Steve Jordan, Bhaumil Patel
2481843, 2651785
CIS632
Project Final Report

Image transfer and Software Defined Radio using USRP and GNU Radio

Overview:

Software Defined Radio (SDR) refers to the process of creating software that performs radio functionality that normally would be implemented in hardware. The purpose is to implement code that comes as close to the antenna as possible. In this way, radio hardware problems can now be resolved in software. The software performs all signal processing, including modulation/demodulation, filtering, sampling, encoding/decoding, etc. For this project, the goal is to perform wireless transfer of image data, in JPEG format, using the SDR paradigm.

An idealized SDR would include several "hard" or fixed components including an Antenna, front-end RF Hardware, and Analog to Digital/Digital to Analog Converters (ADC/DAC). The rest of the functionality would be implemented in a "soft" or programmable medium. The most common "soft" device is a general purpose processor. However, processors lack the I/O bandwidth and processing capabilities necessary for implementing SDRs for all but the simplest architectures. Figure 1 shows a block diagram of a typical SDR setup.

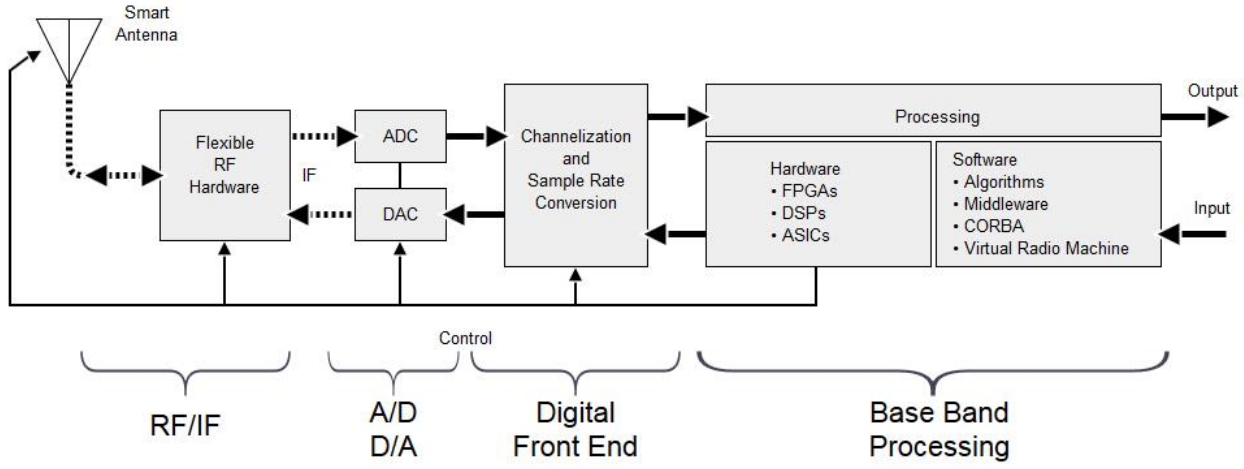

Figure 1

The software portion will be implemented using GNU Radio Companion (GRC). GNU Radio is an open-source signal processing package that provides the necessary tools to implement software radios. It can be used with external RF hardware to create SDRs, or without hardware, for simulations. It provides functions for implementing spectrum analyzers, oscilloscopes, concurrent multichannel receivers, and a collection of modulators and demodulators. It is widely used in commercial environments for both wireless communications research and real-world radio systems. It is licensed under the GNU General Public License (GPL) version 3 or later. All of the code is copyright of the Free Software Foundation. GNU Radio Companion refers to the graphical tool that will be used for creating the signal flow graphs and generating the flow-graph source code. For this project, GNU Radio Release 3.7.10.1 will be used. It will be used in conjunction with Ubuntu Linux 14.04.5 LTS, 64-bit edition.

GNU Radio is a modular, "flow-graph" oriented framework that comes with a comprehensive library of processing blocks that can be readily combined to make complex signal processing applications. Figure 2 shows a simple signal flow graph without any signal processing blocks. It is made up of a source, to generate a signal, and a sink, to display the output.

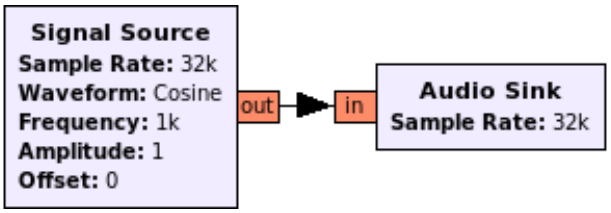

Figure 2

GRC will generate a Python file for each processing block. Each file can then be modified to fit the needs of the developer. It also includes a variety of C++ classes that implement many of signal processing functions. While the signal processing blocks are implemented in C++ and the main application in Python, SWIG (Simplified Wrapper and Interface Generator) is used to connect the two. SWIG is a Linux package that converts the C++ classes to compatible Python classes. Figure 3 shows this relationship.

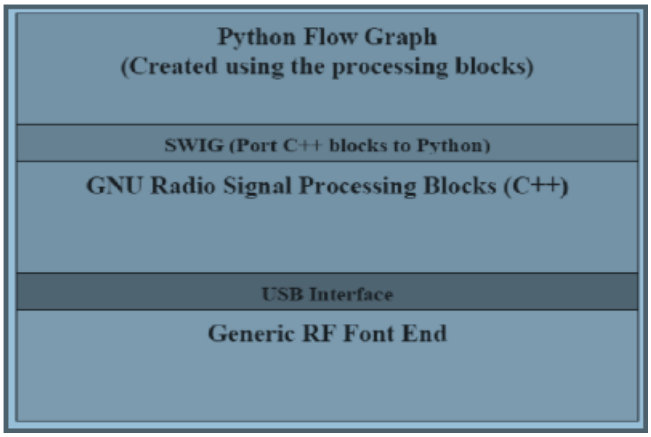

Figure 3

Universal Software Radio Peripheral (USRP) refers to the hardware portion of the SDR. It covers a range of software-defined radios designed and sold by Ettus Research and its parent company, National Instruments. The USRP connects to a host computer through a high-speed link. The host-based software uses this link to control the USRP and transmit/receive data. Some USRP models also integrate the general functionality of a host computer with an embedded processor. This allows the USRP device to function in a stand-alone manner. Figure 4 shows the block diagram for the USRP and Figure 5 shows the actual system.

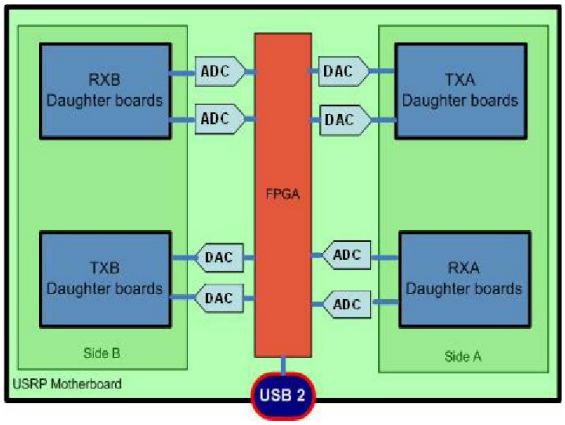

Figure 4: Block diagram of USRP system

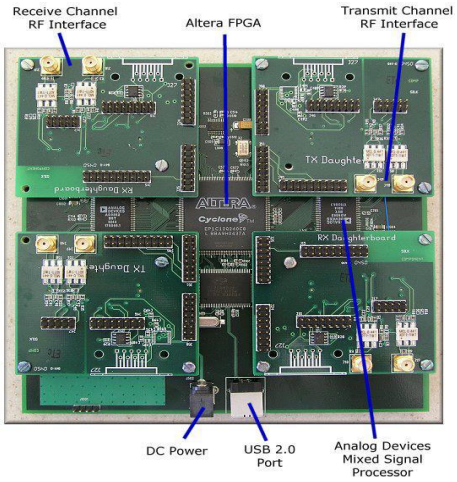

Figure 5: Actual USRP system

The USRP consists of a Programmable Field Gate Array (FPGA), Analog to Digital Converters (ADC), Digital to Analog Converters (DAC), and a Universal Serial Bus controller (USB). It is designed to interface an analog signal generated by, or transferred to, software. It interfaces with SDR libraries, such as Gnu-Radio, LabVIEW, and Simulink. Essentially it is a motherboard with an FPGA as well as a USB microcontroller. It contains RFX2400 daughter-boards which perform both transmit (TX) and receive (RX) functionality. These are equipped with ISM-band RF filters for suppression outside the band 2.402-2.48 GHz. Essentially, they are designed for operation in the 2.4 GHz band. The onboard analog devices capture the data and perform decimation/interpolation as well as filtering. An Altera FPGA outputs the stream of data into the USB microcontroller. The microcontroller accesses the interface between FPGA and the USB enabling data transfer to the PC. Figure 6 shows this process.

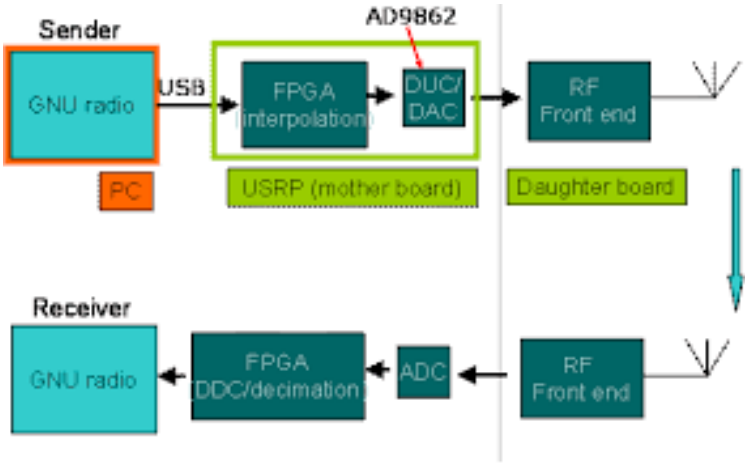

Figure 6

Experiment:

Because all communication systems, especially wireless, are limited due to inherent noise, this project will center on various aspects of that problem. Different modulation schemes and error correction methods and how they affect the signal quality will be considered. Two modulation schemes will be performed, Binary Phase Shift Keying (BPSK) And Quadriphase Phase Shift Keying (QPSK). In BPSK, the carrier sine wave is shifted 180° for each change in binary state (Figure 7). BPSK is coherent as the phase transitions occur at the zero crossing points.

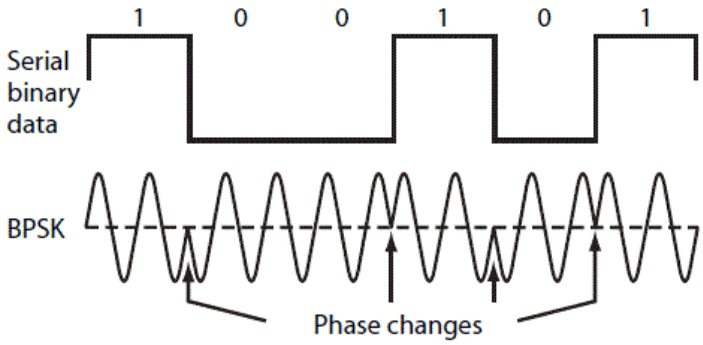

Figure 7

QPSK is a variation of BPSK where the modulator produces two sine carriers 90° apart. The binary data modulates each phase, producing four unique sine signals shifted by 45° from one another. The two phases are added together to produce the final signal. Each unique pair of bits generates a carrier with a different phase. Since each carrier phase represents two bits of data, twice the data rate can be achieved. Figure 8 shows QPSK with a phasor diagram and a constellation diagram.

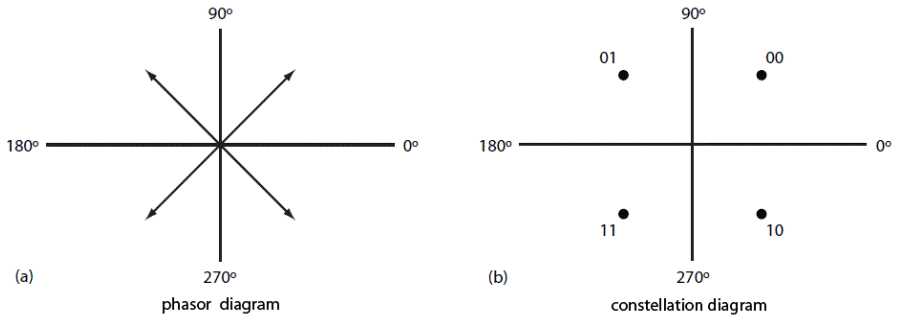

Figure 8

Coding for error handling can be done in several different ways. One "detection without correction" strategy is Automatic Repeat Request (ARQ). In this implementation, the receiver requests retransmission when errors are detected. This is a common method in computer systems. A different approach, and one which will be implemented in this project, is Forward Error Correction (FEC). FEC coding uses redundancy by adding to the original data stream. This allows for errors to be detected and some corrected at the receiver. The drawback to adding FEC is a data rate that is decreased due to the redundant bits. The benefit, however, is the ability of the receiver to correct the errors without retransmission of the message. The error correction process is shown in Figure 9.

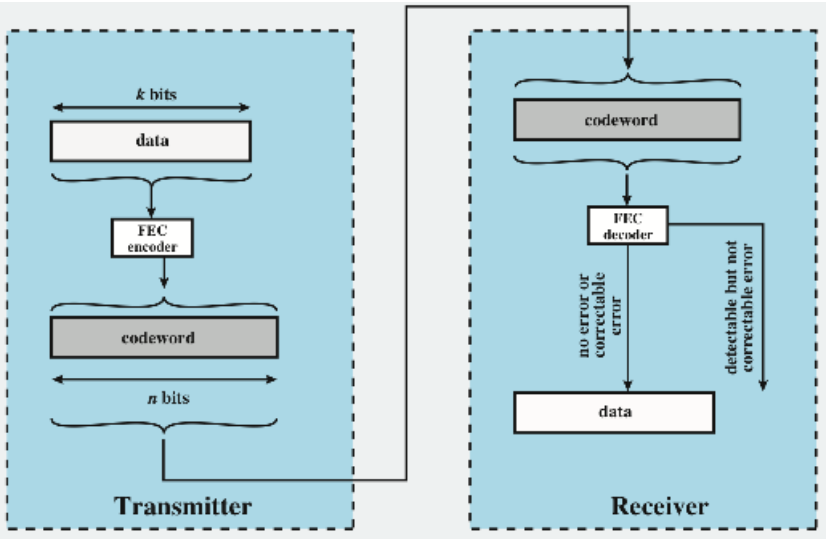

Figure 9

The project will be split into several stages. The 1st stage will be purely simulation. An attempt will be made to modulate/encode and demodulate/decode the image data all within GRC. There will be no real output other than the data accumulated for debugging purposes. Once it has been shown that the image data has been processed correctly, stage 2 will begin. In the 2nd stage, the file transfer between the USRPs will be implemented using a loop back cable. This will confirm that GRC connects and interacts with USRPs in an acceptable manner. The 3rd and final stage will perform the wireless transfer of the file. At this point, the testing will begin. The different modulation schemes will be compared, with and without FEC, and at different distances. It is expected that the early development will be the most difficult and time consuming, due to a lack of familiarity with SDR.

Modulation:

The first step of this project was to create a simulation of the modulated signal in Gnu Radio Companion (GRC). Figure 10 shows our signal flow graph for the simulated modulation sequence.

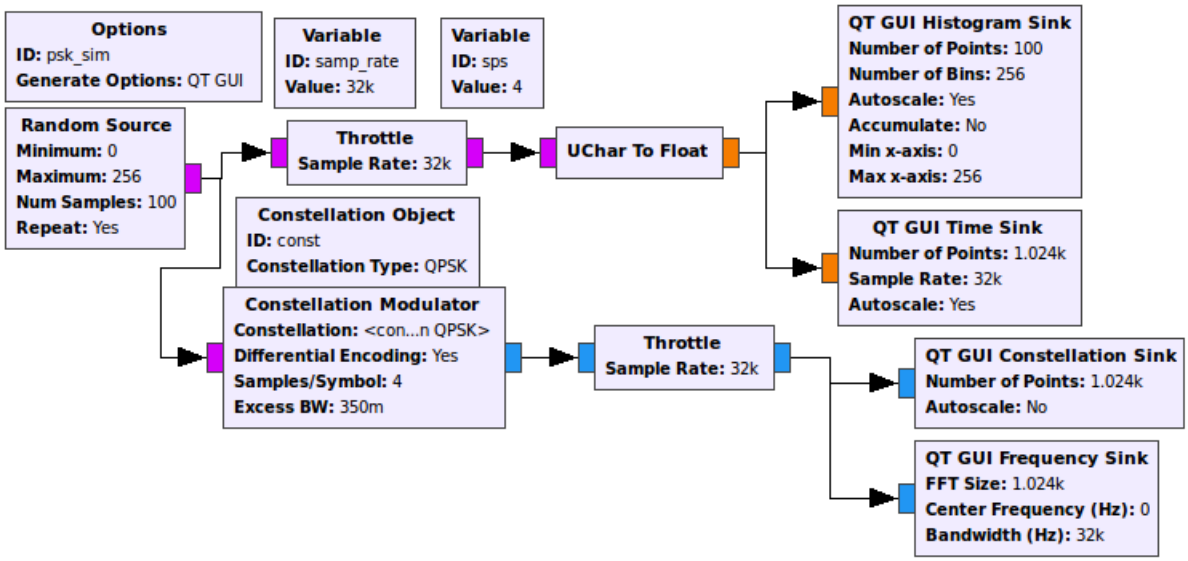

Figure 10: Signal Flow Graph - Simulation

The simulation is run with a random number generator as an input. The source is set to repeat, creating a continuous signal to allow analysis. The range of the random number generator is set from 0 to 256. The GNU-Radio documentation suggests a maximum value that is one greater than the desired value. Since the output values are bytes, a conversion block is used to change the values from unsigned characters to floating point values. This enables the generated signal to conform to something readable on an oscilloscope. The signal is also passed to a modulation block which performs the phase shift keying (PSK). There are two signal paths, one for analysis of the input signal, and one for the modulated output. As this is a simulation, throttle blocks are necessary and placed in both signal paths.

Four different outputs will analyze both the input signal and the modulated signal. For the input signal, a histogram and an oscilloscope will be used. For the modulated signal, a constellation plot as well as the spectral waveform will be used. The outputs are shown using "QT GUI" processing blocks. Figure 11 shows the oscilloscope and histogram outputs.

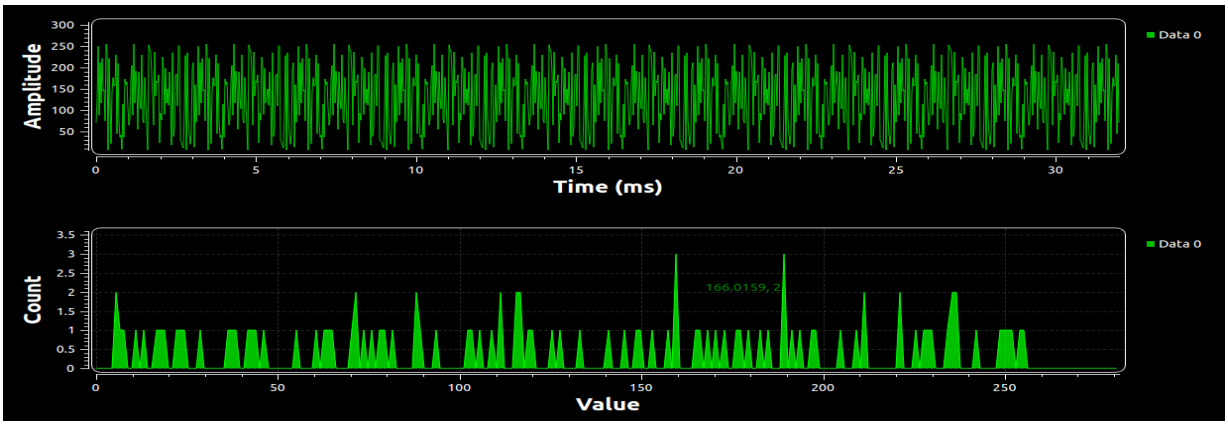

Figure 11: Oscilloscope and Histogram of Input Signal

For the time signal in Figure 11, with the scaling set large enough, the periodic nature of the signal is noticeable. Even though the source is a pseudo random number generator, when in repeat mode the random sequence is no longer random. Although not evident in this report, the histogram plot appears static. This is due to the fact that the number sequence repeats at 100 and the plot shows every 1000 values. When the frame size is changed to something other than a multiple of 100, the graph becomes quite active.

In the second path, the signal first passes through a "Constellation Modulator" block. There are several ways to implement PSK in GNU-Radio. This was chosen, not only because of the ease of choosing the PSK schemes, but because it implements differential encoding within the block. Without this, in the case of BPSK for example, the received data could be inverted. Some form of synchronization is needed between transmitter and receiver. Differential encoding allows for this unambiguous signal reception. Because of exclusive-or operations on the data, what is transmitted depends, not only on the current bit, but the previous one as well. The "Constellation Modulator" block is controlled via the "Constellation Object" block. Any changes in the type of keying being performed, occurs here. For this report, QPSK was designated and is shown in all diagrams. The output of the "Constellation Modulator" block is fed to a Constellation Sink to view the Constellation Plot (Figure 12) and a Frequency sink for spectral analysis (Figure 13). The samples per symbol was increased to 32 (far greater than necessary) to show the transitions between constellation points. The constellation plot shows distinct separation between the four phase values of the QSPK signal. The spectral plot shows a fairly smooth and narrow baseband signal that rolls off into the noise.

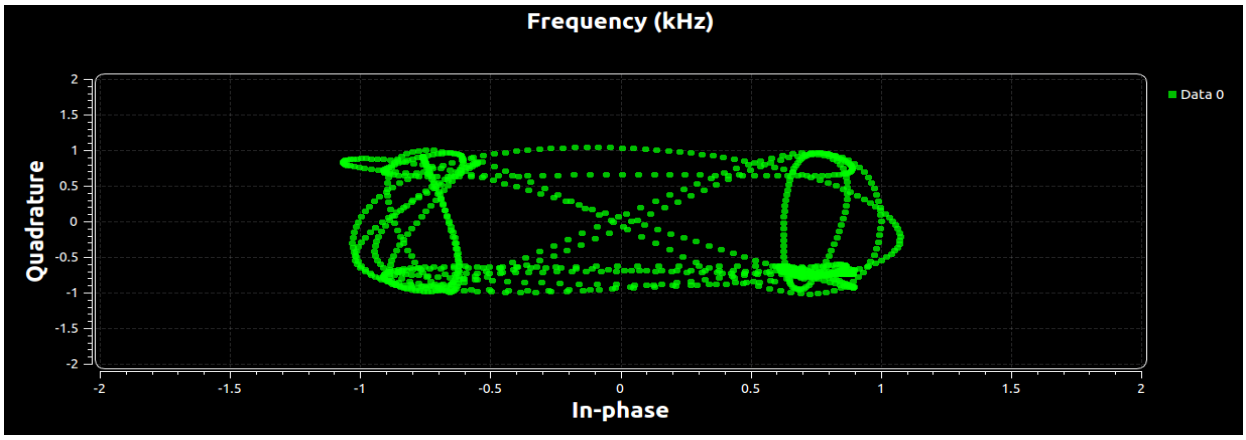

Figure 12: Constellation Plot of Modulated Signal

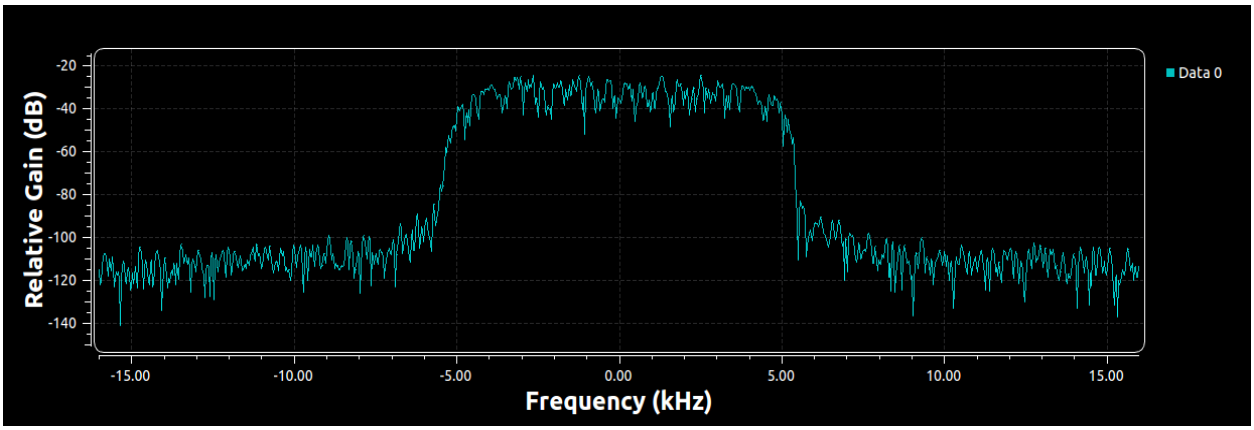

Figure 13: Spectral Plot of Modulated Signal

Also, within the modulator block is an attribute called "Excess Bandwidth." This attribute is used for pulse shaping. Sharp edges on a digital signal are generated by high frequency components. If the edges can be rounded, the overall bandwidth, and thus the energy needed for transmission can be reduced. Figure 14 shows a spectral plot with values of "Excess Bandwidth" that range from 0.1 to 1.0. It's clear that as the value is increased, the bandwidth diminishes. Typical values will usually range between 0.2 and 0.35. It is unclear if this will have an impact on the error rate of the transmission. Until that is determined, this will be monitored.

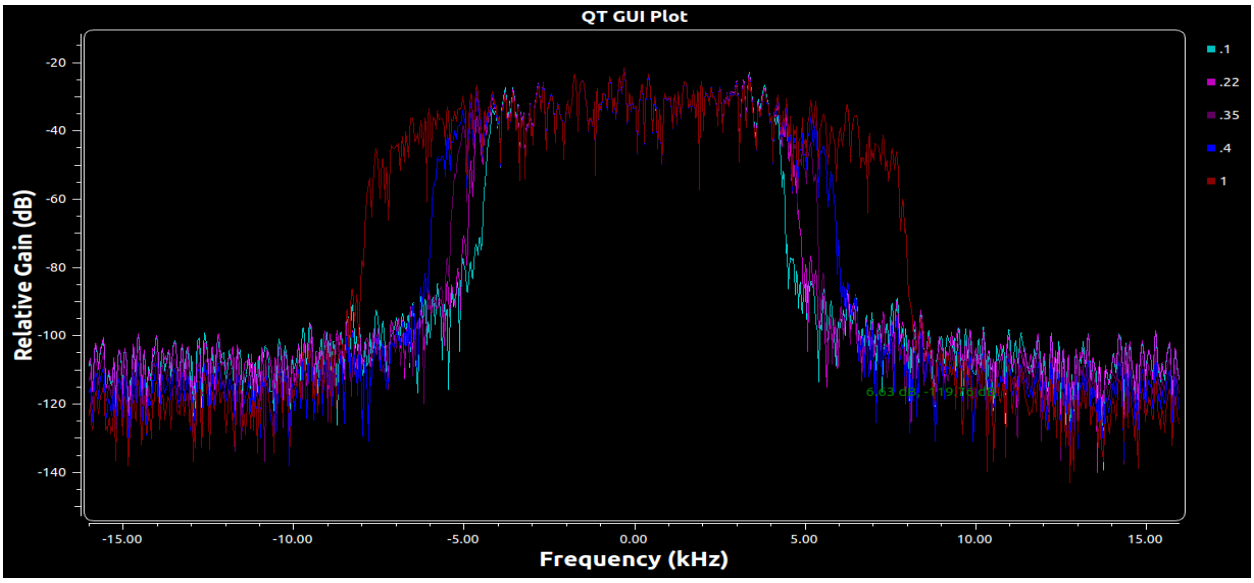

Figure 14: Spectral Plot with varying values of Excess Bandwidth

The drawback of pulse shaping is an effect referred to as Inter-Symbol Interference (ISI). Essentially, the symbols blur together. This is occurs due to the constellation modulator using a root raised cosine (RRC) pulse shaping filter. The filter causes the pulses to spread beyond their time interval. This causes interference with the adjacent pulses. The solution involves using the same filter on the receiver end. The combination of both filters together results in a raised cosine filter, which is a form of a Nyquist filter. A Nyquist filter is defined as a filter which creates no ISI. Figure 15 shows the output time signal from the Constellation Modulator. As can be seen from the plot, the sample points are scrambled. In Figure 16, that same signal is put through a "Decimating FIR Filter," which mimics the receiver end by performing a second RRC filter. In this plot, there are two very distinct lines at 0.7 and -0.7 which happen to correspond to the constellation points specified by the constellation modulator.

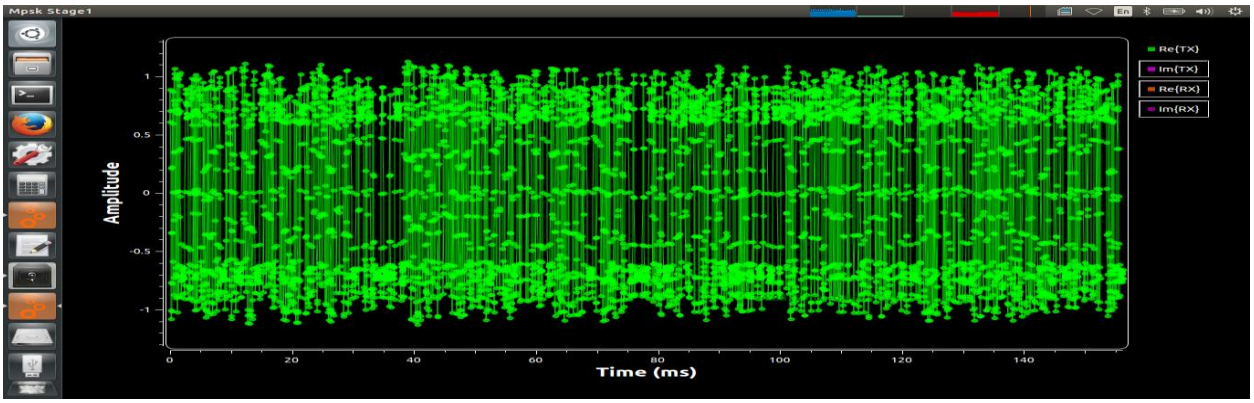

Figure 15: Inter-Symbol Interference

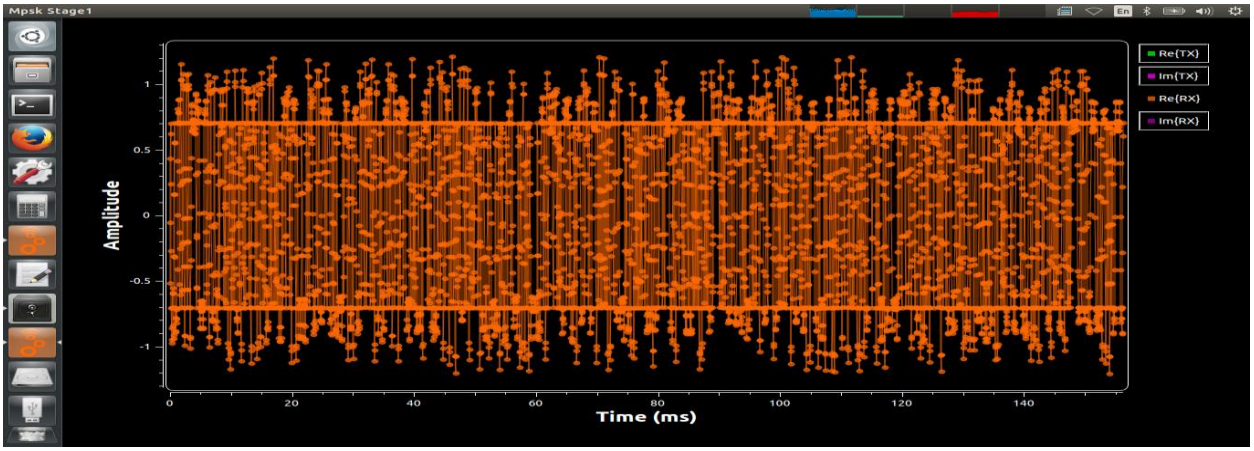

Figure 16: Inter-Symbol Interference Corrected

Channel Simulation:

The next step of this project was to create a simulation of the transmission channel in Gnu Radio Companion (GRC). Figure 17 shows our signal flow graph for this sequence.

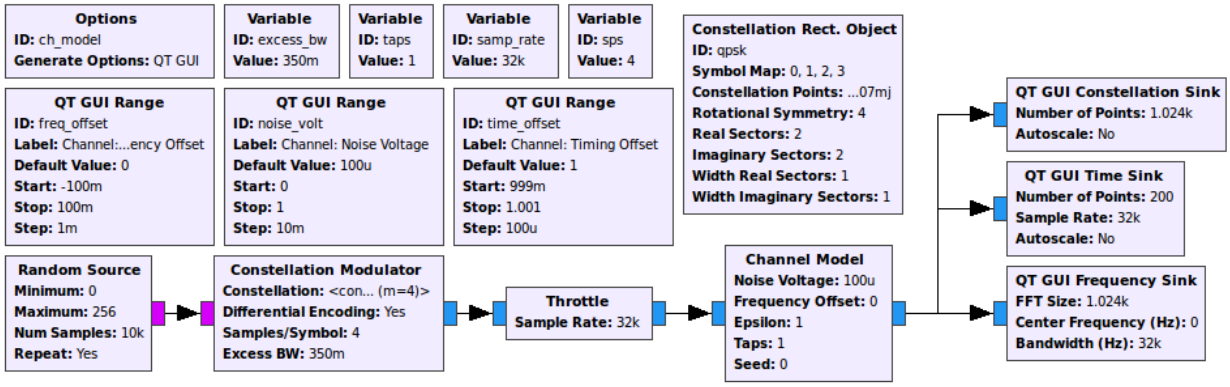

Figure 17: Simulated Transmission Channel

There are a few differences from the original flow graph in this version. First, there is no output for the pre-modulated signal. Second, a "Channel Model" processing block has been added. This will enable the simulation of the typical issues that occur during transmission. Our concern is with three things, noise, timing, and frequency drift. For each of these, a "QT GUI Range" processing block is added. These will allow dynamic simulation of the corresponding issue. Lastly, the output of the channel will be shown in constellation, time, and frequency plots.

The purpose of this portion is look into the effects of the channel and how the signal becomes distorted after it is transmitted and before it is seen at the receiver. The first issue is noise. Thermal noise in the hardware causes noise that is commonly known as additive white Gaussian noise (AWGN). The noise power is set by adjusting the noise voltage of the "Channel Model." The voltage is specified instead of power because of its independence to the signal. This is necessary because the "Channel Model" knows nothing about the incoming signal. The second problem to be resolved occurs because the transmitter and receiver have different clocks, which drive the frequencies of both devices. Imperfections in both clocks cause the specified frequencies to be skewed. Additionally, a lack of synchronization between the two compounds the issue. The end result is a timing offset between origin and destination. A related issue to the timing offset is the problem of finding the ideal sampling point. To minimize the ISI, we need to

sample the signal as close to the original sampling point as possible. Figure 18 and Figure 19 show the modulated signal prior to any channel effects. The time signal is clean, as well as the spectral plot. The constellation plot shows some spreading due to over sampling.

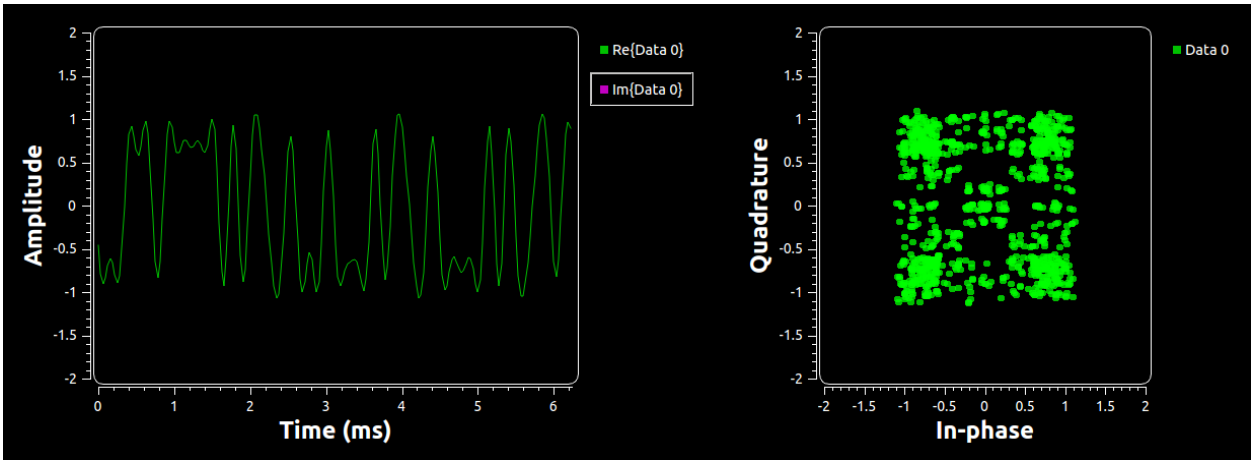

Figure 18: Time Signal and Constellation Plot with no Distortion

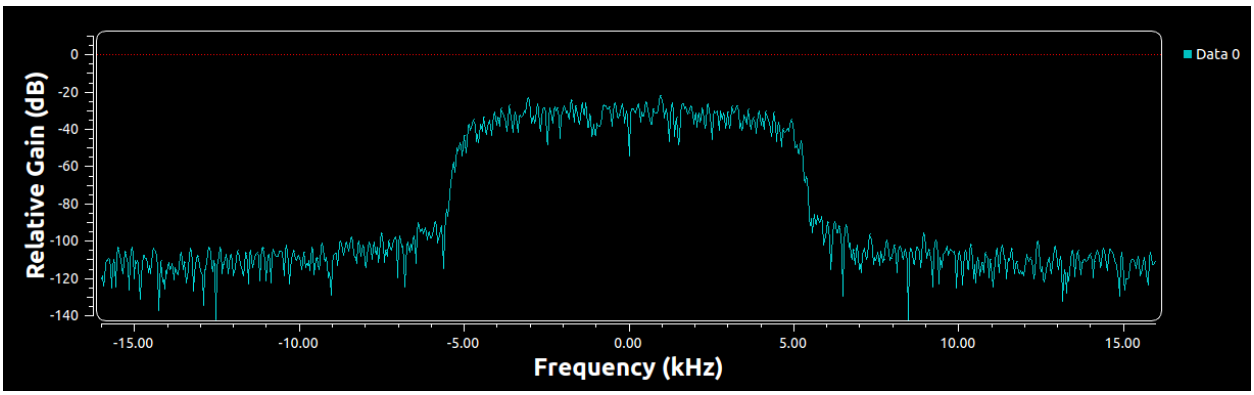

Figure 19: Spectral Plot with no Distortion

Figure 20 and Figure 21 show the effects of additive white Gaussian noise. The time signal is clearly distorted, as well as the constellation plot. The spectral plot is effectively being swallowed by the noise.

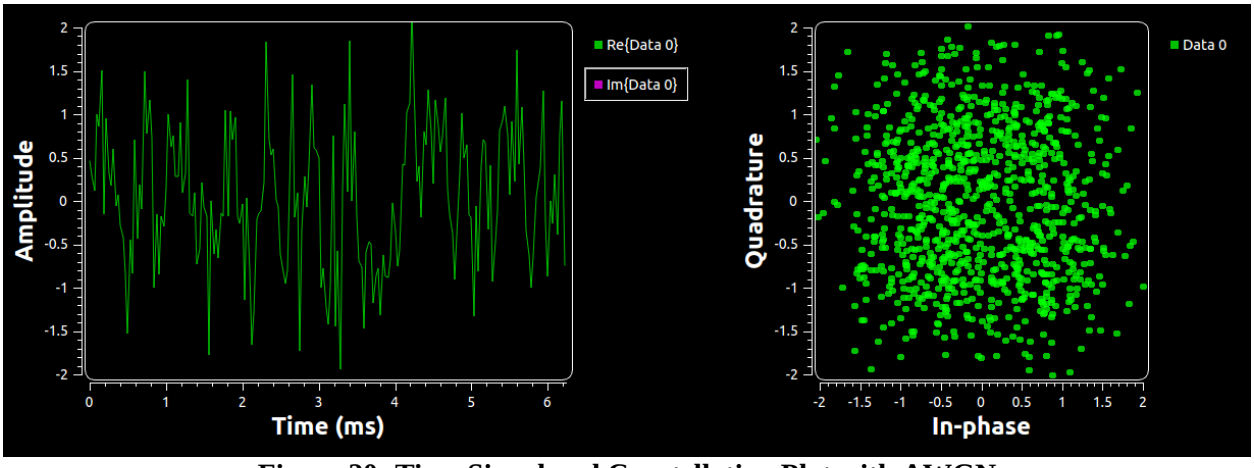

Figure 20: Time Signal and Constellation Plot with AWGN

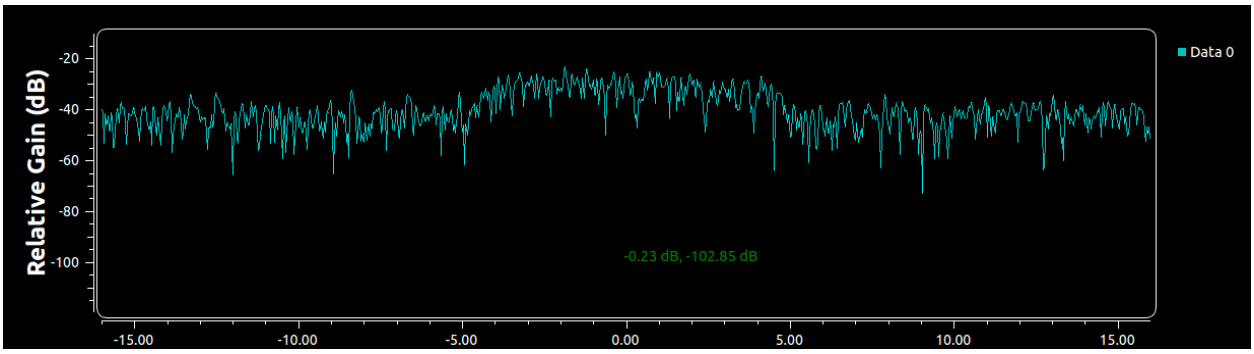

Figure 21: Spectral Plot with AWGN

Figure 22 and Figure 23 show the effects of frequency distortion. The constellation plot is tends to form a circle. The plot takes this shape when there is no timing offset. Therefore, most of the samples fall on the unit circle. Because of the frequency distortion, however, many of the data points move away from the original clusters. The spectral plot simply shifts in frequency.

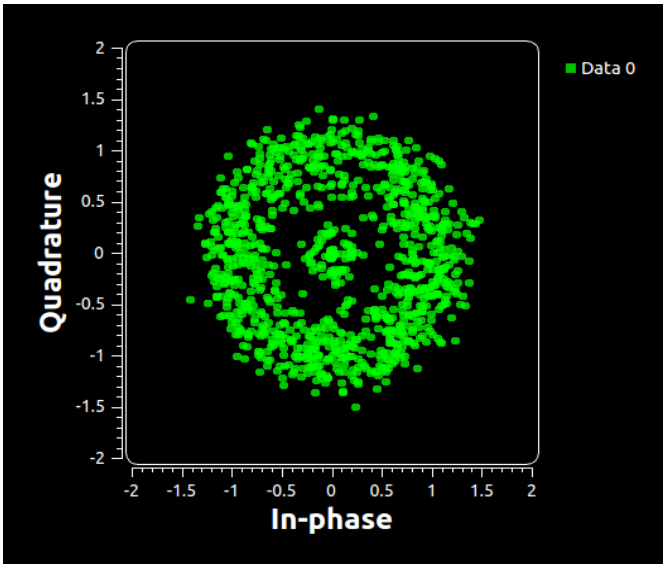

Figure 22: Constellation Plot with Frequency Distortion

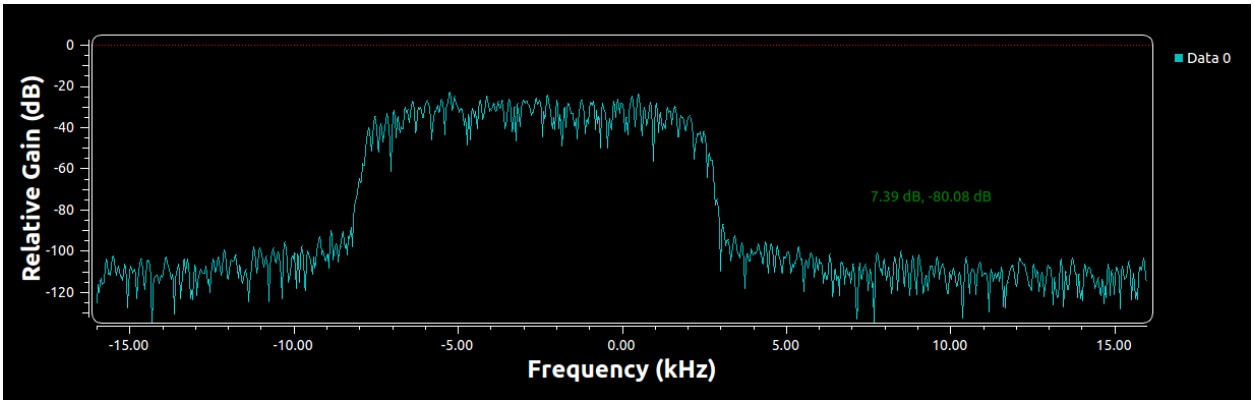

Figure 23: Spectral Plot with Frequency Distortion

Demodulation:

The next step of this project is to create the demodulation sequence, and thus complete this part of the simulation. The first issue to resolve will be timing recovery. The "Poly-phase Clock Sync" processing block will be used for this purpose. This block performs multiple tasks. First, it will achieve clock recovery. Second, it will provide the matched filter needed to remove the ISI. And lastly, it will down-sample the signal to 1 sample per second. Internally, the block uses a poly-phase filter-bank clock recovery technique. The algorithm used calculates the differential of the incoming signal, which is related to the clock offset. This is done with a series of filters, each having a different phase. Eventually, if enough filters are used, one will give the correct phase that has the desired timing. Initially, the block is set to 32 filters. Figure 24 shows the signal flow graph with the addition of the "Poly-phase Clock Sync" processing block. A "QT GUI Range" object was added to dynamically control the timing offset.

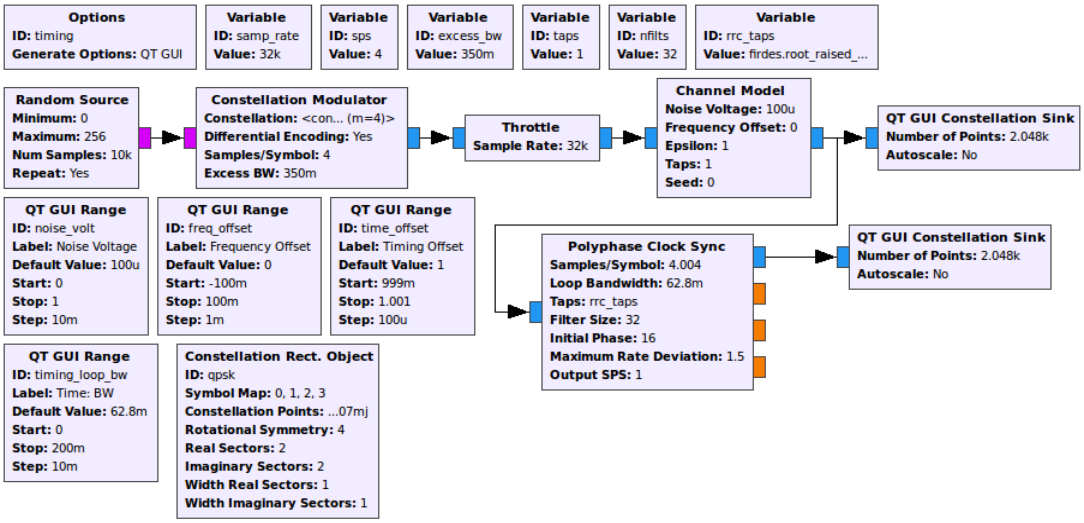

Figure 24: Simulated Timing Recovery

Figure 25 shows the constellation plots before after synchronization. The first plot has been given a very large timing offset. The results resemble a noisy signal. Additionally, the oversampling adds more disarray to the plot. In the second plot not only does the synchronization clean things up, but the down sampling to one sample per symbol gives four distinct clusters that correspond to our four symbols.

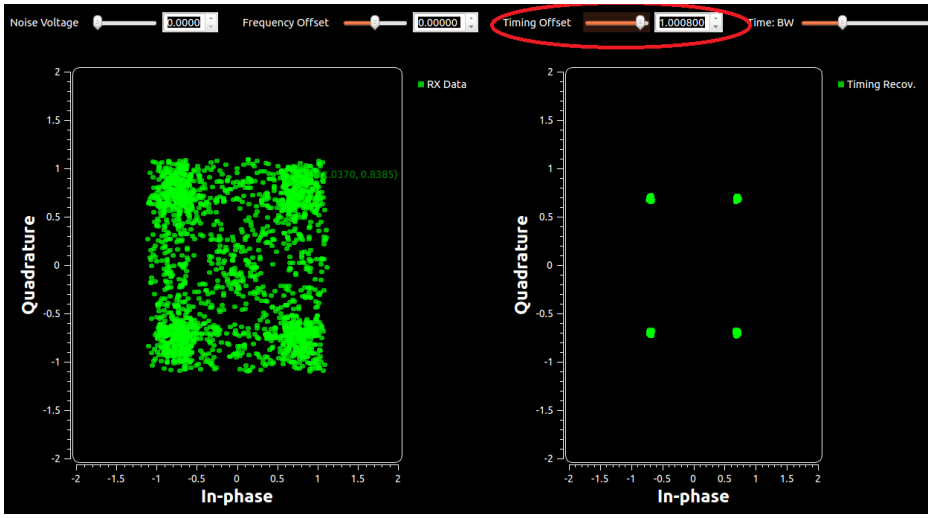

Figure 25: Constellation Plot Before and After Timing Recovery

Next, a Constant Modulus Algorithm (CMA) equalizer will be used to correct multipath distortion. This type of distortion, which is a type of ISI, occurs because most communication environments do not have a single path for the signal to travel from the origin to destination. Surfaces like buildings, walls, trees, etc. all produce reflections. Each of which will show up at the receiver at different times. Summing these together at the receiver causes the multipath distortion. The CMA equalizer will only work on a signal with constant amplitude. Signals such as PSK fit this description. Figure 26 shows the signal flow graph with the addition of the CMA equalizer. A "QT GUI Range" object was added to dynamically control the gain of the equalizer.

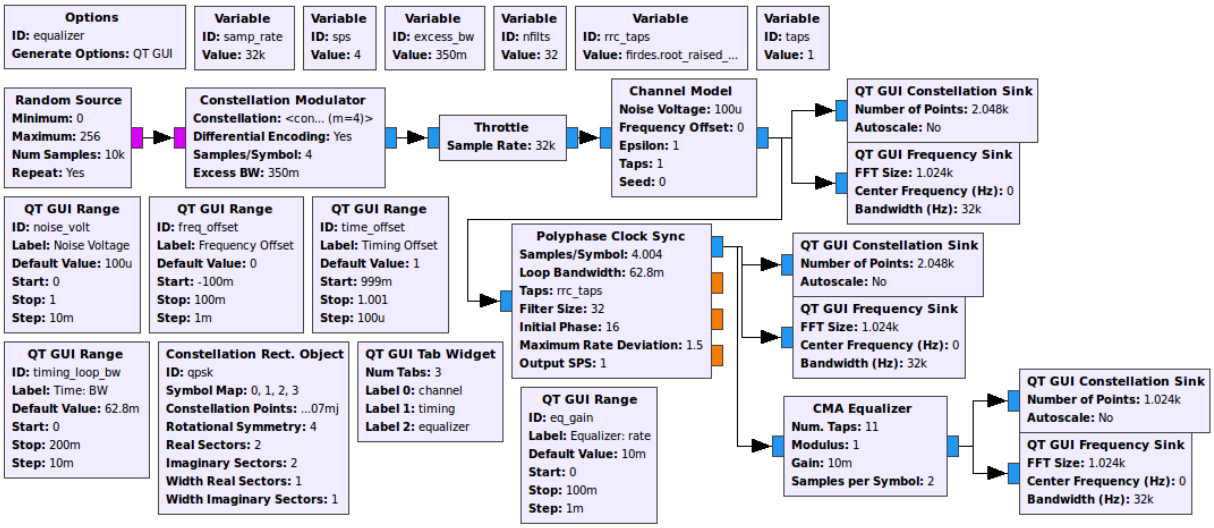

Figure 26: Simulated Equalizer

Figure 27 shows the output constellation and frequency spectrum from the clock synchronization block. The output "samples per symbol" was altered to create the appearance of interference. Figure 28 shows the output of the equalizer plot. Both the constellation and spectral plots are cleaned up. The frequency spectrum in particular has a constant, "equalized" shape.

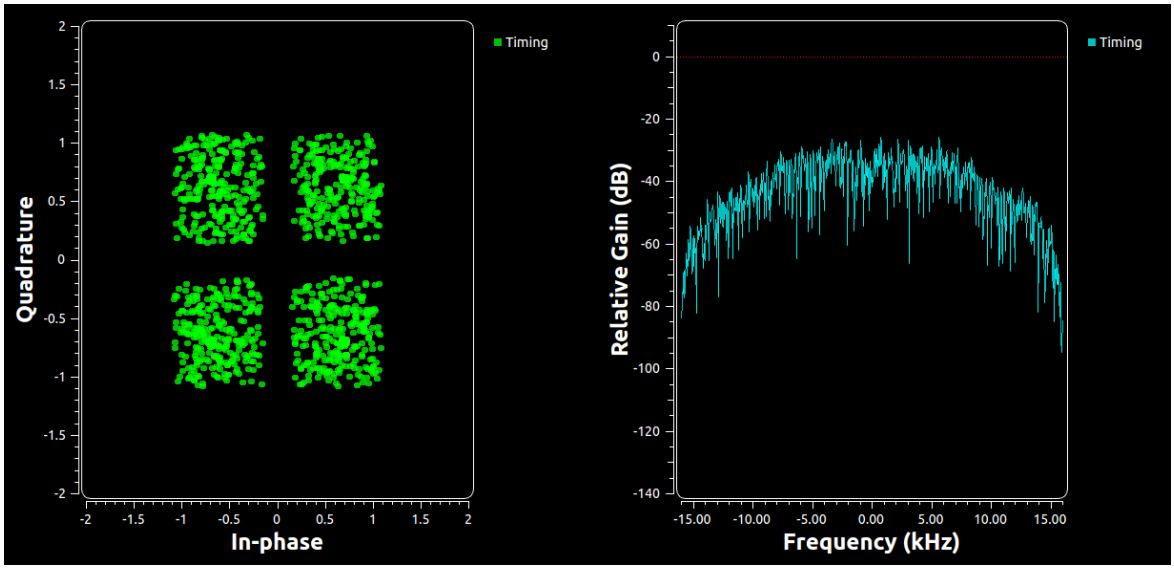

Figure 27: Before Equalization

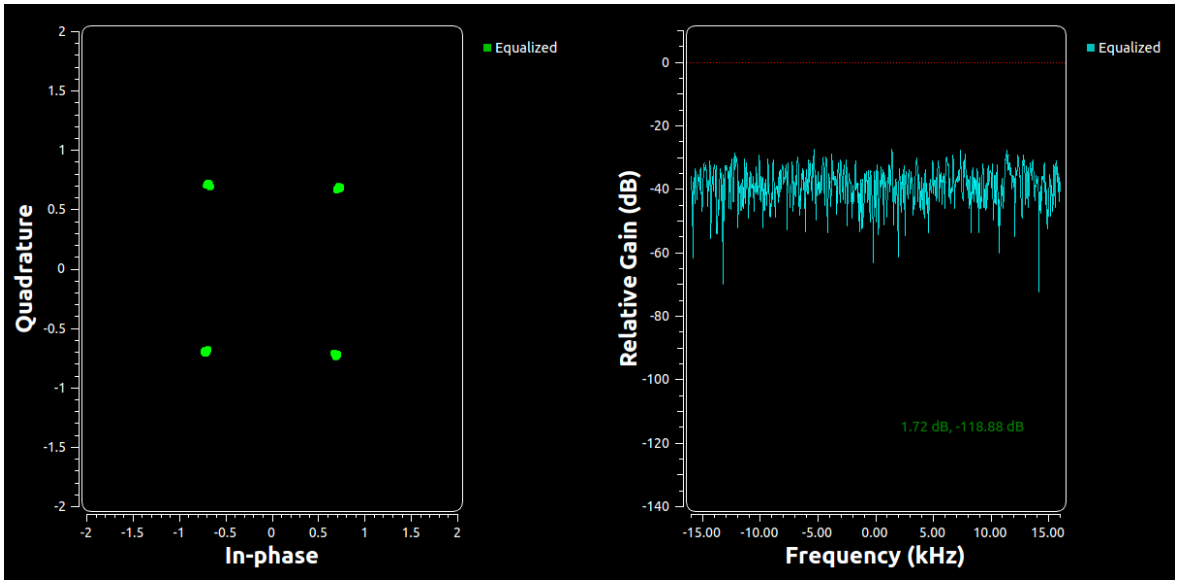

Figure 28: After Equalization

Since the CMA equalizer is only capable of converging to the unit circle, it can lock at any given phase. For this reason, the signal flow graph needs to correct for any phase offset as well as any frequency offset. For this, the "Costas Loop" processing block will be used. The Costas Loop can synchronize to both BPSK and QPSK. Figure 29 shows the signal flow graph with the addition of the Costas loop. A "QT GUI Range" object was added to dynamically control the phase bandwidth.

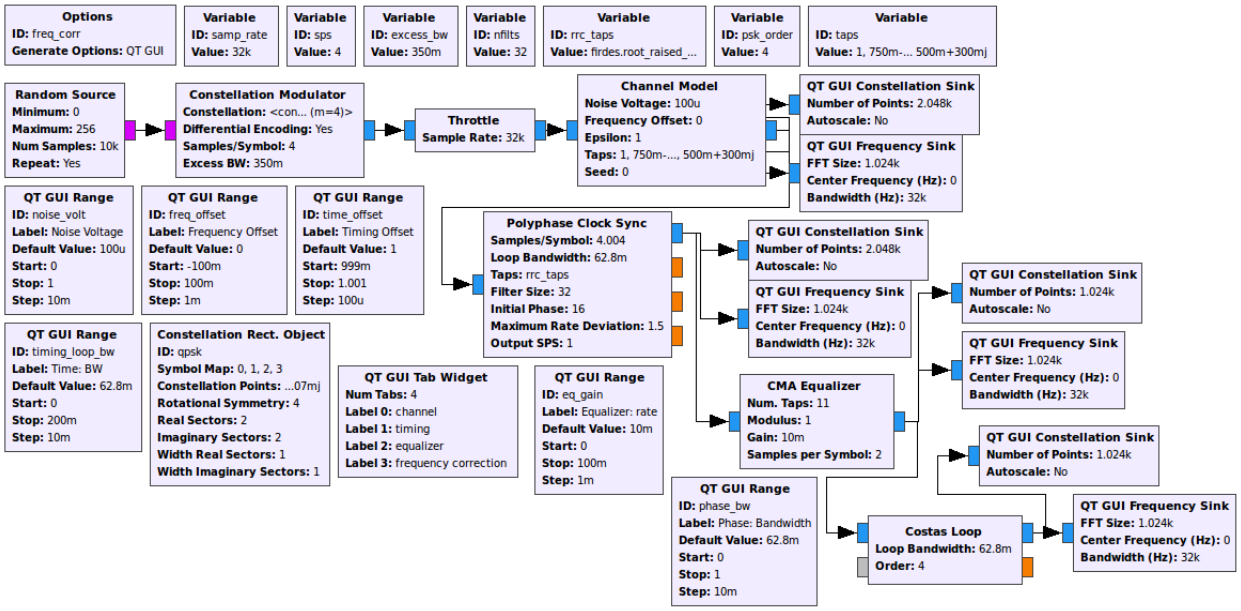

Figure 29: Simulated Frequency Correction

Figure 30 shows the output constellation plots before and after the Costas loop. A frequency offset was dynamically added through the channel model. As the timing block and equalizer cannot compensate for this, the data points accumulate on the unit circle. This is shown in the left plot. The right plot shows the output of the Costas loop. Here the data forms the expected clusters representing the four symbols.

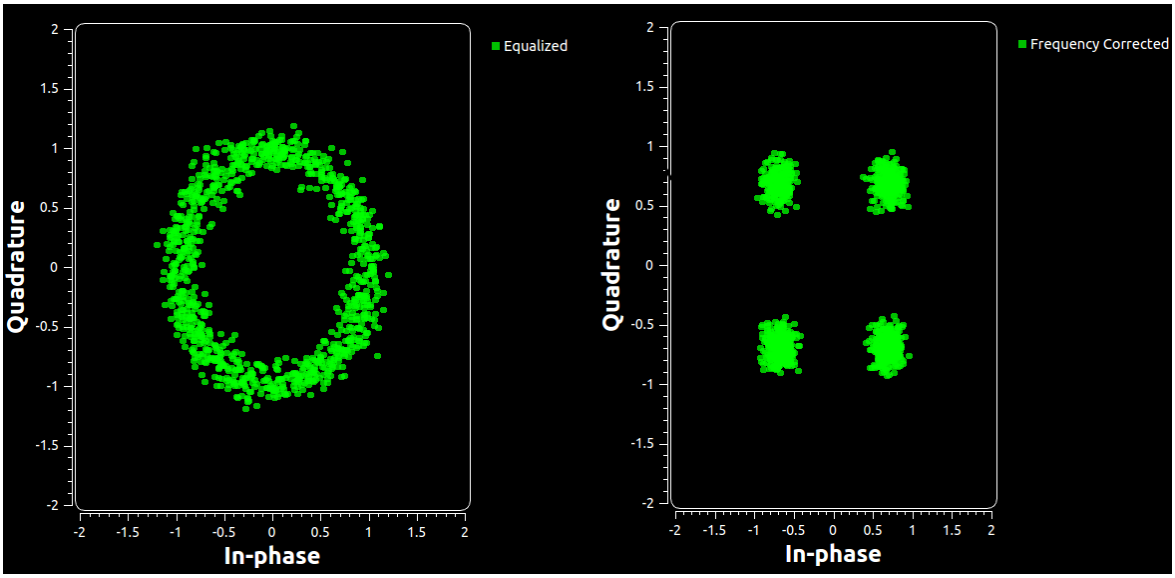

Figure 30: Constellation Plot Before and After Frequency Correction

The next step in the demodulation process was the decoding procedure. In this part, the symbols from the constellation plot are converted back to bits. The signal flow graph is shown in Figure 31. Several processing blocks are used to complete the simulation, although not all will be used in the final flow graph. First, the Costas Loop is followed by a demodulator block. The symbols are then mapped to their binary equivalents. Since differential encoding was used before modulation, differential decoding is used now. The actual constellation was not transmitted. Instead, differential symbols were, and are now converted. Since the symbols represent multiple bits, an "Unpack" processing block is used. The final step is to convert the byte values into floating point values so that the signal can be viewed on a time plot.

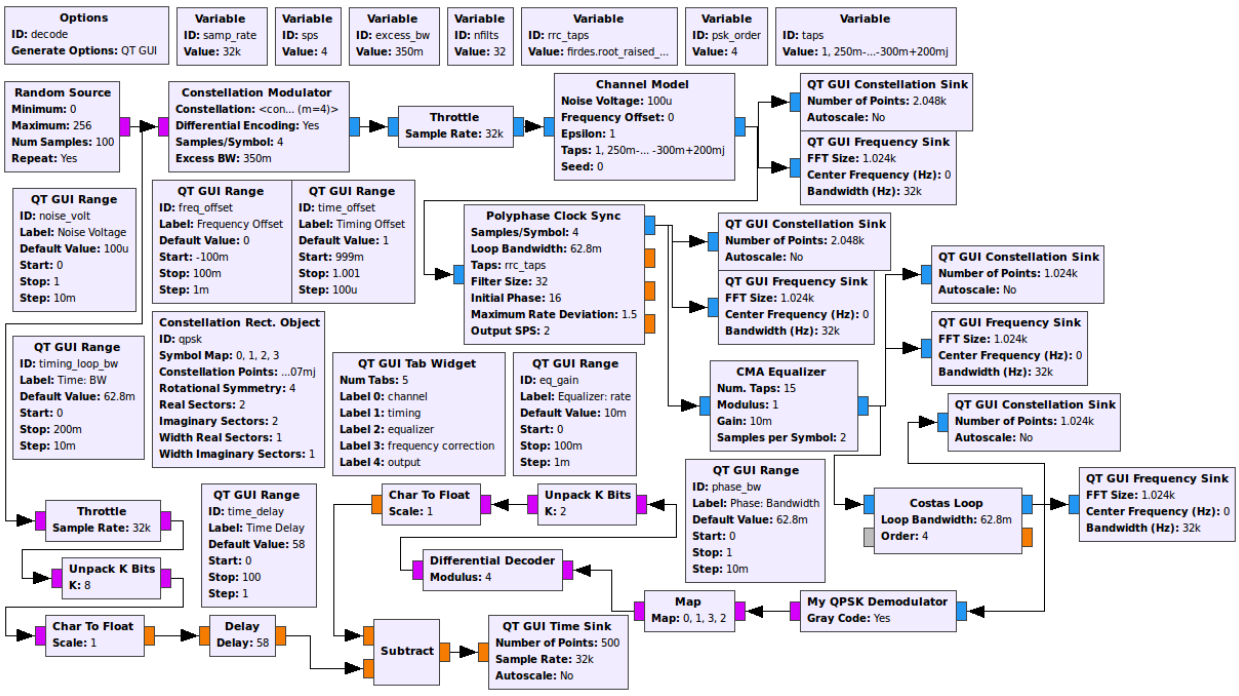

Figure 31: Simulation with Decoding

The output shown in Figures 32 and 33 are the combination of the original signal, with a time delay, subtracted from the decoded output signal. In Figure 32, the output is a constant zero, showing that the input and output signals match exactly. In Figure 33, a significant amount of noise was added to the channel. The resulting signal exhibits numerous errors.

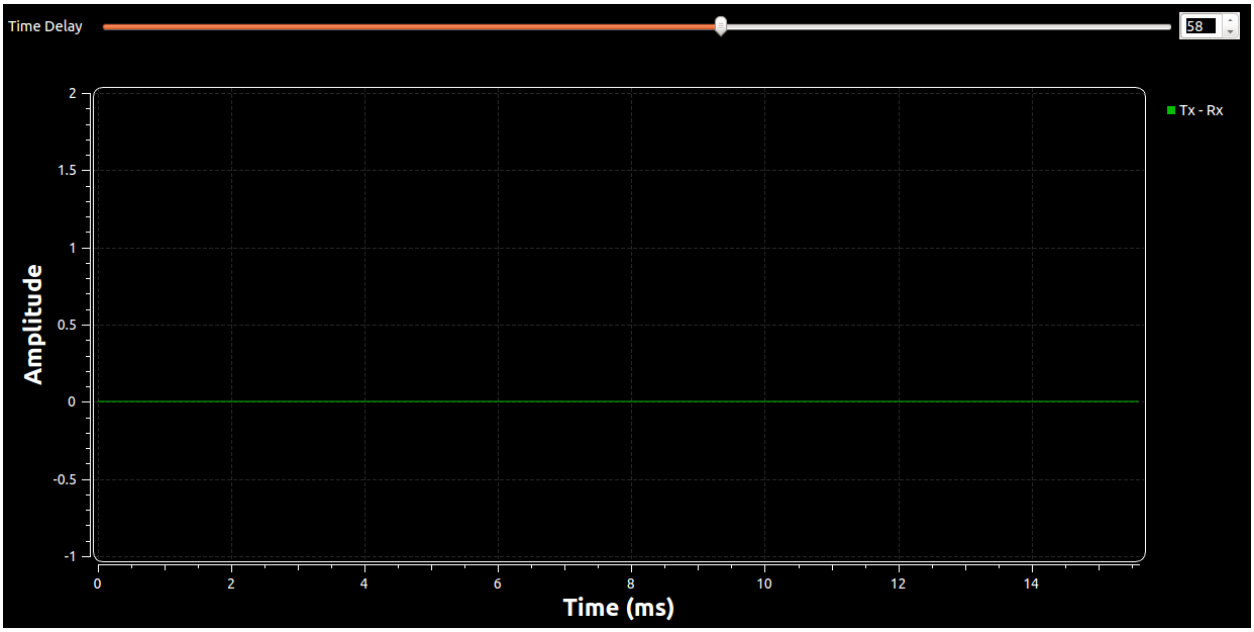

Figure 32: Output Decoded Time Signal

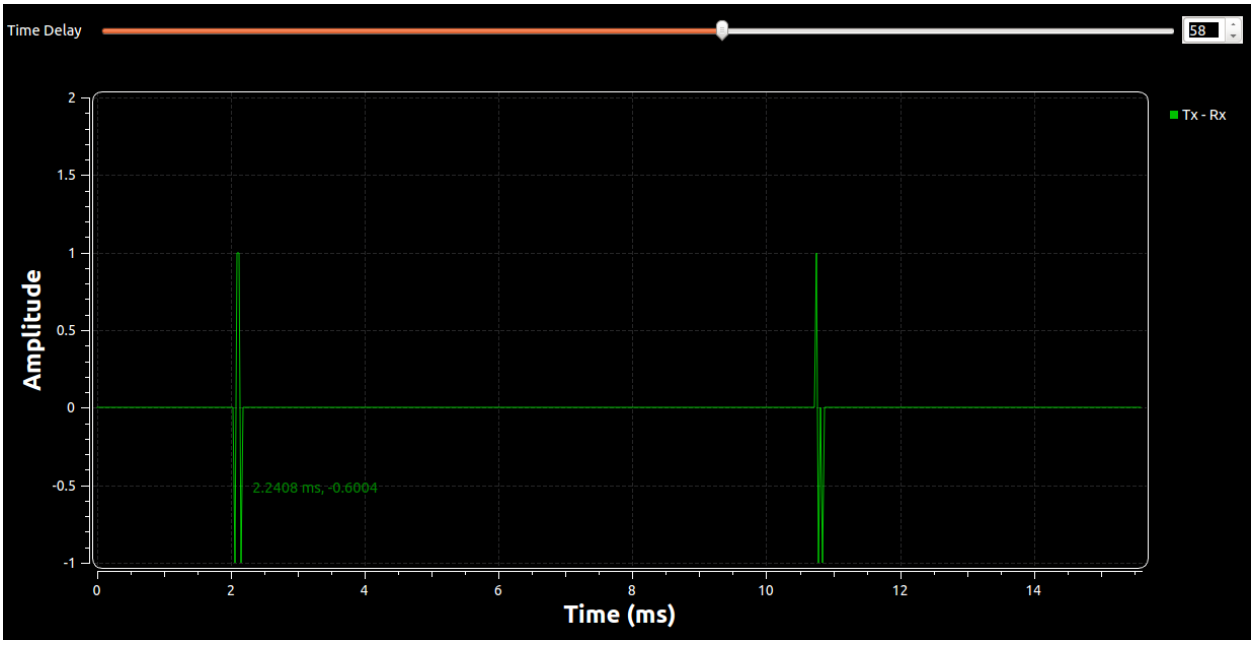

Figure 33: Output Decoded Time Signal with Noise

This confirms that the signal processing works as planned. The next step in the process was to reconstruct the original byte stream. Gnu-radio provides functionality for this purpose. The “Repack”

processing block extracts designated bits from a stream of bytes and reconstructs bytes with 8 valid bits. Figure 34 shows the signal flow graph with this addition. The constellation and spectral plots were removed for simplification. A file source was added to replace the random number generator. Additionally, two oscilloscopes were added for comparison of the input and output streams. The outputs from the added oscilloscopes are shown in Figure 35.

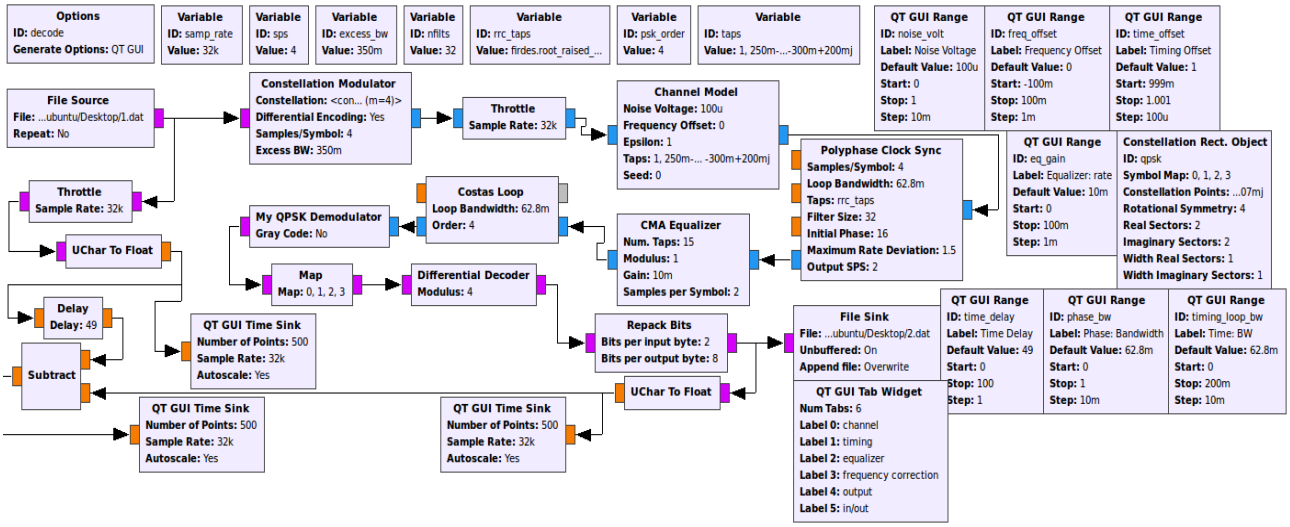

Figure 34: Signal Flow Graph – Byte reconstruction.

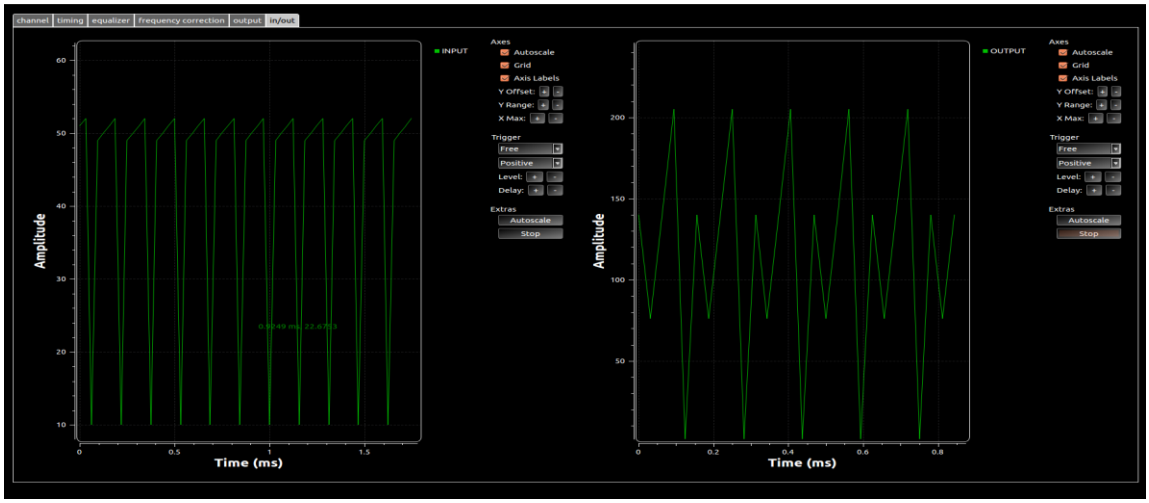

Figure 35: Output of flow graph from Figure 34

As seen in the plots in figure 35, the reconstructed byte streams were not identical. This occurred because the byte streams were misaligned. This issue was due to latency between the demodulator and modulator blocks. In effect, the demodulator does not know when to start. The solution for this issue was the use of "Packet Encoder" and "Packet Decoder" processing blocks. Within the encoder is an attribute referred to as the "preamble." The preamble is a signal used to synchronize the transmission timing. Essentially, it is a warning, that tells the receiver that data is about to be transmitted. Figure 36 shows the updated simulation with the packet encoder and packet decoder included.

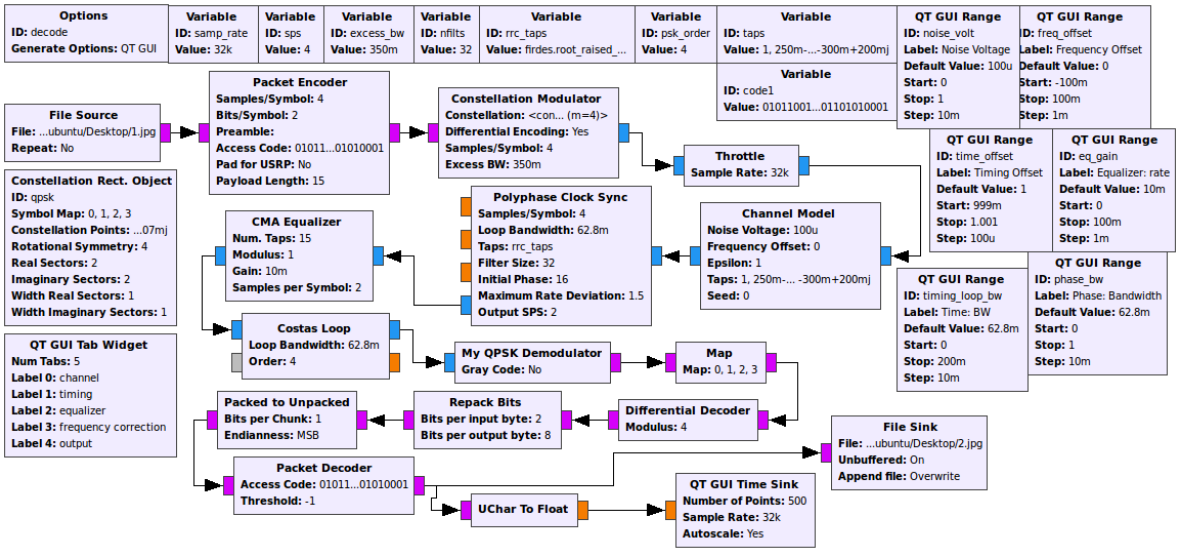

Figure 36: Signal Flow Graph – Completed Simulation

For the preamble, the field was left blank. In this case, a default value is created. A "Payload Length" attribute is also included in the encoder block. This value represents the number of bytes included in each packet. It was found that for the small text files used previously, a value of 1 was appropriate. Larger values caused a failure in transmission. For the larger image files, a larger value worked better. Figure 37 shows an image file in mid transmission. Since our simulation was successful, error correction was the next step.

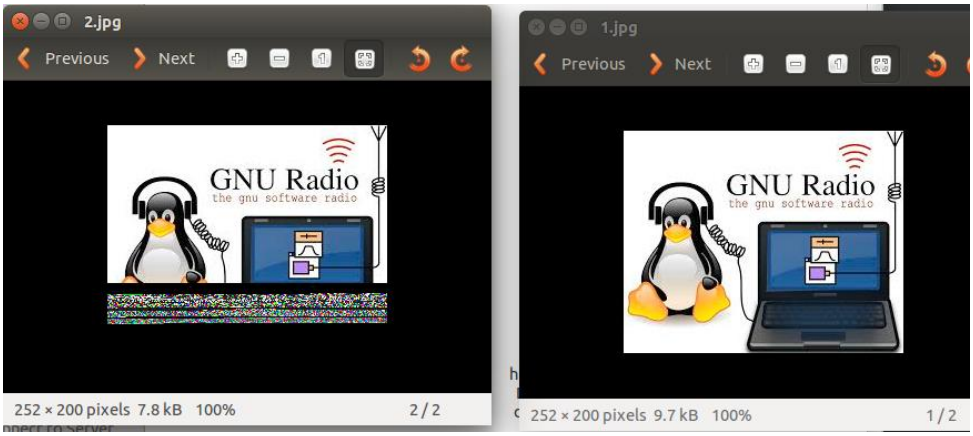

Figure 37: Simulated Image Transfer

Error Correction:

For this project, Forward Error Correction was the chosen method. As stated previously, FEC is a method of adding redundancy to the bit stream to allow the receiver to detect and correct errors. Figure 38 shows the final signal flow graph with the added FEC. The FEC encoding was inserted directly on the output of the file source prior to the packet encoder. The decoding, likewise, was placed on the input of the file sink, after the packet decoder. Since the FEC encoders expect unpacked bytes in and unpacked bytes out, an "Unpack" block was used on its input. Additionally, the FEC decoders expect a floating point stream for their input. For this reason, the output of the FEC encoder was mapped to values of -1 and 1. Also, the input to the FEC decoder was converted to floating

points. The output of the FEC decoder would obviously need to be repacked into a form readable by the file sink. The encoding and decoding blocks also perform the necessary puncturing and de-puncturing operations. Puncturing is the act of removing some of the parity bits after the encoding process.

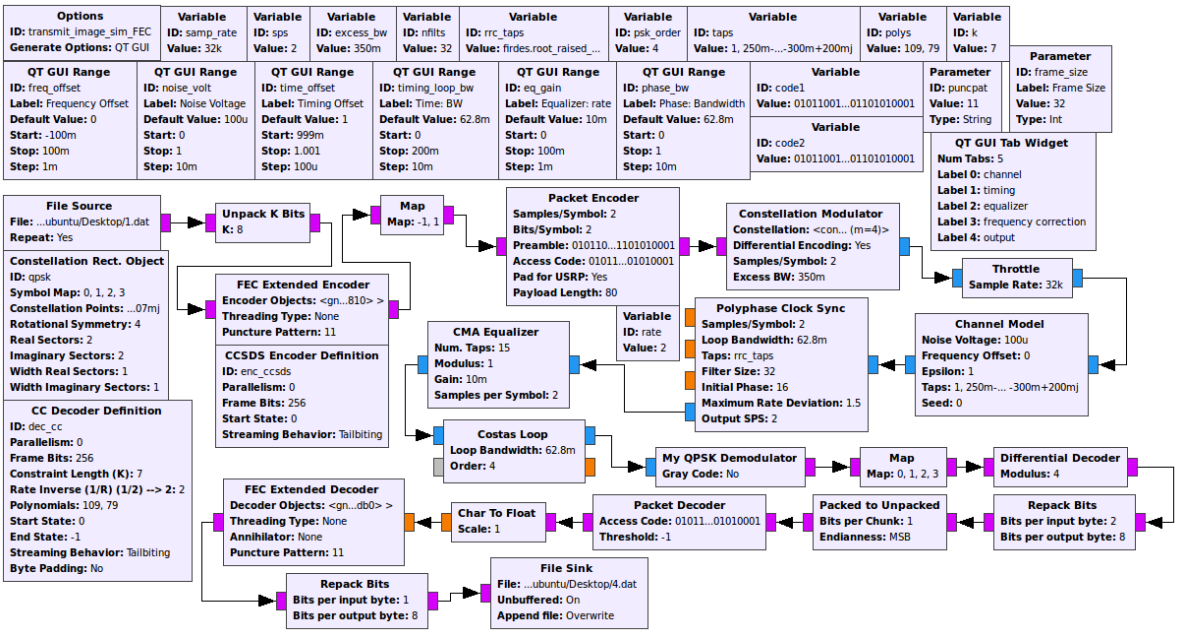

Figure 38: Simulation with FEC

The type of error correction chosen was CCSDS. This is a type of convolutional code. It generates parity symbols through the sliding application of a Boolean polynomial function to a data stream. The sliding portion represents the 'convolution' of the encoder over the data. Additionally, a “tail-biting” behavior was chosen. This involves adding bits onto the end of each packet. It essentially continues the stream between the packets by pre-initializing the state of the new packet based on the state of the last packet. Figure 39 shows a comparison of the input and output data streams with FEC applied.

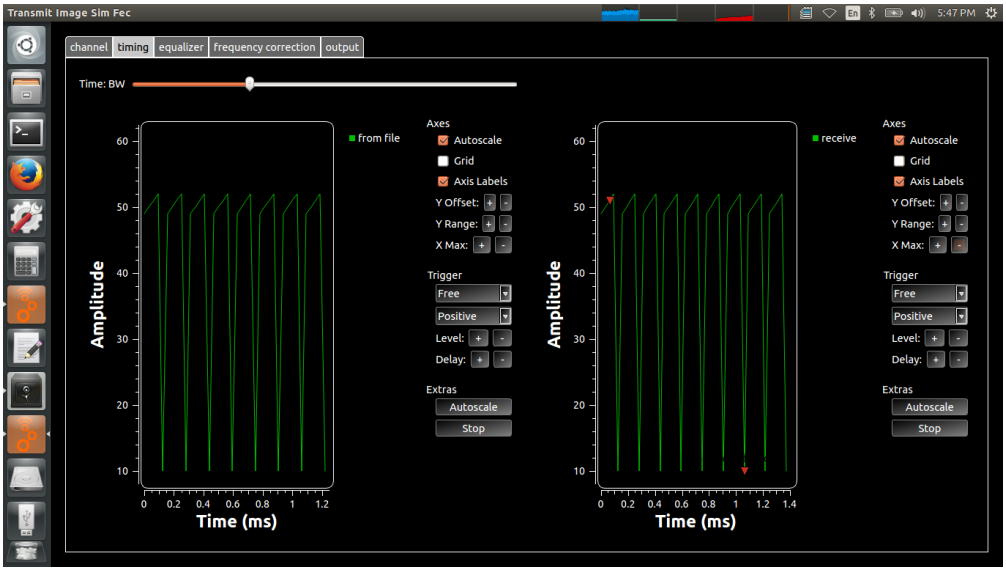

Figure 39: Simulation Input and Output Streams with FEC

Image Transfer - Hardware

For the hardware portion, our simulation flow graph was broken down between transmission side and receiver side. Figure 40 and Figure 41 show the flow graphs.

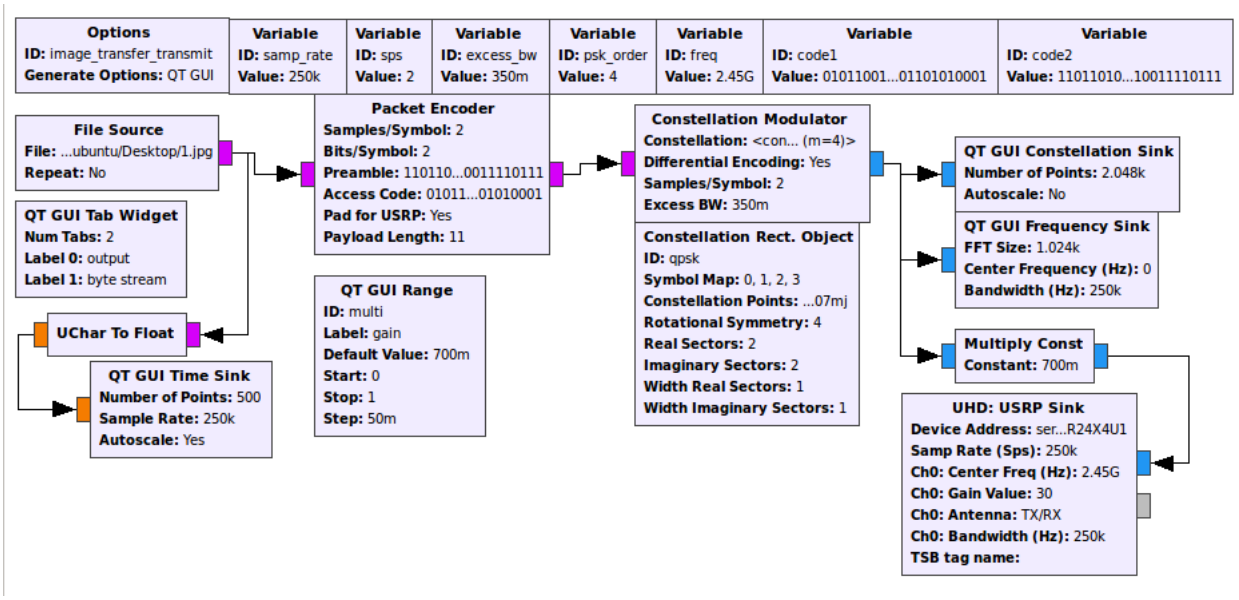

Figure 40: Transmitter Flow Graph

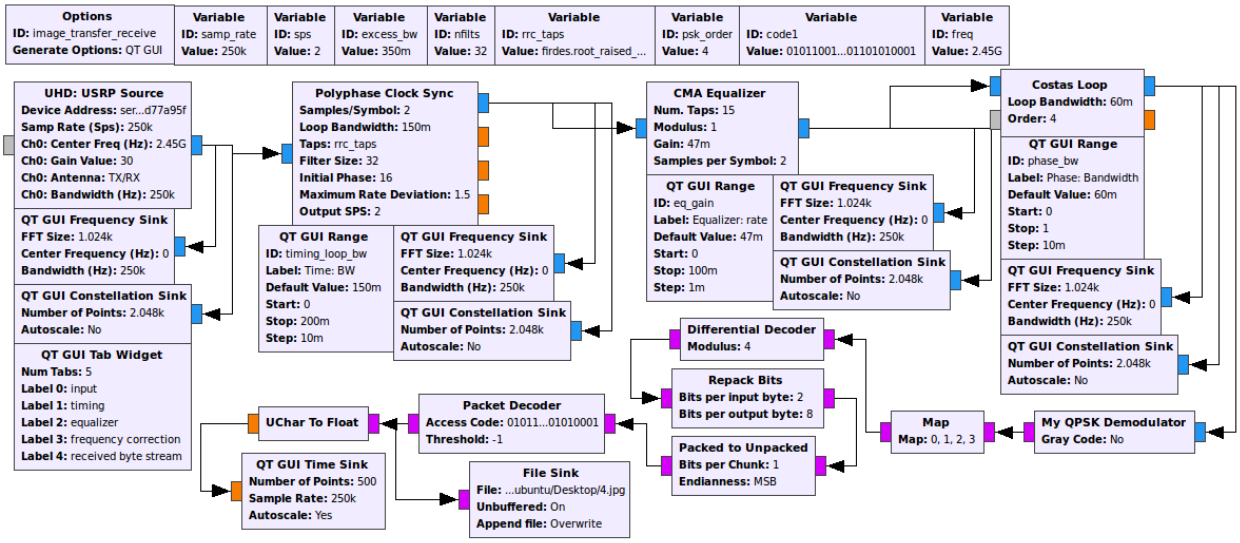

Figure 41: Receiver Flow Graph

Since this is no longer a simulation, the throttle, channel model, and all related aspects were removed. A USRP sink was added to the transmit side. A dynamically controlled "Multiply Constant" processing block was added just prior to the output to control the gain of the signal. Too large of a signal can saturate hardware components and thus distort the signal. Additionally, a USRP source was added to the receive side. For the transmit portion, we show a constellation and spectral plot of the modulated output. Also, the output of the file source is converted to floating point numbers and shown on an oscilloscope. On the receive side, constellation and spectral plots are shown at each stage of the demodulation process.

This will enable the necessary adjustments for the channel conditions. Lastly, the output is sent to a file sink and is also converted to a floating point stream and sent to an oscilloscope.

Image Transfer – Results and Conclusions

The first test of the transmit/receive flow graphs used two USRP devices, a loop back cable connecting them, and one laptop PC running both processes. Figure 42 shows the spectral and constellation plots for both transmit and receive using QPSK. Figure 43 shows the same using BPSK. The plots on the left correspond to the output of the transmit process. The plots on the right correspond to the output of the Costas Loop on the receive side.

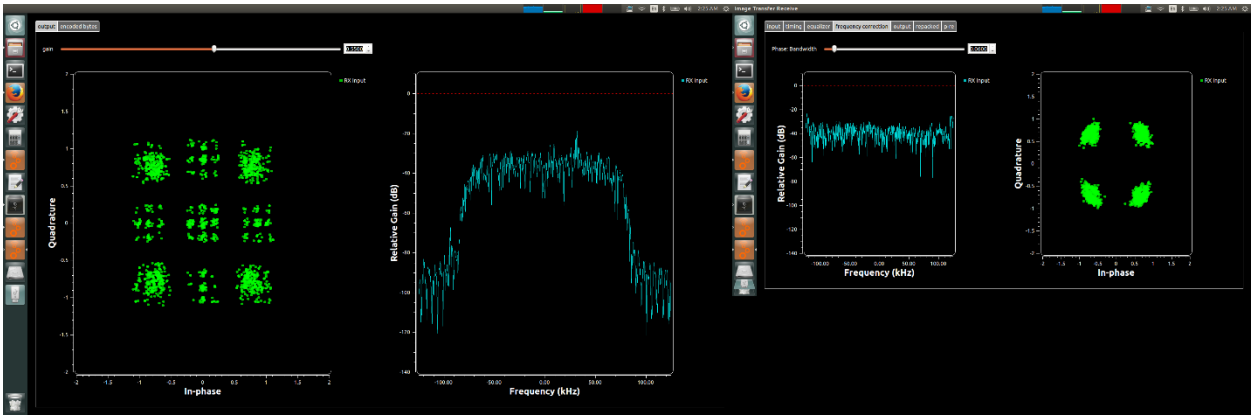

Figure 42: QPSK Transmit and Receive plots using a loop back cable

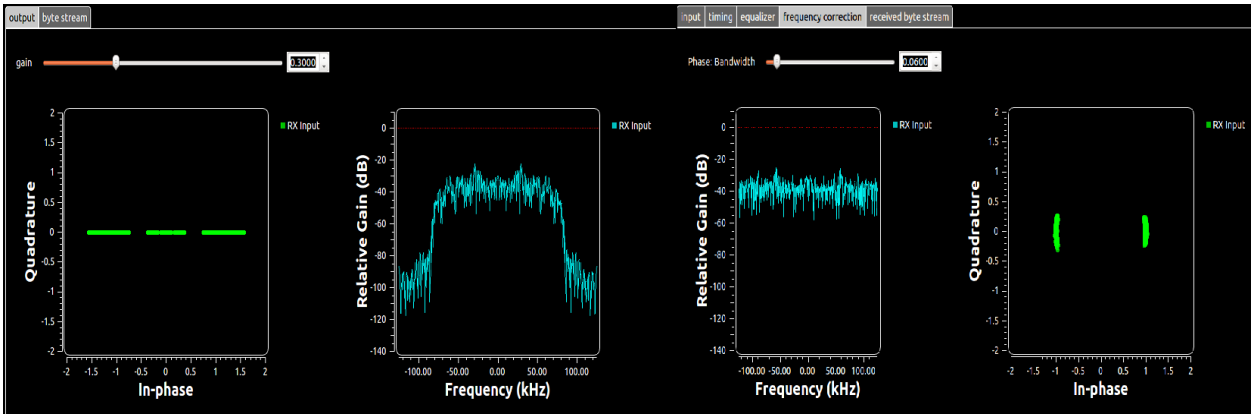

Figure 43: BPSK Transmit and Receive plots using a loop back cable

Image transfer with the loop back cable occurred without errors. The constellation plots on the receive side in Figure 42 and Figure 43 show distinct clusters. It is clear that the signal processing is correct. After removing the loop back cable and using the antennas, some problems occurred. The primary issue seen was during the initial transmission. When the signal is first transmitted, a noise burst occurs as the signal is being received. This has the effect of corrupting the initial bytes transferred. This causes a serious issue with image file transfer. The first bytes in the file represent the file header. If this is corrupted, the file will not open. Even when the rest of the file is transferred correctly, a corrupted header causes any image viewer to reject it. A partial solution to this was in the pre-amble of the packet encoder. By increasing its length, it caused the receiver to delay its starting point. Although it didn't completely fix the issue, it improved from about 10% to about 75% success ratio.

Also, the Forward Error Correction was found to ineffective. As the devices were moved further apart, they were more susceptible to burst errors. FEC is not the proper solution for this. A method that involves the interleaving of data would be more appropriate. In conclusion, we have successfully transmitted our image data at short distances, but failed to realize the severity of the noise distortion.