# Software Defined Radio Projects

S. Racz

ECE 1898
Dr. S. Jacobs
Spring 2021

## Contents

## Table of Acronyms

| | |
|---|---|
| **SDR** | software defined radio |
| **op-amp** | operational amplifier |
| **DSP** | digital signal processing |
| **GR** | GNURadio |
| **GRC** | *GNURadio Companion* |
| **GUI** | graphical user interface |
| **complex64** | complex number where each component is a 32-bit float |
| **FED** | frontend device |
| **AA** | anti-aliasing |
| **LPF** | low-pass filter |
| **BPF** | band-pass filter |
| **HPF** | high-pass filter |
| **FM** | frequency modulation |
| **AM** | amplitude modulation |
| **DSB** | dual side band modulation |
| **ASK** | amplitude shift keying |
| **BASK** | binary amplitude shift keying |
| **OOK** | on-off keying |
| **I** | in-phase component |
| **Q** | quadrature-phase component |
| **IQ** | in-phase quadrature-phase |
| **ADC** | analog-to-digital converter |
| **DAC** | digital-to-analog converter |
| **SqLD** | square-law device |
| **SBR** | symbol-to-bit ratio |
| **NRZ** | non-return-to-zero |
| **RRC** | root-raised cosine |
| **preD** | pre-demodulation |
| **BER** | bit error rate |
| **SNR** | signal-to-noise ratio |
| **sps** | samples-per-symbol |
| **TX** | transmit |
| **RX** | receive |
| **RF** | radiofrequency |
| **FPGA** | field-programmable gate array |

# I Introduction

There exists a trend in nearly all disciplines of electrical engineering that functionality is taken out of the hardware realm and instead realized in software. For example, the operational amplifier (op-amp) is an integrated circuit which filled a need that, for the most part, no longer exists: arithmetic operations on signals. Due to its efficiency and design flexibility, digital signal processing (DSP) has replaced the massive circuits with dozens of op-amps with a sleek embedded device.

This "hardware grim reaper" has met the discipline of radio as well. This era, due to ubiquity of embedded devices and the extreme dependence on digital modes in our society, will be defined by software defined radio (SDR). SDR is a design philosophy where DSP functions are implemented in software or firmware in algorithmic form.

GNURadio (GR) is an SDR development framework. The framework itself is written in Python and includes a tool that allows for the creation of components compiled in C++. It organizes the flow of data, or "flowgraph," into "connections" and "blocks." Many default blocks are provided for a variety of uses; the design philosophy for GR is to develop blocks as generally and as single-function as possible so that they can be easily used in other flowgraphs. An entire flowgraph is described in a single *.py file, containing the "top block," in which is instantiated the blocks for the flow graph and how they connect, and a main function, which instantiates the top block when called. The framework provides a graphical user interface (GUI) to speed up flowgraph development called *GNURadio Companion* (GRC).

The most critical aspect of an SDR system is its hardware frontend. This frontend device (FED) is essentially an antenna, an in-phase quadrature-phase (IQ) modem, and an interface with the host device (often a computer). For example, to receive signals at a certain frequency and sample rate, the FED is commanded by the host to tune its demodulator to that frequency. It then takes samples at the specified sample rate of the spectrum centered around that frequency, converts it down to baseband, and sends this IQ data to the host device over some peripheral interface. To transmit, the opposite process occurs; a stream of IQ data is sent to the frontend, and the frontend converts these data to an analog waveform and modulates them to a specified center frequency.

Complex signals are often regarded as purely theoretical, but this is not so in the realm of SDR. Complex signal representation is imperative to SDR systems as the FED must be able to down-convert or up-convert signals according to any modulation scheme. Every modulation scheme can be described in general as an IQ modem.

This paper reports on the development and testing of three projects: an frequency modulation (FM) commercial broadcast receiver, compatible with mono- or stereophonic audio; a code-level implementation of a narrowband FM detector block using GR; and an on-off keying (OOK) modem that sends and receives textual payloads.

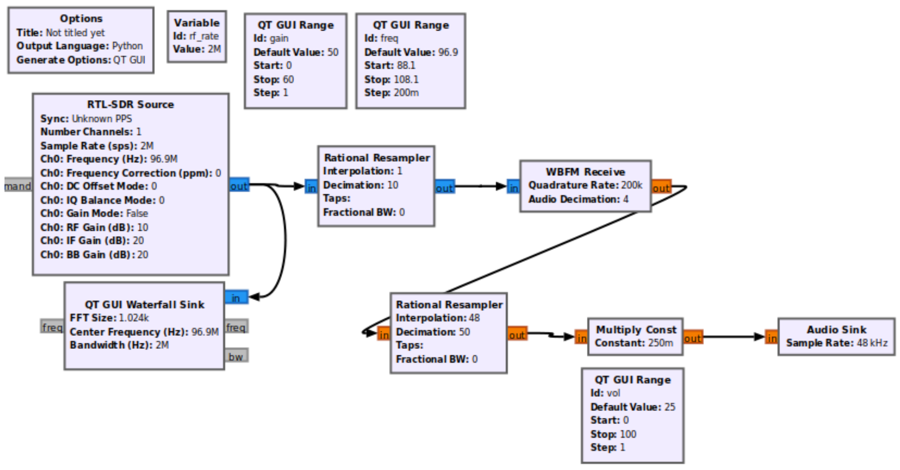
# II FM Commercial Broadcast Receiver

Though it may soon be changing, commercial broadcasting is one of the last champions of analog FM with an audio payload. Analog modulation schemes benefit from DSP in that it tends to save power and increases the flexibility of the device.

## II.A Monophonic Audio

### II.A.i Flowgraph

This flowgraph shown in Figure 1 is the first success of the project. Its goal is to demodulate a standard FM signal from a commercial broadcast source and produce its monophonic audio.


**Options**
**Title:** Not titled yet
**Output Language:** Python
**Generate Options:** QT GUI

**Variable**
**Id:** rf_rate
**Value:** 2M

**QT GUI Range**
**Id:** gain
**Default Value:** 50
**Start:** 0
**Stop:** 60
**Step:** 1

**QT GUI Range**
**Id:** freq
**Default Value:** 96.9
**Start:** 88.1
**Stop:** 108.1
**Step:** 200m

**RTL-SDR Source**
**Sync:** Unknown PPS
**Number Channels:** 1
**Sample Rate (sps):** 2M
**Ch0: Frequency (Hz):** 96.9M
**Ch0: Frequency Correction (ppm):** 0
**Ch0: DC Offset Mode:** 0
**Ch0: IQ Balance Mode:** 0
**Ch0: Gain Mode:** False
**Ch0: RF Gain (dB):** 10
**Ch0: IF Gain (dB):** 20
**Ch0: BB Gain (dB):** 20

**Rational Resampler**
**Interpolation:** 1
**Decimation:** 10
**Taps:**
**Fractional BW:** 0

**WBFM Receive**
**Quadrature Rate:** 200k
**Audio Decimation:** 4

**QT GUI Waterfall Sink**
**FFT Size:** 1.024k
**Center Frequency (Hz):** 96.9M
**Bandwidth (Hz):** 2M

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

**Multiply Const**
**Constant:** 250m

**Audio Sink**
**Sample Rate:** 48 kHz

**QT GUI Range**
**Id:** vol
**Default Value:** 25
**Start:** 0
**Stop:** 100
**Step:** 1

Figure 1: Monophonic FM broadcast demodulator.

A source block is needed to control the frontend device. The “osmocom Source” block is the block that can runs the driver for the *HackRF One*. The output of this block is a complex number where each component is a 32-bit float (complex64). Using the “QT GUI Waterfall Sink,” a waterfall-style spectral display is instantiated in the flowgraph’s GUI.

The actual signal processing begins at the rational resampler. This block can perform fully-filtered upsampling and downsampling. The anti-aliasing (AA) filter can either be specified directly with a list of taps or by specifying the fractional bandwidth directly. When 0 or nothing is specified, the block automatically designs a filter whose fractional bandwidth is 0.4. In this flowgraph, the rational resampler is configured to filter the signal with an AA filter whose cutoff frequency is approximately $0.4 * f_s$, where $f_s$ is the sample rate. The new sample rate is now 200 kHz.

A block is now needed to demodulate the baseband FM signal. GR provides a ready-made block for this purpose, entitled ”WBFM Receive.” This block completes a few processing steps at once. Its demodulation algorithm also directly downsamples the output signal. This algorithm is only capable of an integer decimation factor.

With the data fully demodulated, another resampling step is required to be able to play the data at a rate compatible with the computer's soundcard. A slight adjustment from 50 kHz to 48 kHz is performed with another rational resampler, a multiplier is added for a volume knob, and the data is sent to the soundcard.

## II.B Stereophonic Audio

### II.B.i Transmitting and Receiving Stereophonic Audio Payload

Most FM commercial broadcast stations transmit data sufficient for stereophonic audio recovery. A monophonic audio signal is by definition the sum of the left and right channels as shown in Equation 1. This is how the standard can provide a monophonic version at the center of the station's bandwidth. In order to then recover a stereophonic sound image, the difference between the channels can be used in conjunction with the monophonic signal. The difference signal is generated by subtracting the left channel from the right channel as in Equation 2.

$$s_{mono[n]} = s_{L[n]} + s_{R[n]} \tag{1}$$

$$s_{diff[n]} = s_{L[n]} - s_{R[n]} \tag{2}$$

To shift the difference signal out of baseband, dual side band modulation (DSB) is applied with a carrier frequency of 38 kHz [1]. The monophonic signal is then added to this along with a 19 kHz pilot tone to create the full baseband signal [1]. This entire spectrum is frequency modulated to the carrier frequency of the station.

At the receiver, the spectrum is frequency demodulated to baseband. The DSB difference signal is then also demodulated. Rearranging the equations above, the left and right channels can be recovered with Equations 3 and 4. Figure 2 is the block diagram of the demodulator.

$$s_{L[n]} = s_{mono[n]} + s_{diff[n]} \tag{3}$$

$$s_{R[n]} = s_{mono[n]} - s_{diff[n]} \tag{4}$$

### II.B.ii Flowgraph

Figure 3 shows the initial sample reception and partial demodulation. The frontend source block is configured for a sample rate of 2 MHz with an adjustable center frequency, channelized by a range GUI element. Like in the case of the monophonic flowgraph, the spectrum is displayed in a waterfall and the samples are first downsampled by a factor of 10.

When trying to use the wide band FM receive block, a problem with filtering occured. It appeared that the "WBFM Receive" was applying aggressive filtering, which muddled out the DSB side lobes. This was overcome by using a more basic FM detector and doing manual filtering. To overcome this, a basic FM detector was applied. The FM

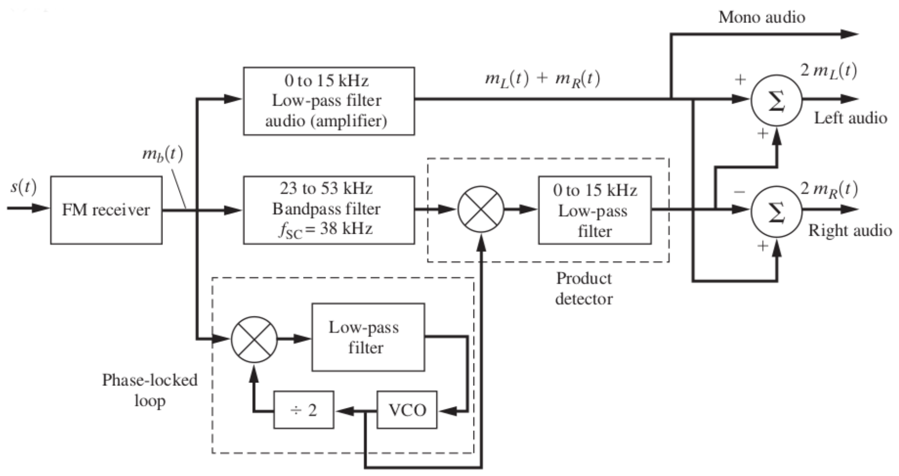
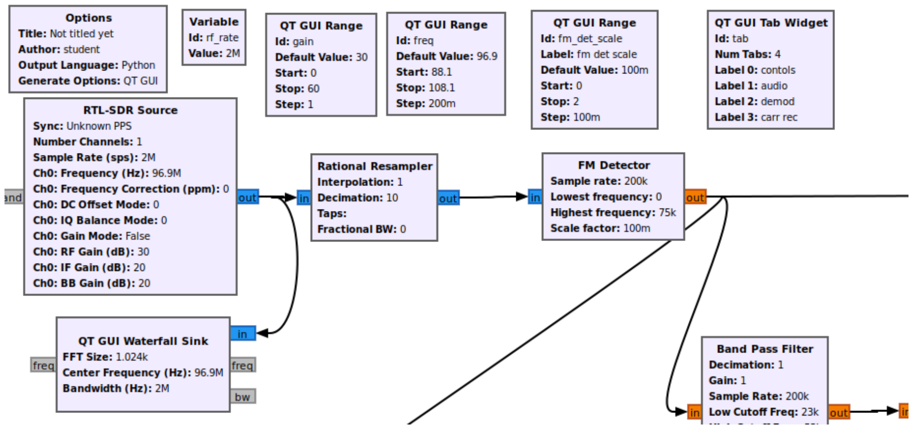


Figure 2: block diagram of a stereophonic FM broadcast demodulator. [1]



**Options**
**Title:** Not titled yet
**Author:** student
**Output Language:** Python
**Generate Options:** QT GUI

**Variable**
**Id:** rf_rate
**Value:** 2M

**QT GUI Range**
**Id:** gain
**Default Value:** 30
**Start:** 0
**Stop:** 60
**Step:** 1

**QT GUI Range**
**Id:** freq
**Default Value:** 96.9
**Start:** 88.1
**Stop:** 108.1
**Step:** 200m

**QT GUI Range**
**Id:** fm_det_scale
**Label:** fm det scale
**Default Value:** 100m
**Start:** 0
**Stop:** 2
**Step:** 100m

**QT GUI Tab Widget**
**Id:** tab
**Num Tabs:** 4
**Label 0:** contols
**Label 1:** audio
**Label 2:** demod
**Label 3:** carr rec

**RTL-SDR Source**
**Sync:** Unknown PPS
**Number Channels:** 1
**Sample Rate (sps):** 2M
**Ch0: Frequency (Hz):** 96.9M
**Ch0: Frequency Correction (ppm):** 0
**Ch0: DC Offset Mode:** 0
**Ch0: IQ Balance Mode:** 0
**Ch0: Gain Mode:** False
**Ch0: RF Gain (dB):** 30
**Ch0: IF Gain (dB):** 20
**Ch0: BB Gain (dB):** 20

**Rational Resampler**
**Interpolation:** 1
**Decimation:** 10
**Taps:**
**Fractional BW:** 0

**FM Detector**
**Sample rate:** 200k
**Lowest frequency:** 0
**Highest frequency:** 75k
**Scale factor:** 100m

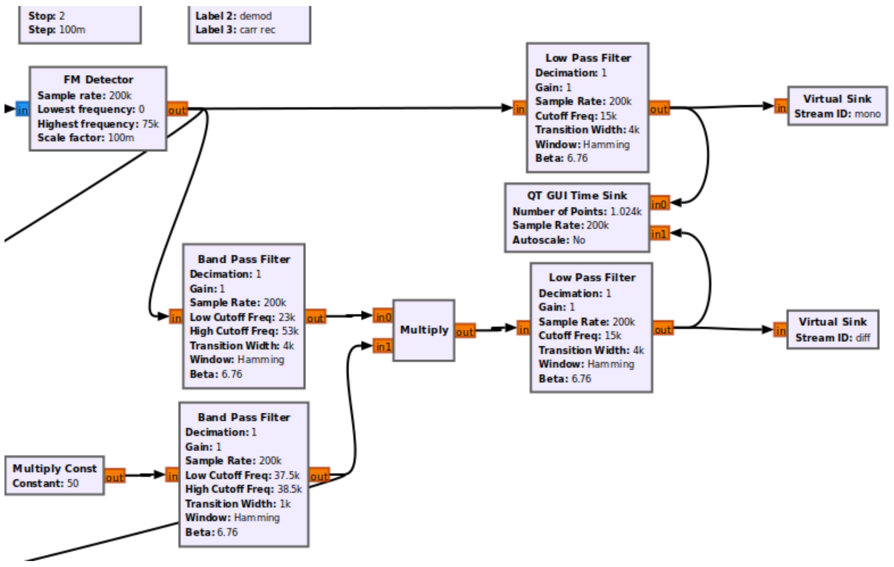
**QT GUI Waterfall Sink**
**FFT Size:** 1.024k
**Center Frequency (Hz):** 96.9M
**Bandwidth (Hz):** 2M

**Band Pass Filter**
**Decimation:** 1
**Gain:** 1
**Sample Rate:** 200k
**Low Cutoff Freq:** 23k

Figure 3: initial channel sampling and FM detection.

detector included by default in GR has some filtering and scaling to map the modulated frequencies to baseband. The maximum deviation of an FM signal is 75 kHz [1]. A scaling factor is also necessary to tune the output waveform to an appropriate amplitude.

The next step requires the monophonic and difference signals to be demodulated simultaneously. Both demodulation steps are conducted with the blocks in Figure 4. The monophonic demodulation from this point is as simple as applying a low-pass filter (LPF) at 15 kHz. The difference signal, however, is still DSB modulated centered at 38 kHz. This is recovered by isolating it with a band-pass filter (BPF), multiplying it by its regenerated carrier, and applying another LPF to select the copy of the spectrum that was shifted to baseband. This topology is known as a product detector [1].



Stop: 2
Step: 100m

Label 2: demod
Label 3: carr rec

FM Detector
Sample rate: 200k
Lowest frequency: 0
Highest frequency: 75k
Scale factor: 100m

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Cutoff Freq: 15k
Transition Width: 4k
Window: Hamming
Beta: 6.76

Virtual Sink
Stream ID: mono

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 200k
Autoscale: No

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Low Cutoff Freq: 23k
High Cutoff Freq: 53k
Transition Width: 4k
Window: Hamming
Beta: 6.76

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Cutoff Freq: 15k
Transition Width: 4k
Window: Hamming
Beta: 6.76

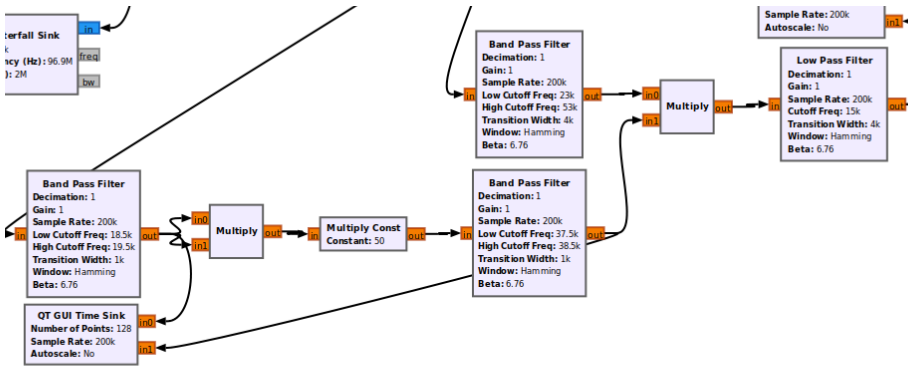
Virtual Sink
Stream ID: diff

Multiply Const
Constant: 50

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Low Cutoff Freq: 37.5k
High Cutoff Freq: 38.5k
Transition Width: 1k
Window: Hamming
Beta: 6.76

Figure 4: concurrent monophonic and difference signal demodulation.

In Figure 5, the carrier is recovered and regenerated from the 19 kHz pilot tone. A narrow BPF is employed in this flowgraph as a tone detector. Squaring the output of the BPF and filtering out the DC component with another BPF results in the recovered 38 kHz carrier wave. Some scaling is applied so that the difference signal is of appropriate magnitude.



terfall Sink
k
ncy (Hz): 96.9M
): 2M

Sample Rate: 200k
Autoscale: No

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Low Cutoff Freq: 23k
High Cutoff Freq: 53k
Transition Width: 4k
Window: Hamming
Beta: 6.76

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Cutoff Freq: 15k
Transition Width: 4k
Window: Hamming
Beta: 6.76

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Low Cutoff Freq: 18.5k
High Cutoff Freq: 19.5k
Transition Width: 1k
Window: Hamming
Beta: 6.76

Multiply Const
Constant: 50

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Low Cutoff Freq: 37.5k
High Cutoff Freq: 38.5k
Transition Width: 1k
Window: Hamming
Beta: 6.76

QT GUI Time Sink
Number of Points: 128
Sample Rate: 200k
Autoscale: No

Figure 5: carrier recovery for the DSB component of a stereophonic FM broadcast signal.

Now that both the monophonic and difference signals have been demodulated, it is time to apply the left and right channel recovery equations given in II.B.i. According to Figure 6, after resampling and volume control, the data are sent to the left and right channels of the soundcard. (N.B. - the virtual sources and sinks are equivalent to arrows going between those with the same stream ID.)

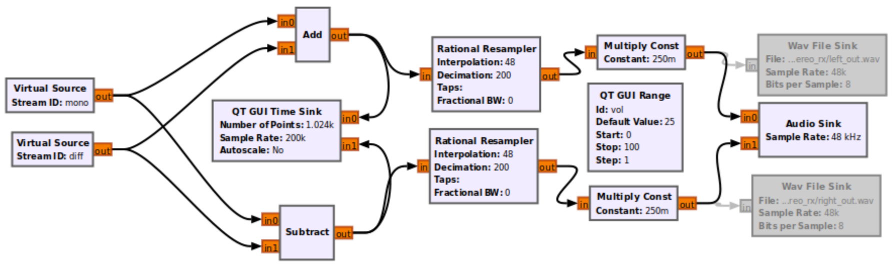


Figure 6: regeneration of left and right channels, volume control, and soundcard output.

### II.B.iii Verification

It is not so obvious when listening to a song on the radio if it is stereophonic or not; therefore, some testing is necessary to verify the output of this flowgraph is in fact stereophonic. To do this, the left and right channels were recorded from the flowgraph using the "WAV File Sink" and placed in a monophonic *.wav file. The files were then imported into *Audacity*. "Effect - Invert" was applied to the right channel and the two tracks were mixed (added) together by "Tracks - Mix and Render to new Track" and then remixing this new track with "Tracks - Mix Stereo down to Mono." What remains should now be the difference signal. Applying this procedure to a clip of music recorded from a commercial radio station, distinct musical elements can be heard well above the noise, demonstrating that this flowgraph preserves a real stereophonic difference between the left and right channels.

## III Narrowband FM Detector Algorithm

A typical FM detection algorithm is depicted in Figure 7. Frequency is defined as the rate of change in phase. Using arctangent, the phase of an IQ signal can be computed and then differentiated in order to compute this change in phase.

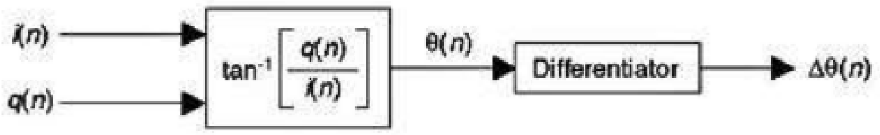


Figure 7: Typical FM detection algorithm using arctangent. [2]

A major drawback of this topology is the high amount of resources required to compute arctangent. This section will report the implementation and testing of a FM detection algorithm that avoids arctangent computations.

### III.A FM Detection Algorithm without Arctangent

A derivation of this algorithm is given in Reference [2] and is beyond the scope of this paper. The algorithm is defined by the equation

$$\Delta\theta_{[n]} = \frac{i_{[n]}\frac{dq}{dn} - q_{[n]}\frac{di}{dn}}{i^2_{[n]} + q^2_{[n]}} \tag{5}$$

where $i_{[n]}$ and $q_{[n]}$ are the in-phase component (I) and quadrature-phase component (Q) signals respectively [2]. This simplifies to the topology given in Figure 8, where the "Scaling" block divides the input by the denominator given in Equation 5. Making the assumption that the incoming signal is pure FM and that the incoming envelope is mostly constant, denominator can be set to a constant [2]. In practice, the author found this simplification necessary to avoid divide-by-zero errors when the input was squelched with zero injection. The second difference here is not used to compute the second derivative, but to compute the two-sided discrete derivative.

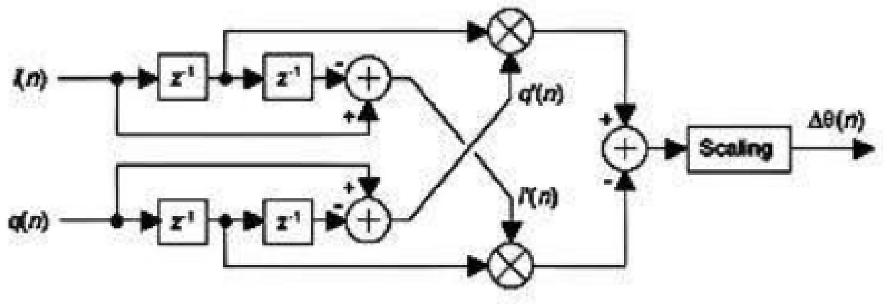


Figure 8: Simplified FM detection algorithm that avoids arctangent. [2]

## III.B Implementation in GNURadio

GR provides a block named ”Embedded Python Block” which allows the user to write Python code to describe its behavior. The code is in Figure 9. It begins by declaring the block as a child of `gr.sync_block`. It overloads the constructor for the `gr.sync_block` with a complex input and float output. `self.scale` is provided to globalize the input `scale` from the constructor arguments. A few additional global variables are declared to act as buffers between data blocks.

The `work()` function allows the programmer to access the input and output buffers for the block in its arguments. The code begins by splitting the input stream into its I and Q components. Buffers are declared for the I and Q second differences as well as for a data output buffer `dtheta`. After doing a check to ensure the data block is long enough to compute the second differences, the first two second differences for the I and Q components are calculated using the buffered values from the previous data block. The rest of the second differences can be computed with Python shorthand in lines 34 and 35. `dtheta` can then be calculated by the equation

$$\Delta\theta_{[n]} = A(i_{[n-1]}\Delta q_{[n]} - q_{[n-1]}\Delta i_{[n]}) \tag{6}$$

where $A$ is equal to `self.scale`. After updating the buffers so the next iteration of `work()` can access the necessary data from the current block, `dtheta` is copied to the block's output buffer.

## III.C Performance

The frequency response of this algorithm is plotted against the theoretical ideal in Figure 10. The concavity of the magnitude response is also given. The arctangent-less algorithm has a favorable amplitude in its response and it remains effectively linear until approximately 5 kHz. At a sample rate of 30 kHz, this means that the algorithm provides a linear response up to one third the nyquist frequency.

```python
import numpy as np
from gnuradio import gr

class blk(gr.sync_block):

    def __init__(self, scale=1.0):
        gr.sync_block.__init__(
            self,
            name='HB FM Detect',
            in_sig=[np.complex64],
            out_sig=[np.float32]
        )
        self.scale = scale
        self.prev_last_i = 0
        self.prev_pent_i = 0
        self.prev_last_q = 0
        self.prev_pent_q = 0

    def work(self, input_items, output_items):
        i = input_items[0].real
        q = input_items[0].imag

        di = np.zeros(i.size)
        dq = np.zeros(q.size)

        dtheta = np.zeros(i.size)

        if i.size > 1:
            di[0] = i[0] - self.prev_pent_i
            di[1] = i[1] - self.prev_last_i
            dq[0] = q[0] - self.prev_pent_q
            dq[1] = q[1] - self.prev_last_q

            di[2:] = i[2:] - i[:-2]
            dq[2:] = q[2:] - q[:-2]

            dtheta[0] = self.scale*(self.prev_last_i*dq[0] - self.
    prev_last_q*di[0])
            dtheta[1:] = self.scale*(i[:-1]*dq[1:] - q[:-1]*di[1:])

            self.prev_last_i = i[-1]
            self.prev_pent_i = i[-2]
            self.prev_last_q = q[-1]
            self.prev_pent_q = q[-2]

        output_items[0][:] = dtheta

        return len(output_items[0])
```

Figure 9: code for the FM detector algorithm that avoids arctangent for use in GNURadio.

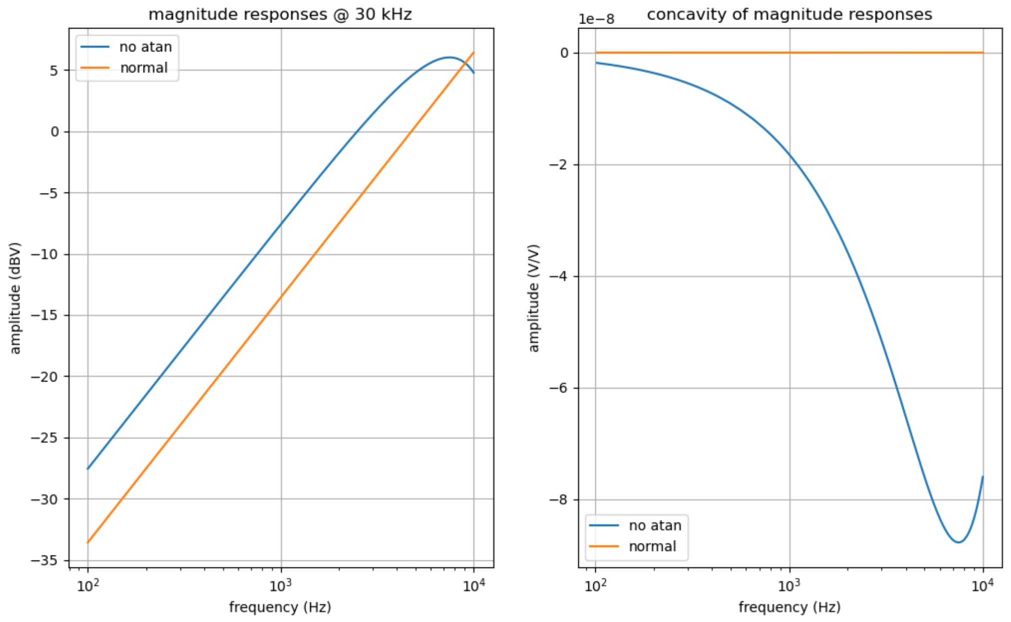


Figure 10: frequency response of the arctangent-less FM detector with the theoretical ideal (left) and the concavity of the response for better understanding the nonlinearity of the algorithm (right).

## IV OOK Text Modem

The projects in this paper so far have covered analog modulation schemes. A great advantage of SDR and DSP in general is that it allows for the processing of digital data. There are several digital versions of all the analog modulation schemes. This section describes the implementation of a unipolar binary amplitude shift keying (BASK) modem, better known as on-off keying (OOK).

### IV.A OOK Theory and Modem Design

OOK is a specific case of amplitude shift keying (ASK) modulation. ASK modulation encodes symbols as different envelope amplitudes of the carrier according to the equation

$$s_{(t)} = g_{(t)} A_m \cos 2\pi f_c t \tag{7}$$

Where $g_{(t)}$ is the complex envelope, $A_m$ is a nonzero trivial amplitude constant, and $f_c$ is the carrier frequency. The discretized version of the complex envelope $g_{[n]}$, which is sent directly to the output buffer of an FED to encode ASK, is given by

$$g_{[n]} = \text{symmap}_k[b] \star w_{[n]} \tag{8}$$

Where symmap$[b]$ is the function that maps an integer $b$ to a sequence with an amplitude deterministic of $b$ of length $k$ and $w_{[n]}$ is a pulse shaping LPF to reduce bandwidth. It is possible to use zero and negative amplitudes, but requires a more complex modem design. OOK is the name given to when an ASK scheme is restricted to the symbol map

$$\text{symmap}_k[b] = \{\{0, +1\}\}_k = \begin{cases} \{0, 0, ..., 0\}_k & b = 0 \\ \{1, 1, ..., 1\}_k & b = 1 \end{cases} \tag{9}$$

where the notation $\{x, x, ..., x\}_k$ means a sequence consisting of $x$ repeated $k$ times.

A demodulator need only ask the question of if a signal is present or not in order to function. The symbols can be recovered from an envelope detector, instantiated by a square-law device (SqLD) and an LPF. The binary data can be finally extracted using symbol clock recovery and a clocked symbol decoder. This symbol clock recovery can be accomplished by an BPF and a hard limiter.

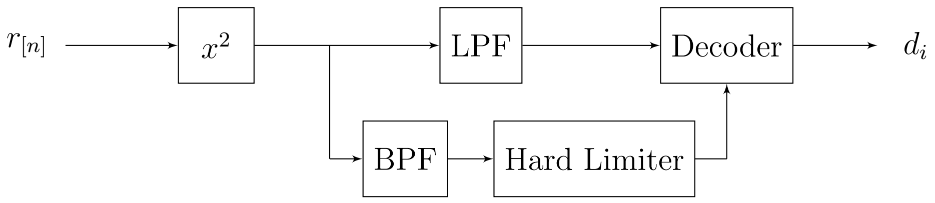


Figure 11: OOK demodulator.

Figure 11 is the block diagram for an OOK demodulator. The signal $r_{[n]}$ is the sampled received signal and $d_i$ is the sequence of recovered bits. Since both the envelope detector and the symbol clock recovery begin with an SqLD, the whole system can be optimized such that they share a single SqLD.

## IV.B Transmission

### IV.B.i Flowgraph

This flowgraph is separated into two smaller flowgraphs for transmission and reception. The transmission flowgraph, captured in Figure 12 begins with a vector source block which converts the payload string to a list of binary values. Each ASCII character is represented by a byte. Since OOK has a symbol-to-bit ratio (SBR) of 1, each bit must be split into a single unit of data. The block "Packed to Unpacked" with a "Bits per Chunk" of 1 takes an 8-bit integer and spits it into a sequence of eight 8-bit integers, with each integer being either 0 or 1, corresponding to the binary value of that bit in the original character. This allows the "Unipolar NRZ Encoder" to read each bit value individually and encode the data into a unipolar non-return-to-zero (NRZ) symbol train.

Pulse shaping is applied using a root-raised cosine (RRC) filter with an excess bandwidth ratio $\beta$ of 0.35. It is critical for this pulse shaping to occur for the receiver to operate correctly, as will be discussed in Section IV.C.i. A conversion to complex data for the symbol train is performed so that the "osmocom Sink" block can receive the correct data type. The complex symbol train is then upsampled and delivered to the *HackRF One*.

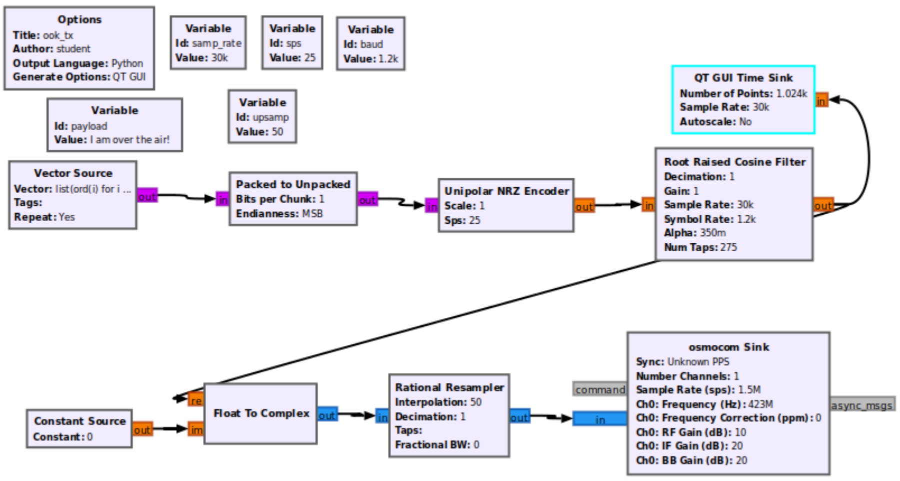


Figure 12: OOK transmitter.

### IV.B.ii Operation and Results

Testing for this modem was done chiefly at 1200 baud and 25 samples-per-symbol (sps), equating to a sample rate of 30 kHz. Figure 13 shows the real symbol train in the time and frequency domains as it is sent to the *HackRF One* before upsampling. Though it is modulated to a frequency of 423 MHz when being output through the antenna on the FED, it remains baseband throughout the entire flowgraph before it reaches the "osmocom Sink" block. The frequency domain shows how the spectrum of the symbol train begins to roll off at approximately ±810 Hz, which would cause a passband bandwidth of 1620 Hz. This matches the theoretical bandwidth from the formula

$$B = \beta R_s \tag{10}$$

Where $\beta$ is the RRC excess bandwidth ratio and $R_s$ is the symbol rate (baud rate).

## IV.C Reception

### IV.C.i Flowgraph

Initial sampling and pre-demodulation (preD) occur in Figure 14. Reception occurs at a high sample rate due to hardware limitations, so the data are first downsampled. Since OOK is realized with real-numbered symbols, the imaginary component of the *RTL-SDR*'s reception stream can be discarded. A hand-tuned scaling factor is then introduced so that other level-related decoding constants later in the flowgraph are easier to conceptualize.

As stated in Section IV.A, the envelope detector and symbol clock recovery can both share a SqLD. The LPF is used to remove the high frequency component from the squaring and leaves only the DC component. This DC component is the envelope of the carrier wave, which is the symbol train. The symbol clock is recovered by a narrow

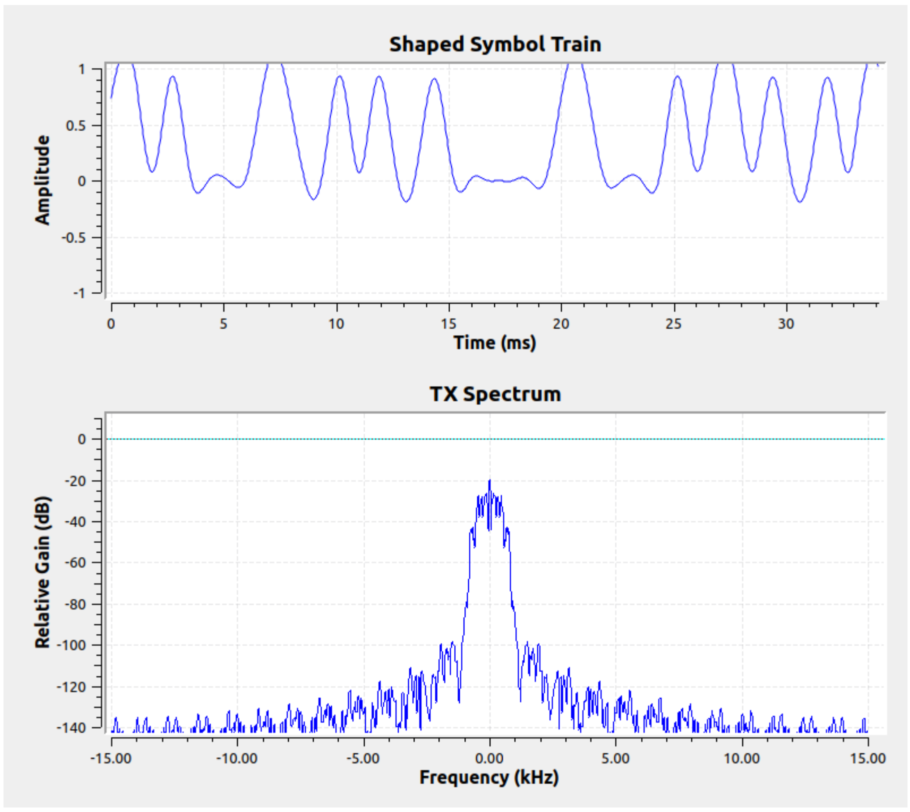
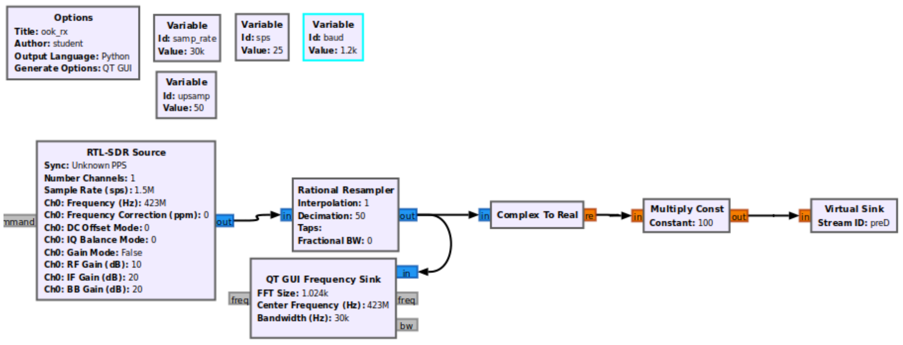


Figure 13: time domain and spectrum of TX symbol train.



**Options**
**Title:** ook_rx
**Author:** student
**Output Language:** Python
**Generate Options:** QT GUI

**Variable**
**Id:** samp_rate
**Value:** 30k

**Variable**
**Id:** sps
**Value:** 25

**Variable**
**Id:** baud
**Value:** 1.2k

**Variable**
**Id:** upsamp
**Value:** 50

**RTL-SDR Source**
**Sync:** Unknown PPS
**Number Channels:** 1
**Sample Rate (sps):** 1.5M
**Ch0: Frequency (Hz):** 423M
**Ch0: Frequency Correction (ppm):** 0
**Ch0: DC Offset Mode:** 0
**Ch0: IQ Balance Mode:** 0
**Ch0: Gain Mode:** False
**Ch0: RF Gain (dB):** 10
**Ch0: IF Gain (dB):** 20
**Ch0: BB Gain (dB):** 20

**Rational Resampler**
**Interpolation:** 1
**Decimation:** 50
**Taps:**
**Fractional BW:** 0

**QT GUI Frequency Sink**
**FFT Size:** 1.024k
**Center Frequency (Hz):** 423M
**Bandwidth (Hz):** 30k

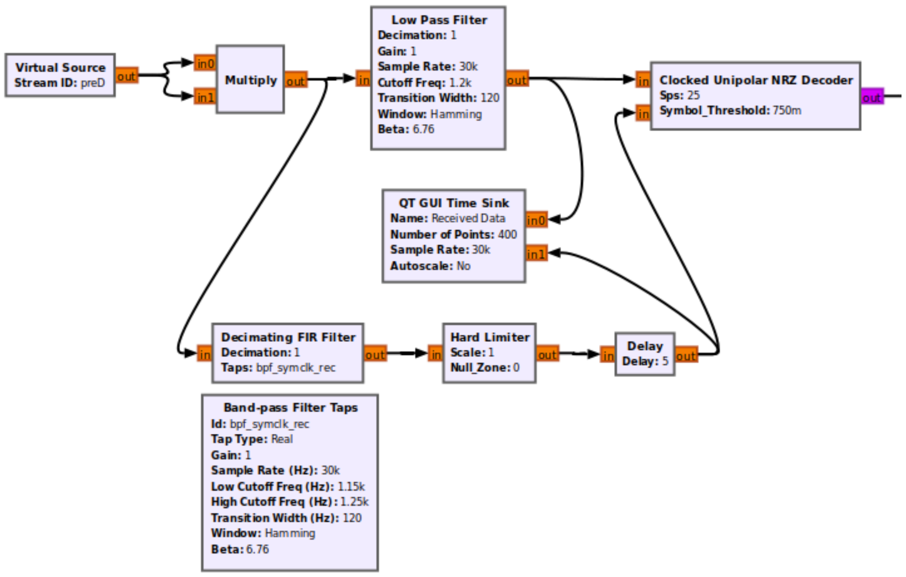
**Complex To Real**

**Multiply Const**
**Constant:** 100

**Virtual Sink**
**Stream ID:** preD

Figure 14: initial pre-demodulation processing.

BPF and a hard limiter. This topology, provided in Figure 11, is instantiated into the flowgraph as seen in Figure 15.



Virtual Source
Stream ID: preD

Multiply

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 30k
Cutoff Freq: 1.2k
Transition Width: 120
Window: Hamming
Beta: 6.76

Clocked Unipolar NRZ Decoder
Sps: 25
Symbol_Threshold: 750m

QT GUI Time Sink
Name: Received Data
Number of Points: 400
Sample Rate: 30k
Autoscale: No

Decimating FIR Filter
Decimation: 1
Taps: bpf_symclk_rec

Hard Limiter
Scale: 1
Null_Zone: 0

Delay
Delay: 5

Band-pass Filter Taps
Id: bpf_symclk_rec
Tap Type: Real
Gain: 1
Sample Rate (Hz): 30k
Low Cutoff Freq (Hz): 1.15k
High Cutoff Freq (Hz): 1.25k
Transition Width (Hz): 120
Window: Hamming
Beta: 6.76

Figure 15: envelope detection and symbol clock recovery using a combined SqLD.

The symbol clock recovery in this flowgraph is an example of practical implementation extending beyond what is noted in theory. A delay block is included after the hard limiter in Figure 15 to realign the symbol clock onto the center of the symbols in the symbol train. The delay for this block is calculated by the formula

$$N = N_{gd,BPF} \% N_{sps} + N_{sps}//8 \tag{11}$$

Where $N_{gd,BPF}$ is the group delay of the BPF, $N_{sps}$ is the samples per symbol of the encoding scheme, % is the modulo operator, and // is the integer division operator. The group delay of a filter is the amount of samples that a feature in a signal is delayed as a result of being filtered and is given by

$$N_{gd} = \frac{L-1}{2} \tag{12}$$

Where $L$ is the length of the filter's set of taps.

The "Clocked Unipolar NRZ Decoder" on the right side of Figure 15 does the work of the decoder in Figure 11. The blocks works by evaluating the amplitude of the symbol train on every rising edge of the symbol clock input. If the symbol train input is greater than "Symbol_Threshold," the symbol is determined to be a +1. If not, it is a 0. These symbol values are output in the same representation as the "Packed to Unpacked" block in the transmission flowgraph.

Since this flowgraph does not include data synchronization, the user must tune it himself during runtime. Figure 16 depicts the final processing stage to accomplish this. It utilizes a GUI control allowing the user to select a value from 0 to 7 and applies that much delay to the bits. With some patience, the correct delay is applied such that the "Unpacked to Packed" block frames the data exactly on the boundaries of each ASCII character. The decoded payload is then streamed to a terminal window to be viewed in real time.

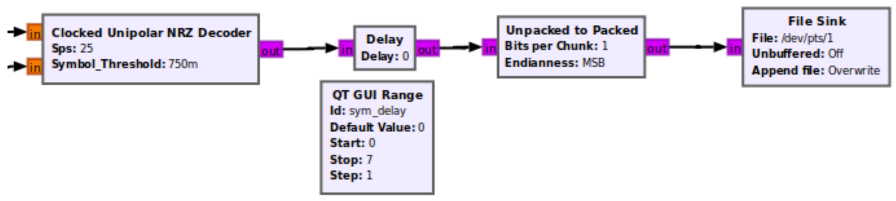


Figure 16: data alignment and output stage.

## IV.C.ii Operation and Results

This OOK receiver is successful in demodulating and decoding a payload sent by the OOK transmitter made in Section IV.B.i. The receiving antenna was placed a few feet from the transmitting antenna and both transmitter and receiver were activated. Figure 17 shows the output of the envelope detector and the recovered symbol clock. The "ideal" symbol amplitude is present at every rising edge of the clock.

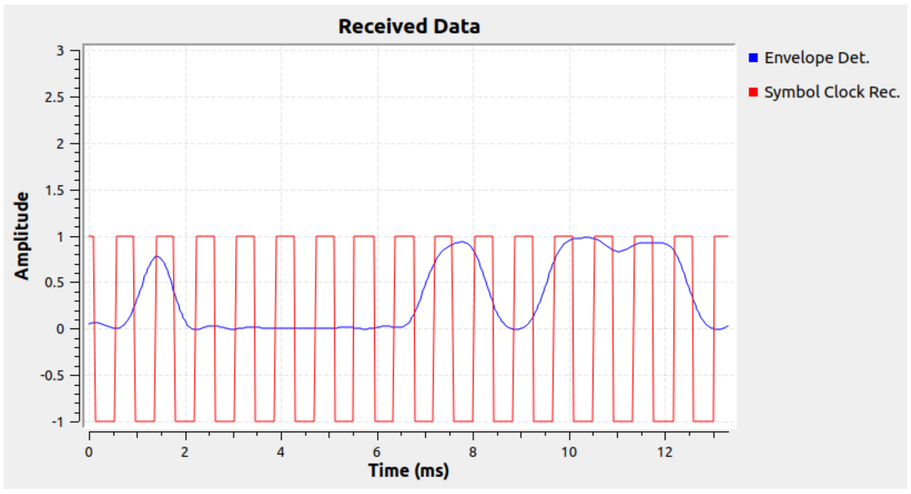


Figure 17: time domain output of the envelope detector and symbol clock recovery.

The received spectrum in Figure 18 shows that the local oscillators in each FED are not perfectly synchronized, hence why the desired signal appears to have an offset of approximately 7 kHz. This offset is not an issue for this receiver as it is an incoherent receiver.

Figure 19 shows both transmitter and receiver running. The hidden spectrum on the bottom is the transmitter's GUI. The receiver and the terminal window receiving its output are seen on the top.

The bit error rate (BER) at different baud rates and signal-to-noise ratio (SNR) was calculated by transmitting one-thousand "a" characters through an equivalent loopback flowgraph (that is, the two flowgraphs were connected together within GRC and not via radios) with a channel model block to simulate the noise. Figure 20 shows the results from this test.

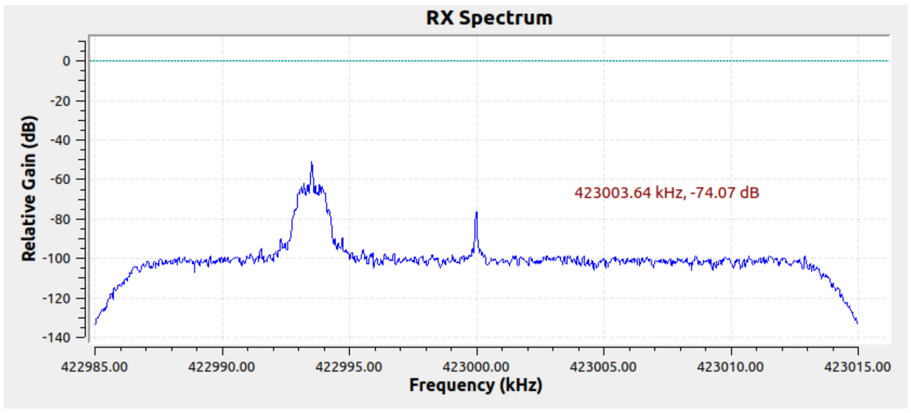
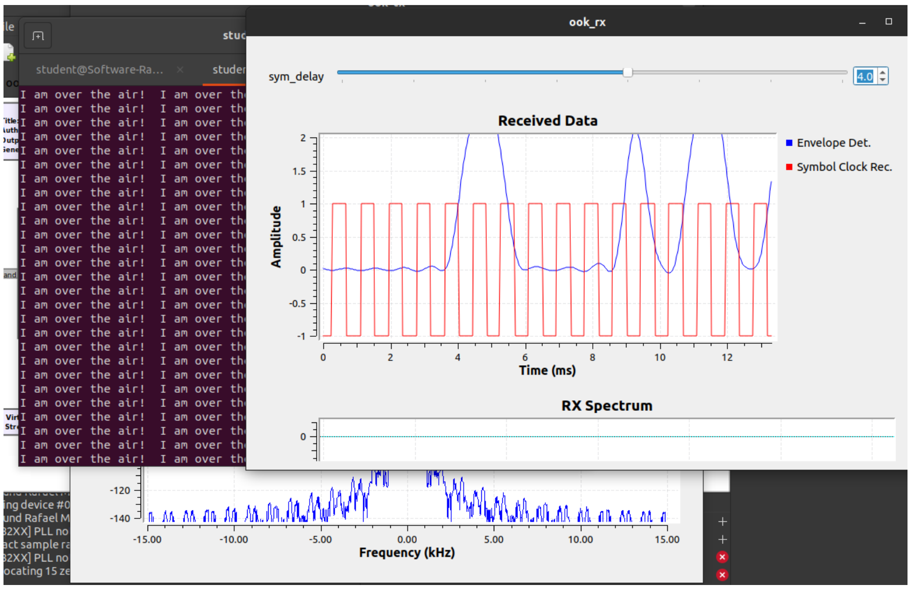


Figure 18: spectrum as seen from the *RTL-SDR*.



```
I am over the air!  I am over the
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
I am over the air!  I am over th
```

Figure 19: (behind) TX GUI; (top right) RX GUI; (top left) decoded output.

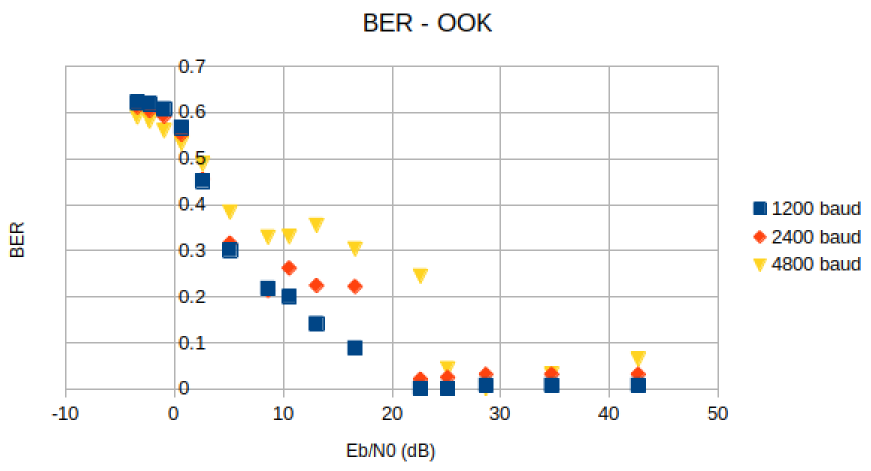


Figure 20: BER of OOK modem at various baud rates.

## V Conclusion

These projects demonstrate the flexibility of SDR, as well as the challenges of developing an SDR system. In the coming future, all RF applications will be driven by DSP on computers, FPGAs, and embedded processors.

Further areas of exploration could include expanding the OOK modem to other modes, forward error correction, and automatic frame synchronization.

## Acknowledgements

Dr. Juan Manfredi provided initial guidance for the direction of the project and on choosing legal and optimal frequency bands.

## References

[1] L. W. Couch, *Digital and Analog Communication Systems, Eighth Edition*, 2013.

[2] Embedded Staff. (2011) DSP Tricks: Frequency demodulation algorithms. [Online]. Available: https://www.embedded.com/dsp-tricks-frequency-demodulation-algorithms/