Pennsylvania State University

World Campus

# EE 460 - Communications Systems II

Dr. Tofighi

*256-QAM, SDR Applications, And Viterbi Error Correcting*

*Author*

Chase Timmins

December 18, 2020

# Contents

## List of Figures

## 1 Purpose

Quadrature Amplitude Modulation (QAM) is a digital communication scheme that uses a singular carrier frequency with the addition of a Hilbert Transformation to accommodate a separate orthogonal carrier. This accomplishes a low-cost and less complex implementation of OFDM (Orthogonal Frequency-Domain Multiplexing) with the use of a single carrier that can be considered two carriers. QAM is a very popular digital modulation technique due to it's minimal hardware requirements and relative scalability which makes it ideal for small-scale, high-speed communication systems. It is popular for high speed modem applications as well as small-scale cellular applications. According to Fierce Wireless, a publishing company that specifically monitors the wireless industry, 256-QAM is also gaining popularity for macro-model communication or communication on the larger scale specifically due to the high-data rates and increased spectral efficiency (Alleven [2016]).

The purpose of this paper will to cover the general scheme and mechanisms behind Quadrature Amplitude Modulation as well as the setup and results of creating a 256-QAM implementation using a Pluto SDR by Analog Devices. The use of 256-QAM on its own will be compared to the use of 256-QAM with the Viterbi Algorithm, an error correcting algorithm, which will be discussed in the later sections. In addition, the use of error detection codes and algorithms will be discussed and comment on their use with QAM will be provided . This paper will also provide a proof of concept in a simulation using Python and Matlab.

# 2 Introduction

## 2.1 Quadrature Amplitude and I-Q Modulation

QAM is a single-carrier modulation technique that acts effectively as dual carrier modulation. To be specific, the single carrier undergoes a Hilbert transformation creating a 90° separation between the initial carrier and the created carrier, which is coincidentally orthogonal[1] to the original carrier when demodulation is concerned.

The initial concept that was used for the basis of QAM is known as I-Q Modulation which strictly stands for In-phase and Quadrature Modulation. I-Q Modulation was developed to be a bandwidth-cheap option to increase data rates or send separate message signals on adjacent frequencies. An example of this is present in Figure 1 showing the general hardware diagram for a super-heterodyne implementation of IQ Modulation.

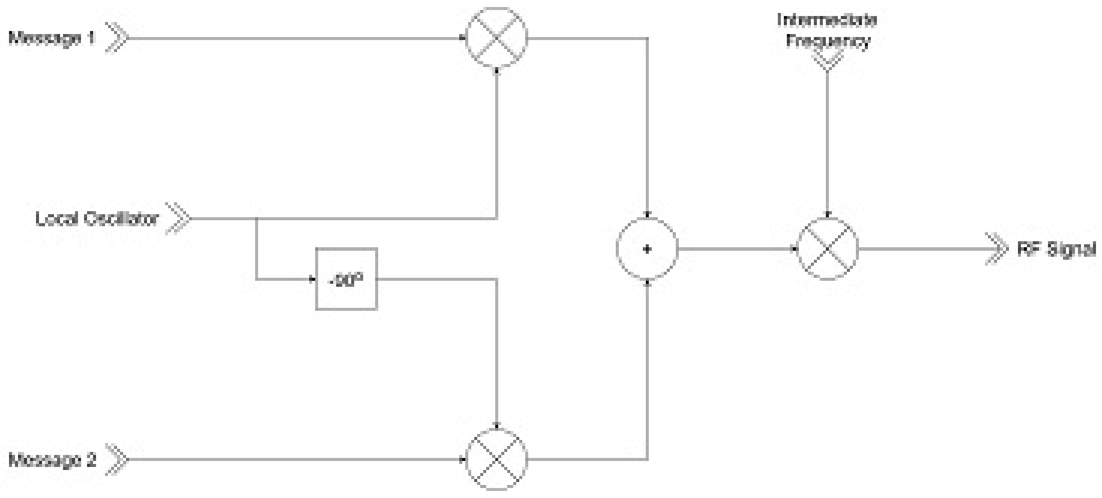


Figure 1. Super-Heterodyne IQ Modulation Hardware Diagram

An example of a strict I-Q Modulator can be see in Figure 2 below:

[1]A signal when demodulated by an orthogonal signal results in a reduced or a zero signal

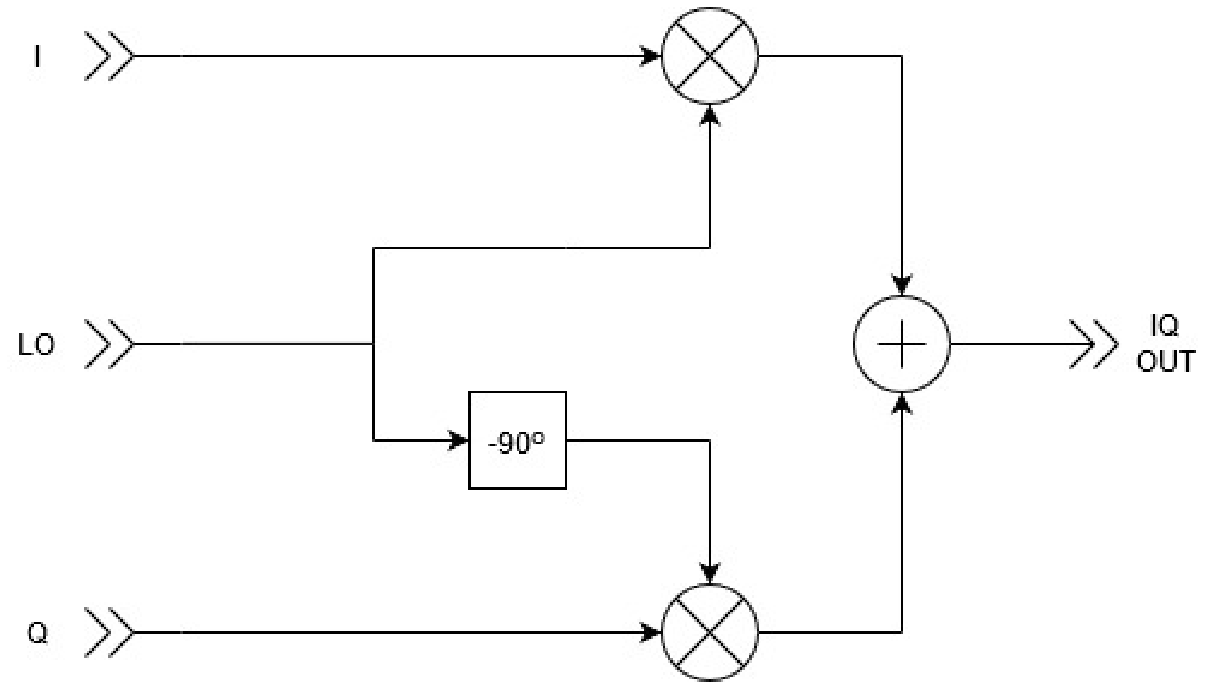


Figure 2. I-Q Modulator Hardware Diagram

I-Q Modulation was developed for standard RF Communication, not necessarily digital, to effectively double the data rate of a given signal without increasing its bandwidth. This meant that at least twice as much data could be sent at a given time by a system using this technique, which requires relatively minimal hardware compared to those that achieve a similar feat. For example, Transmitting an audio signal via AM while trying to separate the left and the right channels can be done by implementing single sideband modulation for each channel resulting in one channel occupying the upper sideband and the other channel occupying the lower sideband. This method of communication requires more hardware which can cause issues regarding signal integrity via cross-talk and other noise related issues from the added components. The modulator / demodulator combination will be more costly as a result, either in terms of signal integrity or fiscally if the noise issue is to be mitigated by other means. I-Q Modulation satisfies the same requirements as this method of communication with reduced hardware and reduced overall cost.

Quadrature Amplitude Modulation further simplifies the model of I-Q modulation by simplifying the message signals to be discrete-time and discrete valued signals correlating to different hexadecimal values depending on the level of QAM that is used. QAM is considered a digital communication scheme, hence the Amplitude Modulation will depend on the binary values being sent. A general system diagram of how QAM can look can be seen in Figure 3.

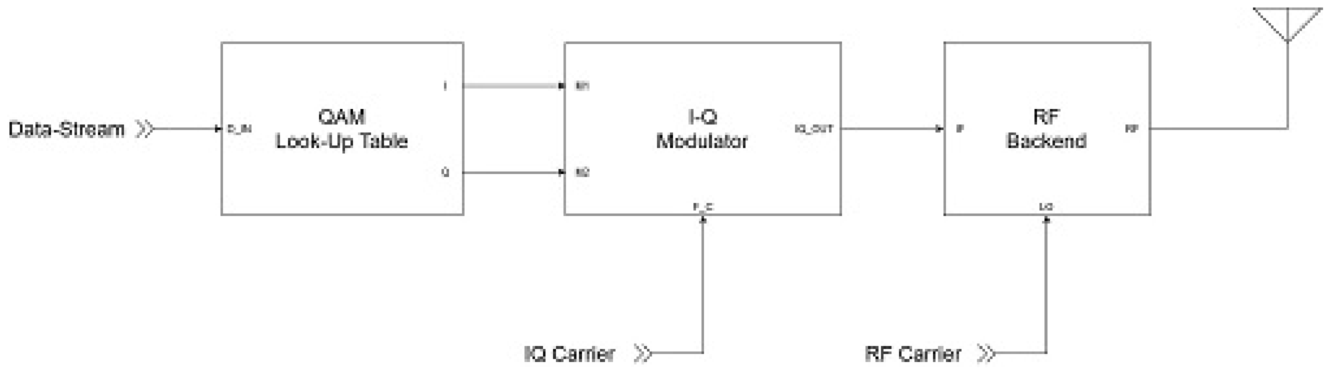


Figure 3. Quadrature Amplitude Modulation Block Diagram

The QAM Look-Up table is more easily comprehended as a constellation diagram seen in Figure 4. Each input byte from ’0x00’ to ’0xFF’ maps to its own point in the constellation diagram which represents a In-Phase and a Quadrature amplitude to be relayed to its given I-Q Modulator. These amplitude quantities are represented by ’I’ for ’In Phase’ and ’Q’ for ’Quadrature’ in Figure 3.

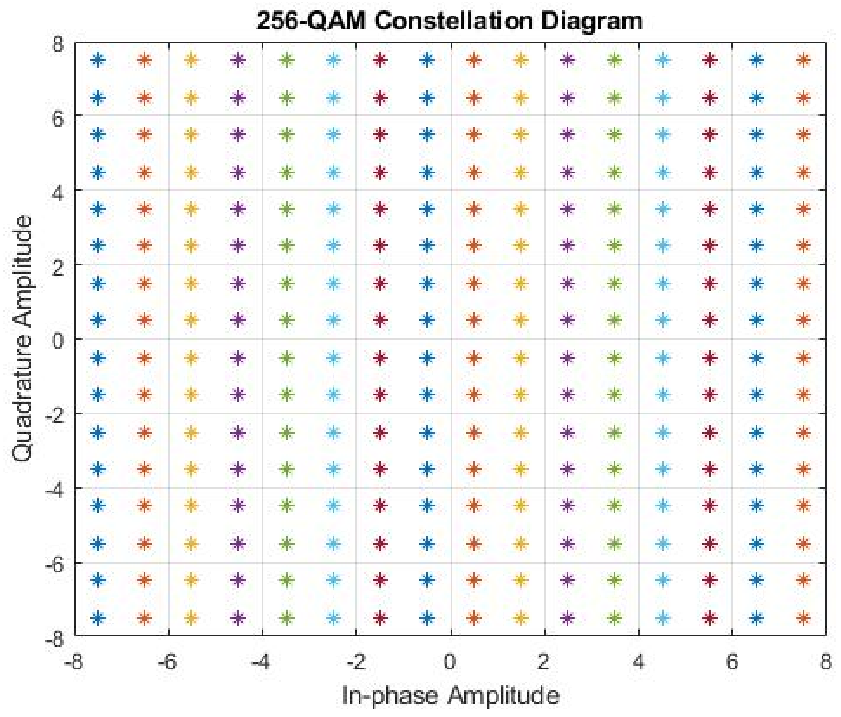


Figure 4. Constellation Diagram for 256-QAM Communication Scheme

## 2.2 QAM and Error-Correction

One major reason that digital communication systems are more popular than their analog predecessors is their compatibility with error-detecting and error-correcting codes or algorithms. Since the output of a QAM Demodulator can feed directly into a Microprocessor or an FPGA, there are a multitude of algorithms that could be used to detect and correct errors. One popular algorithm is known as the Viterbi algorithm which will be discussed in the next subsection.

Error-detecting codes and algorithms tend to rely on certain properties of the data being sent that are computed and usually added to the signal as additional bits. Three of the more common examples of error-detecting codes are Parity Check, Checksum, Cyclical Redundancy Checking.

### 2.2.1 Parity Check

Parity Check is the simplest of the listed error detecting codes. A Parity Checking codes essentially checks to see if the sent message contains either even or odd parity[2] and will add an additional bit known as the 'parity' to the end. The parity is usually implemented such that the message signal ends up with even parity. The major pro of using Parity Check is its simplicity. If the transmission occurs using digital hardware, for instance between the FPGAs of Microprocessors of two different Actuation Control Module in an aircraft, and if a platform of use is used where noise the floor is relatively low when compared to wireless transmission, the Parity Check error detection code is reasonable choice for digital communication. Its draw back is that the code as a potential to not correctly detect errors in noisy environments. The Parity Check error code only works assuming that an odd number of bits are corrupted in the transmission of the message, not to mention that the parity bit itself can get corrupted in transmission. This means that the reliability of the Parity Check error detection method is only as reliable as the Bit Error Ratio (BER).

[2]Parity, in this context means the number of 1's in a given binary message; even parity implies an even number of 1's

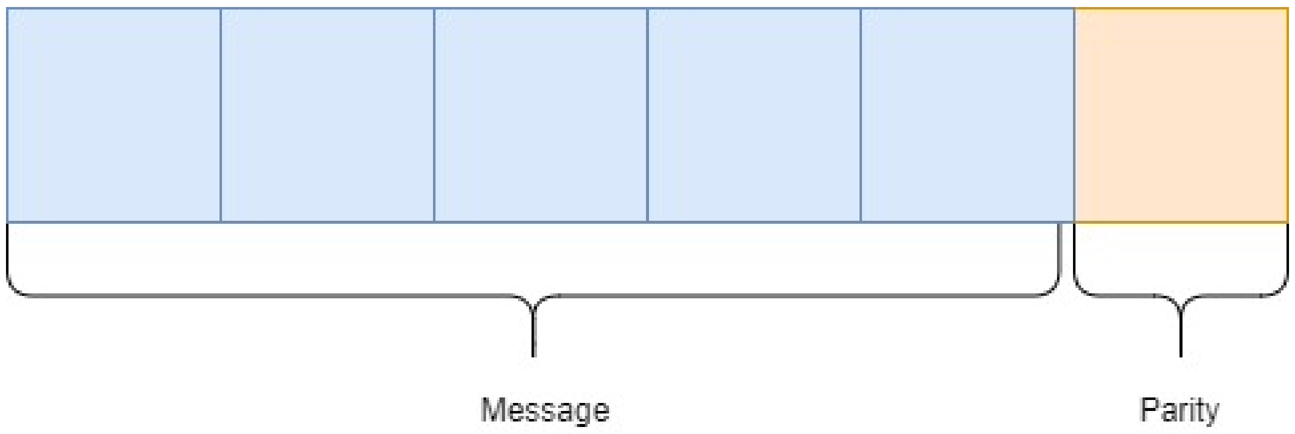


Figure 5. Parity Bit as Portion of Sent Message

Parity check will be the only error detection code that will be discussed at length in this paper as it is the most easily understood. The demonstration of the shortcomings of Parity Check can be seen in Figure 7 and Figure 8 as well as the accompanying write up. The code for the simulation can be seen in Appendix A

### 2.2.2 Checksum

Checksum error-detection is implemented by having the transmit station and receive station both having knowledge of an agreed-upon divisor. The bits that are added to the message, also known as the checksum, are calculated by determining the remainder after the message signal is divided by the divisor. The checksum is usually added to the back end of the initial message and when the receive station receives the signal it can divide it by the known divisor and see whether the remainder is the same as the checksum.

### 2.2.3 Cyclical Redundancy Check

Cyclical redundancy checking (CRC) utilizes checksum, however the divisor is chosen via mathematical means (Koopman and Chakravarty [2004]) such that the CRC is very hard to fool. When developing a CRC divisor, a metric known as the Hamming Distance, the minimum number of bits that would misclassify a message. Specific divisors are chosen for different bit lengths such that the Hamming Distance is maximized and a myriad of different divisors, also called generator polynomials in this context can be found at the CRC Wikipedia Page (Wikipedia). Since referenced by the CRC Wikipedia Page and further supported by their references, specifically ETSI [1999] provided major credibility to the fact that a specific implementation of CRC known as "CRC-14-GSM", which is also known as the "GSM Standard" at least in Europe, was required to be implemented as a layer of channel encoding. A similar implementation of CRC would have yielded positive results when used in this system. An implementation was not actually implemented, however it was necessary to state that error-detecting codes can be implemented and remain effective in wireless communication systems such as 256-QAM.

### 2.2.4 Viterbi and Hidden Markov Models

**Note for this section:** *When the "Viterbi Algorithm" is referred to in a technical context (not historical or introductory), it refers to the author's implementation of the Viterbi Algorithm. It is not a statement as to the exact nature of the Viterbi Algorithm itself, but rather the author's implementation of it with a statistical approach.*

There are many error-correcting codes and algorithms present that range from time-efficient to high accuracy, there is only one discussed in this paper and it is known as the Viterbi Algorithm. The Viterbi Algorithm is quite versatile and has extensive use when related to digital communication system. This is specifically due to the implementation of Viterbi as a maximum likelihood sequence algorithm, which is essentially the implementation that will be observed and discussed in this paper.

The workings of the Viterbi Algorithm were first discovered and coined by Andrew Viterbi in 1967 as a decoder for convolutional codes in noisy digital communication systems (Moon [2005]), though its initial reception was not well received for this use. It instead found its early uses in Natural Speech Processing in 1987. Once technology could sustain high-intensity algorithms in parallel with data collection, the Viterbi Algorithm was then appreciated for its applications in digital communications.

The key mechanisms behind the Viterbi Algorithm are known as Hidden Markov Models. Hidden Markov Models (HMM) use principles of arbitrary states and use the statistics of state-transitions to find forbidden state transitions and relevant information. Once the user defines what these hidden states might be and what defines a transition, the user can then use Hidden Markov Models to develop relevant data regarding their subject matter. The catch with HMM is that the state transitions need to be direct (Eddy [2004]). HMM's prove to be unreliable when trying to relate long-term correlation between state transitions (see Figure 6 for a visual of HMM States and State Transition Notation).

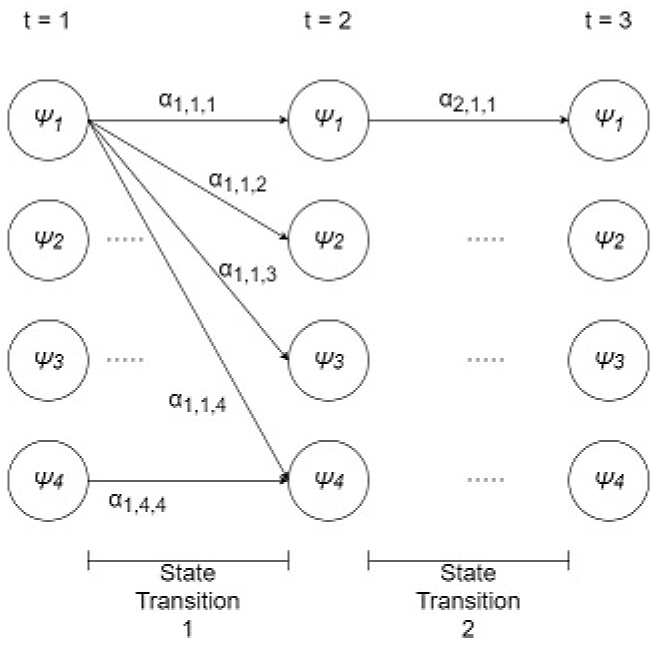


Figure 6. Hidden Markov Models: State and State Transition Visual and Notation

Note that the weights on each arrow in the trellis in Figure 6 represent probabilities. In general terms, $\alpha_{i,a,b}$ represents the probability of the $i^{\text{th}}$ state transition from $\Psi_a$ to $\Psi_b$. Also note that each $\Psi_a$ represents a distinct state ($\Psi_a = \Psi_a$ but $\Psi_a \neq \Psi_c$ regardless of event[3]).

[3]I used the subscript $c$ here to avoid confusion since $a$ and $b$ can be equal in that context.

This is where Viterbi comes in. Viterbi uses the statistics from HMM and state transitions to determine a maximum likelihood sequence for a given set of binary data of a given sample size. The Viterbi Algorithm does so by calculating the $\alpha$ coefficients seen in Figure 6 for each state transition. Each state, denoted by $\Psi$, represents a hexdecimal symbol being received at time $t$ or conversely in bit $t$. The algorithm works regardless of the semantics, but let us assume that $t$ in Figure 6 represents the bits in a received message. We can assume, as an initial condition at least, that all bits are equally likely to appear at any point in a message since the message generation at this point is random. Until the system is implemented where the transmitted command or data is not completely random there is no reason to assume that any character in any position, prior to testing, is more likely to appear than another. As the subscripts of $\alpha$ in Figure 6 denote, this implementation of Viterbi considers each state transition to be separate and unrelated. Due to them being separate, each bit to bit transition must be considered separately. This is not much of an issue when collecting multiple samples of a given signal in a noisy environment, as it is quite common for a transmit station to send multiple copies of a given message in order to ensure that the message is most likely to reach its destination.

In statistical terms, the multiple received messages that copies of the one intended message are treated as the sample population and each bit to bit transition is treated as a separate and unrelated event. A "Transition Matrix" was implemented for each state transition, assume there are $N$ state transitions and consequently $N$ bits for this example. In this example there are also $N$ matrices since each event is considered separate and unrelated. Think of each element within a given transition matrix as one of the $\alpha$ coefficients which serve as the weights on the decision trellis (another name for the diagram in Figure 6), $\alpha_{i,1,1}$ through $\alpha_{i,16,16}$.

$$T_i = \begin{bmatrix} \alpha_{i,1,1} & \alpha_{i,1,2} & \alpha_{i,1,3} & \cdots & \alpha_{i,1,16} \\ \alpha_{i,2,1} & \cdots & \cdots & \cdots & \alpha_{i,2,16} \\ \alpha_{i,3,1} & \cdots & \cdots & \cdots & \alpha_{i,3,16} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \alpha_{i,16,1} & \cdots & \cdots & \cdots & \alpha_{i,16,16} \end{bmatrix}$$

The $\alpha$ coefficients can be easily calculated using multiple methods. The most straight forward method is to calculate the statistic by counting the number of occurrences of a state transition from $a$ to $b$ from position $i$ to $i+1$ (where $i=0$ implies the starting state) and divide that number by the total number of times that state $a$ occurred in position $i$. This probability is defined, less wordily, in the following equation:

$$\alpha_{i,a,b} = \frac{N_{i,a,b}}{\sum_{j=1}^{\forall\Psi} N_{i,a,j}}$$

Once all $\alpha$ coefficients are calculated for all state transition matrices, the optimal path can be determined. This is achieved by two primary methods, which will be referred to as "Fast Path-finding" and "Multi-path Path-finding". Fast path-finding involves the creation of only one path which saves a lot of time and is accurate most of the time depending on the sample population (exact results can be seen in subsection 3.3). The program or hardware proceeds through the decision trellis event by event and chooses the transition with the highest probability. This procedural transition using highest-likelihood state transitions creates the optimal path in the fast path-finding method in a fast and reasonable accurate manner. The multi-path path-finding method works be generating all possible state sequences of legal state transitions (forbidden state transitions occur where the $\alpha$ coefficient is 0), calculates the probability of each path occurring and choose the path with the highest probability. One possible implementation of this method that would be less taxing on the hardware would be to implement a cost paradigm instead of probability by using logarithms on the given probability. This would cause multiplication to be replaced by addition, which is much friendlier and faster on digital hardware. Both of these methods will be discussed in subsection 3.3.

# 3 Test Procedures, Results & Discussion

## 3.1 Parity-Check Simulations

A program was written in Python to emulate a Parity Check system in a ”noisy” environment. The ”noisiness” was accomplished by implementing a user-defined bit error ratio (BER), which was a threshold used to determine if a given bit was to be flipped, or more concisely in this case, the BER is used to determine if a symbol is corrupted due to ”noise” as well as if the parity bit itself were transmitted and possibly flipped due to the user-defined BER.

Regarding Simulation Conditions: The Parity Check simulation was performed under known BERs with 20 separate data points. The sample size for each data point consisted of 1000 separate trials, each trial classified 1000 different samples as correct or corrupted. The metric that was logged using this program was misclassification, also called an error in this context, which occurs when a corrupted sample is found to be correct by parity check, or when a correct sample is found to be corrupt by the parity check.

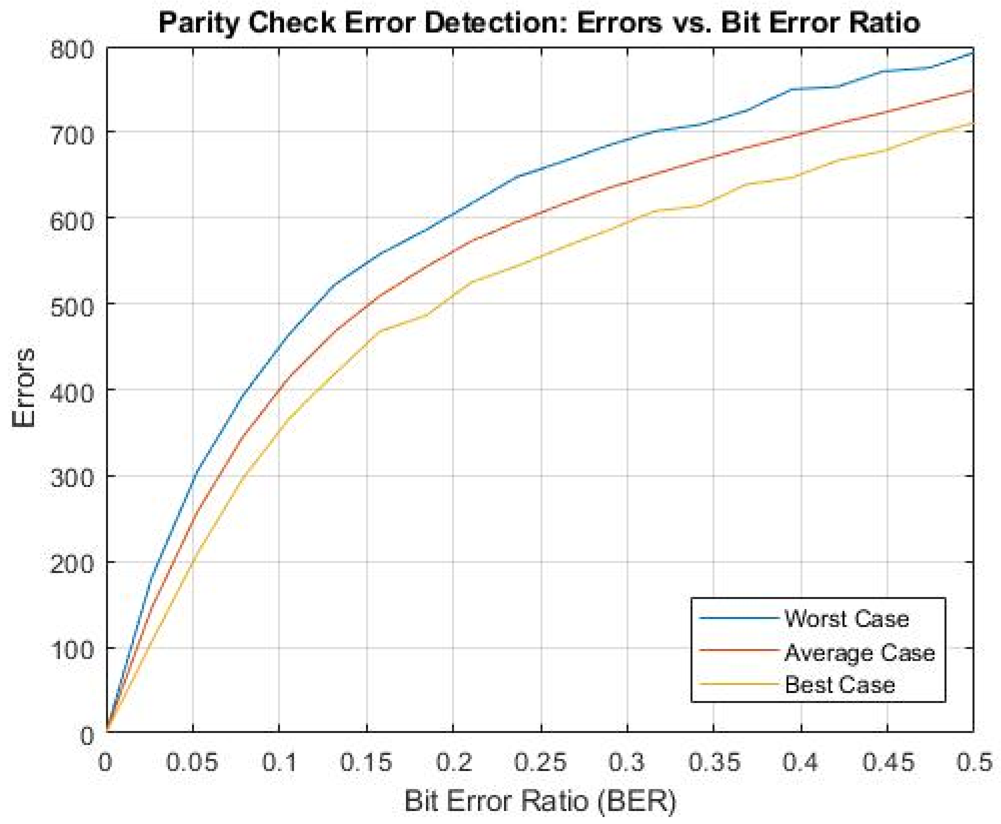


Figure 7. Parity Check Error-Detection: Errors vs. Bit Error Ratio

Errors here are defined to be the number of misclassifications of a given message. More specifically, an error is a misclassification. The misclassification condition is met under one of the following conditions:

- The parity bit is not corrupted and the message parity does not match the parity bit
- The parity bit is corrupted (Parity bit received different from parity bit sent)

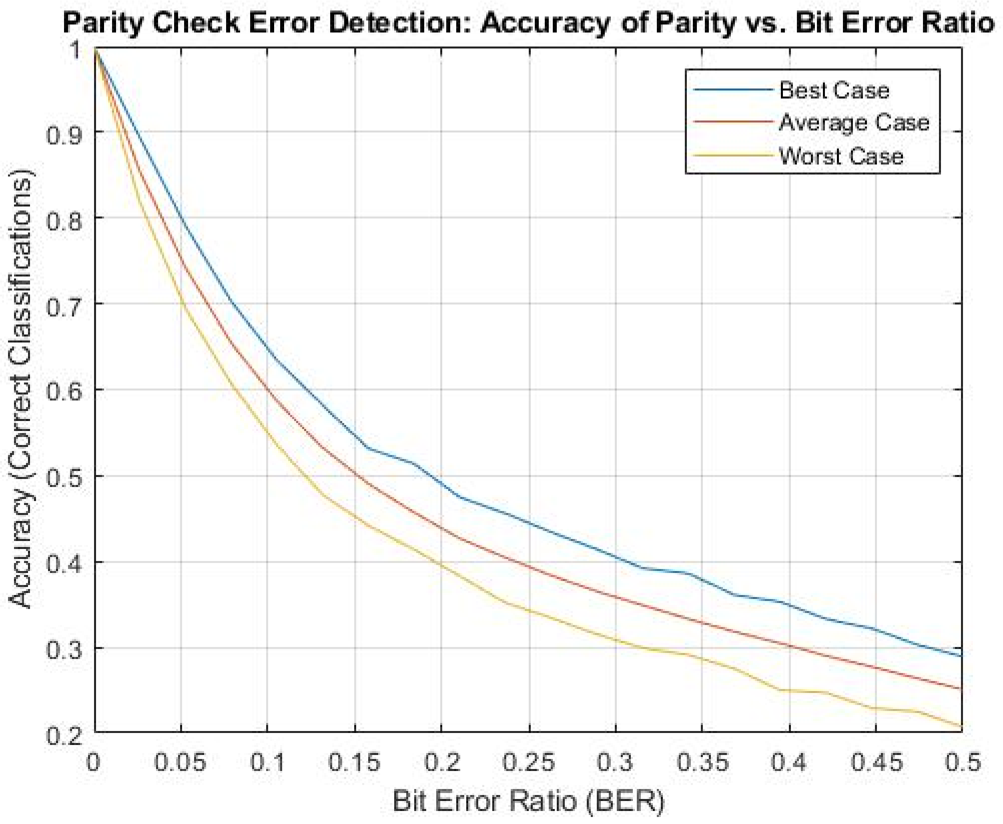


Figure 8. Parity Check Error-Detection: Classification Accuracy vs. Bit Error Ratio

Accuracy is an easier metric to digest once the error metric is understood. Accuracy can be defined by the following formula:

$$\mathbf{Accuracy} = 1 - \frac{\mathbf{Misclassifications}}{\mathbf{TotalTestCases}}$$

This small-scale simulation clearly demonstrates both in terms of errors and accuracy why the use of a parity check as an error-detecting code in the case of 256-QAM is a poor fit due to the bit error rate of 256-QAM is higher than most due to its high susceptibility to noise (see Figure 4 for the minute distances between symbols which are easily bridged by noise in freespace). The next section will demonstrate why using Viterbi and omitting error-detecting codes altogether yields a greater result.

## 3.2 Matlab Proof of Concept Simulation: 256-QAM

The purpose of this simulation was to demonstrate the mechanisms of QAM transmission using a highly sensitive rendition (i.e. 256-QAM). This was achieved by simulating hardware seen in Figure 23 and Figure 24 in the form of software, Matlab specifically. All code for this simulation can be seen at the link to the GitHub repository seen in Appendix A. The wave forms generated by this simulation can be seen in Figure 10 through Figure 17.

These simulations were conducted using constant frequency deviation withing the FM Transmitter functionality in the Communications Toolbox within Matlab of 2.667 MHz. Random Message signals were used to ensure fairness of test cases. The simulation is represented by a block diagram as seen in Figure 9

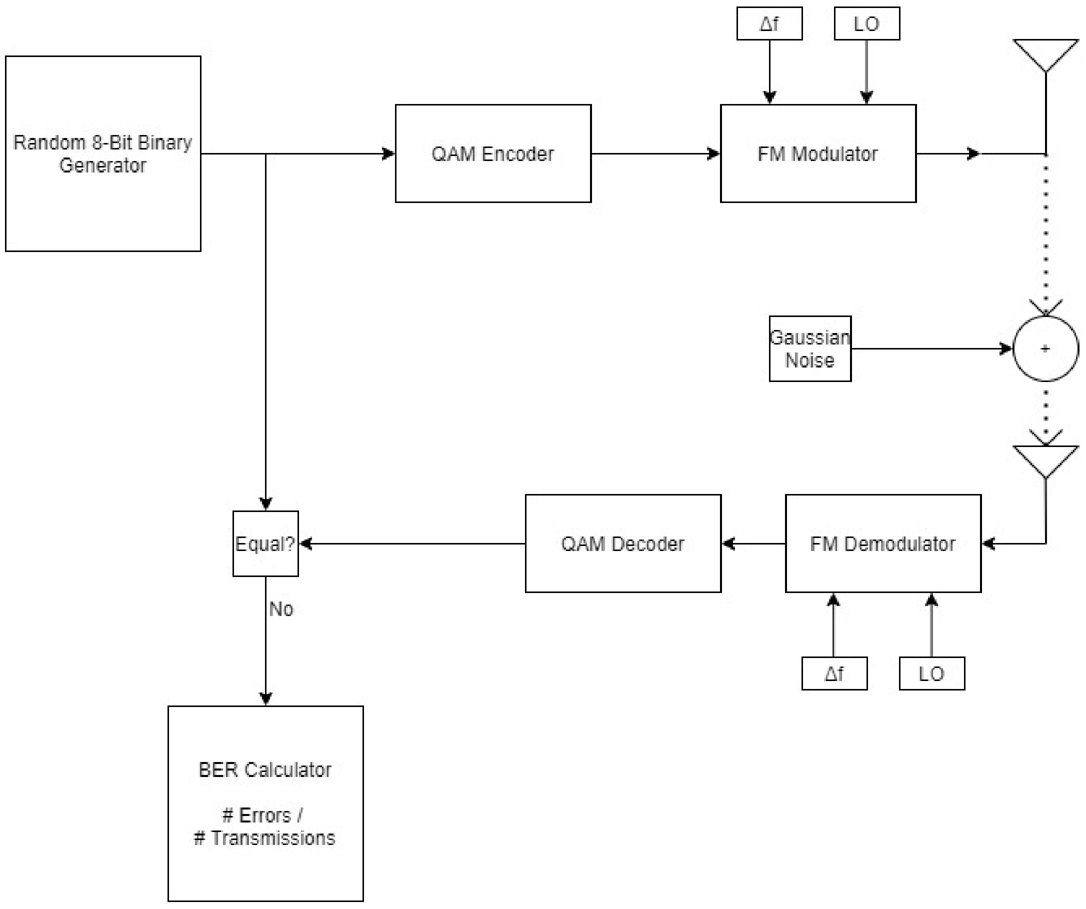
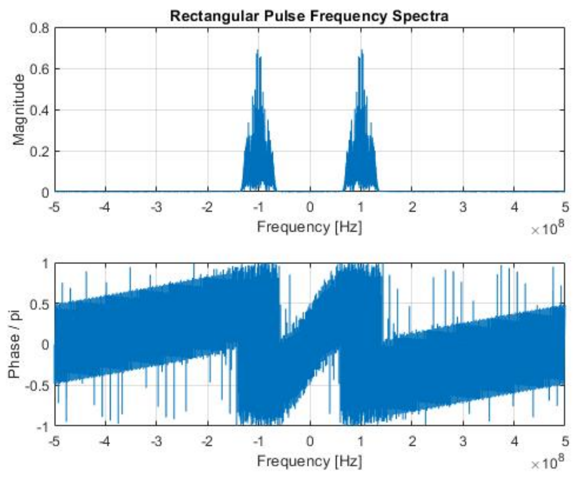
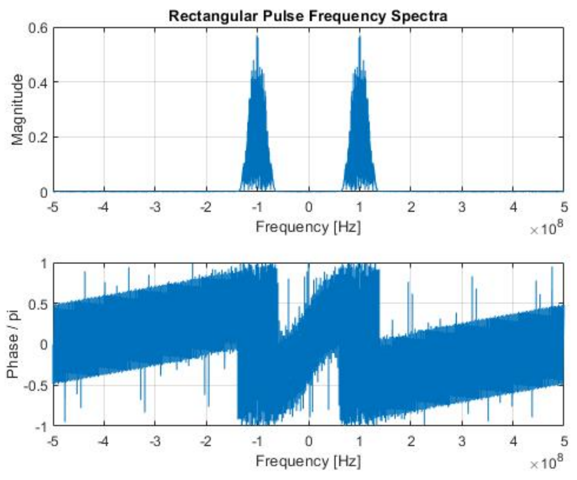


Figure 9. Matlab Software Diagram: Simulation Hardware Equivalent



(a) In-Phase



(b) Quadrature

Figure 10. Transmitted Message Signal: Frequency Domain; Noise Power = 15dB

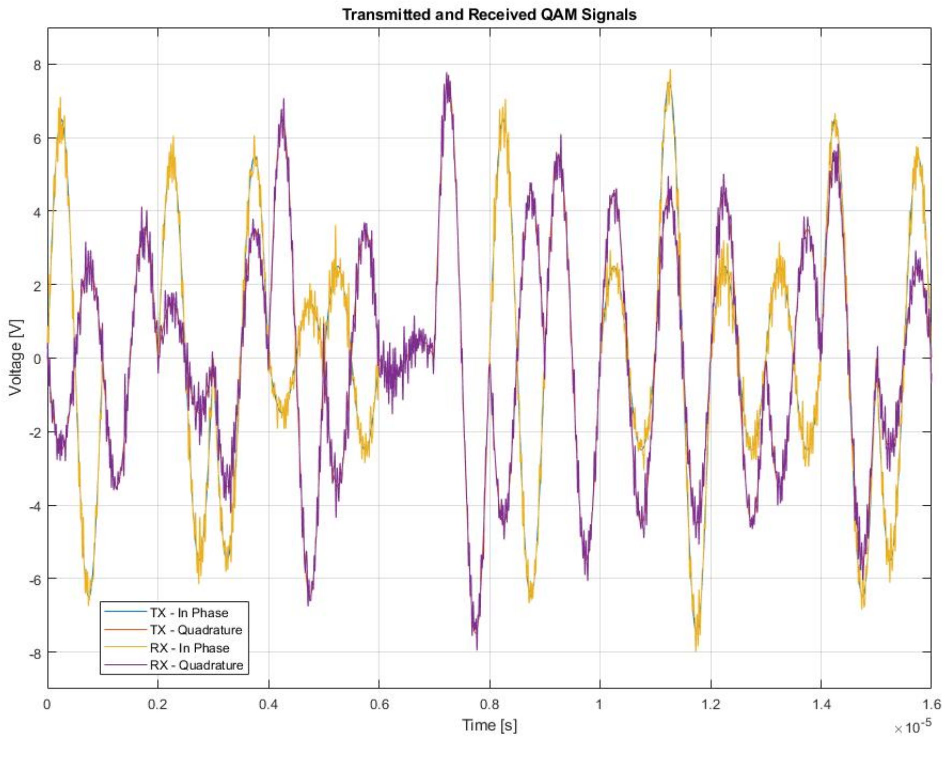
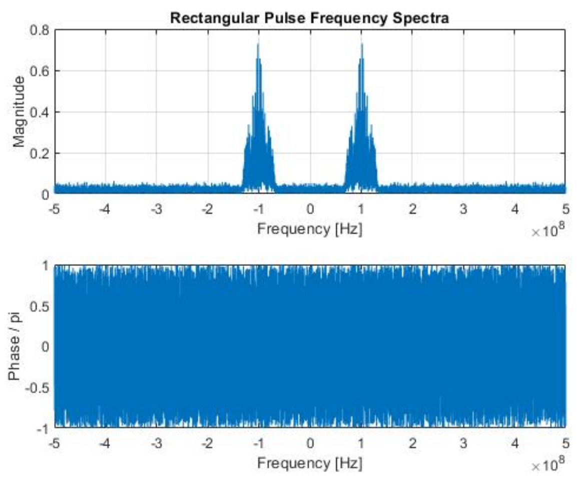
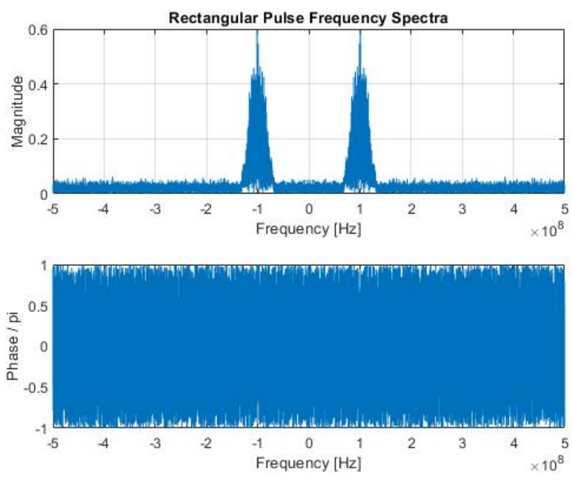


Figure 11. Transmitted and Received Message Signal: Time Domain; Noise Power = 15dB

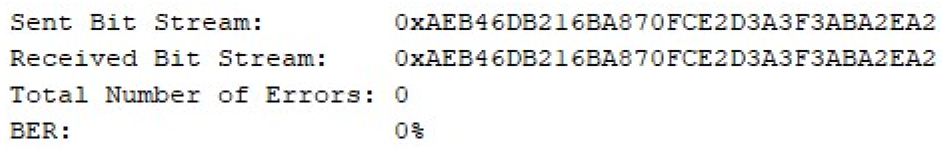


(a) In-Phase



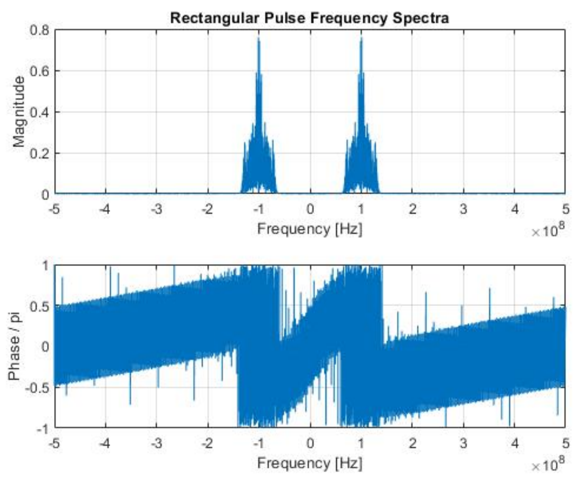
(b) Quadrature

Figure 12. Received Message Signal: Frequency Domain; Noise Power = 15dB

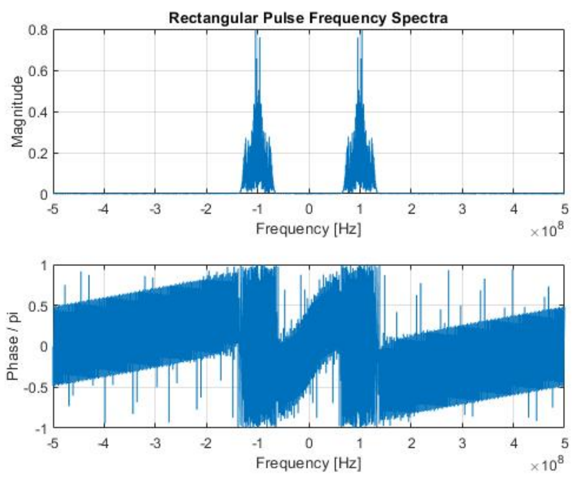


```
Sent Bit Stream:         0xAEB46DB216BA870FCE2D3A3F3ABA2EA2
Received Bit Stream:     0xAEB46DB216BA870FCE2D3A3F3ABA2EA2
Total Number of Errors: 0
BER:                     0%
```

Figure 13. Matlab Simulation Output: Noise Power = 15 dB



(a) In-Phase



(b) Quadrature

Figure 14. Transmitted Message Signal: Frequency Domain; Noise Power = 17dB

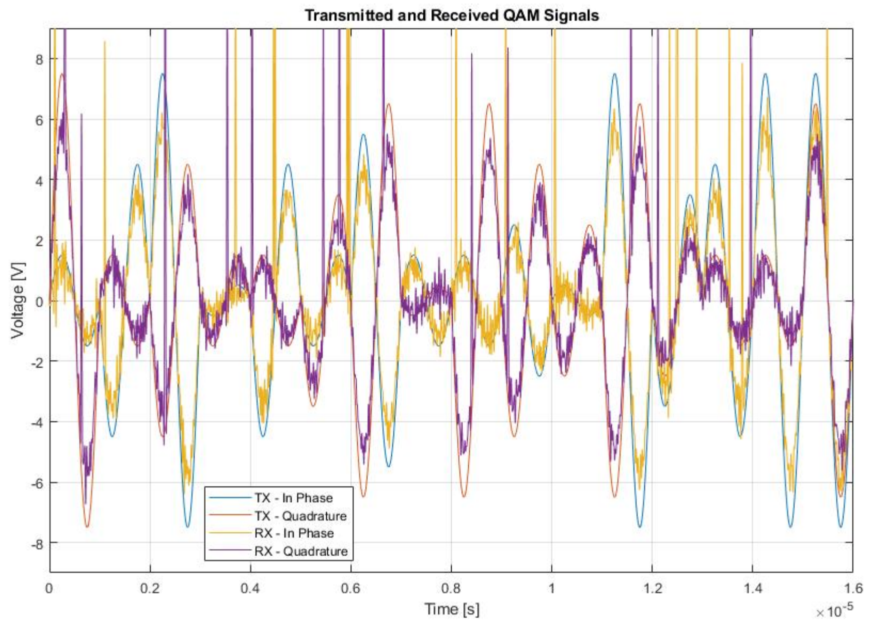
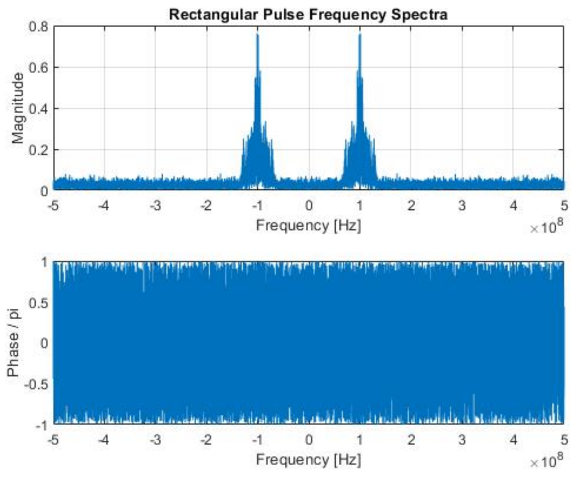
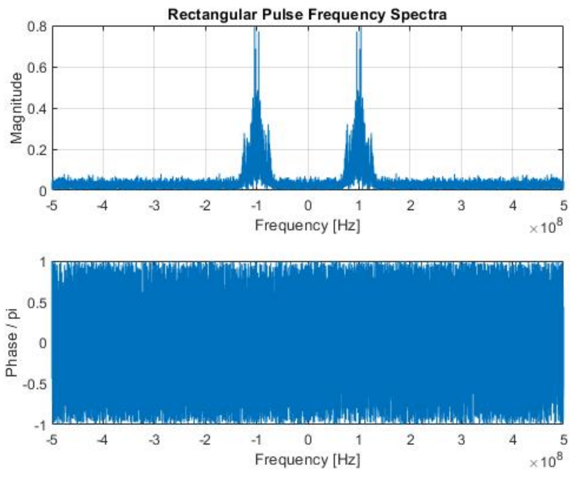


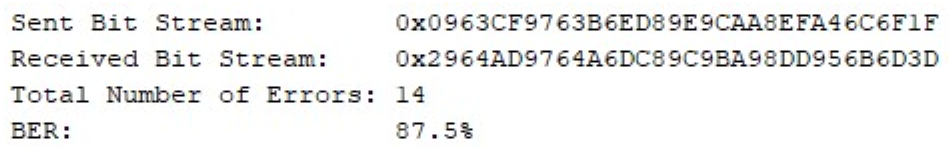
Figure 15. Transmitted and Received Message Signal: Time Domain; Noise Power = 17dB



(a) In-Phase

(b) Quadrature

Figure 16. Received Message Signal: Frequency Domain; Noise Power = 17dB



```
Sent Bit Stream:          0x0963CF9763B6ED89E9CAA8EFA46C6F1F
Received Bit Stream:      0x2964AD9764A6DC89C9BA98DD956B6D3D
Total Number of Errors: 14
BER:                      87.5%
```

Figure 17. Matlab Simulation Output: Noise Power = 17 dB

A second simulation took place that utilized the first simulation. The second iteration took controlled the noise power and frequency deviation of the FM Modulator and monitored the bit error rate of signals that mapped to specific SNR. This simulation is essentially the same as the first however it is performed over 10 to 100 iterations per test case with 42 different test cases: 21 different cases for Noise Power (SNR by extent) of the Gaussian noise generator and 2 cases for frequency deviation. The code can be seen in the script titled "stat_script.m" in the subfolder listed as Matlab\Scripts in the GitHub repository. Results are seen in Figure 18 and Figure 19.

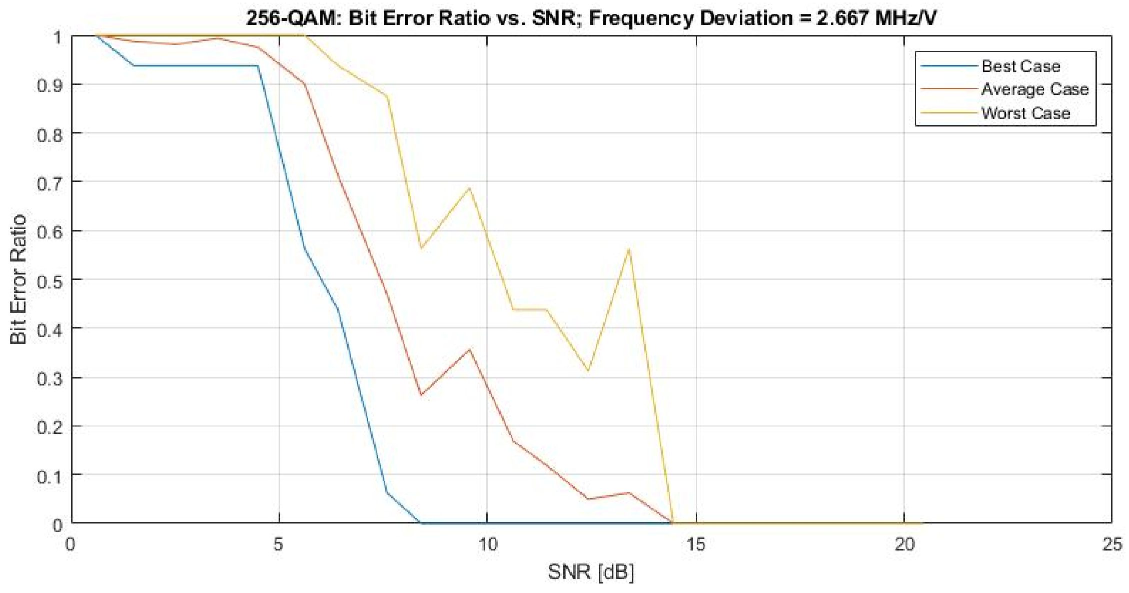


Figure 18. BER vs. SNR: Frequency Deviation 2.667 MHz

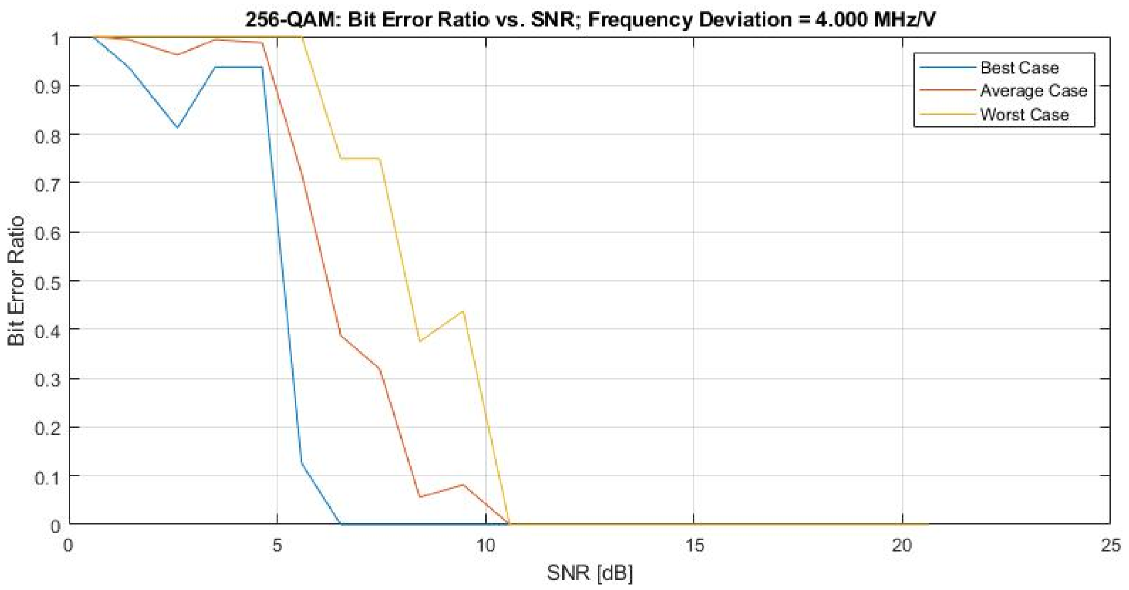


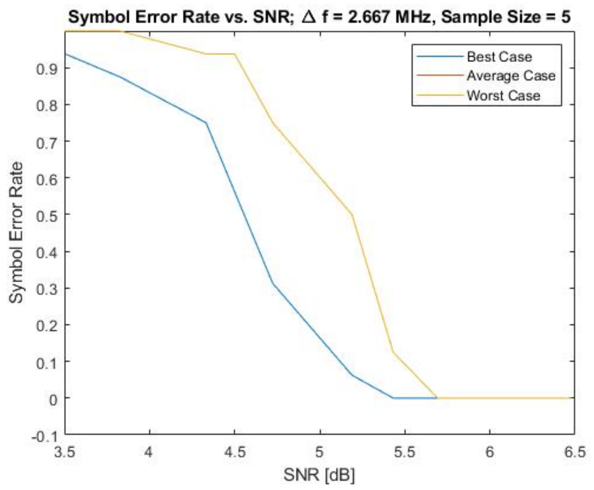
Figure 19. BER vs. SNR: Frequency Deviation 4.000 MHz

The results above were expected; the Bit Error Ratio (BER) should increase as Signal to Noise Ratio decreases. SNR is defined here to be ratio of transmitted RF signal to the generated Gaussian White Noise. The interesting note is that with the increased frequency deviation the decline of the signal integrity was not as intense. This demonstrates that the Gaussian white noise does not interfere with FM Modulation with a broader spectrum. These results could be further improved with the addition of a bandpass filter at the RF frontend of the receiver simulation as well as a similar filter the the end of the RF backend prior to transmission in the transmit simulator.

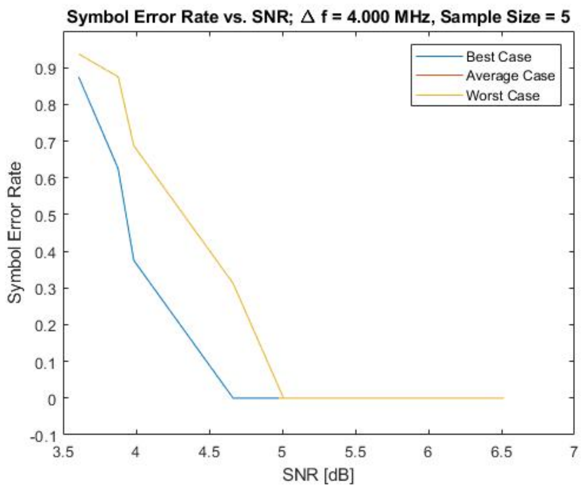
## 3.3 Viterbi Error Correction

A library was written in Python that implemented the Viterbi Algorithm as a maximum likelihood path-finding algorithm, titled 'Viterbi.py' in the Python directory in GitHub. This Python library was integrated into a Matlab environment using 'pyenv' which is quite a nifty tool for using custom Python libraries and modules in Matlab. It was specifically integrated into the Matlab Simulation script that simulates the encoding of QAM, RF Channel, and decoding of QAM with control over noise power and input sequence. The added parameter for the Viterbi Algorithm was to control the sample size, the number of message signals that the algorithm would consider its basis for calculating the optimum path.

This experiment was conducted using sample sizes of 5, 8, and 10 with 5 iterations per sample size to accrue a best, worst, and average case for each sample size. This statistical calculation is performed once per test case of which there are 20: 10 separate test cases for noise power and 2 separate test cases for frequency deviation. Results can be observed in Figure 20 to Figure 22

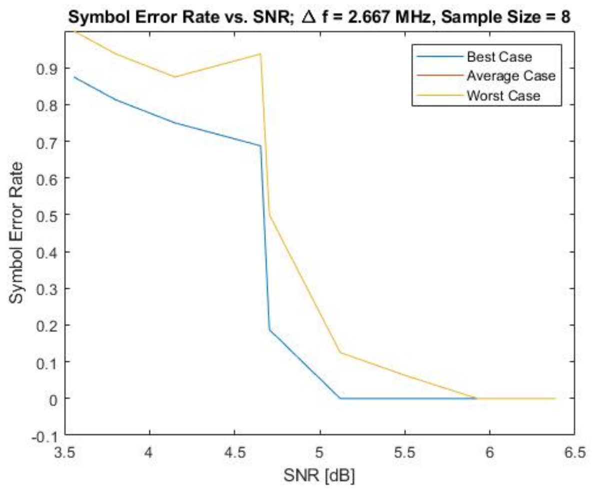


(a) Δf = 2.667 MHz

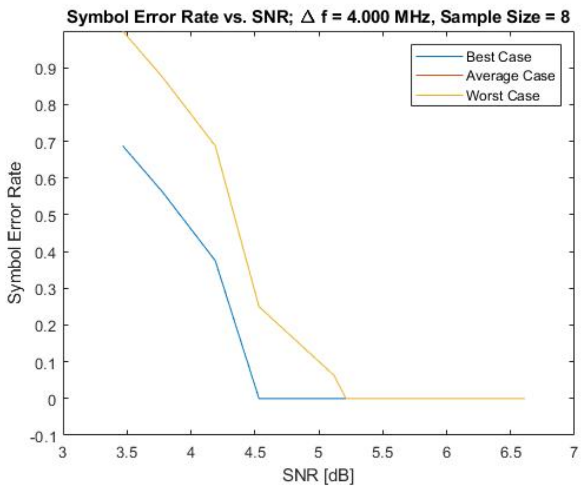


(b) Δf = 4.000 MHz

Figure 20. Symbol Error Rate vs. SNR; Sample Size = 5

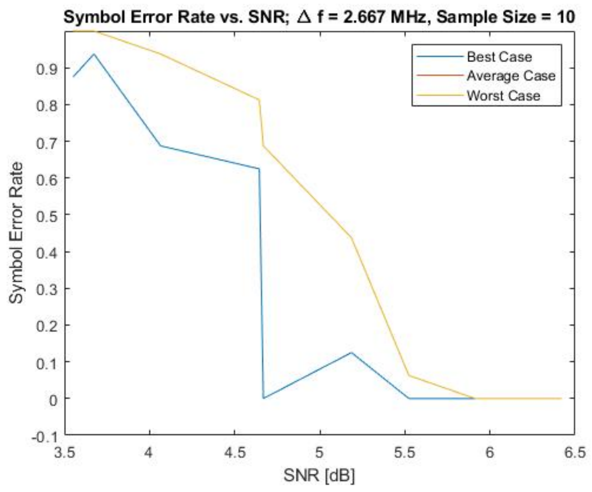


(a) Δf = 2.667 MHz

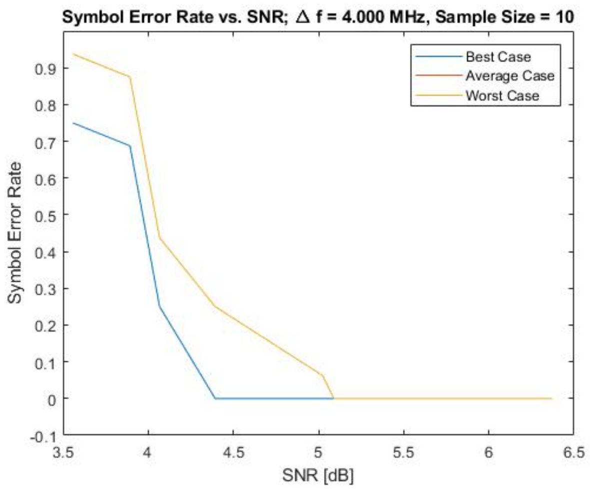


(b) Δf = 4.000 MHz

Figure 21. Symbol Error Rate vs. SNR; Sample Size = 8



(a) Δf = 2.667 MHz



(b) Δf = 4.000 MHz

Figure 22. Symbol Error Rate vs. SNR; Sample Size = 10

Take note that sample size had little to no effect on the trend of this system, however the introduction of Viterbi Error Correction did cause the system to perform with a lower Symbol Error Ratio (approximately equal to the Bit Error Ratio) with sample sizes of 5 through 10. Only the faster version of the Viterbi Path-finding algorithm was used due to the author's computer being needed for other matters and the simulations using the multi-path simulation were more accurate but took much longer.

The data shown in Figure 20, Figure 21, and Figure 22 clearly demonstrate that the Viterbi Algorithm can be used with a digital communication system to improve the overall Bit Error Ratio considerably in noisy communication channels, indicated by the shift from Figure 19 to Figure 20 in where the BER begins to rise at 10 dB in Figure 19 as opposed to around 5 dB in Figure 20.

The code for this simulation can be viewed in a file titled 'viterbi$_{s}im.m'intheMatlabsubdirectoryontheC$

## 3.4 Software Defined Radio Implementation

A library was developed in Python to facilitate the setup, transmission, and reception of data using the Pluto SDR, a Software Defined Radio developed by Analog Devices. This library uses multiple techniques to emulate a matched filter and a threshold detector to determine whether or not a signal has been received. This library also performs all encoding and decoding of the QAM symbols.

This library was tested using scripts (all of which can be seen in the Test sub-folder of the Python folder in the GitHub repository mentioned in Appendix A). The SDR transmitted messages of many different compositions (random messages and known messages specifically) which were used to trouble test and analyze both the SDR as well as the library developed for this project. The received waveforms were recorded both in graphical and textual form, specifically in CSV files.

Various parameters within the SDR were altered in order to receive a meaningful transmission (meaningful here meaning that it, in any way, resembled the transmitted signal. Specifically the TX and RX channel gains were modified such that the transmitted signal occupied the entire ADC range of the SDR but also not so much as to saturate and distort the signal as well as prevent the signal from getting lost in the noise. Signal Attenuation over the channel was not as much of an issue due the channel length at the time was approximately one inch and remained constant.

Tests were performed using the default sample rate, $f_s$, of the SDR and an FM carrier frequency of 1 GHz. The QAM used a pulse shape of a sinusoid defined by the following formula (or basis function):

$$p(t) = sin(2\pi f_c t) \prod(t/T_c)$$

where $f_c = f_s/100$, $T_c = 1/f_c$, and $f_s$ of the SDR is approximately 30.72 MHz.

The software written for this test specifically is meant to model the block diagrams shown in Figure 24 and Figure 23. A real-time implementation of this application should be conceivable in hardware with the use of a number of ADCs and either an FPGA or a Microprocessor, but for proof-of concept purposes this application was developed using Python in post processing as well as the SDR itself.

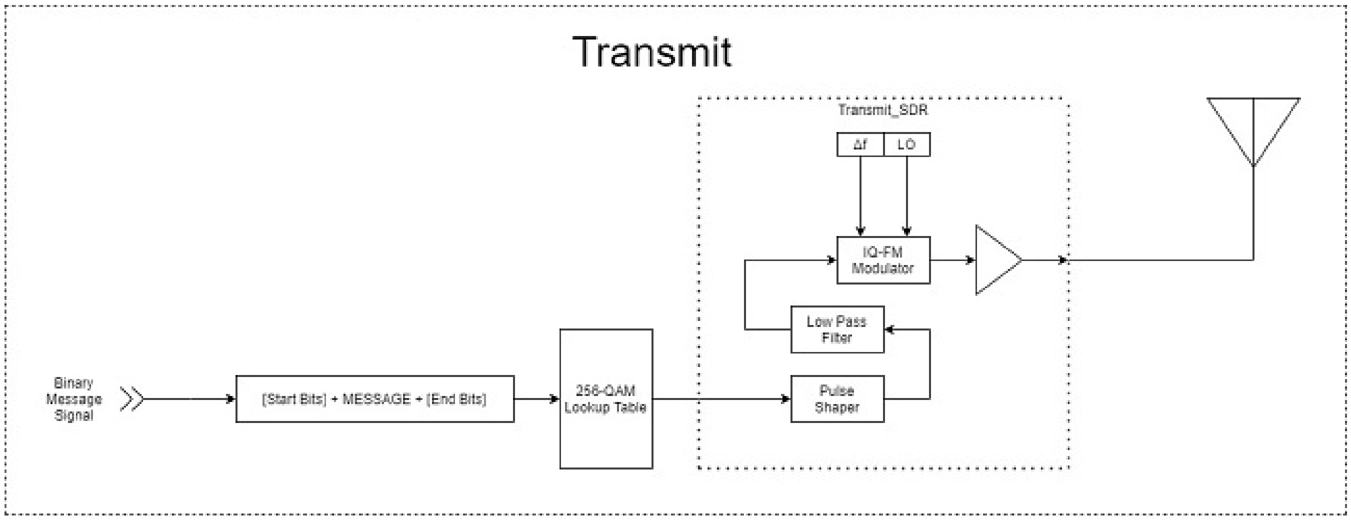


Figure 23. Ideal TX Hardware Block Diagram for QAM Modulation

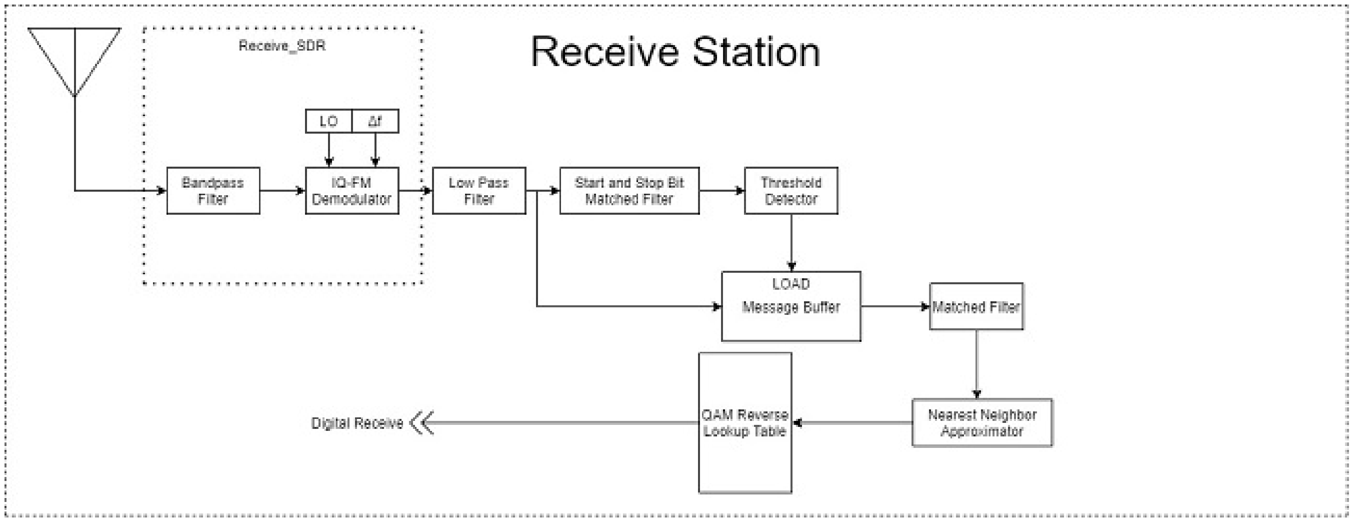


Figure 24. Ideal RX Hardware Block Diagram for QAM Modulation

This was the most difficult portion of the experiment to successfully run. An internal gain for the signal going into the SDR needed to have some gain applied to it such that the signal occupied the entire dynamic range of the SDR's DAC. The process of determining the proper constant for conversion from signal to DAC value yielded varied results from the system as evident in Figure 26, Figure 27, and Figure 28. Figure 28 was settled upon as the standard for signal transmission as the signal was not overtaken by noise (e.g. Figure 26) nor did it saturate the ADC of the RX system post FM Demodulation (Figure 27).

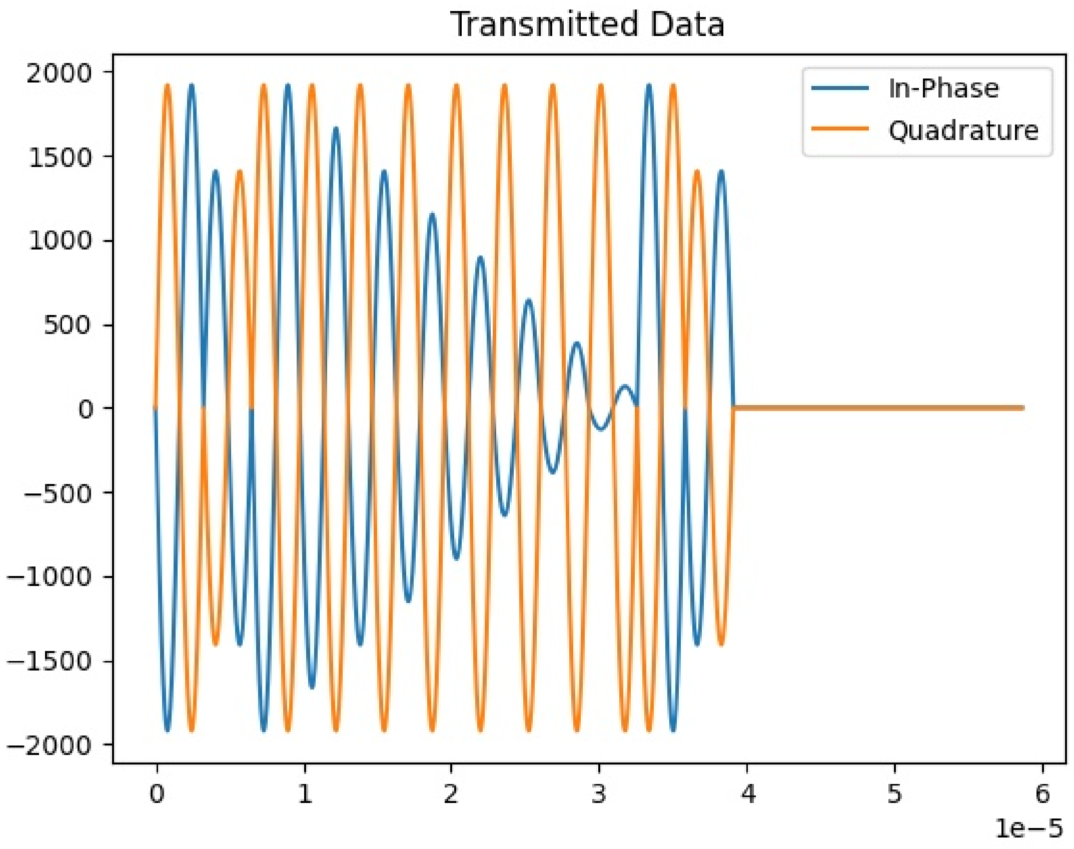


Figure 25. Transmitted Data from SDR

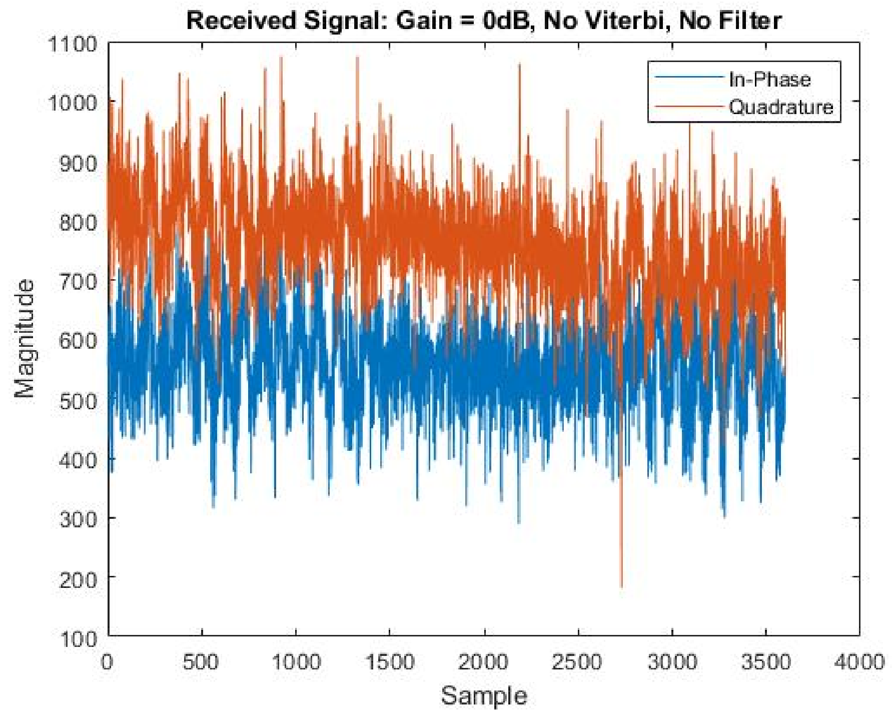


Figure 26. Anomalous Behaviour Demonstrated by the SDR; Item 1 - Absence of Signal

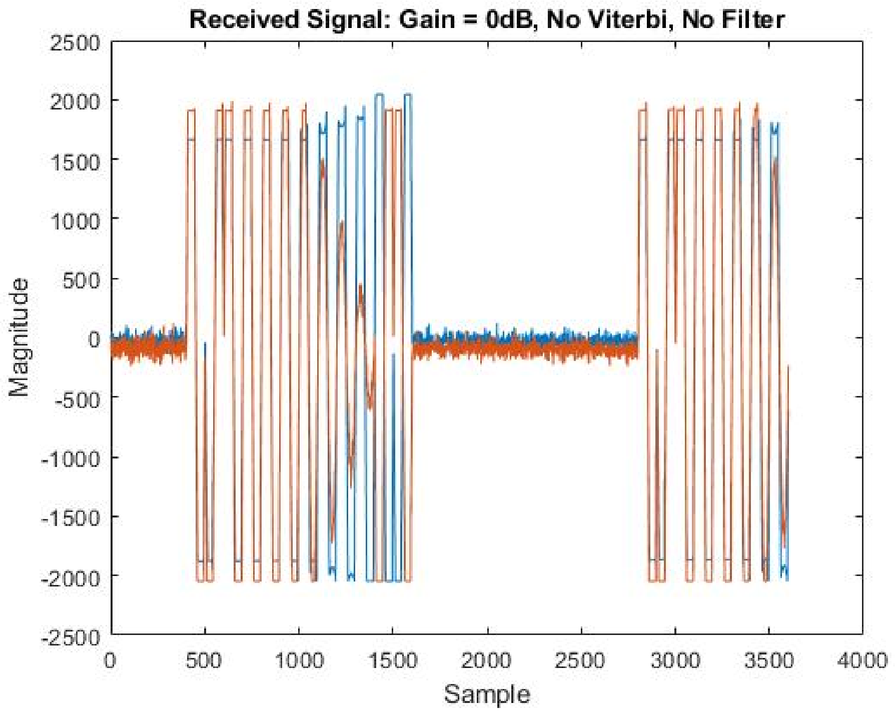


Figure 27. Anomalous Behaviour Demonstrated by the SDR; Item 2 - Signal Saturation

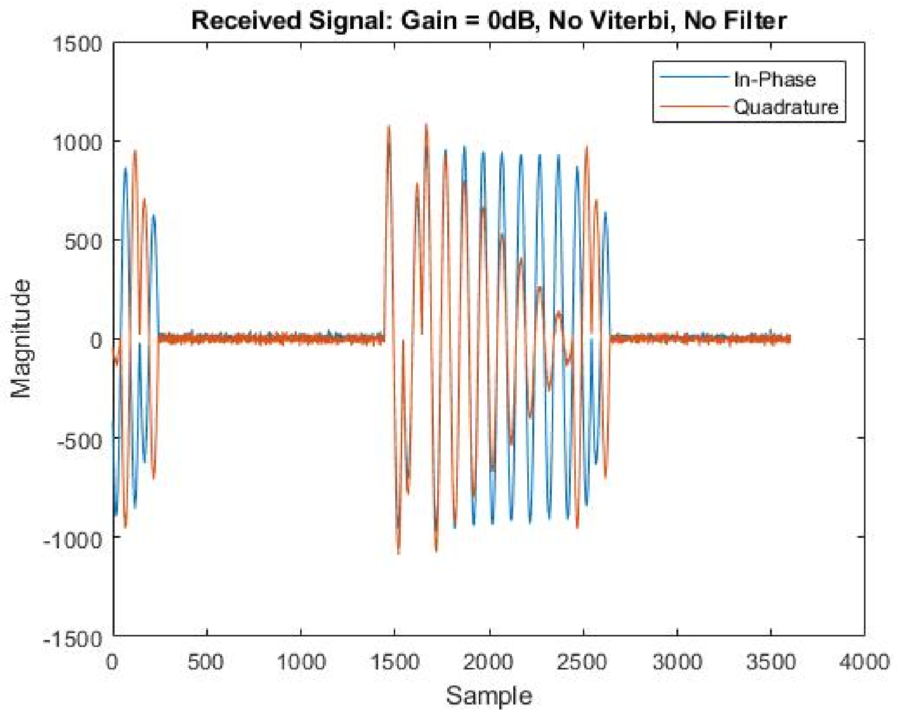


Figure 28. Predicted Behaviour of the SDR; Signal Transmission

Do note that Figure 26, Figure 27, and Figure 28 were all performed using an identical message signal, start bits, end bits, as well as TX / RX RF bandwidth. After reevaluation of the results found in Figure 28, it was found that the bits that were received were not identical to the bits that were transmitted. More specifically, the signal that was transmitted was supposed to show that the in-phase voltage decreased with time in the message signal whereas Figure 28 shows that it is the quadrature. Moreover, the in-phase and the quadrature should be of opposite polarity due to the signal that was transmitted (can be seen in Figure 25). This combination of these two points indicates that the transmission distance is accountable for some time delay between the transmission and reception, that cannot be ignored, and that delay caused the quadrature to be picked up by the in-phase of the IQ modulator within the SDR and vice versa. This variability can be accounted for using a correlation detector and altering around the received signal until the maximum correlation is detected. This method was performed with 8 different test cases: no inversion or switching, inverted I, inverted Q, both inverted, repeat of the prior test cases but with the switching of I and Q. The addition of that algorithm, dubbed as "iq_find" in this case, yielded the following:

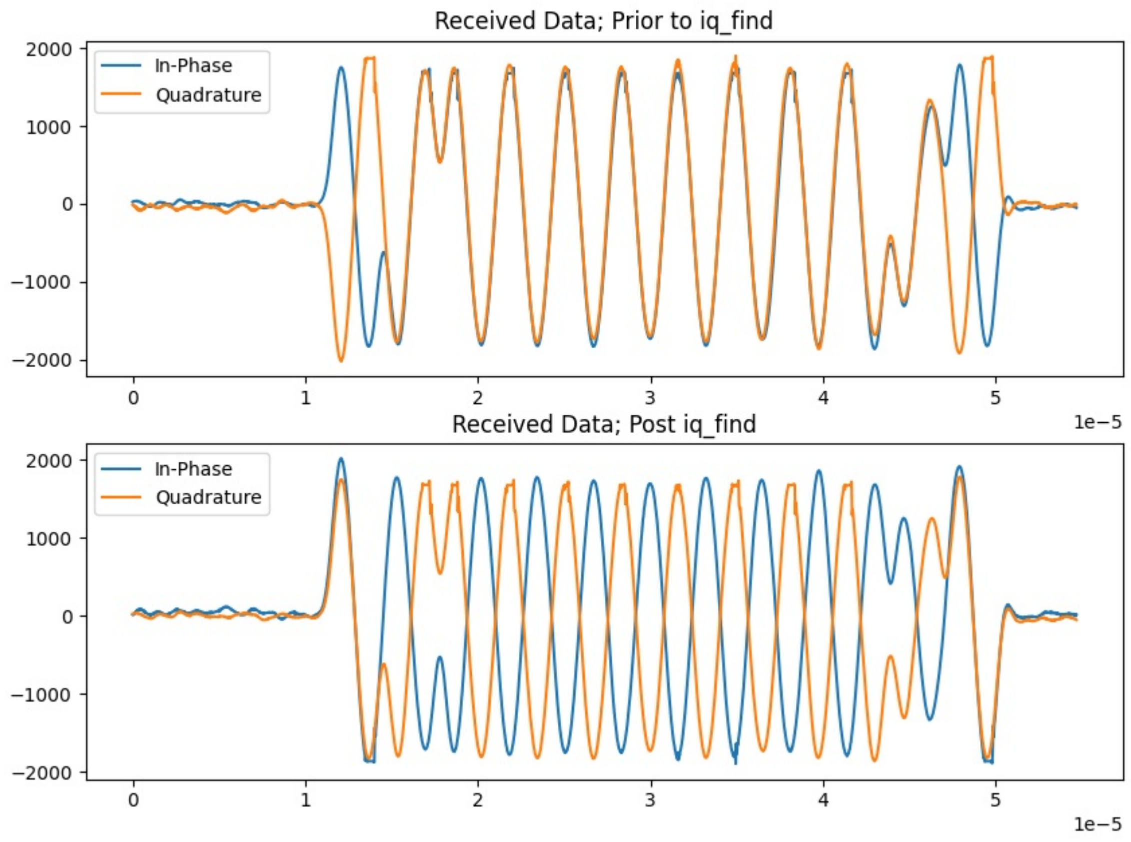


Figure 29. Algorithm Demonstration of 'iq_find': Prior To and Post Utilization

It is seen that the algorithm 'iq_find' can successfully detect when an IQ dataset has been altered by the transmission channel through the use of known start and end bits or symbols (see Figure 29). With that issue resolved, the system could be properly tested with the SDR and associated library designed specifically to operate it. This was performed by transmitting a single signal using a cyclical buffer to continually sample it and observe any changes that might occur. Multiple message signals were used to determine if Inter-Symbol Interference (ISI) affected the results, however the pulse that was used to shape the QAM waveform was chosen such that in an ideal channel ISI should be 0. The channel used is by no means ideal, however this was chosen as optimal given prior experiments and changes to the pulse shape would little marginal changes that would go largely unnoticed. The results of receiving a message signal multiple times can be observed in Figure 30 and Figure 31 with the result of Figure 31 seen in Figure 32.

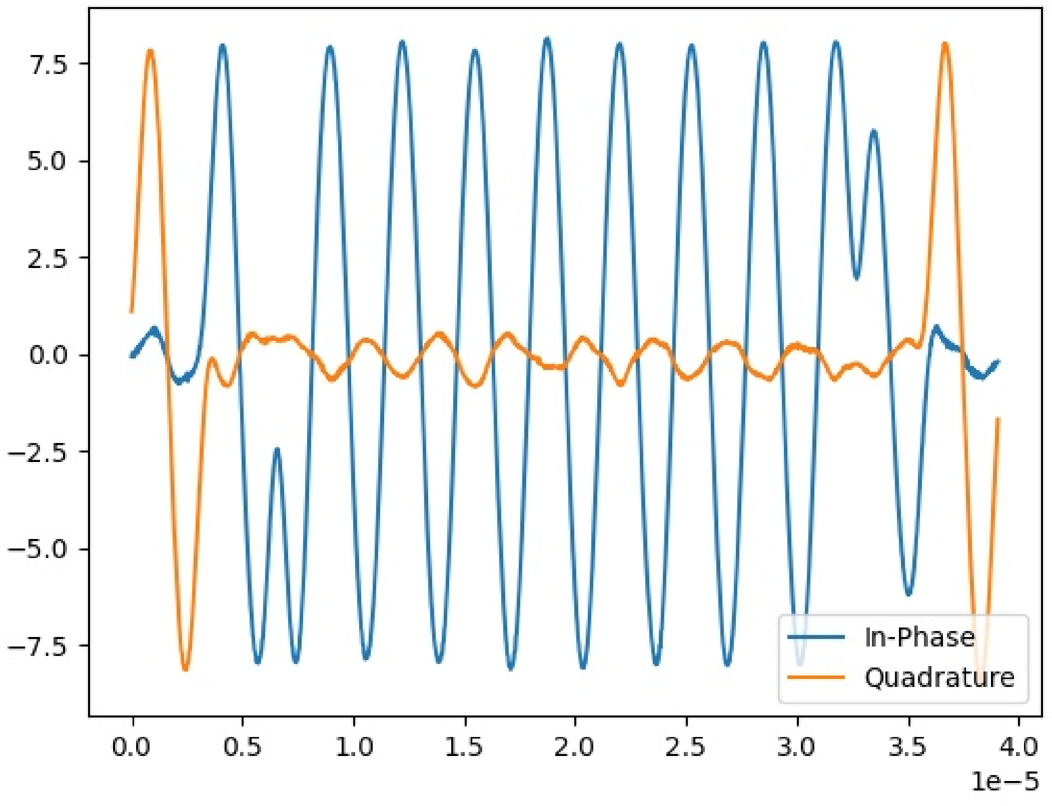


Figure 30. Software Defined Radio: Received Message Signal; Non-Uniform Amplitude Abnormality

Given a sample size of 10 transmission, 7 were received identical or similar to the waveform in Figure 30. The direct cause is unknown, however the most probable cause is that the power delivered to the SDR from the computer's USB port is insufficient causing attenuation in the quadrature and only half of the IQ transmission is driven to the proper voltage. The plug used to connect the SDR and the computer is primarily for data transmission purposes and not capable of transmitting larger amounts of power. This however does not account for the reason why there were still come waveforms that were received free of such distortion, for example in Figure 31.

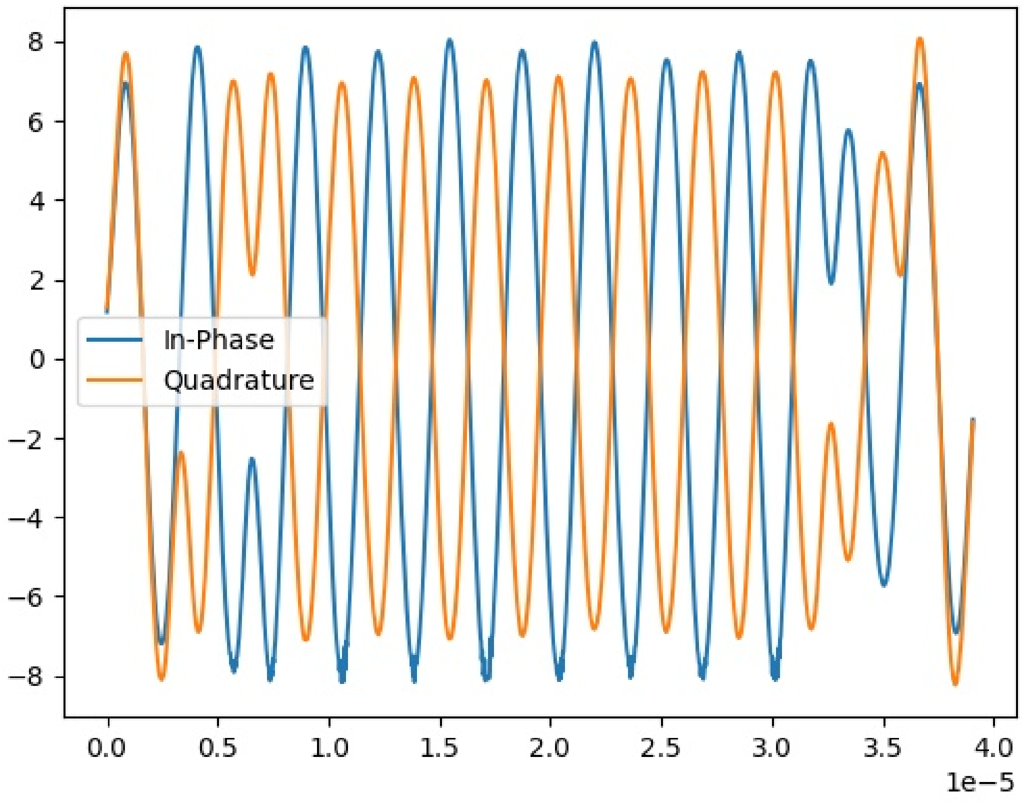
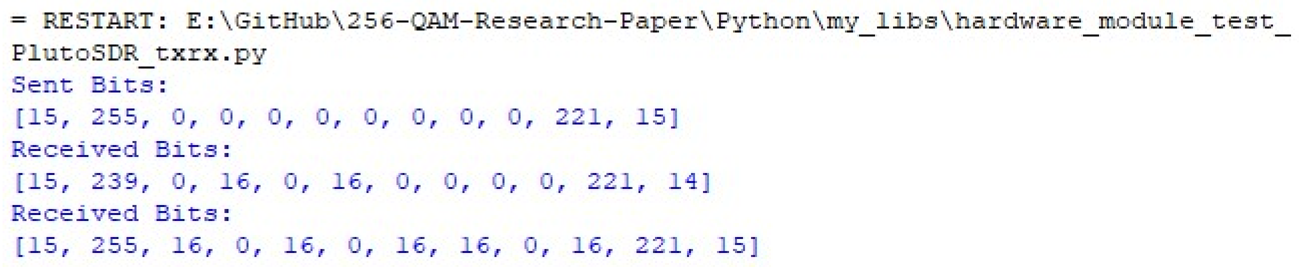


Figure 31. Software Defined Radio: Received Message; Semi-Uniform Amplitude



```
= RESTART: E:\GitHub\256-QAM-Research-Paper\Python\my_libs\hardware_module_test_
PlutoSDR_txrx.py
Sent Bits:
[15, 255, 0, 0, 0, 0, 0, 0, 0, 0, 221, 15]
Received Bits:
[15, 239, 0, 16, 0, 16, 0, 0, 0, 0, 221, 14]
Received Bits:
[15, 255, 16, 0, 16, 0, 16, 16, 0, 16, 221, 15]
```

Figure 32. Software Defined Radio: Received Message; Decoded QAM

It is worth noting that in the message signal sent in Figure 31, the same symbol was sent for the entirety of the message, with the exception of the start and the end symbols, and yet the amplitude of the in-phase as well as the quadrature components were not uniform. This caused the decoding software to recognize the bit as a '16' instead of '0' since they are adjacent to one another in the constellation diagram (Figure 4, '0' is the bit in the very top left and '16' is the symbol directly beneath it). This misclassification is indicative of the quadrature amplitude to fluctuate slightly in the channel or within the hardware causing the algorithm to misclassify the symbol as a '16'. The strict reason for such an anomaly is unknown, however in future iterations of this project, the means for this could be investigated.

Another possible cause for this amplitude abnormality is the issue of TX and RX isolation since both operations occur on the same device and the transmit is accomplished using a cyclical buffer, hence it is always transmitting. More investigation is needed to provide evidence, however due to lack of hardware such as a Vector Network Analyzer or Spectrum Analyzer this investigation could not be accomplished.

There were no recorded test cases that yielded 100% symbol transmission. This was specifically due to the two specified cases as shown in Figure 30 and Figure 31. These anomalies coupled with the observation that the Pluto SDR specifically is supposed to be used as an "Active Learning Module" (Analog-Devices [2019]), this specific application of Analog Devices' ADI AD9363 RF Agile Transmitter might not be fit for applications as sensitive to amplitude distortion as this one.

# 4 Conclusion

Digital communication systems are an incredibly powerful tool, especially when paired with error detection algorithms, error correction algorithms, and future implications with computational intelligence and general AI. New schemes and error-detection and correction methods will rise with technology and perseverance. 256-QAM is no exception.

With the proper hardware 256-QAM is incredibly powerful: with any given symbol rate, the bit rate is eight-fold which yields much more time-effective transmission of data packets. This complexity of QAM is not without its drawbacks. If sub-optimal hardware is used to implement 256-QAM, problems seen in Figure 30 or Figure 31 could arise again which would be quite costly for whichever entity is responsible. 256-QAM is highly sensitive to channel noise as well any possible leakage from the transmit portion of the SDR, although in a more practical system this would be less of an issue. As seen in the mentioned figures, 256-QAM was not a proper digital communication system to implement on this SDR that is meant to be used as an active learning module. A communication scheme of this magnitude requires hardware that is picked specifically for this application due to the issues that sensitivity to various factors such as reverse isolation and noise will incur.

It was observable in the difference specifically between Figure 18, Figure 19, and Figure 22 that Viterbi pairs quite well with 256-QAM since it uses a statistical approach, in this implementation, and can improve Bit Error Rate or Symbol Error Rate. Parity Check can also prove to be effective in communication systems like this, however the noise in the environment can compromise the integrity of such a method. Error detection codes such as 12-bit CRC or other codes of similar complexity would work well in accompaniment with 256-QAM and Viterbi to provide a layered approach to signal integrity.

## A Source Code

Instead of importing walls of text for the reader to see, it was decided that posting code in a consolidated location for all interested parties to see would be preferred (since not necessarily all readers would care about such things. All source code for this project can be found in a GitHub Repository that was setup for this project: `https://github.com/ctimmins96/256-QAM-Research-Paper`

Source code is free to use under an MIT License. Author should be credited if code is reused but that is not necessarily required.

## B Unused References

The following references are not directly referenced in this paper but were used for general fact checking and research purposes:

- Lathi and Ding [2009]
- Molkenthin [2019]
- Jorgensen [2015]
- Anonymous
- Agilent-Tech
- National-Instruments

## C Optimal Normalization Formula: Proof

There was a formula used in the SDR Library that was dubbed "Optimal Normalization Constant" that needed to be addressed. The theory behind this is that if you have two signals that are relatively of similar form you can define a constant, $\Gamma$, that is defined to be the following:

$$\Gamma_{xy} = \int_{-\infty}^{\infty} \big(x(t) - y(t)\big)^2 dt$$

This is done assuming that both $x(t)$ and $y(t)$ are energy signals in that they have a finite duration. Now, assume that you want to treat $\Gamma$ as a function of some other constant $c$ that you are multiplying $x(t)$ by in order to minimize $\Gamma$. This results in the following:

$$\Gamma(c) = \int_{-\infty}^{\infty} \big(cx(t) - y(t)\big)^2 dt \qquad \frac{\partial \Gamma}{\partial c} = \int_{-\infty}^{\infty} \big(cx(t) - y(t)\big) 2x(t) dt = 0$$

$$c \int_{-\infty}^{\infty} x^2(t) dt = \int_{-\infty}^{\infty} x(t)y(t) dt \qquad c = \frac{\int_{-\infty}^{\infty} x(t)y(t)dt}{\int_{-\infty}^{\infty} x^2(t)dt}$$

The computation for the constant $c$ can also be computed discretely and that is the implementation of this formula that is used in the SDR Python Module.

# References

Agilent-Tech. I/q modulation. URL https://www.keysight.com/upload/cmc_upload/All/IQ_Modulation.htm?cmpid=zzfindnw_iqmod&cc=GB&lc=eng. Accessed 2020-Dec-13.

M. Alleven. With 256-qam, what's good for the small cell is good for the macro: Report, Oct 2016. URL https://www.fiercewireless.com/tech/256-qam-what-s-good-for-small-cell-good-for-macro-report. Accessed on 2020-Dec-2.

Analog-Devices. Adalm-pluto detailed specifications, 2019. URL https://wiki.analog.com/university/tools/pluto/devs/specs. Accessed 2020-Dec-13.

Anonymous. Theory of operation of i/q demodulation. URL https://www.microwavejournal.com/ext/resources/pdf-downloads/IQTheory-of-Operation.pdf?1336590796. Accessed 2020-Dec-13.

S. R. Eddy. What is a hidden markov model. *Howard Hughes Medical Institute and Washington University School of Medicine, Department of Genetics*, Aug 2004. URL https://www.fing.edu.uy/~alopeza/biohpc/papers/hmm/Eddy-What-is-a-HMM-ATG4-preprint.pdf.

ETSI. *Digital Cellular Telecommunications System (Phase 2+); Channel Coding.* European Telecommunication Standards Institute, 1999. URL https://www.etsi.org/deliver/etsi_ts/100900_100999/100909/08.09.00_60/ts_100909v080900p.pdf. Version 8.9.0.

D. Jorgensen. The why and when of iq mixers for beginners, June 2015. URL https://www.markimicrowave.com/blog/the-why-and-when-of-iq-mixers-for-beginners/. Accessed 2020-Dec-13.

P. Koopman and T. Chakravarty. Cyclic redundancy code (crc) polynomial selection for embedded networks. *The International Conference on Dependable Systems and Networks*, 2004. URL http://users.ece.cmu.edu/~koopman/roses/dsn04/koopman04_crc_poly_embedded.pdf.

B. P. Lathi and Z. Ding. *Modern Digital and Analog Communication Systems.* Oxford University Press, 198 Madison Avenue, New York, New York 10016, 2009.

B. Molkenthin. Understanding and implementing crc (cyclical redundancy check) calculation, 2019. URL `http://www.sunshine2k.de/articles/coding/crc/understanding_crc.html`. Accessed 2020-Dec-17.

T. K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms.* John Wiley  Sons, Hoboken, New Jersey, 2005.

National-Instruments. Quadrature amplitude modulation (qam). URL `https://www.ni.com/en-us/innovations/white-papers/06/quadrature-amplitude-modulation--qam-.html`. Accessed 2020-Dec-13.

Wikipedia. Cyclical redundancy check. URL `https://en.wikipedia.org/wiki/Cyclic_redundancy_check`. Accessed 2020-Dec-13.