Implementation of Analog and Digital Communications Transceivers on SDR Platforms using GNU Radio Companion

Mr. Joshua David Edgcombe, Grand Valley State University

Joshua Edgcombe is a Graduate Student at Grand Valley State University pursuing his Masters of Science in Engineering in Computer and Electrical Engineering. He has experience with signal processing and communication systems as well as analog and digital circuit and filter design. He also has a strong background in software development and has developed software in a variety of environments including web, native, systems, and embedded.

Dr. Bruce E. Dunne, Grand Valley State University

Bruce E. Dunne received the B.S.E.E. (with honors) and M.S. degrees from the University of Illinois at Urbana-Champaign in 1985 and 1988, respectively, both in Electrical and Computer Engineering. He received the Ph.D. degree in Electrical Engineering from the Illinois Institute of Technology, Chicago, in 2003. In the Fall of 2003, he joined the Padnos College of Engineering and Computing, Grand Valley State University, Grand Rapids, MI, where he is currently a Professor of Engineering. Prior to this appointment, he held several research and development positions in industry. From 1991 to 2002, he was a Staff Engineer with Tellabs, Naperville, IL. Additionally, in 1991, he was with AT&T Bell Telephone Laboratories, Naperville, IL; from 1988 to 1991, he was with R. R. Donnelley & Sons, Lisle, IL; and from 1985 to 1986, he was with Zenith Electronics, Glenview, IL. His interests include adaptive filtering, speech enhancement, wireless and wireline communications, and engineering education. Dr. Dunne is a senior member of the IEEE and a member of Eta Kappa Nu and the ASEE.

Implementation of Analog and Digital Communications Transceivers on SDR Platforms using GNU Radio Companion

Joshua Edgcombe and Bruce E. Dunne
School of Engineering, Grand Valley State University

Abstract

In the recent literature, Software Defined Radio (SDR) has been promoted as a powerful and low-cost approach to offering laboratory experiments in the field of analog and digital communications. Furthermore, using the freeware graphical software GNU Radio Companion (GRC), a wide variety of experiments can be easily and quickly assembled by students on the SDR hardware. The GRC software includes built-in instrumentation blocks that allow visualization of the signals at any point in the modulation and/or demodulation process, lending strong experimental observation to reinforce theoretical concepts. Certain SDR hardware platforms provide duplex processing, allowing implementation of both the transmitter and receiver, for short distances.

The advantages of an SDR/GRC approach to offering communication laboratory experimentation is well described; however, the specific implementation details are less well documented. While conceptually not overly difficult, there are many non-trivial pitfalls and obstacles that must be overcome to actualize such communication experimentation, especially for RF over-air communications. The intent of this paper is to address this knowledge gap and provide clear implementation details for a turn-key laboratory in a first or second course in analog and digital communications. To do so, a series of communications experiments are described, including all processing at both the transmitter and the receiver (including timing considerations), the interface to external files, the RF interface, and beneficial points to observe signals in either the time or frequency domain, or as appropriate, constellation plots. The configuration of the GRC blocks are described along with complete GRC flow graph diagrams for each modulation format presented. Dealing with issues such as timing alignment are also discussed.

This paper includes an overview of the use of SDR/GRC in communication laboratory experimentation as well as a description of the recommended hardware and development environment. Some general remarks about the development of GRC flow graphs is then followed by a detailed discussion of transceiver implementation. Particular transceivers discussed include the analog modulation formats AM and FM and the digital modulation formats FSK and PSK. The paper concludes with recommendations for additional and more advanced communication experimentation.

Introduction

Software Defined Radio[1,2] (SDR) offers a powerful alternative to conventional communication system design. In conventional design, specially-built hardware is implemented to perform communication for a particular, radio-specific modulation scheme, usually over a limited frequency range. In contrast, SDR systems offer much more flexibility by implementing the modulation/demodulation functionality in software. Connected to the antenna through an RF mixer is a high-speed ADC/DAC (for receiver/transmitter, respectively) such that the SDR processes the communication signals using DSP algorithms implemented in software. Particularly powerful is the concept of flexibility; if the radio modulation scheme changes, new DSP software is loaded to perform the necessary processing and no hardware modification is required. This approach allows for ease of adaptability, shortens development effort and greatly reduces cost and complexity. Furthermore, mid-range capable SDR systems are highly affordable.

The most popular tool used for SDR software development is GNU Radio[3]; a free open-source program with a large online support community. Typically, GNU Radio serves as the signal processing engine (executing on the host computer) while the SDR hardware provides the RF front end and digitization (note that GNU Radio can also run in a simulation mode without any SDR hardware connected or from recorded data). This software provides the necessary drivers for communicating with the SDR hardware and host system I/O, as well as signal processing blocks for encoding, modulation, filtering, packet handling, stream manipulation and other functions. The software for GNU Radio is written in Python and C++, where Python is the glue code and C++ performs most of the heavy signal processing. Additional user-generated custom signal processing blocks can be written in either Python or C++ and added to the signal chain. Fortunately, GNU Radio is relatively mature; most functions for general signal processing applications and communications have already been written and optimized, making GNU Radio highly modular.

Communication system development using GNU Radio is greatly simplified with the use of GNU Radio Companion (GRC). GRC is the graphical user interface for GNU Radio, where users place functional blocks into a processing chain known as a flowgraph. Blocks exist for the vast majority of communication system functions, requiring users to simply configure the block with a handful of parameters particular to their system. If desired, a user can create a custom graphical flow block (in either Python or C++); many such blocks are available via the user community and by default in GRC. Once a flowgraph has been created, the user generates a Python file with a click of a button in GRC, where the signal processing blocks (written typically in C++) are connected together, hence, Python is seen as the glue code.

The literature has widely discussed and promoted the advantages of SDR for use in an academic laboratory setting for instructional purposes. Such papers present an overview of various experiments and projects[4,5,6,7] including discussions of both analog and digital communications laboratories, at varying levels of detail. Others include aspects of RF over-air full transceiver implementation[8,9].

Despite this large array of reference material, what is missing is a clear presentation of implementation detail with SDR and GRC. For the educator new to SDR, there is a considerable learning curve to get a laboratory up and running, particularly for full over-air RF digital transceiver systems. While aided by the large GRC library of function blocks, there are many pitfalls that are not easily avoided. The intent of this paper is to address this knowledge gap and provide clear implementation details for a turn-key laboratory in a first or second course in analog and digital communications. Flowgraphs for realizing full transceiver RF implementations of analog systems including AM and FM as well as digital systems including FSK and PSK are presented. Discussion on how to configure these flowgraphs is also included.

We begin by discussing some preliminaries including a recommended hardware setup and other details. We then present flowgraphs for analog communications, followed by flowgraphs for digital communications, along with explanations and a link to a repository containing up to date versions of the flowgraphs discussed in this paper. We briefly present an assessment survey regarding student preferences for working with SDR systems. Finally, we conclude with a summary of our findings and recommendations for other communication experiments.

Preliminaries

Recommended Hardware and Development Environment

There is a wide selection of SDR hardware available, with many good choices for the purposes presented herein[10]. Given the requirements of the projects, the recommended choice is the HackRF One open-source SDR, along with ANT500 antenna[11]. The HackRF One offers half-duplex transceiver capability, sampling rates up to 20 MSPS, operating frequency of 1 MHz to 6 GHz, USB powered connection, SMA RF connection with programmable gain, and full compatibility with GRC. For experimentation when only a receiver is needed, or for simple analog communication experimentation, use of the very low-cost RTL-SDR (for the receiver function) is also a good choice. This device offers receive frequency range of 25 MHz to 1.766 GHz, a bandwidth of 2.4 MHz, USB powered connection (thumb-drive) and GRC compatibility. A promising new platform (not yet experimented with by the authors) is the LimeSDR[12], with competitive features to the HackRF One at a lower price-point.

The recommended development environment is a Linux-based operating system such as Ubuntu OS[13], as Linux-based environments are most compatible with GRC. Flowgraphs are developed with the GRC application, compiled and downloaded onto the SDR hardware. The hardware configuration is programmed with the GRC osmocom sink block (for transmission) and the osmocom source block (for reception). A typical development setup is given below in Figure 1. Shown in the figure are separate transmit and receive stations, each running GRC and connected to transmit and receive HackRF One SDRs (note that it is possible to run both transmit and receive on one computer, with two instances of GRC). Also shown, but unconnected, is a particular vendor's version of the RTL-SDR device.

Note that with the wide choice of frequency operation, it is usually possible to select a communication frequency in an ISM band so as to avoid interference with commercial broadcasts. Furthermore, if so desired, these devices (in most cases) have a tuning range compatible with broadcast frequencies, allowing reception of these channels. Finally, given the comparatively low power of the HackRF One transmitter, use over even commercial bands would not be significantly disruptive (low power FM commercial band use is permitted).

Complex (Quadrature) Sampling

Complex (or quadrature) sampling is a natural consequence of SDR processing due to the desire to capture as much bandwidth as possible while being constrained by the device's sampling hardware. Maximizing bandwidth allows for more control for the developer (for example, it might be advantageous to digitize the entire FM broadcast bandwidth of 20 MHz at once rather than selectively tuning for individual channels one-by-one).

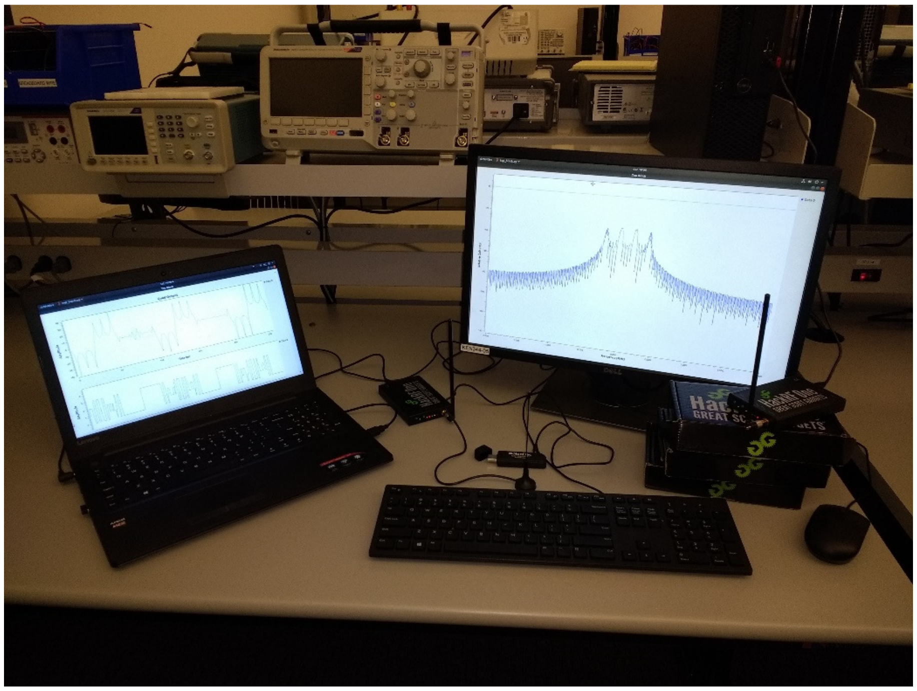


Figure 1: Laboratory Development Setup

The details of complex sampling are left to the interested reader[14], but essentially, complex sampling provides the user with a frequency band equal to the sampling frequency at a desired center frequency. That is, assume a center frequency of f_c Hz and a sampling frequency of f_s samples per second. SDR complex sampling then provides the developer with a frequency band described as

$$f_c - \frac{f_s}{2} < f < f_c + \frac{f_s}{2} \text{ Hz.} \tag{1}$$

Here, with complex sampling at f_s Hz, f_s Hz of signal bandwidth is provided, in seeming violation of Nyquist sampling theory (Nyquist sampling theory states that $f_s/2$ Hz of bandwidth is the maximally available bandwidth). This seeming discrepancy is resolved by considering that the sampled data is no longer real-valued but instead complex-valued, with separate real and imaginary components. Hence, there are actually two data elements per sample, and with twice the data, it is possible to obtain f_s Hz of signal bandwidth while sampling at f_s Hz. Note that since the time-based data is no longer real-valued, the resulting spectrum is *no longer conjugate symmetric* such that the center frequency f_c is not a center reflective point. The developer needs to be cognizant of this fact when processing signals, in particular, when isolating signals and applying symmetric filters or converting the complex-valued signal into a real-valued (floating-point format).

Signals used in SDR processing are normally complex in keeping with the above description. For many analog flowgraphs, the complex sampling is unnecessary and the signal can be converted to be real-valued (note that "corruption" of the signal band may occur due to the reintroduction of previously nullified spectral components as discussed above). Alternatively, for many digital flowgraphs (where in-phase and quadrature signals are processed), the complex sampling is useful and typically maintained.

The signal format is color-coded onto the flowgraph. Blue colored I/O tabs indicate complex-valued signals while orange colored I/O tabs indicate real-valued signals.

Synchronization

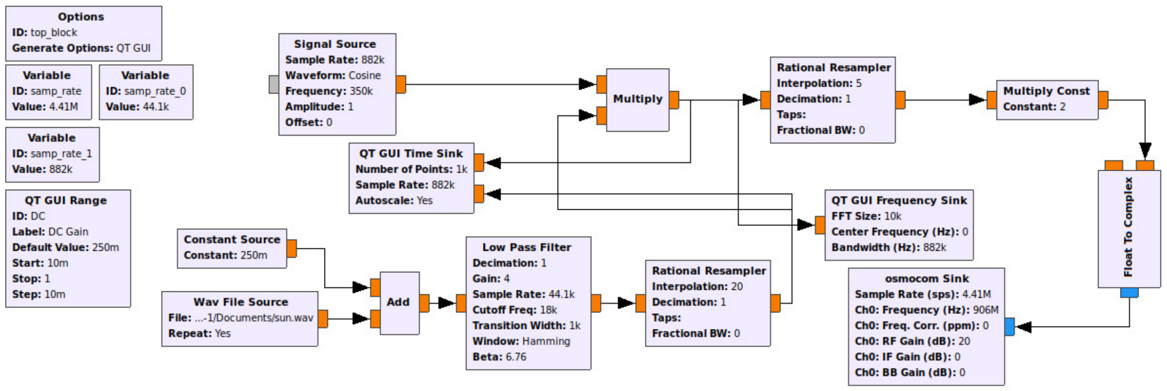
The HackRF One includes easily accessible SMA connections to perform frequency synchronization, known as CLKIN and CLKOUT. As a lower-cost device, the HackRF One apparently does not have a temperature compensated frequency synthesizer, and frequency drift is possible over the course of operation, which is potentially problematic for certain digital modulation schemes. One solution is to frequency lock the transmitting and receiving HackRF One devices together by either connecting CLKOUT of one device to CLKIN of the other or using an external 10 MHz square-wave source connected to both CLKIN ports. For the experiments described in this paper, it was found that using GRC software signal processing alignment blocks was sufficient to maintain frequency synchronization.

Analog Communications

Broadcast AM Transmitter

The HackRF One SDR is used as the AM transmitter, set to operate in the ISM-band near 900 MHz. The flowgraph for the AM transmitter is given below in Figure 2. Note that the format is broadcast AM (includes the carrier). In Figure 2, a CD-rate wave audio signal on the computer's hard drive is chosen as the message (aka information) signal (note that "repeat" is checked). Following a low pass filter option to limit the bandwidth, a DC offset (via the "constant source" block) is added to the information signal (controlled by a "range" block) to eventually represent the carrier in the transmitted signal. The actual value is left as a variable in the QT GUI Block to allow for adjustment of the modulation depth. Note that interpolation is performed to properly increase the sampling rates. These signals are real-valued; however, the SDR is only configured for RF transmission (and reception) of complex-valued signals, and it is therefore necessary to convert the signal to the complex domain.

An osmocom sink block creates the interface to the RF hardware and antenna. For short distances (as used in the lab), RF gains can be set moderately – in this case, only the RF gain of 20 dB is employed. A channel should be chosen in the range of 902 MHz to 928 MHz (ISM band) – let 906 MHz be chosen for this particular experiment.


Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 4.41M

Variable
ID: samp_rate_0
Value: 44.1k

Variable
ID: samp_rate_1
Value: 882k

QT GUI Range
ID: DC
Label: DC Gain
Default Value: 250m
Start: 10m
Stop: 1
Step: 10m

Signal Source
Sample Rate: 882k
Waveform: Cosine
Frequency: 350k
Amplitude: 1
Offset: 0

QT GUI Time Sink
Number of Points: 1k
Sample Rate: 882k
Autoscale: Yes

Constant Source
Constant: 250m

Wav File Source
File: ...-1/Documents/sun.wav
Repeat: Yes

Add

Low Pass Filter
Decimation: 1
Gain: 4
Sample Rate: 44.1k
Cutoff Freq: 18k
Transition Width: 1k
Window: Hamming
Beta: 6.76

Multiply

Rational Resampler
Interpolation: 5
Decimation: 1
Taps:
Fractional BW: 0

Multiply Const
Constant: 2

Rational Resampler
Interpolation: 20
Decimation: 1
Taps:
Fractional BW: 0

QT GUI Frequency Sink
FFT Size: 10k
Center Frequency (Hz): 0
Bandwidth (Hz): 882k

Float To Complex

osmocom Sink
Sample Rate (sps): 4.41M
Ch0: Frequency (Hz): 906M
Ch0: Freq. Corr. (ppm): 0
Ch0: RF Gain (dB): 20
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0

Figure 2: AM Transmitter

Included in the AM transmitter of Figure 2 are time-based and frequency based plots of the signal at points in the processing chain. Shown below in Figure 3 in the top plot are segments of the original message signal (blue) as well as the AM modulated waveform (red). The bottom part half of Figure 3 shows the frequency content of the AM signal, with the carrier and message signal sidebands, centered about 350 kHz.

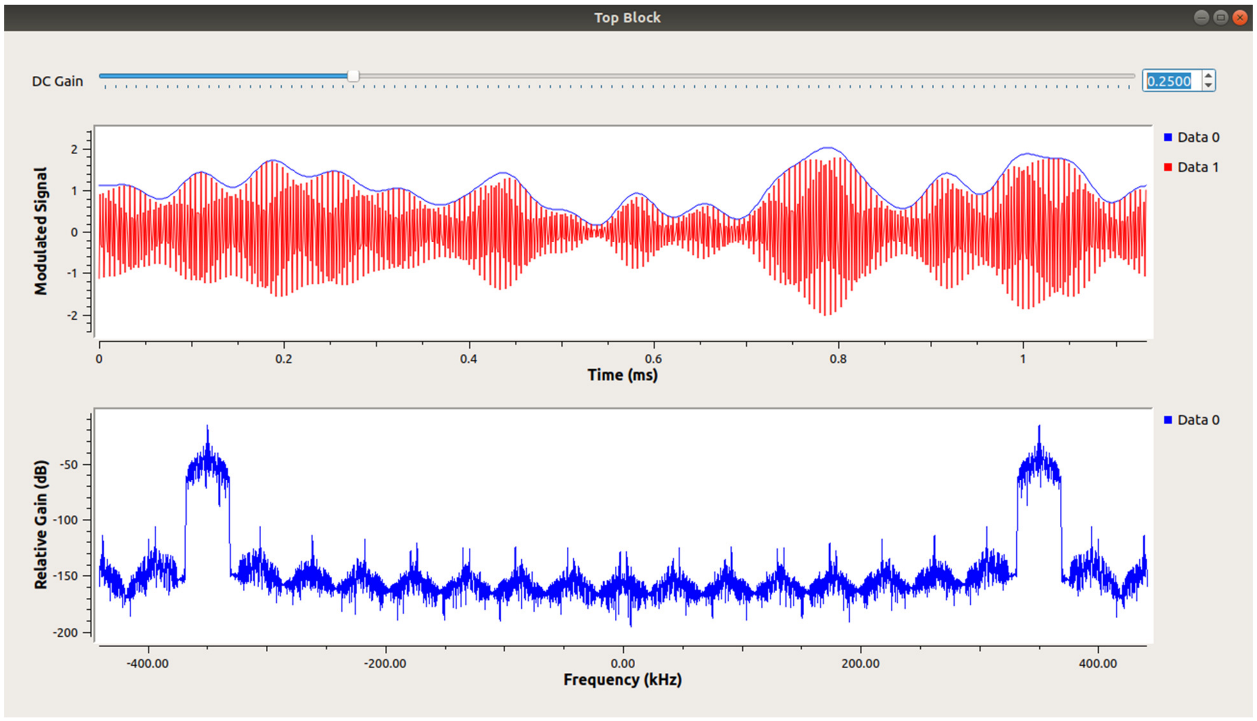
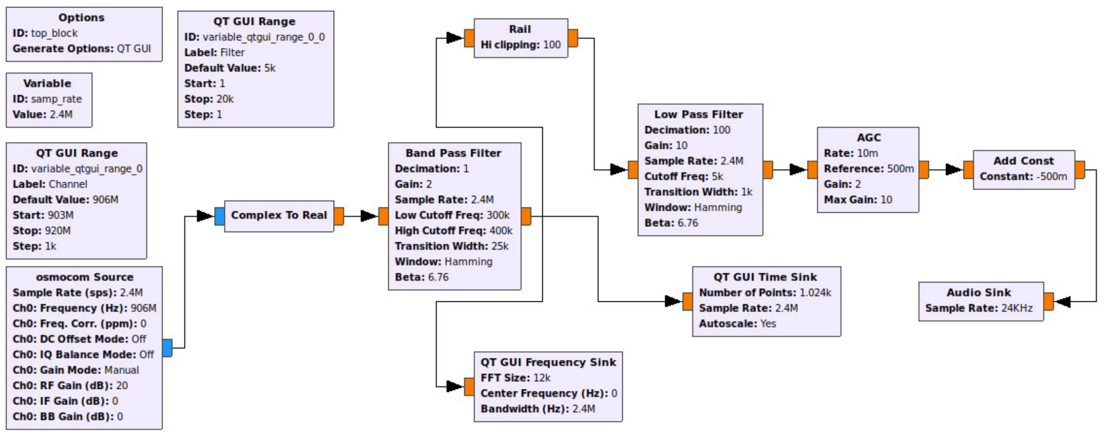


Figure 3: AM Transmitter Signals

Broadcast AM Receiver

A second HackRF One device is programmed as the AM receiver. The demodulation is performed as a simple envelope detector, as is generally done for broadcast AM and is shown below in Figure 4.

As noted earlier, the natural format for SDR processing is complex-valued signals. In this case, to apply the envelope detector, a conversion from complex to real-valued signals was explicitly employed (alternatively, this conversion is an option for the osmocom block). Because we are in a controlled environment where the transmit signal is known to be symmetric and real (we are generating the AM signal), the conversion will not introduce unintended reflected spectral components (as can happen with capturing a more general spectrum).



Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 2.4M

QT GUI Range
ID: variable_qtgui_range_0_0
Label: Filter
Default Value: 5k
Start: 1
Stop: 20k
Step: 1

QT GUI Range
ID: variable_qtgui_range_0
Label: Channel
Default Value: 906M
Start: 903M
Stop: 920M
Step: 1k

osmocom Source
Sample Rate (sps): 2.4M
Ch0: Frequency (Hz): 906M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 20
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0

Complex To Real

Band Pass Filter
Decimation: 1
Gain: 2
Sample Rate: 2.4M
Low Cutoff Freq: 300k
High Cutoff Freq: 400k
Transition Width: 25k
Window: Hamming
Beta: 6.76

Rail
Hi clipping: 100

QT GUI Frequency Sink
FFT Size: 12k
Center Frequency (Hz): 0
Bandwidth (Hz): 2.4M

Low Pass Filter
Decimation: 100
Gain: 10
Sample Rate: 2.4M
Cutoff Freq: 5k
Transition Width: 1k
Window: Hamming
Beta: 6.76

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2.4M
Autoscale: Yes

AGC
Rate: 10m
Reference: 500m
Gain: 2
Max Gain: 10

Add Const
Constant: -500m

Audio Sink
Sample Rate: 24KHz

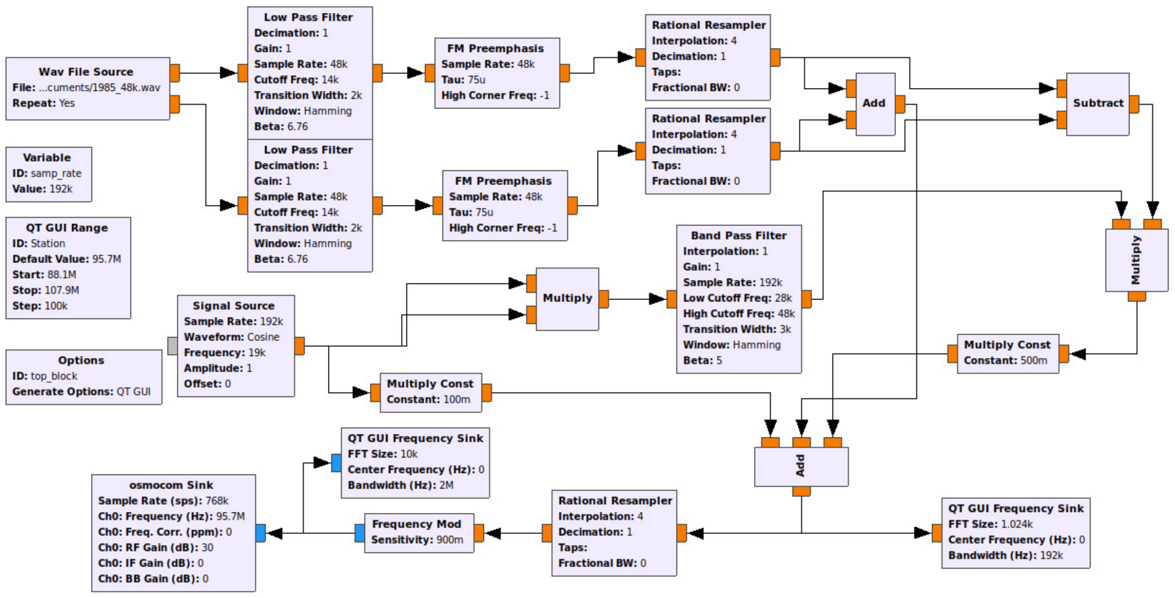
Figure 4: AM Receiver

In Figure 4, the AM signal is centered about 350 kHz, and the BPF isolates this signal. The “Rail” block is used to half-wave rectify the RF signal, which is then processed by a LPF to leave the envelope, which contains the message signal, along with significant decimation to lower the overall rate. Following signal level normalization, the DC bias is removed and the signal audio is played back to the user.

FM Transmitter

For FM transmission (show below in Figure 5), it was decided to use a frequency in the FM commercial band, as this allowed easy conversion to connecting to broadcast FM. As long as the power is kept low (as is true with the HackRF One), use of the FM band is allowed.

The FM transmitter flowgraph builds the composite FM signal, starting with a stereo signal at 48 kHz, utilizing a LPF to limit the frequency content to below 15 kHz. This LPF is followed by pre-emphasis filtering on both L and R channels. The sum $L + R$ and difference $L - R$ signals are then formed, with the later DSB-SC AM modulated by a doubled 19 kHz (i.e., 38 kHz) pilot tone, with all three signals summed into one 53 kHz band. This composite signal is then frequency modulated (note that the WBFM GRC block is not used – unfortunately, this block includes low pass filtering which negates the use of the composite signal, instead performed by the Frequency Mod block) after being interpolated to a rate that will support the wider FM signal. This signal is then sent to the osmocom source, operating (in this case) at 95.7 MHz.



Wav File Source
File: ...cuments/1985_48k.wav
Repeat: Yes

Variable
ID: samp_rate
Value: 192k

QT GUI Range
ID: Station
Default Value: 95.7M
Start: 88.1M
Stop: 107.9M
Step: 100k

Options
ID: top_block
Generate Options: QT GUI

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 48k
Cutoff Freq: 14k
Transition Width: 2k
Window: Hamming
Beta: 6.76

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 48k
Cutoff Freq: 14k
Transition Width: 2k
Window: Hamming
Beta: 6.76

FM Preemphasis
Sample Rate: 48k
Tau: 75u
High Corner Freq: -1

FM Preemphasis
Sample Rate: 48k
Tau: 75u
High Corner Freq: -1

Rational Resampler
Interpolation: 4
Decimation: 1
Taps:
Fractional BW: 0

Rational Resampler
Interpolation: 4
Decimation: 1
Taps:
Fractional BW: 0

Signal Source
Sample Rate: 192k
Waveform: Cosine
Frequency: 19k
Amplitude: 1
Offset: 0

Band Pass Filter
Interpolation: 1
Gain: 1
Sample Rate: 192k
Low Cutoff Freq: 28k
High Cutoff Freq: 48k
Transition Width: 3k
Window: Hamming
Beta: 5

Multiply Const
Constant: 500m

Multiply Const
Constant: 100m

QT GUI Frequency Sink
FFT Size: 10k
Center Frequency (Hz): 0
Bandwidth (Hz): 2M

osmocom Sink
Sample Rate (sps): 768k
Ch0: Frequency (Hz): 95.7M
Ch0: Freq. Corr. (ppm): 0
Ch0: RF Gain (dB): 30
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0

Frequency Mod
Sensitivity: 900m

Rational Resampler
Interpolation: 4
Decimation: 1
Taps:
Fractional BW: 0

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 192k

Figure 5: FM Transmitter

Shown in Figure 6 are plots of the FM spectrum. In the top plot, the modulated FM signal spectrum is shown. In the second graph in the plot, the composite FM spectrum is shown, where it is possible to identify all three components: $L + R$, $L - R$ and the pilot tone.

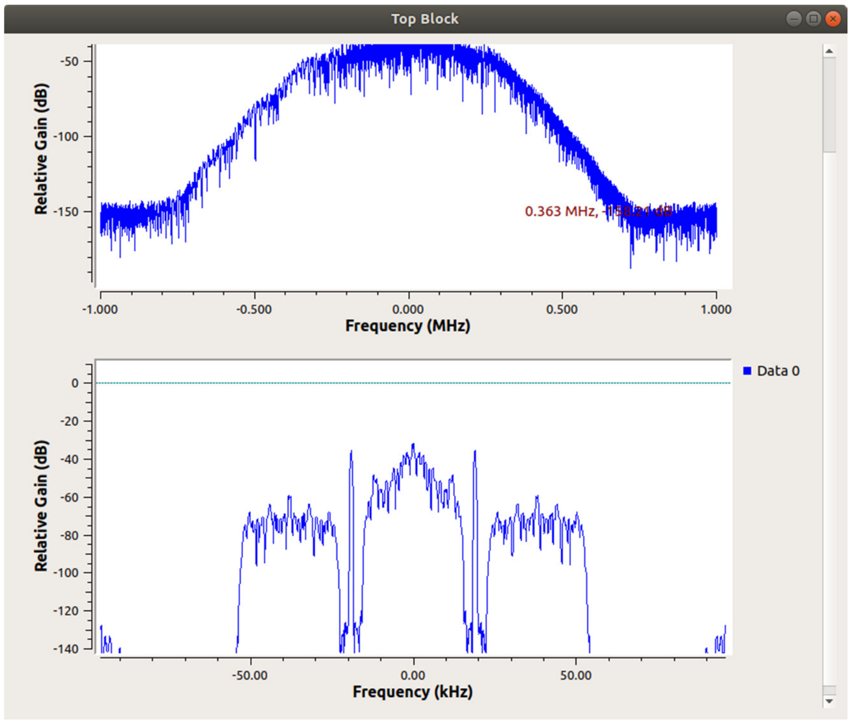
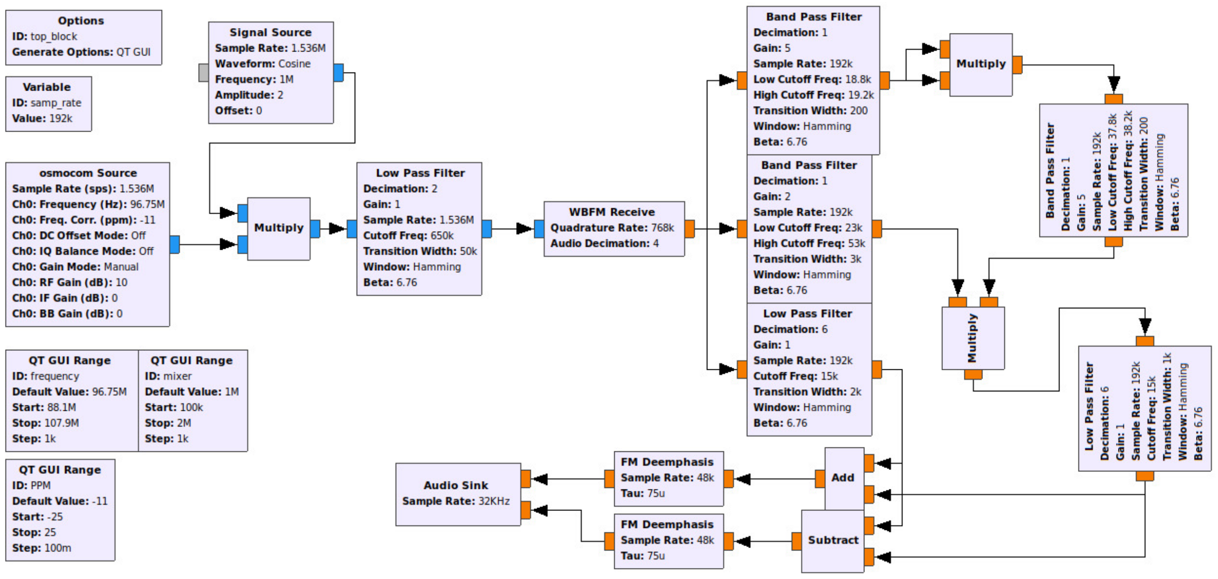


Figure 6: FM Transmitted Signal and Composite Signal

FM Receiver

Shown in Figure 7 is the FM receiver. Note that this receiver is intentionally off-tuned by 1 MHz, and then mixed and filtered back to DC. This was done to correct for spurious content present at DC if direct tuning was employed. Following FM demodulation with the WBFM Receive block, the composite FM signal is disassembled. Note that the recovery of the pilot tone includes a squaring and BPF operation to double its frequency; this tone is used to DSB-SC AM demodulate the $L - R$ signal. $L - R$ is combined with $L + R$ to obtain the separated stereo signals L and R, which are then deemphasized. The resulting stereo audio is then played back, this time at 32 kHz, in keeping with the filtering done at the transmitter.


Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 192k

Signal Source
Sample Rate: 1.536M
Waveform: Cosine
Frequency: 1M
Amplitude: 2
Offset: 0

osmocom Source
Sample Rate (sps): 1.536M
Ch0: Frequency (Hz): 96.75M
Ch0: Freq. Corr. (ppm): -11
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0

Multiply

Low Pass Filter
Decimation: 2
Gain: 1
Sample Rate: 1.536M
Cutoff Freq: 650k
Transition Width: 50k
Window: Hamming
Beta: 6.76

WBFM Receive
Quadrature Rate: 768k
Audio Decimation: 4

Band Pass Filter
Decimation: 1
Gain: 5
Sample Rate: 192k
Low Cutoff Freq: 18.8k
High Cutoff Freq: 19.2k
Transition Width: 200
Window: Hamming
Beta: 6.76

Band Pass Filter
Decimation: 1
Gain: 2
Sample Rate: 192k
Low Cutoff Freq: 23k
High Cutoff Freq: 53k
Transition Width: 3k
Window: Hamming
Beta: 6.76

Low Pass Filter
Decimation: 6
Gain: 1
Sample Rate: 192k
Cutoff Freq: 15k
Transition Width: 2k
Window: Hamming
Beta: 6.76

Multiply

Band Pass Filter
Decimation: 1
Gain: 5
Sample Rate: 192k
Low Cutoff Freq: 37.8k
High Cutoff Freq: 38.2k
Transition Width: 200
Window: Hamming
Beta: 6.76

Multiply

Low Pass Filter
Decimation: 6
Gain: 1
Sample Rate: 192k
Cutoff Freq: 15k
Transition Width: 1k
Window: Hamming
Beta: 6.76

QT GUI Range
ID: frequency
Default Value: 96.75M
Start: 88.1M
Stop: 107.9M
Step: 1k

QT GUI Range
ID: mixer
Default Value: 1M
Start: 100k
Stop: 2M
Step: 1k

QT GUI Range
ID: PPM
Default Value: -11
Start: -25
Stop: 25
Step: 100m

Audio Sink
Sample Rate: 32KHz

FM Deemphasis
Sample Rate: 48k
Tau: 75u

FM Deemphasis
Sample Rate: 48k
Tau: 75u

Add

Subtract

Figure 7: FM Receiver

Digital Communications

Digital communications with SDR/GRC has proven more challenging as compared to analog communications. The main challenges include:

- Picking the optimal sampling time in the oversampled data stream;
- Identifying the start and stop of a message (beginning and end in a stream);
- Dealing with symbol ambiguity

The implementation of the digital communications flowgraphs used in this paper were highly suggested by the publically available works of other researchers. In particular, examples of GRC packetization[15] were quite useful. Additionally, numerous example flowgraphs illustrating methods to perform constellation encoding and decoding, clock and phase recovery, and matched filter implementation were likewise suggested and employed[16].

To address the digital communications challenges, there were several key blocks that found their way into many of the GRC flowgraphs. These include:

- Costas loop: used to phase align the input;
- Clock recovery MM: Mueller/Muller clock recovery to adaptive pick the optimal sampling instant;
- Constellation decoder: using a constellation defined in the constellation object block, maps complex input samples to the nearest symbol;
- Protocol Formatter: using a protocol format specified in a format object, a header is generated for the tagged stream supplied at the blocks input;
- Correlate Access Code – Tag Stream: determines the beginning of a packet based on a predetermined (user specified) access code and creates a tagged stream of data bits;
- Tagged Stream to PDU: Constructs a Protocol Data Unit (PDU) asynchronous message out of the received packet;
- Vector source: a utility to provide a repeating message along with a preamble.

All digital communication schemes were designed and tested using two HackRF One SDRs attached to separate computers, communicating over a short RF link.

FSK Transmitter

For FSK transmission (shown below in Figure 8), 4-FSK was used as an introductory implementation. The 4-FSK transmitter repeatedly transmits the message defined in the variable "*msg_bytes*". A Vector Source block includes the text message that functions as the payload for the packet.

The payload stream must first must be converted to a tagged stream. Converting a stream to a tagged stream associates key-value pairs of metadata with the supplied stream of data, where generally this includes the data length as a minimum. A tagged stream is necessary in this case for GNU Radio to handle the packetization process using pre-created blocks. The resulting tagged stream is passed through the protocol formatter block to generate a packet header to be associated with the data packet. The header is then prepended to the data packet by the "Tagged Stream Mux" block, which requires the name of the key for packet length. The full data packet is then split into two-bit "nibbles" via the "Repack Bits" block and gray-coded by the "Chunks to Symbols" block. Following centering the sample values about zero, these symbols are supplied to a VCO which generates a frequency proportional to that symbol's value. Supplying the output of the VCO to the SDR mixes the generated frequency with the configured intermediate frequency of the SDR. The 4 MHz sampling rate with 100 samples per symbol indicates that we have a 40 ksps or 80 kbps data rate. The resulting FSK signal is transmitted about 912 MHz, within the ISM band.

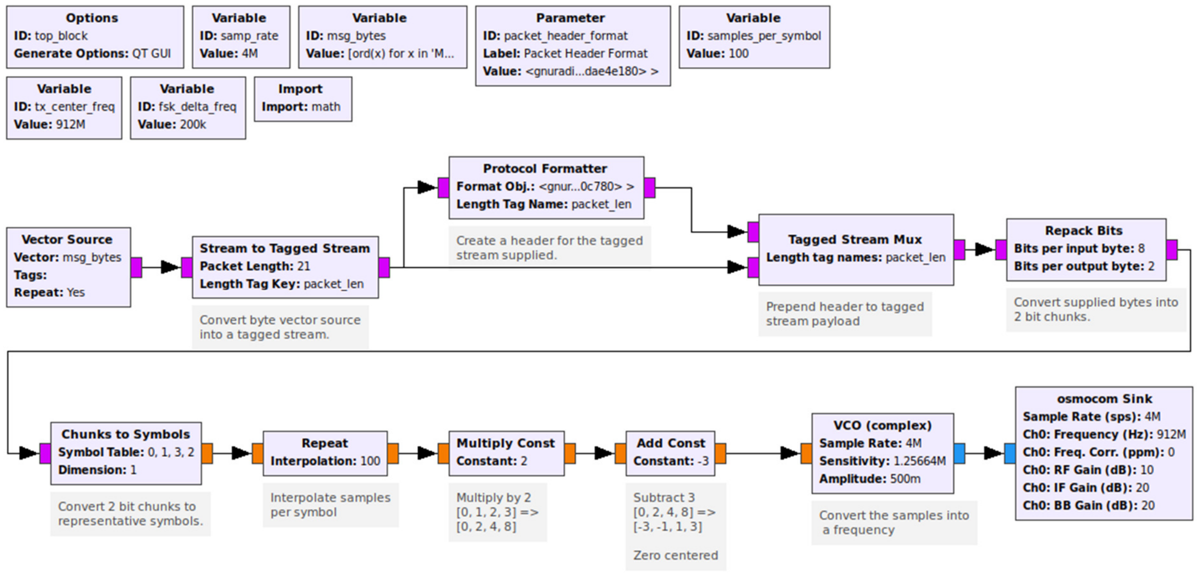


Figure 8: 4-FSK Transmitter

FSK Receiver

Shown below in Figure 9 is the receiver for the 4-FSK system. The 4-FSK receiver reads in the samples received by the SDR and performs packet detection. When a packet is detected the payload of the packet is written to the socket specified in the PDU Socket block. In order to view this data, a simple utility like "*nc*" on Linux can be used.

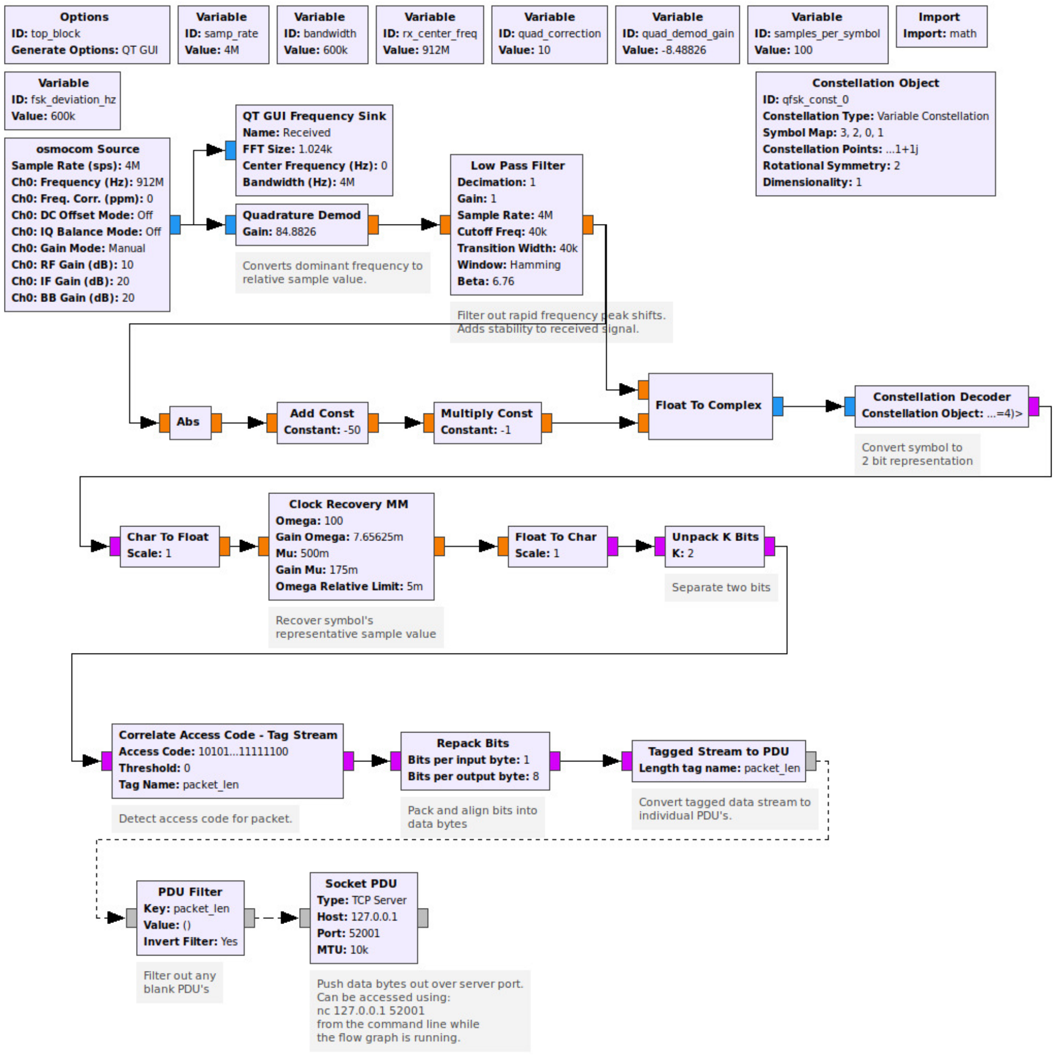
The received FSK signal is first passed through the "Quadrature Demod" block which returns a sample value representing the predominant frequency in the current sample set. The Quadrature Demod block behaves in a manner that can be conceptualized as the inverse of a VCO in this case. The sample value associated with the peak frequency is generally scaled by some gain value. An appropriate gain value can be calculated using:

$$gain = \frac{f_s}{\left(\frac{2\pi f_\Delta}{8}\right)}, \qquad (2)$$

where f_Δ is the frequency deviation of the FSK signal.

The high frequency content of the Quadrature Demod block output is then filtered out using a low pass filter to increase symbol detection accuracy. The low pass filter increases the symbol detection accuracy by supplying the clock recovery block a smoother signal with mostly low frequency content. This increases the clock recovery block's ability to better select appropriate samples. The output of the Quadrature Demod block is 4-PAM. The blocks following the low pass filter (preceding the "Clock Recovery MM" block) use the constellation detection capabilities of GRC to identify the symbol being transmitted. The Clock Recovery MM block is then used to select the appropriately timed sample to represent the symbol. The symbol is then translated to its representative bits using the "Unpack K Bits" block.

The bits are passed through the “Correlation Access Code – Tag Stream” block that detects the user-defined predetermined header access code and associates a packet length tag with the stream for the duration of the number of message bytes specified in the header. The data stream is then packed into full bytes and converted to a PDU. Empty PDUs are filtered out and the PDU is broadcast using a network socket on the local machine. To view the output of the socket, the *network connect* (nc) command line utility available in most Linux distributions can be used.



Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 4M

Variable
ID: bandwidth
Value: 600k

Variable
ID: rx_center_freq
Value: 912M

Variable
ID: quad_correction
Value: 10

Variable
ID: quad_demod_gain
Value: -8.48826

Variable
ID: samples_per_symbol
Value: 100

Import
Import: math

Variable
ID: fsk_deviation_hz
Value: 600k

Constellation Object
ID: qfsk_const_0
Constellation Type: Variable Constellation
Symbol Map: 3, 2, 0, 1
Constellation Points: ...1+1j
Rotational Symmetry: 2
Dimensionality: 1

osmocom Source
Sample Rate (sps): 4M
Ch0: Frequency (Hz): 912M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

QT GUI Frequency Sink
Name: Received
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 4M

Quadrature Demod
Gain: 84.8826

Converts dominant frequency to relative sample value.

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 4M
Cutoff Freq: 40k
Transition Width: 40k
Window: Hamming
Beta: 6.76

Filter out rapid frequency peak shifts. Adds stability to received signal.

Abs

Add Const
Constant: -50

Multiply Const
Constant: -1

Float To Complex

Constellation Decoder
Constellation Object: ...=4)>

Convert symbol to 2 bit representation

Char To Float
Scale: 1

Clock Recovery MM
Omega: 100
Gain Omega: 7.65625m
Mu: 500m
Gain Mu: 175m
Omega Relative Limit: 5m

Recover symbol's representative sample value

Float To Char
Scale: 1

Unpack K Bits
K: 2

Separate two bits

Correlate Access Code - Tag Stream
Access Code: 10101...11111100
Threshold: 0
Tag Name: packet_len

Detect access code for packet.

Repack Bits
Bits per input byte: 1
Bits per output byte: 8

Pack and align bits into data bytes

Tagged Stream to PDU
Length tag name: packet_len

Convert tagged data stream to individual PDU's.

PDU Filter
Key: packet_len
Value: ()
Invert Filter: Yes

Filter out any blank PDU's

Socket PDU
Type: TCP Server
Host: 127.0.0.1
Port: 52001
MTU: 10k

Push data bytes out over server port.
Can be accessed using:
nc 127.0.0.1 52001
from the command line while
the flow graph is running.

Figure 9: 4-FSK Receiver

Shown below in Figure 10 is the received spectrum of the 4-FSK receiver. Clearly visible in the plot are the four frequencies associated with the FSK system (note: complex spectrum).

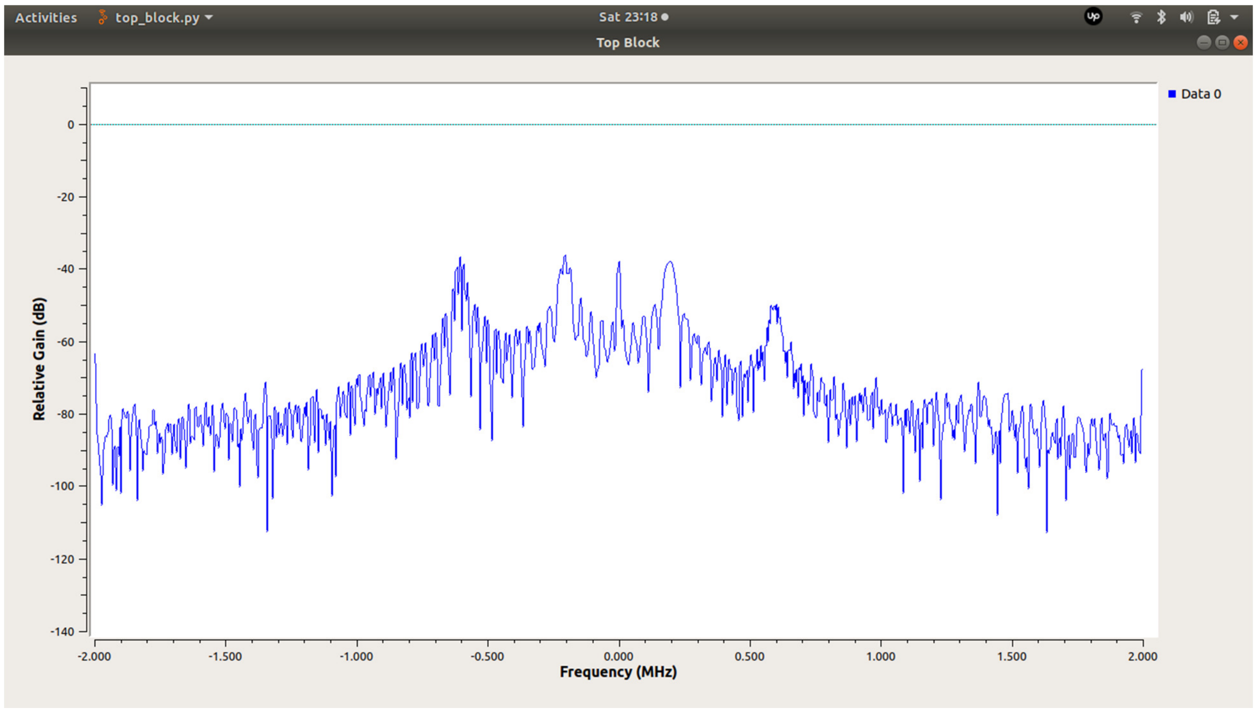
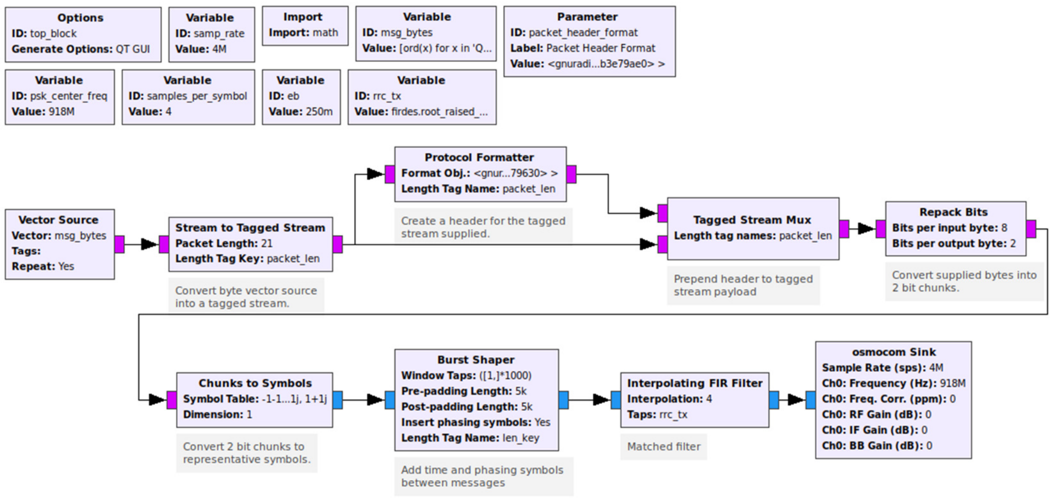


Figure 10: FSK Received Spectrum

PSK Transmitter

Given in Figure 11 is the flowgraph for the 4-PSK transmitter. The PSK implementation uses a constellation with points [0,1,2,3] at $[1 - j, 1 + j, -1 + j, -1 - j]$. The combination of the real and imaginary values leverages the way that the SDR encodes data by mixing the supplied values with the device's configurable intermediate frequency. The real part of the sample value supplied to the SDR will be mixed with a sinusoidal signal that is phase shifted from the imaginary part's mixing signal by 90 degrees. The two independent sinusoidal signals are then summed prior to the RF interface.

Similar to the FSK system, the source for the payload of the packet is a "Vector Source" block which contains a text message (again, the variable denoted "*msg_bytes*"). The source stream is then converted to a tagged stream using the "Stream to Tagged Stream" block that specifies the key to be used for the packet length. The tagged stream is then passed through the "Protocol Formatter" block to generate a packet header. The header is prepended to the data packet in the "Tagged Stream Mux" block. The samples are then split into two-bit "nibbles" via the "Repack Bits" block, and then converted to their complex representation in the "Chunks to Symbols" block. A "Burst Shaper" block is used to add space between the messages being sent and to keep sending alternating "dummy" symbols to maintain phase alignment. The symbols are then passed through a root raised cosine matched filter to increase the SNR and minimize ISI at the receiver. The 4 MHz sampling rate with 4 samples per symbol implies a 1 Msps or 2 Mbps data rate. The resulting PSK signal is then transmitted at 918 MHz, within the ISM band.


Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 4M

Import
Import: math

Variable
ID: msg_bytes
Value: [ord(x) for x in 'Q...

Parameter
ID: packet_header_format
Label: Packet Header Format
Value: <gnuradi...b3e79ae0> >

Variable
ID: psk_center_freq
Value: 918M

Variable
ID: samples_per_symbol
Value: 4

Variable
ID: eb
Value: 250m

Variable
ID: rrc_tx
Value: firdes.root_raised_...

Vector Source
Vector: msg_bytes
Tags:
Repeat: Yes

Stream to Tagged Stream
Packet Length: 21
Length Tag Key: packet_len
Convert byte vector source into a tagged stream.

Protocol Formatter
Format Obj.: <gnur...79630> >
Length Tag Name: packet_len
Create a header for the tagged stream supplied.

Tagged Stream Mux
Length tag names: packet_len
Prepend header to tagged stream payload

Repack Bits
Bits per input byte: 8
Bits per output byte: 2
Convert supplied bytes into 2 bit chunks.

Chunks to Symbols
Symbol Table: -1-1...1j, 1+1j
Dimension: 1
Convert 2 bit chunks to representative symbols.

Burst Shaper
Window Taps: ([1,]*1000)
Pre-padding Length: 5k
Post-padding Length: 5k
Insert phasing symbols: Yes
Length Tag Name: len_key
Add time and phasing symbols between messages

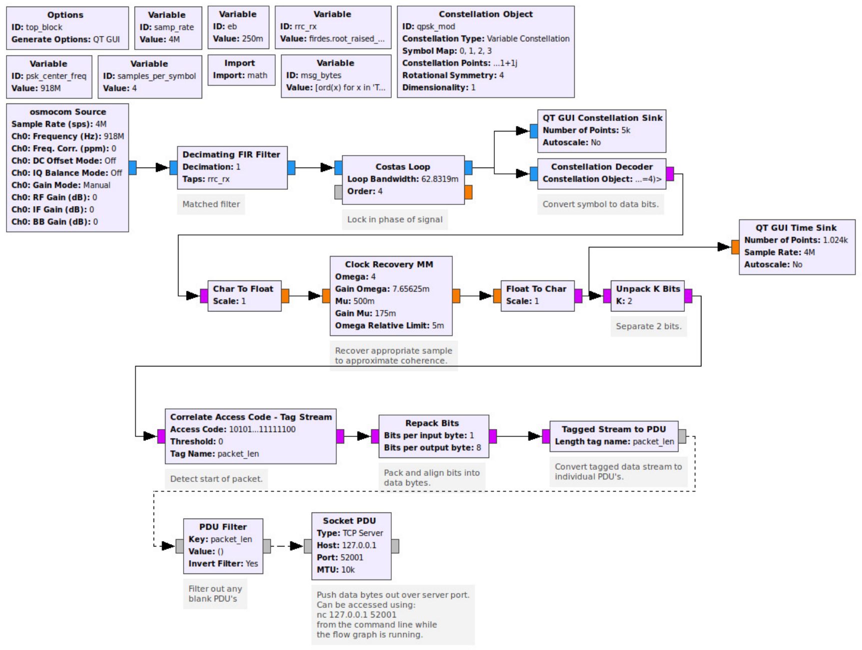
Interpolating FIR Filter
Interpolation: 4
Taps: rrc_tx
Matched filter

osmocom Sink
Sample Rate (sps): 4M
Ch0: Frequency (Hz): 918M
Ch0: Freq. Corr. (ppm): 0
Ch0: RF Gain (dB): 0
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0

Figure 11: 4-PSK Transmitter

<u>PSK Receiver</u>

The PSK receiver flowgraph is given in Figure 12. The signal is first passed through a root raised cosine matched filter to match the processing done at the transmitter. The signal is phase aligned using a 4th order Costas loop and decoded using GRC's constellation decoder block with a QPSK constellation defined using the "Constellation Object" block. The stream is then passed through a "Clock Recovery MM" block to select an appropriately representative sample from the stream. The symbol is then unpacked into its representative bits and run through packet detector, repacked into bytes, and converted to a PDU (which behaves identically to the 4-FSK receiver) to make the received packets payload available on a localhost network socket.


Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 4M

Variable
ID: eb
Value: 250m

Variable
ID: rrc_rx
Value: firdes.root_raised_...

Constellation Object
ID: qpsk_mod
Constellation Type: Variable Constellation
Symbol Map: 0, 1, 2, 3
Constellation Points: ...1+1j
Rotational Symmetry: 4
Dimensionality: 1

Variable
ID: psk_center_freq
Value: 918M

Variable
ID: samples_per_symbol
Value: 4

Import
Import: math

Variable
ID: msg_bytes
Value: [ord(x) for x in 'T...

osmocom Source
Sample Rate (sps): 4M
Ch0: Frequency (Hz): 918M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 0
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0

Decimating FIR Filter
Decimation: 1
Taps: rrc_rx
Matched filter

Costas Loop
Loop Bandwidth: 62.8319m
Order: 4
Lock in phase of signal

QT GUI Constellation Sink
Number of Points: 5k
Autoscale: No

Constellation Decoder
Constellation Object: ...=4)>
Convert symbol to data bits.

Char To Float
Scale: 1

Clock Recovery MM
Omega: 4
Gain Omega: 7.65625m
Mu: 500m
Gain Mu: 175m
Omega Relative Limit: 5m
Recover appropriate sample to approximate coherence.

Float To Char
Scale: 1

Unpack K Bits
K: 2
Separate 2 bits.

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 4M
Autoscale: No

Correlate Access Code - Tag Stream
Access Code: 10101...11111100
Threshold: 0
Tag Name: packet_len
Detect start of packet.

Repack Bits
Bits per input byte: 1
Bits per output byte: 8
Pack and align bits into data bytes.

Tagged Stream to PDU
Length tag name: packet_len
Convert tagged data stream to individual PDU's.

PDU Filter
Key: packet_len
Value: ()
Invert Filter: Yes
Filter out any blank PDU's

Socket PDU
Type: TCP Server
Host: 127.0.0.1
Port: 52001
MTU: 10k
Push data bytes out over server port.
Can be accessed using:
nc 127.0.0.1 52001
from the command line while
the flow graph is running.

Figure 12: 4-PSK Receiver

Digital Communications Performance

The performance of both of the digital modulation communication schemes described above is marginal and substandard. Constellation diagrams indicate higher than expected noise corruption and other impairments. As implemented, performance was insufficient to allow large payload data transfers. Therefore, only short strings of text were used for testing. As observed, packetization works well, but payload corruption occurs due to noise in the system that leads to the undesirable performance. The performance of the digital systems could likely be improved with additional noise reduction techniques and the integration of error checking within the packetization process. Increased signal levels may also help to improve performance.

Assessment

In the most recent offering of the course EGR 415 Communication Systems at Grand Valley State University, students were anonymously surveyed in order to obtain their opinions on the use of SDR as a platform for the instruction of analog and digital communications material. Each question allowed for free-form comments. The following questions were asked of the students (the enrollment was limited to 10 students in the Fall 2018 offering; all students responded to the survey):

Q1: The laboratory component of this course helps me to grasp the theoretical concepts presented in lecture.

Q2: Real-time signal modulation/demodulation is preferred to off-line simulation

Q3: The SDR/GRC approach combines the best of both worlds: block-based system design with real-time transmission of radio signals.

Q4: Development of labs and projects using the SDR/GRC approach is as straightforward as doing so in a simulation environment such as MATLAB/Simulink.

Q5: The possibilities of what I can develop with the SDR/GRC approach seem significantly more numerous as compared to using MATLAB/Simulink.

The results of the first three questions (mean and standard deviation) are displayed below in Figure 13 (key: 5: strongly agree; 4: agree; 3: neutral; 2: disagree; 1: strongly disagree).

The survey results indicate that students clearly appreciate the laboratory component of the course to aid them in their understanding of the theoretical material. The students also preferred the transmission and reception of “real” signals over the RF interface to those generated in simulation, and agreed that the SDR/GRC approach was a good way to achieve this goal. Students were less in agreement that GRC was as easy to use as MATLAB/Simulink, seemingly mostly due to the fact that the GRC documentation is highly incomplete. Finally, students did seem to think that the SDR/GRC approach is quite powerful, even compared to MATLAB/Simulink. These rating are supported by the selection of student comments presented below.

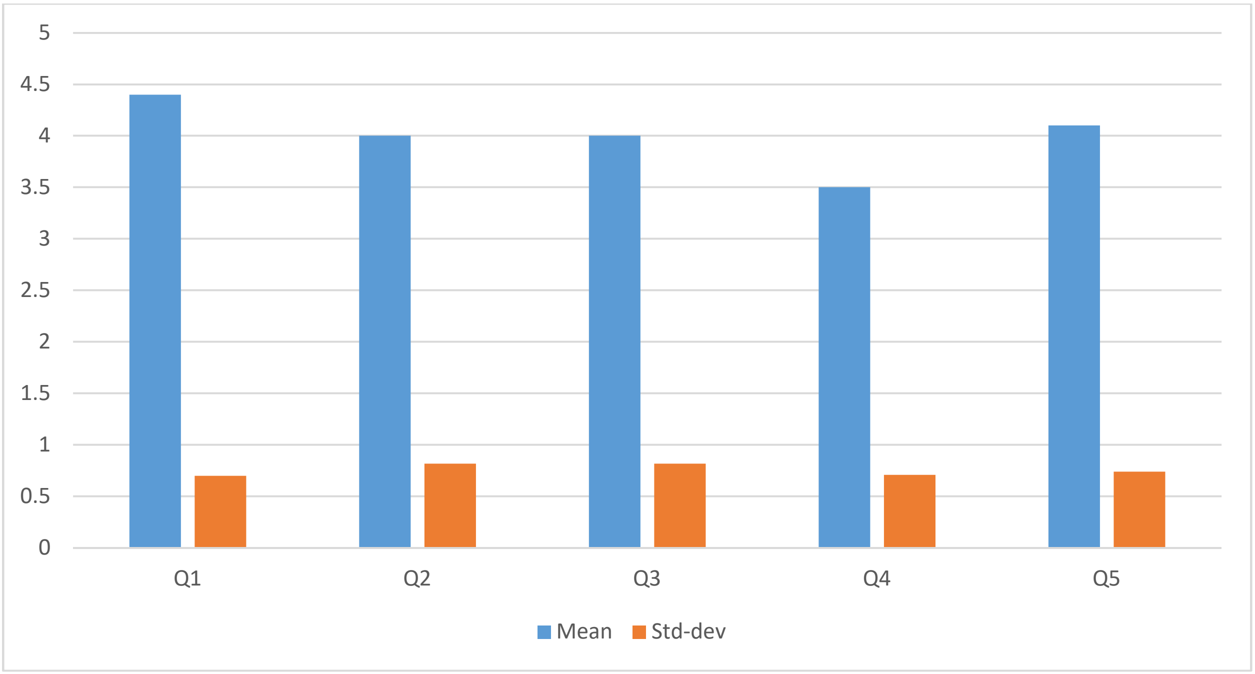


Figure 13: Student Survey Results Summary

Each question solicited student comments. A selection of insightful comments is given below:

1. *GRC flowgraphs are difficult to debug. Something doesn't work and it is very difficult to find your error.*

2. *Some of the public-domain materials for SDR/GRC are very exciting. These go beyond the course material, but we now have the pre-requisites to do our own investigation.*

3. *SDRs are amazing powerful and versatile!*

Conclusion

The point of this paper was to attempt to provide an educator a very strong starting point for delivering SDR-based laboratories in a first or second semester analog and digital communications course. For a few hundred dollars per workstation, students are quickly able to develop sophisticated communication systems. Students seem to greatly appreciate this very powerful approach to delivering communication laboratory experimentation.

The flowgraphs presented consider the more difficult case of actual RF over-air communication (as opposed to simulating transceiver operation in a single flowgraph). To that end, flowgraphs for analog communications (AM, FM) and digital communications (4-FSK, 4-PSK) were presented, along with explanations that should significantly help the motivated educator get started.

While this paper gives a good start, there is plenty of opportunity to expand on the materials given. The actual crafting of laboratories for students (what they should develop, what they should measure) need to be developed as appropriate for the instructor teaching the course.

Unfortunately, in both the analog and digital communication cases, over-the-air performance was found to be less than perfect. This was especially true for the digital modulation experiments. The exact remedy for this substandard performance remains an active area of investigation for the authors. However, that being said, the overall approach is still very satisfying for academic laboratory instruction, and the subpar performance demonstrates the challenges faced in the design of communication systems. Students get to see first-hand how the impairments impact quality.

Beyond improvements to current methods, other modulation schemes need to be considered – plans for future work include flowgraphs for higher M-ary levels and other modulation formats including ASK and QAM. Following these relatively straightforward implementations, flowgraphs to implement OFDM and DSSS are possible.

Bibliography

1. Bard, J. & Kovarik, V., "*Software Defined Radio: The Software Communications Architecture*," Wiley Series in Software Radio, 2007.
2. Reed, J., "*Software Radio: A Modern Approach to Radio Engineering*," Prentice Hall, 2005.
3. URL: www.gnuradio.org last visited January 31, 2019.
4. Mao, S., & Huang, Y., & Li, Y. (2014, June), *On Developing a Software Defined Radio Laboratory Course for Undergraduate Wireless Engineering Curriculum* Paper presented at 2014 ASEE Annual Conference, Indianapolis, Indiana. https://peer.asee.org/22880
5. Wu, Z., & Wang, B., & Cheng, C., & Cao, D., & Yaseen, A. (2014, June), *Software Defined Radio Laboratory Platform for Enhancing Undergraduate Communication and Networking Curricula* Paper presented at 2014 ASEE Annual Conference, Indianapolis, Indiana. https://peer.asee.org/23023
6. Wyglinski, A. M., & Cullen, D. J. (2011, June), *Digital Communication Systems Education via Software-Defined Radio Experimentation* Paper presented at 2011 ASEE Annual Conference & Exposition, Vancouver, BC. https://peer.asee.org/17783
7. Cao, D., & Wu, Z., & Wang, B., & Cheng, C. (2018, June), *Undergraduate Research: Adaptation and Evaluation of Software-defined Radio-based Laboratories* Paper presented at 2018 ASEE Annual Conference & Exposition , Salt Lake City, Utah. https://peer.asee.org/31170
8. Zhang, Z., & Wu, Z., & Wang, B., & Cheng, C., & Cao, D. (2016, June), *Software Defined Radio-based General Modulation/Demodulation Platform for Enhancing Undergraduate Communication and Networking Curricula* Paper presented at 2016 ASEE Annual Conference & Exposition, New Orleans, Louisiana. https://peer.asee.org/25833
9. Gandhi Raja, R., Ranjini Ramb, & Soman, K.P. (2011, December), *Analog and Digital Modulation Toolkit for Software Defined Radio* Paper presented at 2011 International Conference on Communication Technology and System Design, Coimbatore, India.
10. VonEhr, K., & Neuson, W., & Dunne, B. E. (2016, June), *Software Defined Radio: Choosing the Right System for Your Communications Course* Paper presented at 2016 ASEE Annual Conference & Exposition, New Orleans, Louisiana. https://peer.asee.org/25838
11. URL: https://greatscottgadgets.com/hackrf/ last visited January 31, 2019.
12. URL: https://limemicro.com/products/boards/limesdr-mini/ last visited April 21, 2019.

13. URL: https://www.ubuntu.com/ last visited January 31, 2019.
14. Dunne, B.E. (2019 March), *The What, How and Why of Complex Sampling for SDR Transceivers* Paper presented at 2019 ASEE North Central Section Conference, Grand Rapids, Michigan.
15. URL: http://aaronscher.com/ last visited April 26, 2019.
16. Mathys, P. (2016 September), *Motivating Undergraduate Communication Theory Using GNU Radio* Paper presented at 2016 6th GNU Radio Conference, Boulder, Colorado.