Ryan Kearns

Digital Communications

Lab 1

# Introduction

In this lab, we built an AM radio receiver. We used the Nooelec NESDR SMArt XTR SDR with an antenna to capture ambient radiation. GNU Radio Companion was used to build a software defined radio. The below sections outline how this is done.

# GNU Radio

The below flow diagram in Figure 1 was built for this lab.

A diagram of a computer flowchart

Description automatically generated with medium confidence

Figure : GNU Flow Diagram

The key components of this flow diagram is the RTL-SDR Source, Low Pass Filter, AM Demod, Rational Resampler. Multiply constants, and the various GUI functions. Each of these sections is detailed below.

## RTL-SDR Source

The RTL-SDR source is the USB compatible SDR device “Nooelec NESDR SMArt XTR SDR.” This device has an operational bandwidth of 55 MHz-2300MHz. It connects to an antenna via SMA connector. The software defined radio (SDR) is based on the E4000. For further details on the E4000, see the associated E4000 datasheet, attached to this folder. The antenna is a telescopic antenna which varies the frequency band based on the length of the antenna.

The GNU interface allows for variation in the sample rate and the center frequency. The sample rate is controlled by the variable sample\_rate. The sample rate chosen for this device is 1 MSps. The center frequency is selected by the QT GUI Range parameter center\_freq which is further detailed below.

The output of the RTL-SDR source is complex. These values are fed into the next value within the module “Low Pass Filter.”

## Low Pass Filter

The role of the low pass filter is the module which immediately follows the RTL-SDR source. It is responsible for reducing the operational bandwidth of the signal to just the signal of interest. It accomplishes this by using a hamming window with the parameters given in Table 1.

|  |  |
| --- | --- |
| Parameter | Value |
| Filter Type | Hamming Window |
| Beta | 6.76 |
| Cutoff Frequency | 5 KHz |
| Transition Width | 1 KHz |

Table : Hamming Window Parameters

The first parameter which can be tweaked is the “beta.” The larger the beta value, the longer the transition region of the filter. However, the larger value also means lower sidelobes of the filter. In order to meet the transition region determined by the other parameters, extra filter taps must be added to meet the filter parameters. By fine-tuning this parameter, an engineer can minimize the number of taps required to meet the design needs.

Without a low pass filter, out of band noise would be passed through to the speakers. This noise often can result in interference and may make demodulation more difficult by reducing the signal to noise ratio. A tighter filter reduces this out of band noise but also requires increased computational intensity. The cutoff frequency and transition width define this transition region. The cutoff frequency defines the 3db point of the filter attenuation. The transition width defines the stopband frequency of the filter. A balance between the steepness of a filter and the amount of allowable out of band noise is an important parameter for an engineer to trade.

The module accepts complex inputs and outputs complex values. The output should have a nearly unchanged passband but an attenuated stopband. This should increase the SNR of the signal of interest. The output of the low pass filter is the AM Demod.

## AM Demod

The AM Demod takes the input signal and demodulates it so that the original information can be recovered. It accomplishes this by applying filtering to the signal and converting it from complex to floating point. The filtering is in many ways redundant given the low pass filtering completed in the previous section. Instead, the AM Demod is relevant for converting the signal from complex to floating point.

The filtering for the AM demod has a passband of 5 kHz and a stopband of 6 kHz. This is the same filter parameters for transition width that we see in the low pass filter. Similar performance is expected between these two filters.

The action of converting from complex to floating point is done via Inverse Hilbert Transform. The Hilbert Transform is a mathematical function which converts real only signals into complex signals. The Inverse Hilbert Transform instead converts from complex into real only signals. The real only signals are required for the audio output as the quadrature portion of an imaginary value has no physical interpretation for a speaker. By converting to real only signals, the output can be heard via normal speakers.

The output for the AM demod is the throttle.

## Throttle

The throttle acts as a limiter on data rates to ensure sample rates do not exceed the value listed. While the SDR is responsible for ensuring that data rates remain constant, the limiter will throttle the data in the event that data rates begin to exceed the pre-determined limit.

The output of the throttle is the rational resampler.

## Rational Resampler

The rational resampler acts as a sample rate converter to ensure datarates are compatible with downstream needs. In this case, the datarate needs to be 32 kHz to ensure it is compatible with the device speakers which act as output. It accepts an interpolation rate and a decimation rate independently. Both must be integer values.

Resampling acts in two stages. While computationally, it can be completed in a single step, they are two independent stages. A resampler first upsamples then downsamples. The filter for a resampler is always designed such that it handles the most difficult of the two stages.

Upsampling is the act of putting zeros in between each sample to accomplish a desired sampling rate. The act of upsampling creates images of the original signal at each point past the original Nyquist point. Filtering removes these images and leaves the user with just the original signal. The act of upsampling then filtering is known as interpolation.

Downsampling is the act of throwing away every Nth sample to reduce the sample rate. Removing samples inherently will reduce the Nyquist bands of the sampled signal. To avoid aliasing, filtering must be applied to the signal such that the stopband is before the new Nyquist frequency. The act of filtering then decimating is known as decimating.

These two can be combined to create what is known as a resampler. In this case, decimation is greater than interpolation and therefore, the filter must be designed to account for the decimation. GNU Radio automatically populates and creates this filter.

The resample rates are chosen such that the output signal is compatible with the 32 kHz speaker used by the computer. Therefore, a resample ratio of 2/25 is used such that the 1 MSps input sample rate is converted to 32 kHz. The data is output to the multiply constant.

## Multiply Constant

The multiply constant controls the amplitude of the signal. The multiply constant multiples the entire signal in the time domain by a scaling factor such that the amplitude can be controlled. In this case, the constant acts as a volume control so that the signal can be “louder” at the output.

The output of multiply constant is the audio sink.

## Audio Sink

The audio sink is the speaker of the device being used. A speaker on a device is a DAC which converts the floating point input values into discrete voltages. As these voltages oscillate with a signal, a diaphragm is moved with the same frequencies generated from the signal. This converts the speaker values back into noise which an end user can listen to. The speaker sample rate are standard values and in this case, 32 kHz was chosen.

# GUI

The GNU Radio allows a user to display many different aspects of the signal. These tools are utilized in this software to help visualize the spectrum. The below photo details this GUI.

A screenshot of a computer

Description automatically generated

Table : GUI Output

## Volume Control

As mentioned in Section Multiply Constant, the volume is controlled via the multiply constant module. A QT GUI Range variable was used to allow the user to modify the volume in the GUI. The variable assigned to this module is “volume” and it is passed directly to the multiply constant module such that a user can change the volume from within the GUI.

## Center Frequency

The center frequency is another QT GUI Range variable. This time, the variable is passed to the RTL-SDR Source module so that the center frequency can be controlled from within the GUI. The user is given both a slider and a textbox from which they can enter a value.

## QT GUI Waterfall Sink

The QT GUI Waterfall Sink is a module which allows us to see the entire spectrum which is being passed to the audio sink. A red value represents frequencies which have large amplitudes while a blue value represents low amplitudes. The waterfall is an FFT based representation which shows how the spectrum is changing over time. As the data flows down the screen, any changes in amplitude will be captured.

## QT GUI Frequency Sink

The QT GUI Frequency Sink is like the waterfall sink except that it is an instantaneous snapshot of the frequency spectrum. It is an FFT representation of the spectrum where the y-axis represents power and the x-axis represents frequency. In the image, a tone can be clearly seen.