP Source — Fig. 1.11
7. Audio Source — Fig. 1.12
8. WAV File Source — Fig. 1.13
9. UHD: USRP Source — Fig. 1.14
10. osmocom Source — Fig. 1.15
11. RTL-SDR Source — Fig. 1.16
12. Funcube Dongle — Fig. 1.17

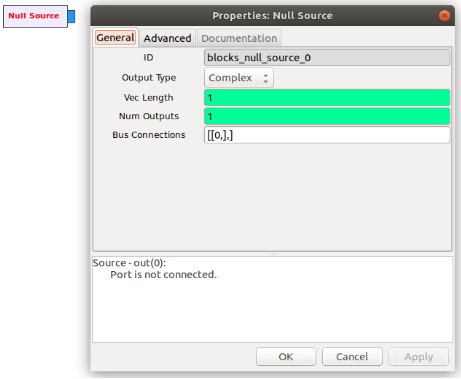

Figure 1.6: GNU Radio sources — ***Null Source***

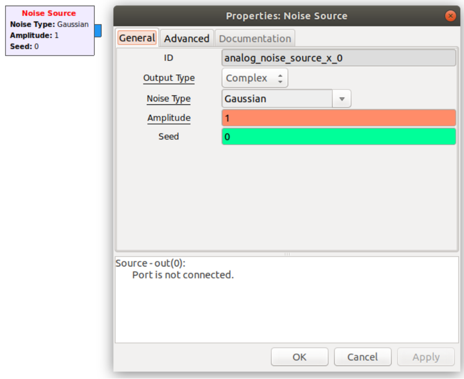

Figure 1.7: GNU Radio sources — ***Noise Source***

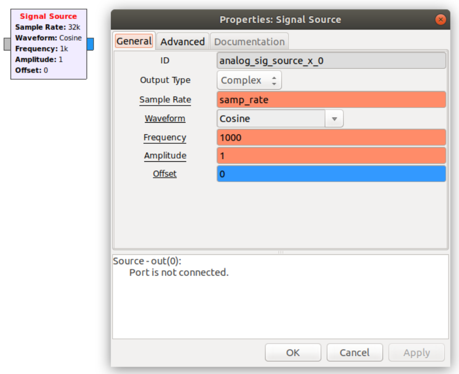

Figure 1.8: GNU Radio sources — ***Signal Source***

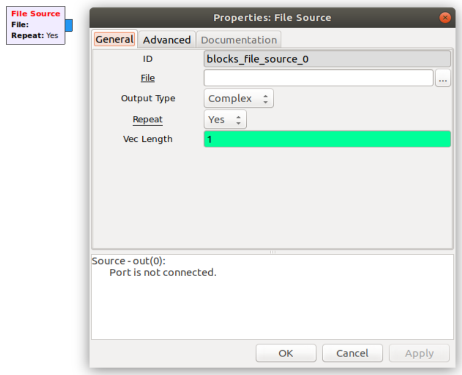

Figure 1.9: GNU Radio sources — ***File Source***

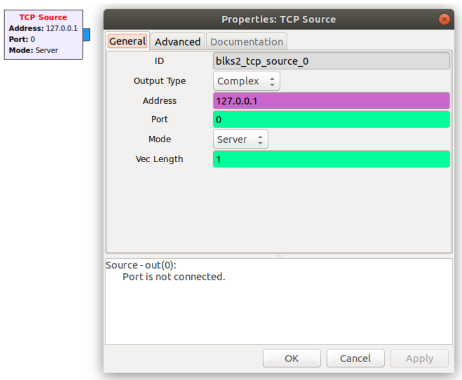

Figure 1.10: GNU Radio sources — ***TCP Source***

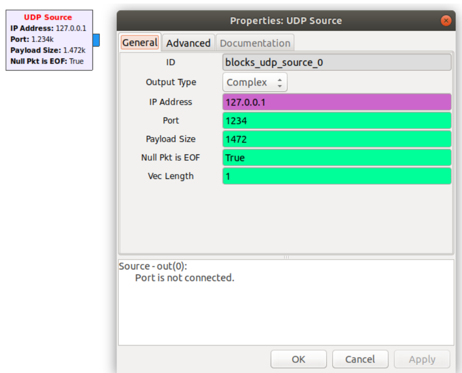

Figure 1.11: GNU Radio sources — ***UDP Source***

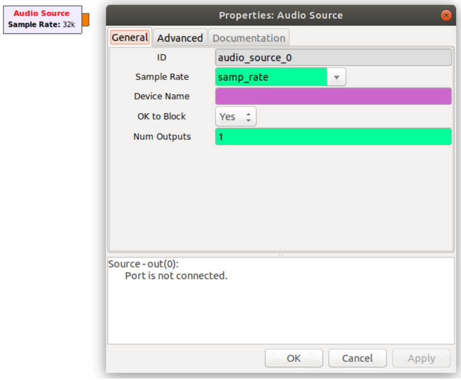

*Figure 1.12: GNU Radio sources — **Audio Source***

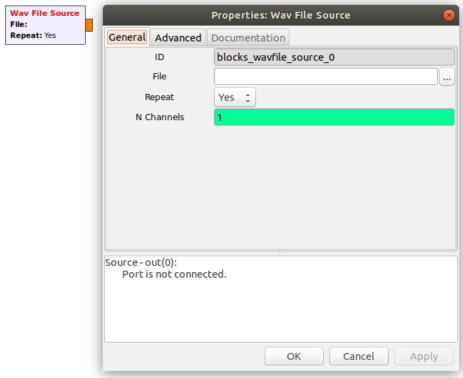

*Figure 1.13: GNU Radio sources — **WAV File Source***

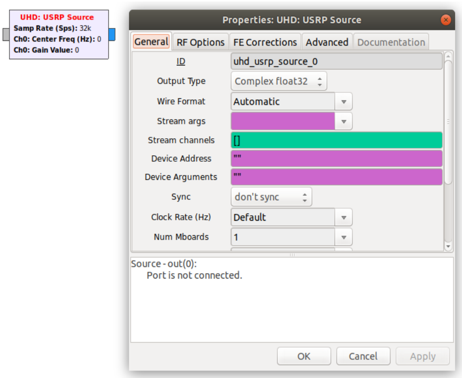

*Figure 1.14: GNU Radio sources — **UHD: USRP Source***

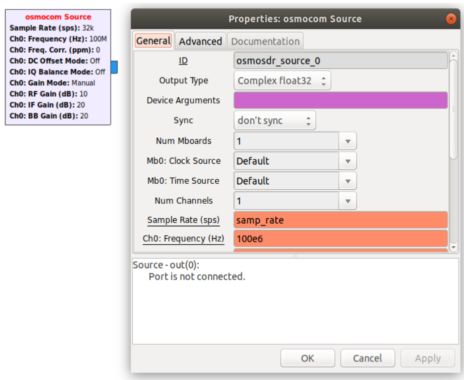

*Figure 1.15: GNU Radio sources — **osmocom Source***

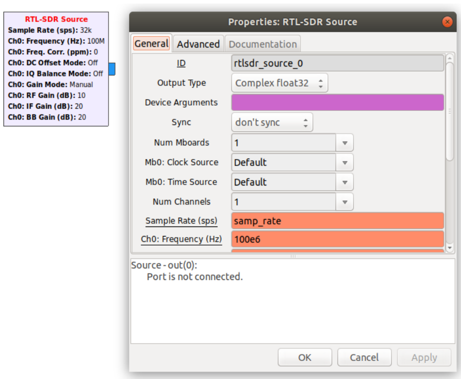

*Figure 1.16: GNU Radio sources — **RTL-SDR Source***

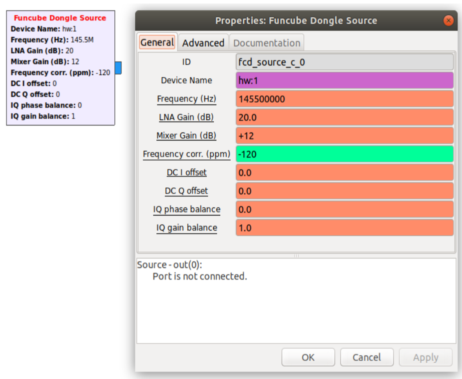

*Figure 1.17: GNU Radio sources — **Funcube Dongle Source***

1.4.2 GNU Radio Sinks

1. Null Sink — Fig. 1.18
2. File Sink — Fig. 1.19
3. TCP Sink — Fig. 1.20
4. TCP Server Sink — Fig. 1.21
5. UDP Sink — Fig. 1.22
6. Audio Sink — Fig. 1.23
7. WAV File Sink — Fig. 1.24
8. UHD: USRP Sink — Fig. 1.25
9. osmocom Sink — Fig. 1.26

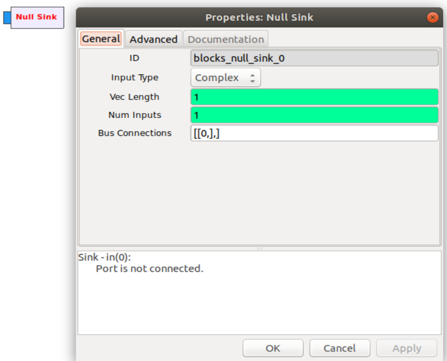

Figure 1.18: GNU Radio sinks — ***Null Sink***

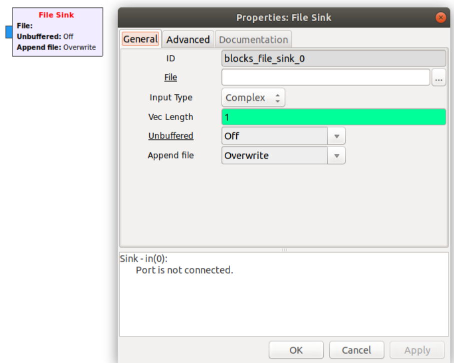

Figure 1.19: GNU Radio sinks — ***File Sink***

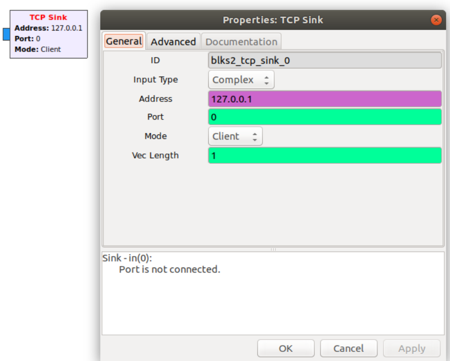

Figure 1.20: GNU Radio sinks — ***TCP Sink***

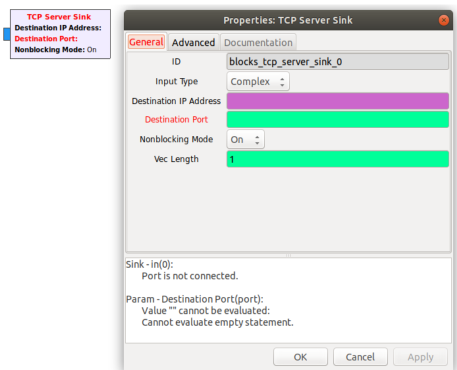

Figure 1.21: GNU Radio sinks — ***TCP Server Sink***

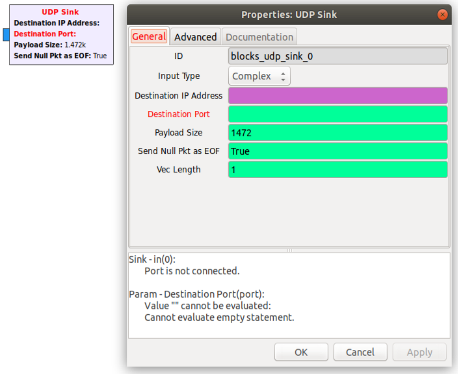

Figure 1.22: GNU Radio sinks — ***UDP Sink***

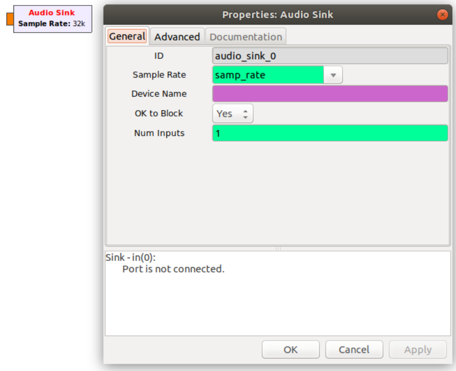

Figure 1.23: GNU Radio sinks — ***Audio Sink***

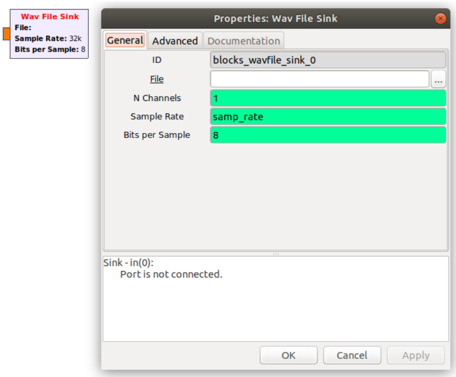

Figure 1.24: GNU Radio sinks — ***WAV File Sink***

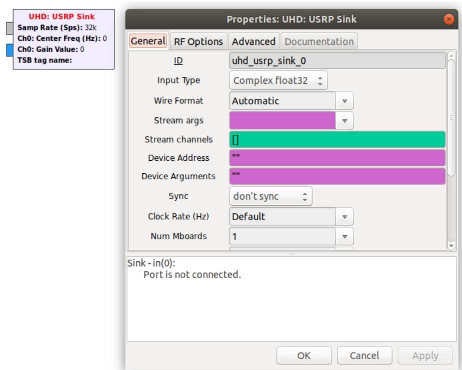

Figure 1.25: GNU Radio sinks — ***UHD: USRP Sink***

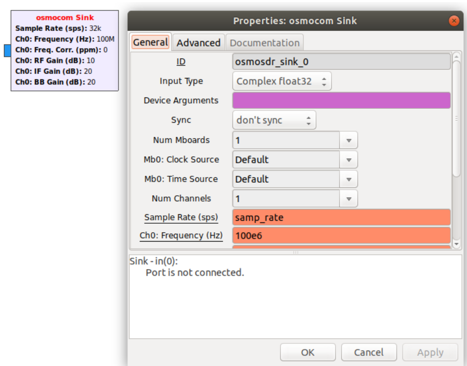

Figure 1.26: GNU Radio sinks — ***osmocom Sink***

1.4.3 GNU Radio Instrumentation Sinks

1. QT GUI sinks

   (a) QT GUI Sink — Fig. 1.27

   (b) QT GUI Constellation Sink — Fig. 1.28

   (c) QT GUI Frequency Sink — Fig. 1.29

   (d) QT GUI Histogram Sink — Fig. 1.30

   (e) QT GUI Number Sink — Fig. 1.31

   (f) QT GUI Time Raster Sink — Fig. 1.32

   (g) QT GUI Time Sink — Fig. 1.33

   (h) QT GUI Vector Sink — Fig. 1.34

   (i) QT GUI Waterfall Sink — Fig. 1.35

2. WX GUI sinks

   (a) WX GUI Constellation Sink — Fig. 1.36

   (b) WX GUI FFT Sink — Fig. 1.37

   (c) WX GUI Histo Sink — Fig. 1.38

   (d) WX GUI Number Sink — Fig. 1.39

   (e) WX GUI Scope Sink — Fig. 1.40

   (f) WX GUI Terminal Sink — Fig. 1.41

   (g) WX GUI Waterfall Sink — Fig. 1.42

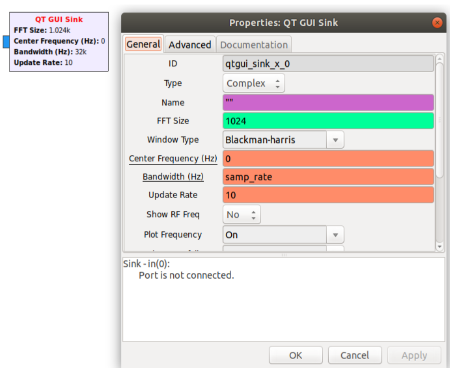

*Figure 1.27: GNU Radio instrumentation sinks — **QT GUI Sink***

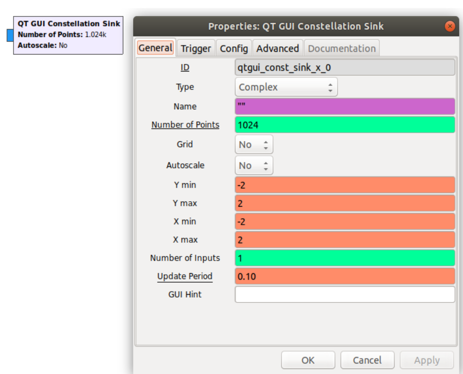

*Figure 1.28: GNU Radio instrumentation sinks — **QT GUI Constellation Sink***

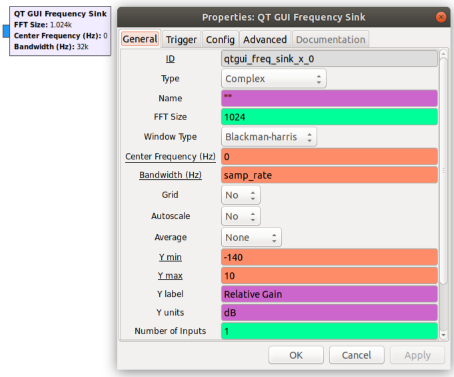

Figure 1.29: GNU Radio instrumentation sinks — *QT GUI Frequency Sink*

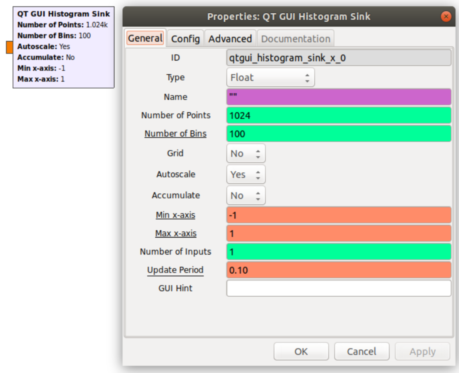

Figure 1.30: GNU Radio instrumentation sinks — *QT GUI Histogram Sink*

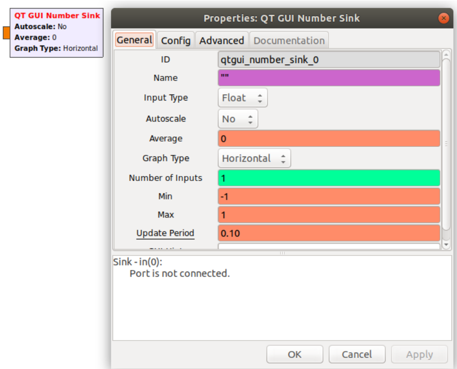

Figure 1.31: GNU Radio instrumentation sinks — *QT GUI Number Sink*

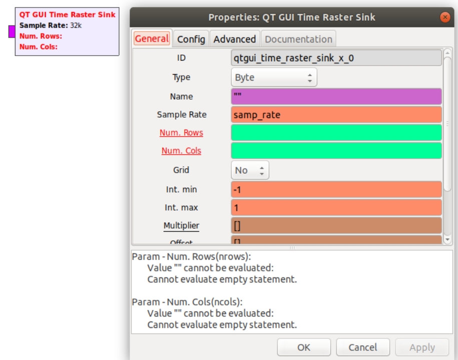

Figure 1.32: GNU Radio instrumentation sinks — *QT GUI Time Raster Sink*

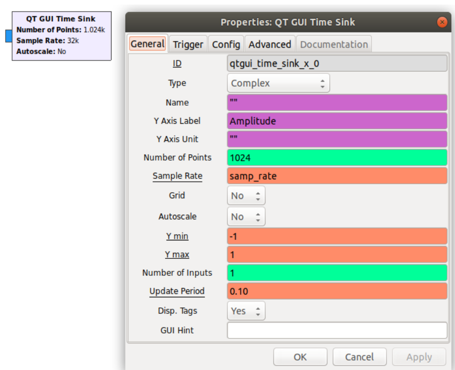

Figure 1.33: GNU Radio instrumentation sinks — *QT GUI Time Sink*

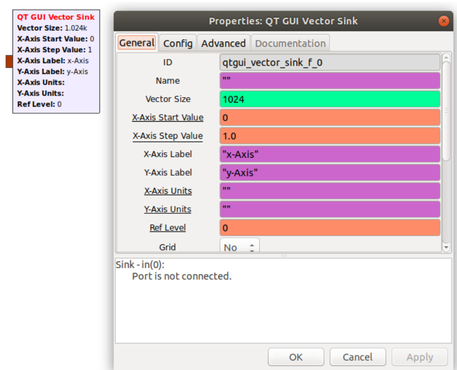

Figure 1.34: GNU Radio instrumentation sinks — *QT GUI Vector Sink*

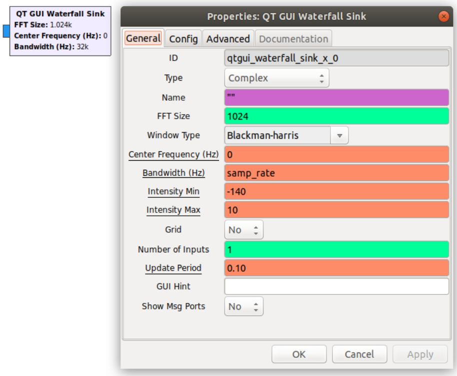

Figure 1.35: GNU Radio instrumentation sinks — **QT GUI Waterfall Sink**

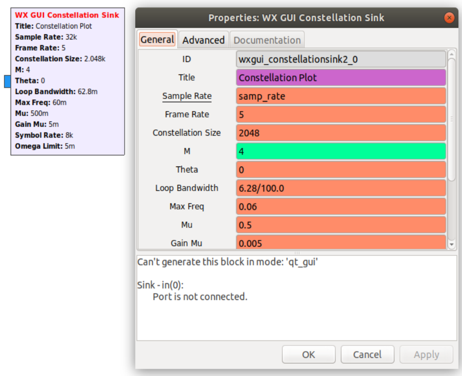

Figure 1.36: GNU Radio instrumentation sinks — **WX GUI Constellation Sink**

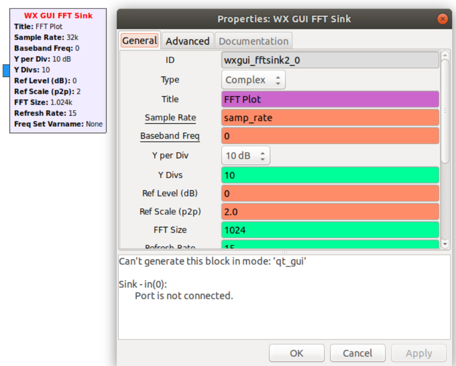

Figure 1.37: GNU Radio instrumentation sinks — **WX GUI FFT Sink**

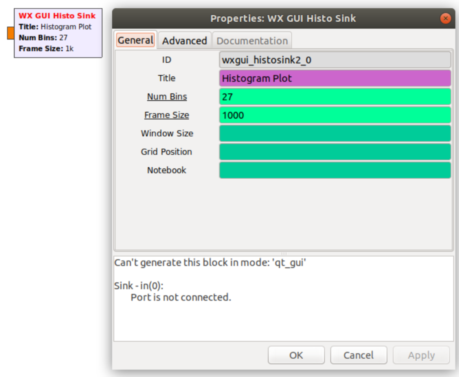

Figure 1.38: GNU Radio instrumentation sinks — **WX GUI Histo Sink**

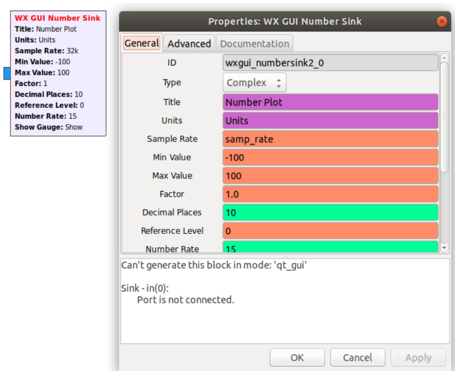

Figure 1.39: GNU Radio instrumentation sinks — **WX GUI Number Sink**

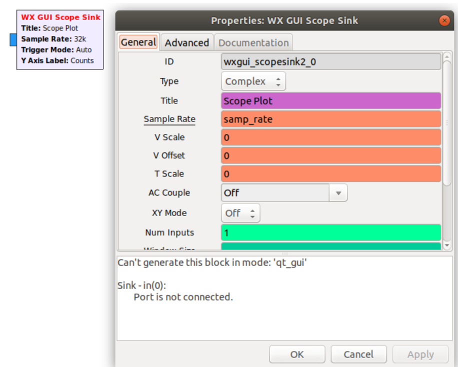

Figure 1.40: GNU Radio instrumentation sinks — **WX GUI Scope Sink**

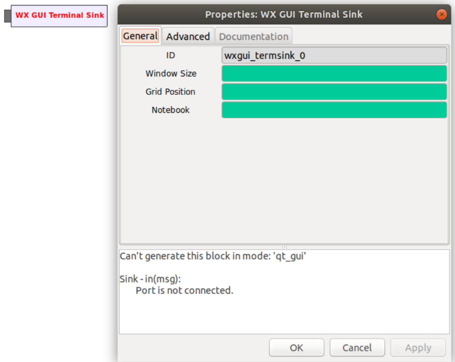

Figure 1.41: GNU Radio instrumentation sinks — ***WX GUI Terminal Sink***

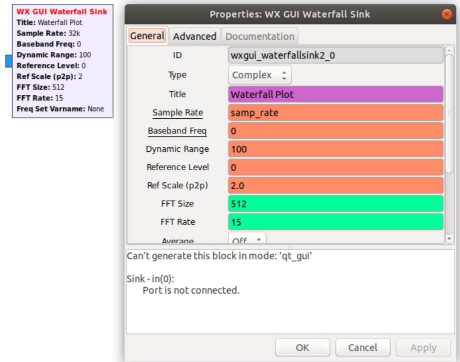

Figure 1.42: GNU Radio instrumentation sinks — ***WX GUI Waterfall Sink***

Selected Radio Signal Applications in GNU Radio

This section contains proposals of the receivers and transmitters for three basic analog modulations: Amplitude Modulation (AM), Frequency Modulation (FM) and Single Side Band (SSB) Modulation. Proposed receivers and transmitters do not utilize real RF hardware (with the exception of the last one example), the RF signal from the transmitter to the receiver is sent via a network socket using ZMQ blocks. All of the presented proposals are based on the ideas shown in official GNU Radio Tutorials [3], however they are made from scratch showing all the parameters necessary to make the flowgraphs running withour errors. These examples were tested under GNU Radio 3.7.11.

The descriptions of the modulations schemes are omited here since they are widely spread in numerous sources.

2.1 Amplitude Modulation (AM)

This subsection presents Amplitude Modulation (AM) Transmitter and Receiver.

2.1.1 AM Transmitter

AM transmitter flowgraph is shown in Fig. 2.1. The parameters of the building blocks are following:

- Variable:
  - **samp_rate** = **768 kHz**

  (gives the 48 kHz carrier frequency 16 samples in every cycle)
- Audio Source:
  - Sample Rate = **48 kHz**
- Repeat:
  - Interpolation = **16**

  (boosts the audio sample rate to the system sample rate)
- QT GUI Range defines Audio gain (= **volume** variable) controls
- Multiply Const:
  - Constant = **volume** variable

- Add Const:
  - Constant = **1**

  (creates AM carrier in the absence of the audio signal)
- Signal Source:
  - Frequency = **48 kHz**
  - Amplitude = **1**

  (generates carrier signal frequency)
- QT GUI Time Sink:
  - Number of Points = **4096**

  (shows visual representation of the transmitted signal)
- ZMQ PUB Sink:
  - Address = **tcp://127.0.0.1:50222**

  (generated signal is sent to a network data socket connected to the receiving section on the same computer)

After compiling and executing the flowgraph, it will transmit the AM signal carrying audio signal recorded with microphone. QT GUI Time Sink will show changing pattern of the signal, modulation level could be adjusted with the volume control in QT GUI Range block. The output signal can be demodulated by the receiver described in the next subsection.

2.1.2 AM Receiver

AM receiver flowgraph is shown in Fig. 2.2. The parameters of the building blocks are following:

- Variable:
  - **samp_rate** = **768 kHz**
- Variable:
  - **decim** = **16**

  (defines the decimation factor to reduce the incoming sample rate by 16 in order to get an audio sample rate of 48 kHz for the Audio Sink block)
- ZMQ SUB Source:
  - Address = **tcp://127.0.0.1:50222**

  (the signal is received from a network data socket connected to the transmitting section on the same computer)

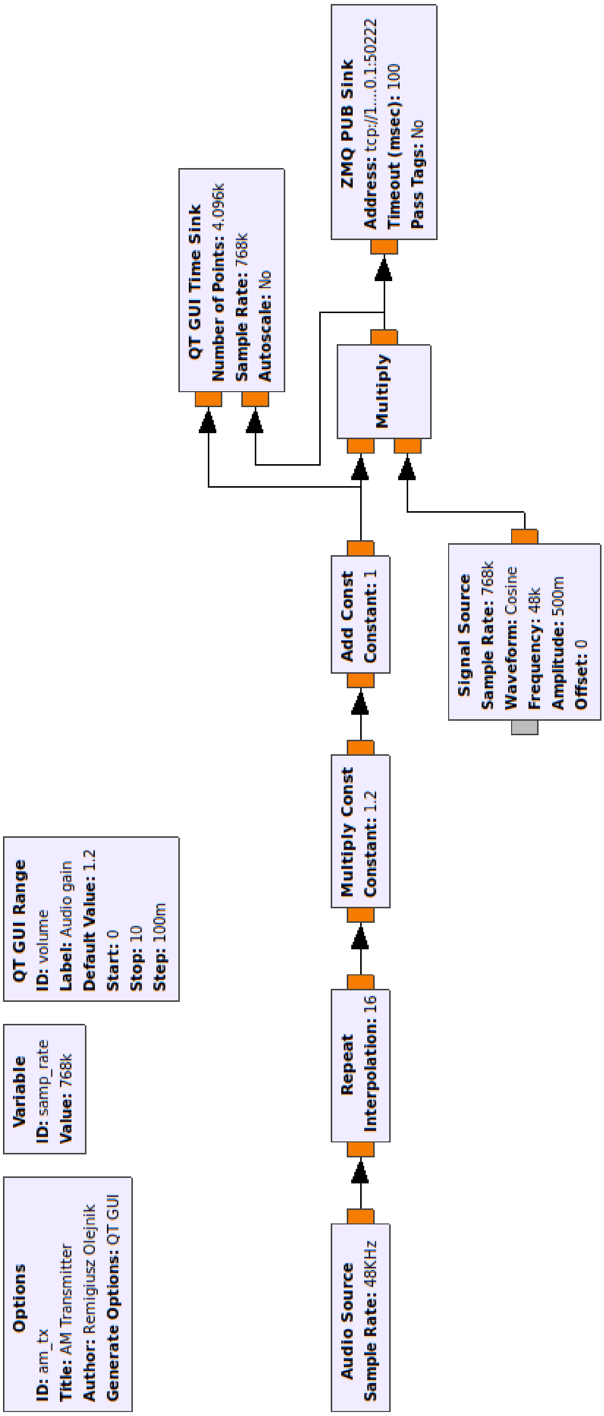

Figure 2.1: GNU Radio — AM transmitter flowgraph

- Frequency Xlating FIR Filter:
  - Type = **Float->Complex (Real Taps)**
  - Decimation = **decim**
  - Taps = **firdes.low_pass(1,samp_rate,samp_rate/(2*decim), 2000)**
  - Center Frequency = **48 kHz**
  - Sample Rate = **samp_rate**

  (performs frequency translation, filtering and decimation)
- AGC (Automatic Gain Control):
  - default values

  (adjusts the input signal to the given reference level)
- Complex to Mag:
  - no values

  (calculates magnitude of the complex samples in order to restore original modulation signal)
- Band Pass Filter:
  - FIR Type = **Float->Float (Real Taps)(Decim)**
  - Decimation = **1**
  - Gain = **1**
  - Sample Rate = **(int)(samp_rate/decim)**
  - Low Cutoff Freq = **500 Hz**
  - High Cutoff Freq = **6 kHz**
  - Transition Width = **400**
- QT GUI Range defines Audio gain (= **volume** variable) controls
- Multiply Const:
  - Constant = **volume** variable
- QT GUI Time Sink:
  - Sample Rate = **(int)(samp_rate/decim)**
  - Number of Points = **256**

  (shows visual representation of the received signal)
- Audio Sink:
  - Sample Rate = **48 kHz**
  - OK to Block = **Yes**

After compiling and executing the flowgraph, it will receive the AM signal from a network data socket connected to the transmitting section described in the previous subsection. QT GUI Time Sink will show changing pattern of the signal.

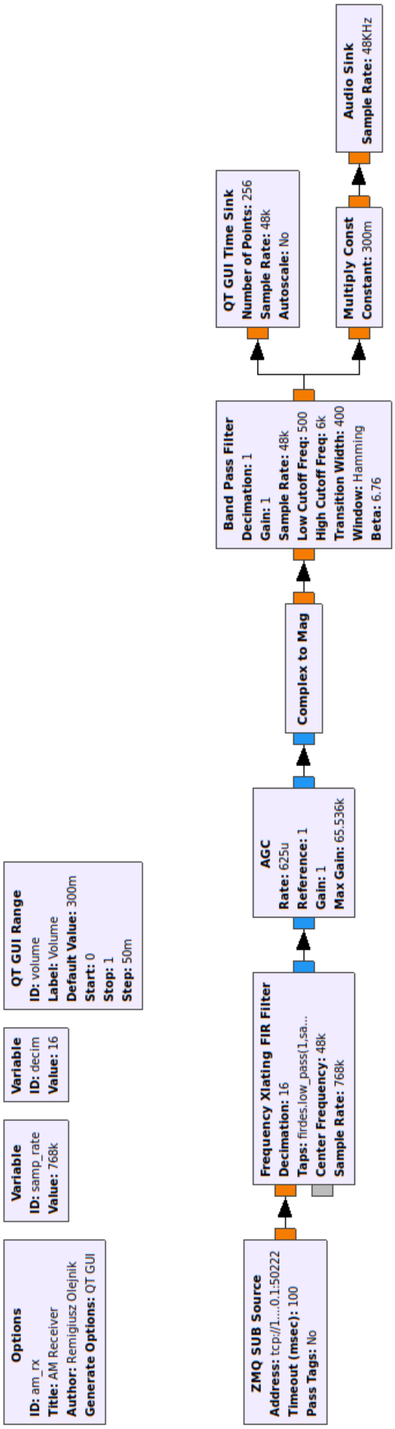

Figure 2.2: GNU Radio — AM receiver flowgraph

2.2 Frequency Modulation (FM)

This subsection presents Narrow Band Frequency Modulation (NBFM) Transmitter and Receiver.

2.2.1 NBFM Transmitter

FM transmitter flowgraph is shown in Fig. 2.3. The parameters of the building blocks are following:

- Variable:
  - **samp_rate** = **48 kHz**
- Variable:
  - **usrp_rate** = **576 kHz**
- Variable:
  - **if_rate** = **192 kHz**
- QT GUI Range defines Audio gain (= **audio_gain** variable) controls
- QT GUI Chooser defines three PL tones — **pl_freq** variable: **0 Hz**, **67 Hz** and **71.9 Hz**
- Audio Source:
  - Sample Rate = **48 kHz**
- Band Pass Filter:
  - Decimation = **1**
  - Gain = **1**
  - Sample Rate = **samp_rate**
  - Low Cuttoff Freq = **300 Hz**
  - High Cutoff Freq = **5 kHz**
  - Transition Width = **200**
  - Window = **Hamming**
  - Beta = **6.76**
- Multiply Const:
  - **audio_gain** variable
- Signal Source:
  - Sample Rate = **48 kHz**
  - Waveform = **Sine**
  - Frequency = **pl_freq**

  - Amplitude = **0.150**
  - Offset = **0**

- NBFM Transmit:
  - Audio Rate = **48 kHz**
  - Quadrature Rate = **if_rate**
  - Tau = **0.000075**
  - Max Deviation = **5000**
  - Preemphasis High Corner Freq = **−1**

- QT GUI Sink:
  - FFT Size = **1024**
  - Center Frequency = **0 Hz**
  - Bandwidth = **if_rate**
  - Update Rate = **10**

- Low Pass Filter:
  - Decimation = **1**
  - Gain = **1**
  - Sample Rate = **if_rate**
  - Cutoff Freq = **5 kHz**
  - Transition Width = **2000**
  - Window = **Hamming**
  - Beta = **6.76**

- Repeat:
  - Interpolation = **3**

  (multiplies **if_rate** in order to get **usrp_rate**)

- ZMQ PUB Sink:
  - Address = **tcp://127.0.0.1:49999**

  (generated signal is sent to a network data socket connected to the receiving section on the same computer)

After compiling and executing the flowgraph, it will transmit the FM signal carrying audio signal recorded with microphone. Modulation level could be adjusted with the volume control in QT GUI Range block. The output signal can be demodulated by the receiver described in the next subsection.

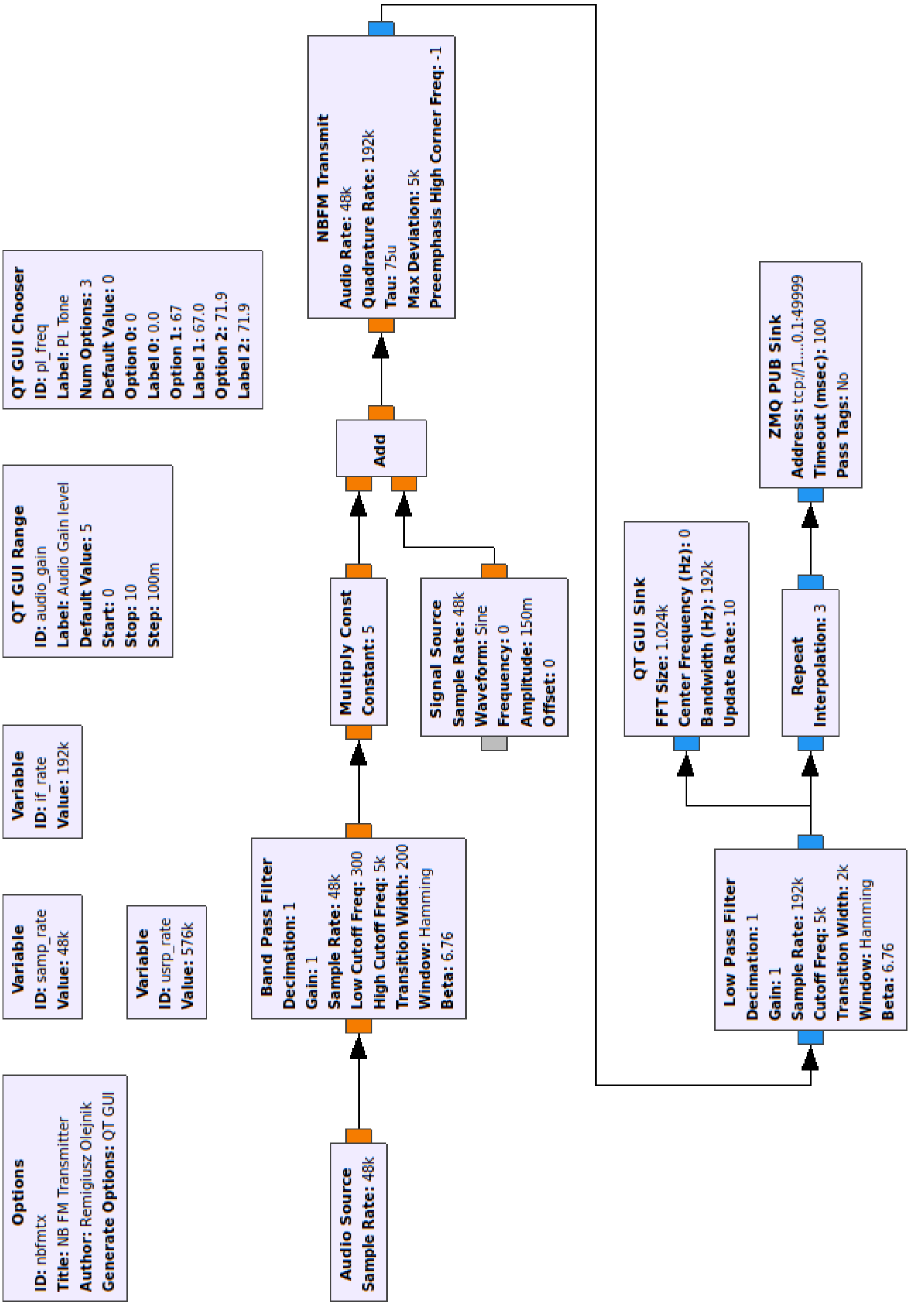

Figure 2.3: GNU Radio — FM transmitter flowgraph

2.2.2 NBFM Receiver

FM receiver flowgraph is shown in Fig. 2.4. The parameters of the building blocks are following:

- Variable:
  - **samp_rate** = **576 kHz**
- Variable:
  - **rf_decim** = **3**
- Band-pass Filter Taps:
  - ID = **channel_filter**
  - Tap Type = **Complex**
  - Gain = **1**
  - Sample Rate = **samp_rate**
  - Low Cuttoff Freq = **−3 kHz**
  - High Cutoff Freq = **3 kHz**
  - Transition Width = **200**
  - Window = **Hamming**
  - Beta = **6.76**

  (defines Filter Taps for the FFT Filter block)
- QT GUI Range defines VOLume level (= **VOL_level** variable) controls
- QT GUI Range defines SQueLch level (= **SQL_level** variable) controls
- ZMQ SUB Source:
  - Address = **tcp://127.0.0.1:49999**

  (the signal is received from a network data socket connected to the transmitting section on the same computer)
- FFT Filter:
  - Type = **Complex->Complex (Complex Taps)**
  - Decimation = **rf_decim**
  - Taps = **channel_filter**
  - Num. Threads = **1**
- Simple Squelch:
  - Threshold = **−50 dB**
  - Alpha = **1**

- NBFM Receive:
  - Audio Rate = **48 kHz**
  - Quadrature Rate = **192 kHz**
  - Tau = **0.000075**
  - Max Deviation = **5000**
- Multiply Const:
  - **VOL_level** variable
- Audio Sink:
  - Sample Rate = **48 kHz**
- QT GUI Waterfall Sink:
  - FFT Size = **1024**
  - Center Frequency = **0 Hz**
  - Bandwidth = **samp_rate**

  (shows a waterfall spectrum display with visual representation of the received signal)

After compiling and executing the flowgraph, it will receive the FM signal from a network data socket connected to the transmitting section described in the previous subsection. QT GUI Waterfall Sink will show changing pattern of the signal. GUI windows with Volume and Squelch controls allow for controling received signal.

2.3 Single Side Band (SSB) Modulation

This subsection presents Single Side Band (SSB) Modulation Transmitter and Receiver.

2.3.1 SSB Transmitter

SSB transmitter flowgraph is shown in Fig. 2.5. The parameters of the building blocks are following:

- Variable:
  - **samp_rate** = 192 kHz
- Variable:
  - **audio_rate** = 48 kHz
- Variable:
  - **carrier_freq** = 16 kHz

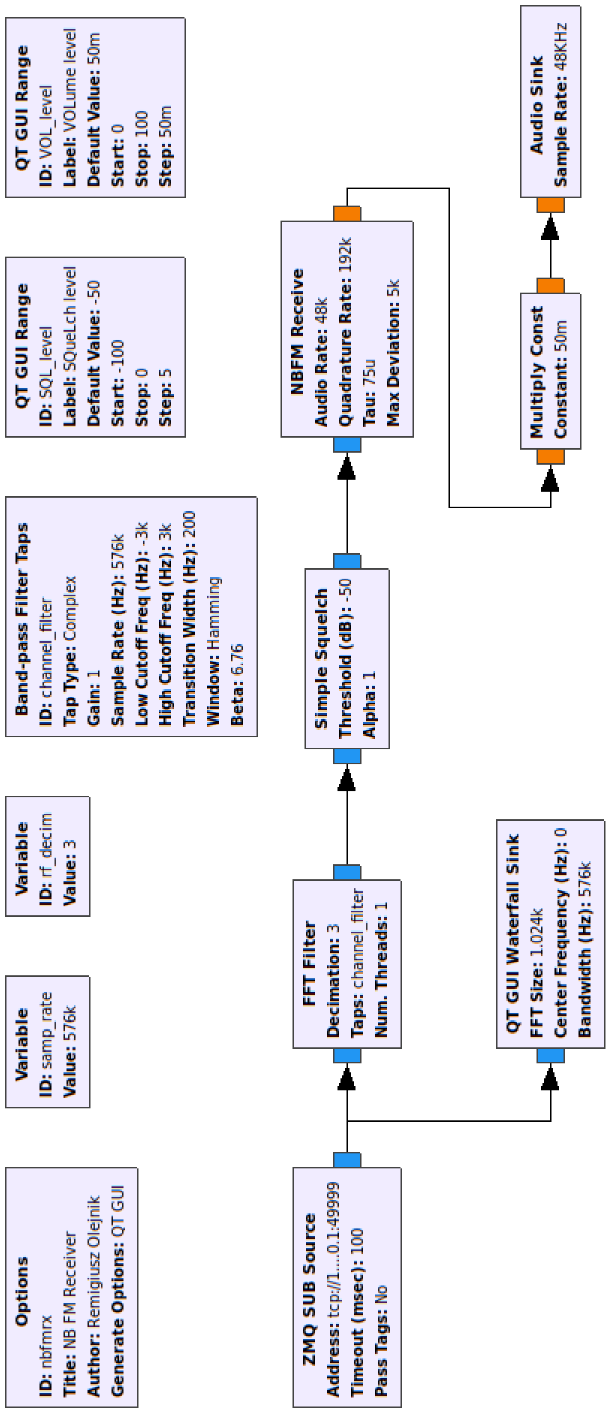

Figure 2.4: GNU Radio — FM receiver flowgraph

- Variable:
  - **interp** = 4
- QT GUI Range defines Audio gain (= **volume** variable) controls
- Audio Source:
  - Sample Rate = **samp_rate**
- Multiply Const:
  - **volume** variable
- Repeat:
  - Interpolation = **interp** variable

  (boosts the audio sample rate to the system sample rate)
- Constant Source:
  - Constant = **0**
- Float to Complex

  (converts float data into complex numbers)
- Signal Source:
  - Sample Rate = **samp_rate**
  - Waveform = **Sine**
  - Frequency = **carrier_freq**
  - Amplitude = **1**
  - Offset = **0**
- Multiply

  (creates modulated SSB signal)
- Band Pass Filter:
  - Decimation = **1**
  - Gain = **1**
  - Sample Rate = **samp_rate**
  - Low Cuttoff Freq = **16.3 kHz**
  - High Cutoff Freq = **19 kHz**
  - Transition Width = **200**
  - Window = **Hamming**
  - Beta = **6.76**

  (creates SSB signal by passing one (upper) sideband only and rejecting the other — the filter method)

- QT GUI Frequency Sink:
  - FFT Size = **1024**
  - Center Frequency = **0 Hz**
  - Bandwidth = **samp_rate**
- ZMQ PUSH Sink:
  - Address = **tcp://127.0.0.1:50333**

  (generated signal is sent to a network data socket connected to the receiving section on the same computer)

After compiling and executing the flowgraph, it will transmit the SSB signal carrying audio signal recorded with microphone. QT GUI Frequency Sink will show changing pattern of the signal, modulation level could be adjusted with the volume control in QT GUI Range block. The output signal can be demodulated by the receiver described in the next subsection.

2.3.2 SSB Receiver

SSB receiver flowgraph is shown in Fig. 2.6. The parameters of the building blocks are following:

- Variable:
  - **samp_rate** = **192 kHz**
- Variable:
  - **audio_rate** = **48 kHz**
- Variable:
  - **carrier_freq** = **16 kHz**
- Variable:
  - **decim** = **4**
- QT GUI Range defines Tuning (= **tuning** variable) controls:
  - Start = **11000**
  - Stop = **21000**
  - Step = **100**
  - Default Value = **17500**
- QT GUI Range defines Fine Tuning (= **bfo** variable) controls:
  - Start = **0**
  - Stop = **3000**

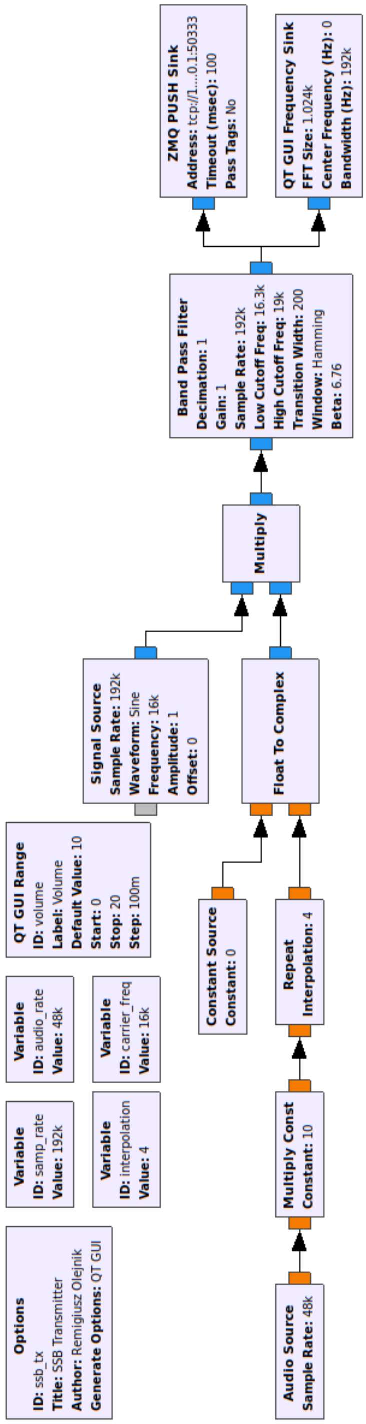

Figure 2.5: GNU Radio — SSB transmitter flowgraph

  - Step = **10**
  - Default Value = **1500**

- QT GUI Range defines Audio gain (= **volume** variable) controls:
  - Start = **0**
  - Stop = **1**
  - Step = **0.050**
  - Default Value = **0.500**

- QT GUI Chooser allows to choose Upper of Lower Sideband (USB/LSB)
- ZMQ PULL Source:
  - Address = **tcp://127.0.0.1:50333**

  (the signal is received from a network data socket connected to the transmitting section on the same computer)

- Frequency Xlating FIR Filter:
  - Type = **Complex->Complex (Complex Taps)**
  - Decimation = **decim**
  - Taps = **firdes.low_pass(1.0, samp_rate, 3000, 100)**
  - Center Frequency = **tuning**
  - Sample Rate = **samp_rate**

  (performs frequency translation, filtering and decimation)

- Complex to Float:
  - no values

  (converts complex numbers into floats)

- Signal Source:
  - Sample Rate = **audio_rate**
  - Waveform = **Cosine**
  - Frequency = **bfo**
  - Amplitude = **1**
  - Offset = **0**

- Multiply:
  - no values

  (multiplies signals)

- Multiply Const:
  - Constant = **1**
- Add:
  - no values

  (adds signals)
- Multiply Const:
  - Constant = **volume** variable
- Audio Sink:
  - Sample Rate = **audio_rate**

After compiling and executing the flowgraph, it will receive the SSB signal from a network data socket connected to the transmitting section described in the previous subsection. QT GUI Range controls allows to tune to the signal, fine tune to the signal and adjust volume. QT GUI Chooser allows to change sideband (USB or LSB).

2.3.3 SSB Receiver — I/Q Signal from the File

The flowgrpah of the SSB receiver taking I/Q signal from the file is shown in Fig. 2.7. The parameters of the building blocks are following:

- Variable
  - **samp_rate** = **256 kHz**
- Variable
  - **audio_rate** = **32 kHz**
- Variable
  - **carrier_freq** = **53 kHz**
- Variable
  - **decim** = **8**
- QT GUI Range defines Tuning (= **tuning** variable) controls:
  - Start = **48000**
  - Stop = **58000**
  - Step = **100**
  - Default Value = **51500**

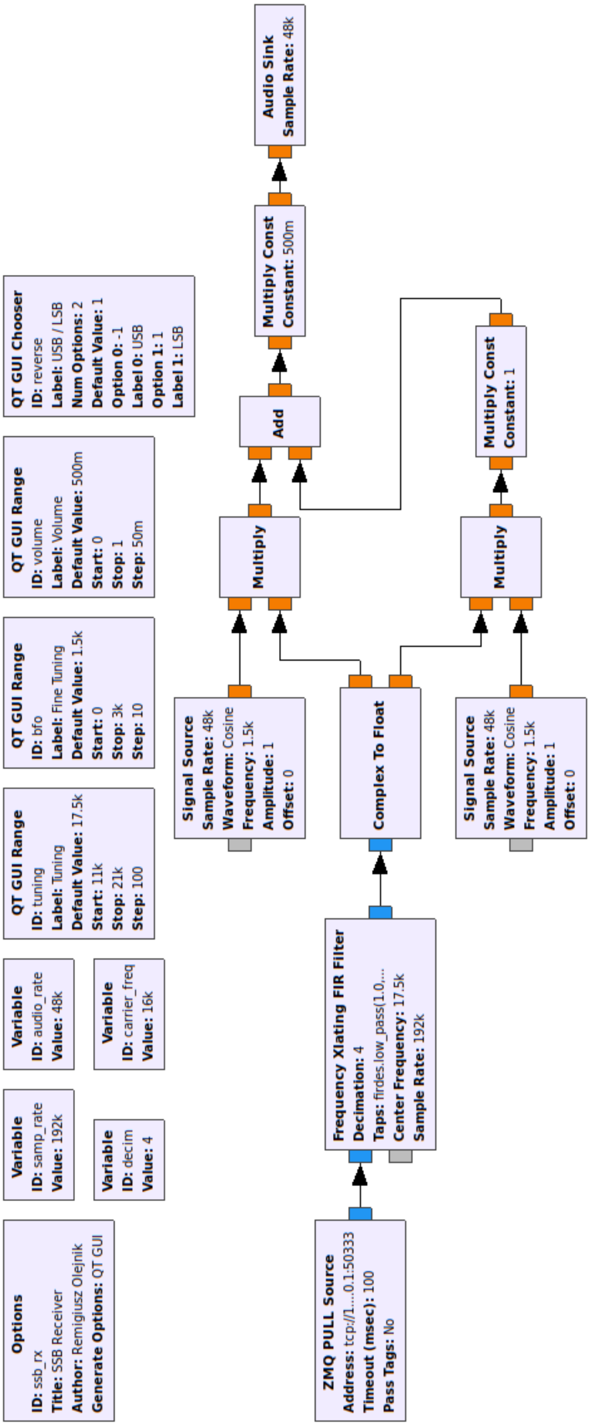

Figure 2.6: GNU Radio — SSB receiver flowgraph

- QT GUI Range defines Fine Tuning (= **bfo** variable) controls:
  - Start = **0**
  - Stop = **3000**
  - Step = **10**
  - Default Value = **1500**
- QT GUI Range defines Audio gain (= **volume** variable) controls:
  - Start = **0**
  - Stop = **1**
  - Step = **0.050**
  - Default Value = **0.200**
- QT GUI Chooser allows to choose Upper of Lower Sideband (USB/LSB)
- File Source:
  - File = **ssb_lsb_256k_complex2.dat**
  - Repeat = **yes**

  (the signal is taken from the file, it is necessary to download it from `https://www.csun.edu/~skatz/katzpage/sdr_project/sdr/ssb_lsb_256k_complex2.dat.zip`)
- Multiply Const:
  - Constant = **0.0001**
- Frequency Xlating FIR Filter:
  - Type = **Complex->Complex (Complex Taps)**
  - Decimation = **decim**
  - Taps = **firdes.low_pass(1.0, samp_rate, 3000, 100)**
  - Center Frequency = **tuning**
  - Sample Rate = **samp_rate**

  (performs frequency translation, filtering and decimation)
- Complex to Float:
  - no values

  (converts complex numbers into floats)
- Signal Source:
  - Sample Rate = **audio_rate**
  - Waveform = **Cosine**
  - Frequency = **bfo**

  - Amplitude = **1**
  - Offset = **0**

- Multiply:
  - no values

  (multiplies signals)
- Add:
  - no values

  (adds signals)
- Multiply Const:
  - Constant = **volume** variable
- Audio Sink:
  - Sample Rate = **audio_rate**

After compiling and executing the flowgraph, it will receive the SSB signal from a provided file. QT GUI Range controls allows to tune to the signal, fine tune to the signal and adjust volume. QT GUI Chooser allows to change sideband (USB or LSB).

2.4 RTL-SDR Based WFM Receiver

In Fig. 2.8 a simple example of the broadcast WFM receiver is presented. It consists of **RTL-SDR Source** block as a radio signal source, **FM Demod** block as a FM demodulator, **Multiply Const** block supplying a volume value for the audio level and **Audio Sink** block that allows playing audio signal.

The parameters of the building blocks are following:

- Variable
  - **samp_rate** = **240 kHz**
- Variable
  - **deviation** = **75 kHz**
- Variable
  - **audio_decim** = **5**
- QT GUI Range defines RF Gain (= **rf_gain** variable) controls:
  - Start = **0**
  - Stop = **70**
  - Step = **1**
  - Default Value = **50**

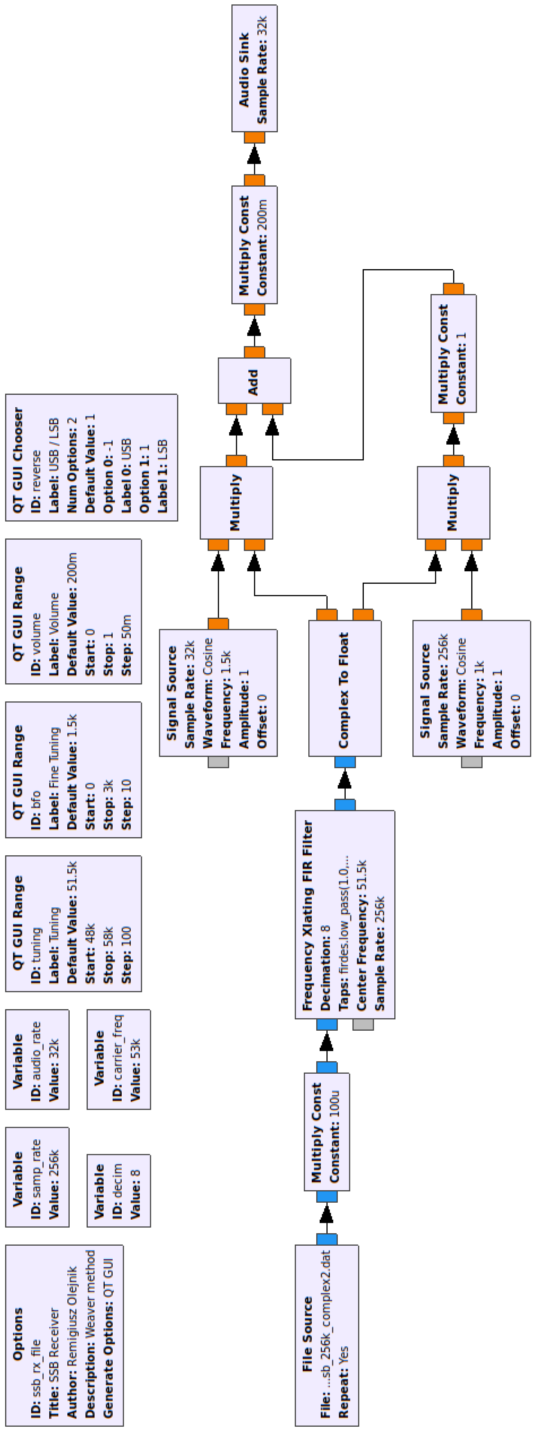

Figure 2.7: GNU Radio — SSB receiver flowgraph — I/Q signal from the file

- QT GUI Range defines Tuning (= **tuning** variable) controls:
  - Start = **87.9 MHz**
  - Stop = **108.1 MHz**
  - Step = **100 kHz**
  - Default Value = **98 MHz**
- QT GUI Range defines Volume (= **volume** variable) controls:
  - Start = **0**
  - Stop = **1**
  - Step = **0.050**
  - Default Value = **0.300**
- RTL-SDR Source:
  - Sample Rate = **samp_rate** variable
  - Frequency = **tuning** variable
  - Freq. Corr. = **0**
  - DC Offset Mode = **Off**
  - IQ Balance Mode = **Off**
  - Gain Mode = **Manual**
  - RF Gain = **rf_gain** variable
  - IF Gain = **20 dB**
  - BB Gain = **20 dB**
- FM Demod:
  - Channel Rate = **samp_rate** variable
  - Audio Decimation = **audio_decim** variable
  - Deviation = **deviation** variable
  - Audio Pass = **15 kHz**
  - Audio Stop = **16 kHz**
  - Gain = **1**
  - Tau = **0.000075**
- Multiply Const:
  - Constant = **volume** variable
- Audio Sink:
  - Sample Rate = **(int)(samp_rate/audio_rate)**

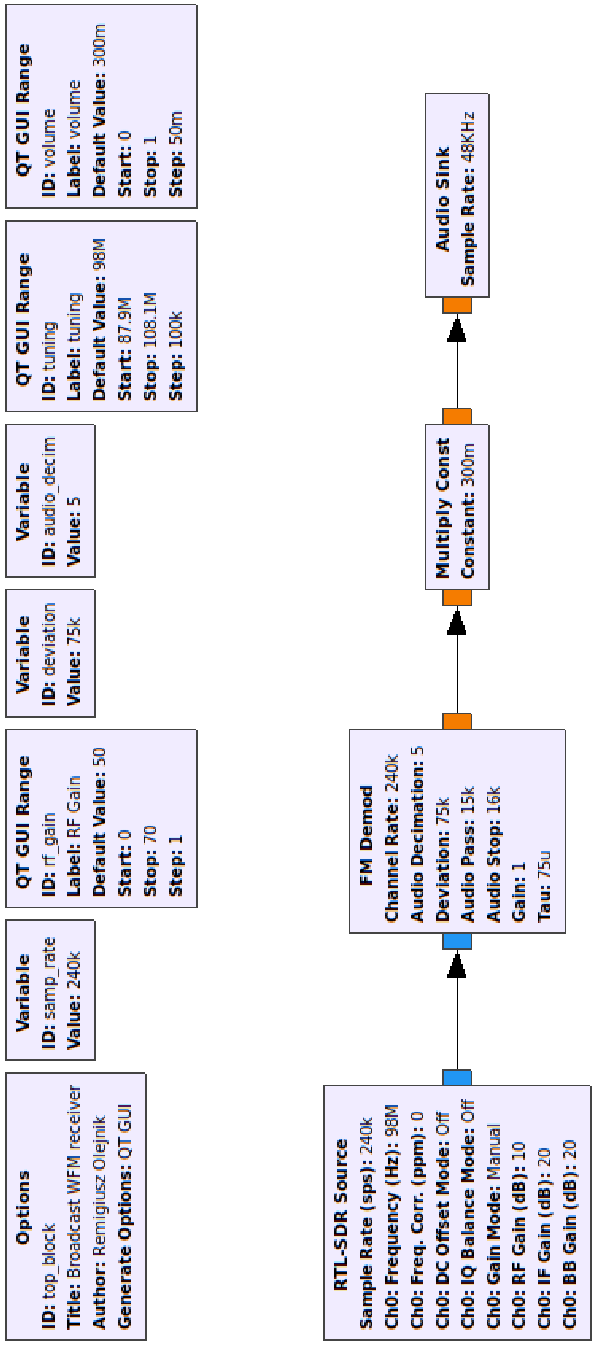

Figure 2.8: GNU Radio — RTL-SDR based broadcast WFM receiver

Bibliography

[1] GNU Radio, https://www.gnuradio.org/

[2] What Is GNU Radio,
https://wiki.gnuradio.org/index.php?title=What_Is_GNU_Radio

[3] GNU Radio Tutorials,
https://wiki.gnuradio.org/index.php?title=Tutorials

List of Figures

Wireless Signal Processing in GNU Radio Environment
Study text

Author: dr inż. Remigiusz Olejnik
West Pomeranian University of Technology in Szczecin

Publisher: Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic
Graphic editing and typesetting: Jiří Rybička, Pavel Haluza
Year of publishing: 2022
First edition
Number of pages: 46

ISBN 978-80-7509-891-7 (online ; pdf)
DOI https://doi.org/10.11118/978-80-7509-891-7