Fundamentals of Radio Communication

1 Introduction

Analog Radio

In a traditional analog radio, like a “Walkie-Talkie”, there is a physical chain of circuits which work to process an incoming or outgoing signal. For a receiver, an antenna and front end collect and amplify an incoming radio frequency (RF) signal. A local oscillator and mixer shift the desired station from its broadcast frequency down to an intermediate frequency where filters can isolate it. A demodulator then recovers the information (e.g. voice, music, etc.) and an audio amplifier drives a speaker. Every change to that radio’s behavior lives in the physical circuits that make up the radio; if you want a different filter or new modulation type, then the hardware must be redesigned.

Software-Defined Radio

A software-defined radio (SDR) keeps only the minimum analog front end and moves everything else into code running on a computer. An analog-to-digital converter (ADC) samples, or measures, the RF signal and turns it into a stream of numbers that can be processed with computers. Tuning, filtering, demodulation, even entire modulation schemes become algorithms that operate on those samples. If you want to change how the radio operates, all you have to do is update the software/algorithm. GNU Radio is a software that allows you to easily build digital signal processing (DSP) systems by connecting together signal processing blocks into a flowgraph.

In this lab we’ll take one more step by keeping the whole radio link inside the computer. Instead of sampling a real RF signal, we generate and process it entirely in software. The “transmitter”, “channel”, and “receiver” are just connected GNU Radio blocks, so nothing is radiated and no hardware needs wiring. The only physical interfaces you’ll need are you computer’s audio devices, e.g. microphone in, headphones out.

2 GNU Radio Flowgraphs

Default rates: `audio_rate = 48000`, `samp_rate = 192000` **Start deviation:** `nbfm_deviation = 5000` Hz

Overview of the two-graph setup (ZMQ)

You will run two separate GNU Radio flowgraphs linked by ZeroMQ (ZMQ) at complex baseband. The transmitter (TX) generates NBFM from your microphone and pushes complex IQ samples over the network; the receiver (RX) pulls those samples, optionally adds impairments, demodulates, and plays audio.

TX flowgraph (block order)

`Audio Source` → `Multiply Const (mic gain)` → `NBFM Transmit` → `ZMQ PUSH Sink`
Settings: Audio Rate = 48000; Quadrature Rate = 192000; Max Deviation = 2000–7000 Hz; Tau = 0 μs (base lab). In ZMQ PUSH Sink set *Bind = On*, Address = `tcp://*:PORT` (e.g. `tcp://*:6001`), High-Water-Mark (HWM) = 25, Linger = 0. Do *not* use a Throttle block on TX when using ZMQ.

RX flowgraph (block order)
`ZMQ PULL Source` → `Channel Model (optional)` → `NBFM Receive` → `Multiply Const (volume)` → `Audio Sink`
Settings: Quadrature Rate = 192000; Audio Rate = 48000; Tau = 0 μs. In ZMQ PULL Source set *Bind = Off*, Address = `tcp://TX_IP:PORT` (e.g. `tcp://192.168.1.50:6001`), HWM = 25.

Required variables (both graphs)
`audio_rate = 48000  samp_rate = 192000  port = 6001` (unique per pair)
TX Address: `tcp://*:${port}`   RX Address: `tcp://TX_IP:${port}`

Instructions for use

Same-machine dry run Start the TX graph first with Address `tcp://*:6001` and Bind = On. Start the RX graph on the same machine with Address `tcp://127.0.0.1:6001` and Bind = Off. Speak into the mic; you should hear your voice with a small delay. If you have a frequency sink on RX, you should see energy centered at 0 Hz that responds to speech.

Two-computer run Choose a unique port per team (e.g. 6001, 6002, ...). On TX, run with Bind = On and Address `tcp://*:PORT`. Determine the TX IP address (Windows: `ipconfig`; macOS/Linux: `ifconfig` or `ip addr`). On RX, set Address `tcp://TX_IP:PORT` with Bind = Off, then run RX. Wear headphones on RX.

Bring-up checklist (Checkpoint A)
Audio heard on RX   RF spectrum active at 0 Hz (if plotted)   Audio waveform visible (if plotted)
If silent: verify matching sample rates (48 k/192 k), TX binds and RX connects to the correct IP:PORT, and OS firewalls allow Python on that port. Set mic gain and volume near 1.0–2.0.

B. Deviation vs. bandwidth (Carson)

Goal. Measure occupied bandwidth vs. deviation Δf and compare to $B \approx 2(\Delta f + 3 \text{ kHz})$.

Set TX Max Deviation to 3, 5, and 7, one at a time. On RX, enable averaging in the frequency sink (if present). While speaking steadily, estimate the main-lobe width down to roughly −20 dB.

Δf (kHz)	Measured BW (kHz)	Carson $2(\Delta f + 3)$ (kHz)	Notes
3	________	12	________________
5	________	16	________________
7	________	20	________________

B1 (1–2 sentences). Does measured bandwidth grow approximately linearly with Δf? If it differs from Carson, give one plausible reason (measurement threshold, speech spectrum, etc.).

Screenshots. Capture RF spectra for one low-Δf and one high-Δf case; label each with Δf.

C. Mic gain and apparent over-deviation

Goal. Show how excessive input level widens the spectrum and degrades audio.

Sweep TX mic gain from ≈ 0.5 to 3.0 while keeping voice level constant. Listen for distortion and watch spectral shoulders widen even if Δf is unchanged.

C1 (1–2 sentences). Why can too much mic gain look like larger Δf in the spectrum? What trade-off did you hear?

D. Noise robustness

Goal. Observe FM's gradual degradation with noise.

On RX, set Channel Model Noise Voltage to 0.00, 0.01, then 0.02. For each, note intelligibility and how the spectrum fattens. Optionally, insert a Power Squelch (Complex) before NBFM Receive (threshold ≈ -60 dB, $\alpha = 10^{-4}$, Gate = True) and toggle it.

D1 (2–3 sentences). At what noise level did speech become hard to follow? Why does FM fail gradually rather than abruptly?

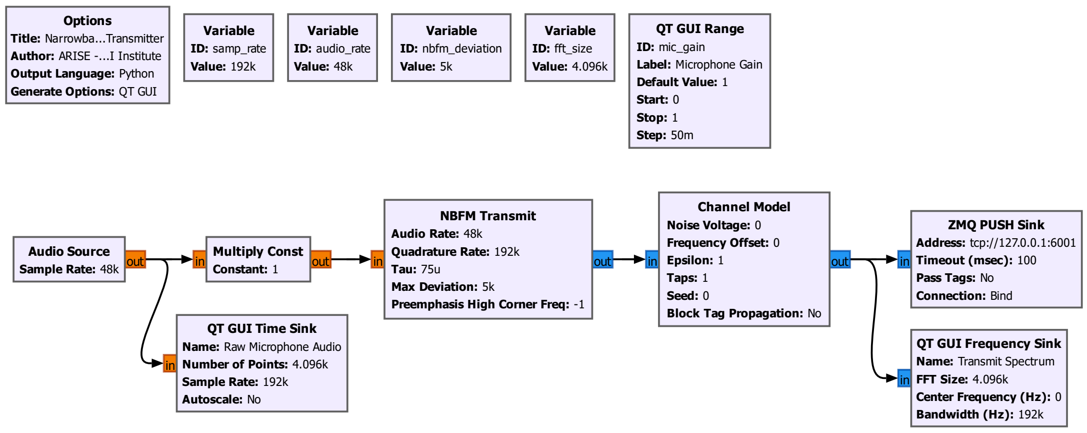
Screenshot. Capture an RF spectrum at your "noisy" setting; label with the Noise Voltage.

E. Carrier-frequency offset (CFO) tolerance

Goal. Explore mistuning tolerance.

On RX, set Channel Model Frequency Offset to 0, +200 Hz, −200 Hz, then ±800 Hz. Note when audio becomes warbly or collapses.

E1 (1–2 sentences). Why is small CFO tolerable in FM? What finally breaks as CFO increases?



Options
Title: Narrowba...Transmitter
Author: ARISE -...I Institute
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 192k

Variable
ID: audio_rate
Value: 48k

Variable
ID: nbfm_deviation
Value: 5k

Variable
ID: fft_size
Value: 4.096k

QT GUI Range
ID: mic_gain
Label: Microphone Gain
Default Value: 1
Start: 0
Stop: 1
Step: 50m

Audio Source
Sample Rate: 48k

Multiply Const
Constant: 1

NBFM Transmit
Audio Rate: 48k
Quadrature Rate: 192k
Tau: 75u
Max Deviation: 5k
Preemphasis High Corner Freq: -1

Channel Model
Noise Voltage: 0
Frequency Offset: 0
Epsilon: 1
Taps: 1
Seed: 0
Block Tag Propagation: No

ZMQ PUSH Sink
Address: tcp://127.0.0.1:6001
Timeout (msec): 100
Pass Tags: No
Connection: Bind

QT GUI Time Sink
Name: Raw Microphone Audio
Number of Points: 4.096k
Sample Rate: 192k
Autoscale: No

QT GUI Frequency Sink
Name: Transmit Spectrum
FFT Size: 4.096k
Center Frequency (Hz): 0
Bandwidth (Hz): 192k

Figure 1: `NBFM_Virtual_Transmitter.grc` GNU Radio flowgraph.

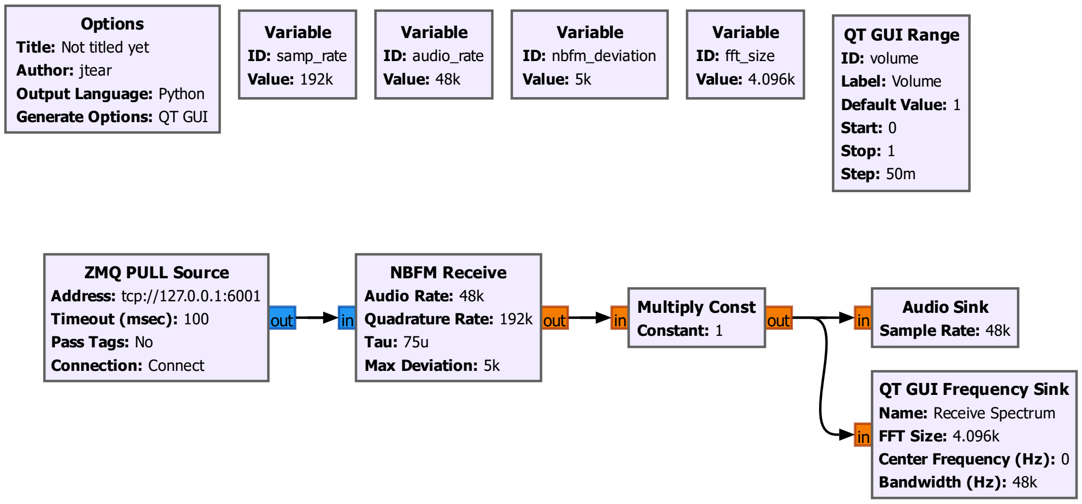


Figure 2: `NBFM_Virtual_Receiver.grc` GNU Radio flowgraph.