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





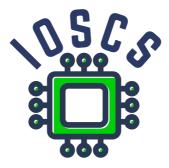
UNIVERSITY OF ŽILINA Mendel
University
in Brno

Reviewer: Ing. Peter Šarafín, PhD., University of Žilina, Slovakia Project: Innovative Open Source Courses for Computer Science Curriculum © Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic ISBN 978-80-7509-891-7 (online; pdf) DOI https://doi.org/10.11118/978-80-7509-891-7



Open Access. This book is licensed under the terms of the Creative Commons Attribution-ShareAlike 4.0 International License, CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/)

#### Project: Innovative Open Source Courses for Computer Science Curriculum



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

#### Erasmus+ Disclaimer

This project has been funded with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

#### **Copyright Notice**

This content was created by the IOSCS consortium: 2019–2022. The content is Copyrighted and distributed under Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).



Co-funded by the Erasmus+ Programme of the European Union

## 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

1	What is GNU Radio and Why it is Worth to Use it6						
	1.1	A Flow	rgraph-Based Approach to Digital Signal Processing	6			
	1.2	Most P	Popular GNU Radio Blocks	8			
		1.2.1	Default Blocks	9			
	1.3	Signal	Data Types in GNU Radio	10			
	1.4	Source	s and Sinks in GNU Radio	10			
		1.4.1	GNU Radio Sources	11			
		1.4.2	GNU Radio Sinks	14			
		1.4.3	GNU Radio Instrumentation Sinks	16			
2	Seleo		lio Signal Applications in GNU Radio	20			
	2.1	Amplit	rude Modulation (AM)	20			
		2.1.1	AM Transmitter	20			
		2.1.2	AM Receiver	21			
	2.2	Freque	ncy Modulation (FM)	25			
		2.2.1	NBFM Transmitter	25			
		2.2.2	NBFM Receiver	28			
	2.3	Single	Side Band (SSB) Modulation	29			
		2.3.1	SSB Transmitter	29			
		2.3.2	SSB Receiver	32			
		2.3.3	SSB Receiver – I/Q Signal from the File $\ldots$ $\ldots$ $\ldots$ $\ldots$	35			
	2.4	RTL-SI	DR Based WFM Receiver	38			

## 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.

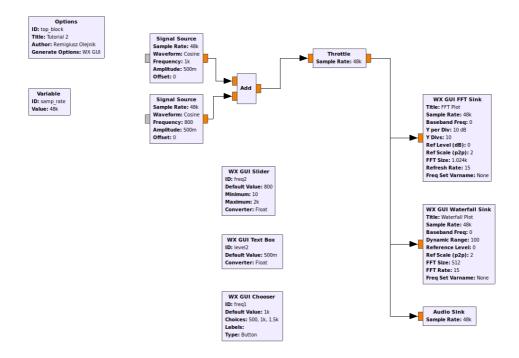


Figure 1.1: GNU Radio – seven blocks connected together form a flowgraph

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.

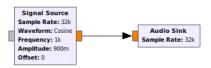


Figure 1.2: GNU Radio – two blocks (Signal Source and Audio Sink) connected with an arrow showing flow of the signal data

## 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.

	Properties: Variable 🛛 🛞					
General Advanced	Documentation					
ID	samp_rate					
Value	32000					
ų	OK Cancel Apply					

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.

	Properties: Options 🛛 😣
General Advanced	Documentation
ID	top_block
Title	
Author	
Description	
Canvas Size	
Generate Options	QT GUI ‡
Run	Autostart 🔻
Max Number of Output	0
Realtime Scheduling	Off ‡
QSS Theme	
u	OK Cancel Apply

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

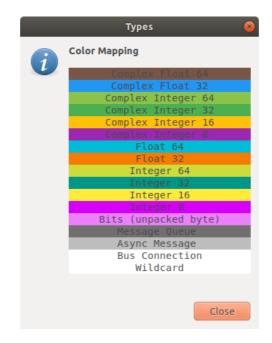


Figure 1.5: GNU Radio signal data types

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

Null Source		Properties: Null Source 🛛 😣	Noise Source Noise Type: Gaussian		Properties: Noise Source 🧧
	General Advanced	Documentation	Amplitude: 1	General Advanced	Documentation
	ID	blocks_null_source_0		ID	analog_noise_source_x_0
	Output Type	Complex 🛟		Output Type	Complex 💲
	Vec Length	1		Noise Type	Gaussian
	Num Outputs	1		Amplitude	1
	Bus Connections	[[0,],]		Seed	0
	Source - out(0):			Source - out(0):	
	Port is not connec	ited.		Port is not connec	:ted.
		OK Cancel Apply			OK Cancel Apply

Figure 1.6: GNU Radio sources – Null Source

Figure 1.7: GNU Radio sources – Noise Source

Signal Source Sample Rate: 32k		Properties: Signal Source 🛛 😣	File Source		Properties: File Source 🛛 😵
Waveform: Cosine Frequency: 1k	General Advanced	Documentation	Repeat: Yes	General Advanced	Documentation
Amplitude: 1 Offset: 0	ID	ID analog_sig_source_x_0		ID	blocks_file_source_0
Offset: 0	Output Type	Complex ‡		File	
	Sample Rate	samp_rate		Output Type	Complex 💲
	Waveform	Cosine		Repeat	Yes 🛟
	Frequency	1000		Vec Length	1
	Amplitude	1			
	Offset	0			
	Source - out(0):			Source - out(0):	
	Port is not connect	ed.		Port is not connect	ed.
		OK Cancel Apply			OK Cancel Apply

Figure 1.8: GNU Radio sources – Signal Source Figure 1.9: GNU Radio sources – File Source

TCP Source Address: 127.0.0.1		Properties: TCP Source 🛛 😣	UDP Source IP Address: 127.0.0.1		Properties: UDP Source 🛛 😵
Port: 0 Mode: Server	General Advanced	Documentation	Port: 1.234k Payload Size: 1.472k	General Advanced	Documentation
Houe. Server	ID	blks2_tcp_source_0	Null Pkt is EOF: True	ID	blocks_udp_source_0
	Output Type	Complex 💲		Output Type	Complex ‡
	Address	127.0.0.1		IP Address	127.0.0.1
	Port	0		Port	1234
	Mode	Server ‡		Payload Size	1472
	Vec Length	1		Null Pkt is EOF	True
				Vec Length	1
	Source - out(0): Port is not connect	.ed.		Source - out(0): Port is not connec	ted.
		OK Cancel Apply			OK Cancel Apply

Figure 1.10: GNU Radio sources – TCP Source

Figure 1.11: GNU Radio sources – UDP Source

Audio Source Sample Rate: 32k		Properties: Audio Source 🛛 🚳	Wav File Source			Properties: Wav File Source	
Sumple Rute. Six	General Advanced	Documentation	Repeat: Yes	Genera	l Advanced	Documentation	
	ID	audio_source_0			ID	blocks_wavfile_source_0	
	Sample Rate	samp_rate 🔹			File		
	Device Name				Repeat	Yes ‡	
	OK to Block	Yes ‡		N	Channels	1	
	Num Outputs	1					
	Source - out(0):			Source	- out(0):		
	Port is not connec	ited.		Por	t is not conne	cted.	
		OK Cancel Apply				OK Cancel	Apply

Figure 1.12: GNU Radio sources – Audio Source

۵

	Properties: UHD: USRP Source					
r Freq (Hz): 0 General RF C	ptions FE Corrections	Advanced	Documentation			
ID	uhd_usrp_sou	rce_0				
Output Ty	pe Complex float	32 🗘				
Wire Form	at Automatic					
Stream ar	gs					
Stream char	nnels []					
Device Add	ress ""					
Device Argun	nents "					
Sync	don't sync	•				
Clock Rate	(Hz) Default					
Num Mboa	rds 1					
Source - out(0) Port is not						

Figure 1.14: GNU Radio sources – UHD: USRP Source

RTL-SDR Source Sample Rate (sps): 32k			Properties: RTL-SDR	Source 😣
Ch0: Frequency (Hz): 100M Ch0: Freq. Corr. (ppm): 0	General	Advanced	Documentation	
Ch0: DC Offset Mode: Off	ID		rtlsdr_source_0	
Ch0: IQ Balance Mode: Off Ch0: Gain Mode: Manual	Out	tput Type	Complex float32 💲	
Ch0: RF Gain (dB): 10 Ch0: IF Gain (dB): 20	Device	Arguments		
Ch0: BB Gain (dB): 20		Sync	don't sync 🛟	
	Num	n Mboards	1	v
	Mb0: 0	lock Source	Default	TT U
	Mb0; 1	lime Source	Default	<b>•</b>
	Num	Channels	1	•
	Sample Rate (sps)	samp_rate		
	Ch0: Frequency (Hz)			
	Source - o Port i	out(0): is not connec	ted.	
			ОК	Cancel Apply

Figure 1.16: GNU Radio sources – RTL-SDR Source

Figure 1.13: GNU Radio sources – WAV File Source

osmocom Source Sample Rate (sps): 32k	Properties: osmocom Source 😣					
Ch0: Frequency (Hz): 100M Ch0: Freq. Corr. (ppm): 0	General Advanced	Documentation				
Ch0: DC Offset Mode: Off	ID	osmosdr_source_0				
Ch0: IQ Balance Mode: Off Ch0: Gain Mode: Manual	Output Type	Complex float32 🛟				
Ch0: RF Gain (dB): 10 Ch0: IF Gain (dB): 20	Device Arguments					
Ch0: BB Gain (dB): 20	Sync	don't sync 🛟				
	Num Mboards	1	v			
	Mb0: Clock Source	Default	T I			
	Mb0: Time Source	Default	T			
	Num Channels	1	T			
	Sample Rate (sps)	samp_rate				
	Ch0: Frequency (Hz)	Ch0: Frequency (Hz) 100e6				
	Source - out(0): Port is not connect	ored.	Cancel Apply			

Figure 1.15: GNU Radio sources – osmocom Source

Funcube Dongle Source	Prop	oerties: Funcube Dongle Source 🛛 🛞
Frequency (Hz): 145.5M LNA Gain (dB): 20	General Advanced	Documentation
Mixer Gain (dB): 12	ID	fcd_source_c_0
Frequency corr. (ppm): -120 DC I offset: 0	Device Name	hw:1
DC Q offset: 0 IQ phase balance: 0	Frequency (Hz)	145500000
IQ gain balance: 1	LNA Gain (dB)	20.0
	Mixer Gain (dB)	+12
	Frequency corr. (ppm)	-120
	DC I offset	0.0
	DC Q offset	0.0
	IQ phase balance	0.0
	IQ gain balance	1.0
	Source - out(0): Port is not connect	ed.
		OK Cancel Apply

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

Null Sink		Properties: Null Sink 🛛 😵	File Sink		Properties: File Sink 🛛 😣
	General Advanced	Documentation	Unbuffered: Off Append file: Overwrite	General Advanced	Documentation
	ID	blocks_null_sink_0	Append met ore mite	ID	blocks_file_sink_0
	Input Type	Complex 💲		File	
	Vec Length	1		Input Type	Complex 💲
	Num Inputs	1		Vec Length	1
	Bus Connections	[[0,],]		Unbuffered	Off v
				Append file	Overwrite v
	Sink - in(0): Port is not connect	ed.		Sink - in(0): Port is not connec	
		OK Cancel Apply			OK Cancel Apply

Figure 1.18: GNU Radio sinks – Null Sink

Figure 1.19: GNU Radio sinks – File Sink

ID Input Type Address Port Mode	ed Documentation blks2_tcp_sink_0 Complex : 127.0.0.1
Input Type Address Port	Complex 🛟 127.0.0.1
Address Port	127.0.0.1
Port	
	0
Mode	×
	Client 💲
Vec Length	1
Sink - in(0): Port is not cor	

Figure 1.20: GNU Radio sinks – TCP Sink

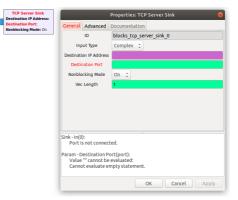


Figure 1.21: GNU Radio sinks – TCP Server Sink

UDP Sink Destination IP Address:		Properties: UDP Sink	Audio Sink Sample Rate: 32k		Properties: Audio Sink
Destination Port: Payload Size: 1.472k	General Advanced	Documentation	Sumple Rule. Sax	General Advanced	Documentation
Send Null Pkt as EOF: True	ID	blocks_udp_sink_0		ID	audio_sink_0
	Input Type	Complex 💲		Sample Rate	samp_rate 🔹
	Destination IP Address			Device Name	
	Destination Port			OK to Block	Yes ‡
	Payload Size	1472		Num Inputs	1
	Send Null Pkt as EOF	True			
	Vec Length	1			
				Sink - in(0):	10
	Sink - in(0): Port is not connect	ed.		Port is not connec	ted.
	Param - Destination Po	ort(port):			
	Value "" cannot be	evaluated:			
	Cannot evaluate en	npty statement.			
		OK Cancel Apply			OK Cancel Apply

Figure 1.22: GNU Radio sinks – UDP Sink

Figure 1.23: GNU Radio sinks – Audio Sink

Way File Sink	Properties: Wav File Sink 🛛 😣		UHD: USRP Sink Samp Rate (Sps): 32k		Properties: UHD: USRP Sink 🛛 😵
Sample Rate: 32k Bits per Sample: 8	General Advanced	Documentation	Ch0: Center Freq (Hz): 0 Ch0: Gain Value: 0	General RF Options	Advanced Documentation
and per complete	ID	blocks_wavfile_sink_0	TSB tag name:	ID	uhd_usrp_sink_0
	File			Input Type	Complex float32 💲
	N Channels	1		Wire Format	Automatic 💌
	Sample Rate	samp_rate		Stream args	▼
	Bits per Sample	8		Stream channels	0
				Device Address	
				Device Arguments	<u> </u>
				Sync	don't sync 🗘
				Clock Rate (Hz)	Default
				Num Mboards	1 v
	Sink - in(0): Port is not connec			Sink - in(0): Port is not connect	ced. ОК Сапсеl Аррју
		OK Cancel Apply			

Figure 1.24: GNU Radio sinks – WAV File Sink

Figure 1.25: GNU Radio sinks – UHD: USRP Sink

osmocom Sink Sample Rate (sps): 32k		Properties: osmocom	Sink 🧯
Ch0: Frequency (Hz): 100M Ch0: Freq. Corr. (ppm): 0	General Advanced	Documentation	
Ch0: RF Gain (dB): 10	<u>ID</u>	osmosdr_sink_0	
Ch0: IF Gain (dB): 20 Ch0: BB Gain (dB): 20	Input Type	Complex float32 ‡	
	Device Arguments		
	Sync	don't sync 💲	
	Num Mboards	1	v
	Mb0: Clock Source	Default	*
	Mb0: Time Source	Default	v
	Num Channels	1	•
	Sample Rate (sps)	samp_rate	
	Ch0: Frequency (Hz)	100e6	
	Sink - in(0): Port is not connect	ted.	

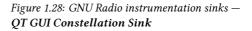
Figure 1.26: GNU Radio sinks – osmocom Sink

### 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

QT GUI Sink FFT Size: 1.024k Center Frequency (Hz): 0		Properties: QT GUI Sink 🛛 😸	QT GUI Constellation Sink Number of Points: 1.024k Autoscale: No		erties: QT GUI Constellation Sink 🛛 🛞
Bandwidth (Hz): 32k	General Advanced	Documentation	Autoscale. No	General Trigger Co	nfig Advanced Documentation
Update Rate: 10	ID	qtgui_sink_x_0		<u>ID</u>	qtgui_const_sink_x_0
	Type	Complex ‡		Туре	Complex ‡
	Name	-		Name	
	FFT Size	1024		Number of Points	1024
	Window Type	Blackman-harris 👻		Grid	No ‡
	Center Frequency (Hz)	0		Autoscale	No ‡
	Bandwidth (Hz)	samp_rate		Y min	-2
	Update Rate	10		Y max	2
	Show RF Freq	No ‡		X min	-2
	Plot Frequency	On 🔻		X max	2
	Sink - in(0):			Number of Inputs	1
	Port is not connect	ed.		Update Period	0.10
				GUI Hint	
		OK Cancel Apply			OK Cancel Apply

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



QT GUI Frequency Sink FFT Size: 1.024k	Pro	perties: QT GUI Frequency Sink 🛛 🔗	QT GUI Histogram Sink Number of Points: 1.024k	Properties: QT GUI Histogr		
Center Frequency (Hz): 0 Bandwidth (Hz): 32k	General Trigger Co	nfig Advanced Documentation	Number of Points: 1.024k Number of Bins: 100 Autoscale: Yes	General Config A	dvanced Documentation	
	<u>ID</u>	qtgui_freq_sink_x_0	Accumulate: No	ID	qtgui_histogram_sink_x_	
	Туре	Complex 2	Min x-axis: -1 Max x-axis: 1	Type	Float 🛟	
	Name	Name		Name		
	FFT Size	1024		Number of Points	1024	
	Window Type	Blackman-harris 💲		Number of Bins	100	
	Center Frequency (Hz)	0		Grid	No ‡	
	Bandwidth (Hz)	samp_rate		Autoscale	Yes ‡	
	Grid	No ‡		Accumulate	No ‡	
	Autoscale	No ‡		Min x-axis	-1	
	Average Y min	None ‡		Max x-axis	1	
		-140		Number of Inputs	1	
	Y max	10		Update Period	0.10	
	Y label	Relative Gain		GUI Hint		
	Y units	dB				
	Number of Inputs	1				
		OK Cancel Apply			ок	

Figure 1.29: GNU Radio instrumentation sinks -QT GUI Frequency Sink

QT GUI Number Sink Autoscale: No		Properties: QT GUI Number Sink 🛛 😣
Average: 0 Graph Type: Horizontal	General Config	g Advanced Documentation
draph type. nonzontal	ID	qtgui_number_sink_0
	Name	**
	Input Type	Float ‡
	Autoscale	No ‡
	Average	0
	Graph Type	Horizontal 👙
	Number of Inpu	outs 1
	Min	1
	Max	1
	Update Period	od 0.10
	Sink - in(0): Port is not cor	
		OK Cancel Apply

Figure 1.31: GNU Radio instrumentation sinks -QT GUI Number Sink

QT GUI Time Sink	F	Properties: QT GUI Time Sink 🛛 😣
Sample Rate: 32k Autoscale: No	General Trigger Co	nfig Advanced Documentation
	ID	qtgui_time_sink_x_0
	Type	Complex 🛟
	Name	-
	Y Axis Label	Amplitude
	Y Axis Unit	
	Number of Points	1024
	Sample Rate	samp_rate
	Grid	No ‡
	Autoscale	No ‡
	<u>Y min</u>	-1
	<u>Y max</u>	1
	Number of Inputs	1
	Update Period	0.10
	Disp. Tags	Yes 💲
	GUI Hint	
		OK Cancel Apply

Figure 1.33: GNU Radio instrumentation sinks -QT GUI Time Sink

Cancel Apply

Figure 1.30: GNU Radio instrumentation sinks -QT GUI Histogram Sink

QT GUI Time Raster Sink Sample Rate: 32k	Prop	erties: QT GUI Time Raster Sink 🛛 🔗
Num. Rows: Num. Cols:	General Config Adv	anced Documentation
	ID	qtgui_time_raster_sink_x_0
	Туре	Byte ‡
	Name	-
	Sample Rate	samp_rate
	Num. Rows	
	Num. Cols	
	Grid	No ‡
	Int. min	-1
	Int. max	1
	Multiplier	0
	Offcot	n 🖳
	Param - Num. Rows(nro Value "" cannot be e Cannot evaluate em	evaluated:
	Param - Num. Cols(nco Value "" cannot be e Cannot evaluate em	evaluated:
		OK Cancel Apply

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

QT C Vect X-Ax X-Ax Y-Ax Y-Ax R-Ax Ref L

t Value: 0	General	Config	Advance	d Documentation
Value: 1		ID		i vector sink f 0
lt y-Axis		lame	qcgc	I_VECCOT_SINK_T_O
s: s:				
0		tor Size	1024	1
	X-Axis :	Start Valu	16 0	
	X-Axis	Step Valu	ie 1.0	
	X-Ax	cis Label	"x-A	xis"
	Y-As	kis Label	"y-A	xis"
	<u>X-A</u>	<u>kis Units</u>	***	
	Y-Ax	cis Units	***	
	Re	f Level	0	
		Grid	No	•
	Sink - in(0) Port is	): s not cor	nnected.	

Figure 1.34: GNU Radio instrumentation sinks -QT GUI Vector Sink

QT GUI Waterfall Sink	Pro	perties: QT GUI Waterfall Sink 🛛 😵	WX GUI Constellation Sink	Prop	erties: WX GUI Constellation Sink	8
Center Frequency (Hz): 0 Bandwidth (Hz): 32k	General Config Adv	vanced Documentation	Sample Rate: 32k	General Advanced	Documentation	
	ID	qtgui_waterfall_sink_x_0	Constellation Size: 2.048k M: 4 Theta: 0 Loop Bandwidth: 62.8m Max Freq: 60m Mu: 500m Gain Mu: 5m Symbol Rate: 8k	ID	wxgui_constellationsink2_0	
	Type	Complex ‡		Title	Constellation Plot	
	Name	**		Sample Rate	samp_rate	
	FFT Size	1024		Frame Rate	5	
	Window Type	Blackman-harris 👻		Constellation Size	2048	
	Center Frequency (Hz)	0	Omega Limit: 5m	м	4	
	Bandwidth (Hz)	samp_rate		Theta	0	
	Intensity Min	-140		Loop Bandwidth	6.28/100.0	
	Intensity Max	10		Max Freq	0.06	
	Grid	No ‡		Mu	0.5	
	Number of Inputs	1		Gain Mu	0.005	
	Update Period	0.10		Can't generate this bl	ock in mode: 'qt_gui'	
	GUI Hint			Sink - in(0):		
	Show Msg Ports	No ‡		Port is not connec	ted.	
		ОК Cancel Apply			OK Cancel Apply	D

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

Figure 1.36: GNU Radio instrumentation sinks -	_
WX GUI Constellation Sink	

u histosink2 0

General Advanced D

Title <u>Num Bins</u> <u>Frame Size</u> Window Size Grid Position Notebook

WX GUI FFT Sink	1	Properties: WX GUI FFT Sink 🛛 😣
Sample Rate: 32k	General Advanced	Documentation
Baseband Freq: 0 Y per Div: 10 dB Y Divs: 10 Ref Level (dB): 0	ID	wxgui_fftsink2_0
	Туре	Complex 💲
Ref Scale (p2p): 2 FFT Size: 1.024k	Title	FFT Plot
Refresh Rate: 15 Freg Set Varname: None	Sample Rate	samp_rate
	Baseband Freq	0
	Y per Div	10 dB 🛟
	Y Divs	10
	Ref Level (dB)	0
	Ref Scale (p2p)	2.0
	FFT Size	1024
	Dofroch Data	lee 🕑
	Can't generate this blo	ick in mode: 'qt_gui'
	Sink - in(0): Port is not connect	ed.
		OK Cancel Apply

Figure 1.37: GNU Radio instrumentation sinks –

WX GUI FFT Sink

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

OK Cancel Apply

Can't generate this block in mode: 'qt\_gui Sink - in(0): Port is not connected.

WX GUI Number Sink Title: Number Plot	Properties: WX GUI Number Sink 🛛 😣	
Units: Units Sample Rate: 32k	General Advance	Documentation
Min Value: -100 Max Value: 100 Factor: 1 Decimal Places: 10 Reference Level: 0	ID	wxgui_numbersink2_0
	Туре	Complex 💠
	Title	Number Plot
Number Rate: 15 Show Gauge: Show	Units	Units
Show dudger show	Sample Rate	samp_rate
	Min Value	-100
	Max Value	100
	Factor	1.0
	Decimal Place	s 10
	Reference Lev	el O
	Number Rate	
	Can't generate th	is block in mode: 'qt_gui'
	Sink - in(0): Port is not co	nnected.
		OK Cancel Apply

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

WX GUI Histo Sink

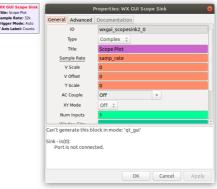


Figure 1.40: GNU Radio instrumentation sinks – **WX GUI Scope Sink** 

WX GUI Terminal Sink	Sink Properties: WX GUI Terminal Sink 🛛 😣		WX GUI Waterfall Sink Title: Waterfall Plot Sample Rate: 32k Baseband Freq: 0 Dynamic Range: 100 Reference Level: 0	Properties: WX GUI Waterfall Sink 🛛 😣		
	General Advanced Documentation			General Advanced Documentation		
	ID wxgui_termsink_0	ID		wxgui_waterfallsink2_0		
	Window Size		Ref Scale (p2p): 2 FFT Size: 512	Туре	Complex 💲	
	Grid Position		FFT Rate: 15	Title	Waterfall Plot	
	Notebook		Freq Set Varname: None	Sample Rate	samp_rate	
				Baseband Freq	0	
				Dynamic Range	100	
				Reference Level	0	
				Ref Scale (p2p)	2.0	
				FFT Size	512	
				FFT Rate	15	
				<u>Averane</u>	Off *	
	Can't generate this block in mode	:: 'qt_gui'		Can't generate this block in mode: 'qt_gui'		
	Sink - in(msg): Port is not connected.			Sink - in(0): Port is not connec	ted.	
		OK Cancel Apply			OK Cancel Apply	

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

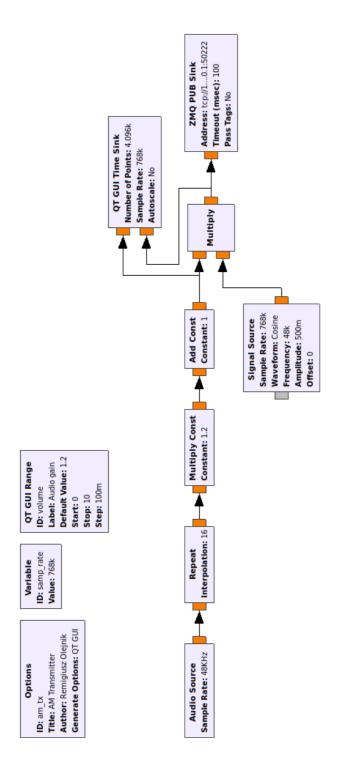
AM receiver flowgraph is shown in Fig. 2.2. The parameters of the building blocks are following:

- 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.

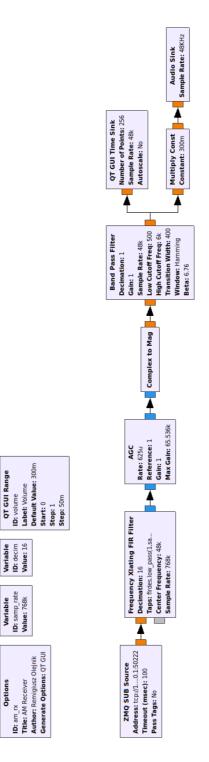


Figure 2.2: GNU Radio – AM receiver flowgraph

QT GUI Range

Options

## 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.

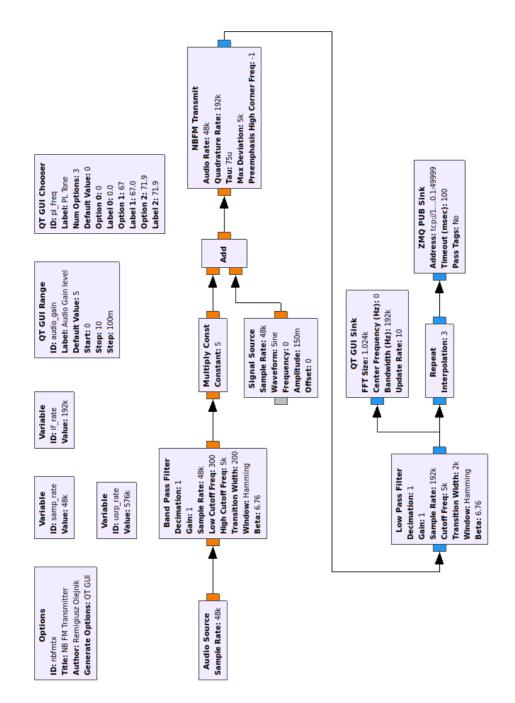


Figure 2.3: GNU Radio – FM transmitter flowgraph

### 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

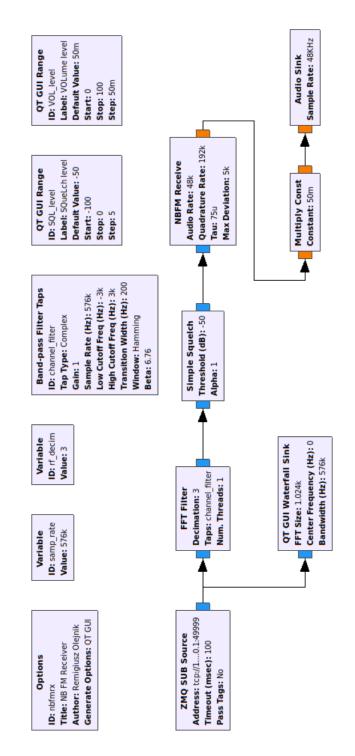


Figure 2.4: GNU Radio – FM receiver flowgraph

- 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

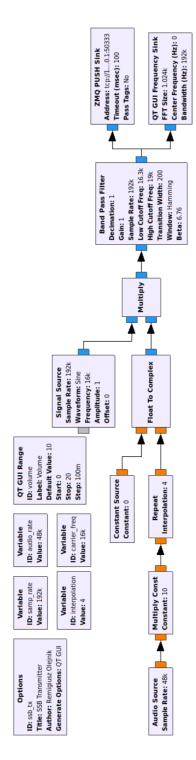


Figure 2.5: GNU Radio – SSB transmitter flowgraph

- 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

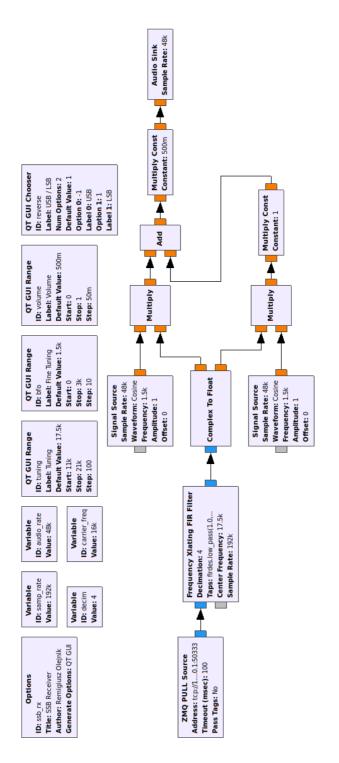


Figure 2.6: GNU Radio – SSB receiver flowgraph

- 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

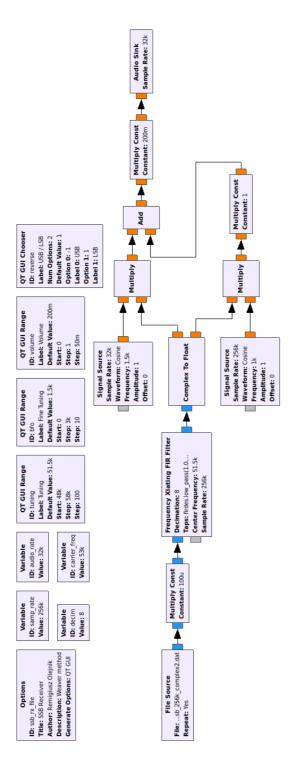


Figure 2.7: GNU Radio - SSB receiver flowgraph - I/Q signal from the file

- 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)

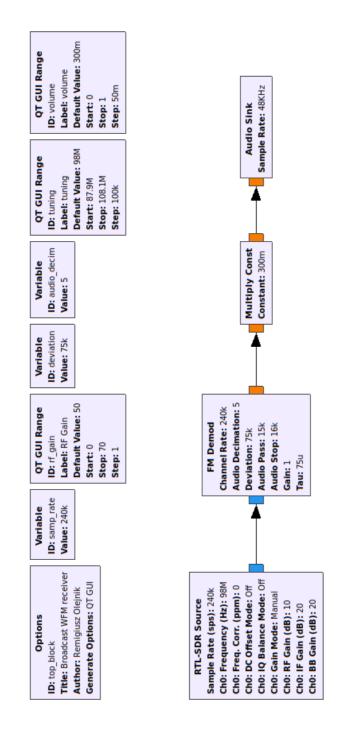


Figure 2.8: GNU Radio – RTL-SDR based broadcast WFM receiver

# 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

1.1: GNU Radio – seven blocks connected together form a flowgraph	7
1.2: GNU Radio – two blocks (Signal Source and Audio Sink) connected with	
$\partial$	7
1.3: GNU Radio – Variable samp_rate block	9
	0
1.5: GNU Radio signal data types	1
1.6: GNU Radio sources – Null Source	2
1.7: GNU Radio sources – Noise Source	2
1.8: GNU Radio sources – Signal Source	2
1.9: GNU Radio sources – File Source	2
1.10: GNU Radio sources – TCP Source	2
1.11: GNU Radio sources – UDP Source	2
1.12: GNU Radio sources – Audio Source	3
1.13: GNU Radio sources – WAV File Source	3
1.14: GNU Radio sources – UHD: USRP Source	3
1.15: GNU Radio sources – osmocom Source	3
1.16: GNU Radio sources – RTL-SDR Source	3
1.17: GNU Radio sources – Funcube Dongle Source	3
1.18: GNU Radio sinks – Null Sink	4
1.19: GNU Radio sinks – File Sink	4
1.20: GNU Radio sinks – TCP Sink	4
1.21: GNU Radio sinks – TCP Server Sink	4
1.22: GNU Radio sinks – UDP Sink	5
	5
1.24: GNU Radio sinks – WAV File Sink	5
1.25: GNU Radio sinks – UHD: USRP Sink	5
1.26: GNU Radio sinks – osmocom Sink	5
1.27: GNU Radio instrumentation sinks – QT GUI Sink	6
1.28: GNU Radio instrumentation sinks – QT GUI Constellation Sink 1	6
1.29: GNU Radio instrumentation sinks – QT GUI Frequency Sink 1	7
1.30: GNU Radio instrumentation sinks – QT GUI Histogram Sink 1	7
1.31: GNU Radio instrumentation sinks – QT GUI Number Sink 1	7
1.32: GNU Radio instrumentation sinks – QT GUI Time Raster Sink 1	7
1.33: GNU Radio instrumentation sinks – QT GUI Time Sink	7
1.34: GNU Radio instrumentation sinks – QT GUI Vector Sink	7
-	8
	8
1.37: GNU Radio instrumentation sinks – WX GUI FFT Sink	8

18
18
18
19
19
22
24
27
30
33
36
39
41

# 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