16-QAM Simulation Project

Adam Aukamp

August 30, 2023

Introduction

The purpose of this project was to implement a form of multi-carrier digital modulation in software. I chose to implement a system using four 16-QAM carriers, capable of transmitting 16 bits at a time. The transmitter portion of this simulation included random bit generation, the generation of a root-raised cosine pulse, and quadrature modulation using a constellation with 16 points. The simulation also included a channel modeled by additive white Gaussian noise and a receiver capable of quadrature demodulation and optimal detection.

This simulation is relevant to the study of digital communications because QAM is used widely in practical applications. Modern technologies like Wi-Fi, LTE, 5G NR, and DOCSIS utilize QAM constellations in their implementation [1][2][3].

Before embarking on this project, I had a rough understanding of some of the concepts used, including QAM and matched filtering. This study greatly advanced my knowledge of these concepts as I got to create an implementation of them firsthand and witness the results.

Procedure

Figure 1 below is a flowchart of the main steps carried out in the code for this simulation. The simulation includes a transmitter model that generates random bits and modulates them onto four 16-QAM carriers using root RCRO pulses. White Gaussian noise is then added to this transmitted signal to model the received signal. A receiver then filters and samples this received signal and decides which bits were most likely transmitted.

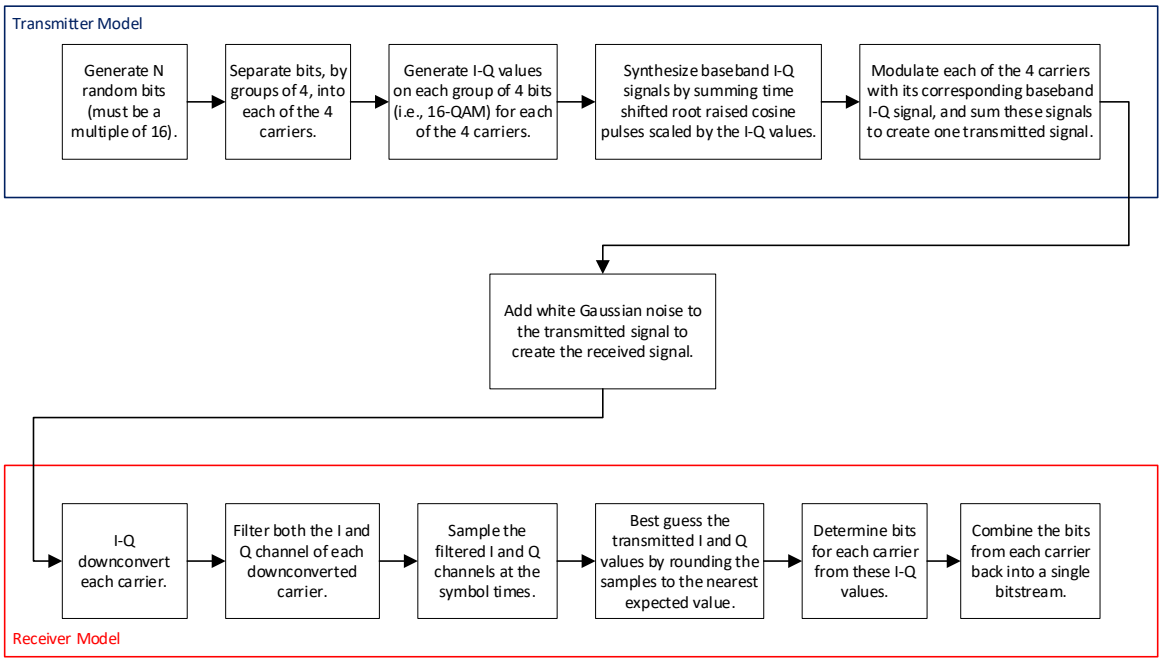

Figure 1: Flowchart Overview of Simulation Code

Root-Raised Cosine Rolloff Pulse Generation

One of the first tasks of this project was to write a MATLAB function to generate a root raised cosine rolloff (RCRO) pulse. The transmitted signals were synthesized from these pulses. A single root RCRO pulse in the time domain can be defined by the equation [4]:

$$h(t) = \begin{cases} 1 - r + \dfrac{4r}{\pi}, & t = 0 \\ \dfrac{r}{\sqrt{2}}\left[\left(1 + \dfrac{2}{\pi}\right)\sin\left(\dfrac{\pi}{4r}\right) + \left(1 - \dfrac{2}{\pi}\right)\cos\left(\dfrac{\pi}{4r}\right)\right], & t = \pm\dfrac{T_{sym}}{4r} \\ \dfrac{\sin[\pi D t(1-r)] + 4Drt\cos[\pi D t(1+r)]}{\pi D t[1-(4Drt)^2]}, & otherwise \end{cases}$$

where T_{sym} is the symbol period, $D = \frac{1}{T_{sym}}$ is the symbol rate, and $0 < r \leq 1$ is the roll off factor.

The root RCRO pulse is desirable for two reasons. First, in the frequency domain, it is absolutely bandlimited to a frequency of $B = \frac{D}{2}(r+1)$. This property is advantageous because it allowed the symbol period and roll off factor to be set such that the modulated carriers did not overlap in frequency. Additionally, the root RCRO pulse convolved with itself produces a standard RCRO pulse, which has zero inter-symbol interference (ISI). In this simulation, the received signal was filtered by convolving it with the same root-RCRO pulse, realizing an RCRO pulse with this zero-ISI property.

The four parameters for this MATLAB function were k_T, T_{sym}, s, and r. The above pulse $h(t)$ was then calculated for $-k_T T_b \leq t < k_T T_b$ with s samples per symbol, i.e., s samples per T_b time. The function returned two vectors, a vector of the pulse values, $h(t)$ and a vector of the corresponding times, t.

The root RCRO pulse generated in MATLAB was then plotted to verify the expected result. An example can be found in the results section of this document.

Random Bit Generation and Allocation to Carriers

The **`randi`** function in MATLAB was used to generate a vector of N random integers. Each integer represented a bit, taking on a value of either 0 or 1. For this simulation, N was required to be a multiple of 16 due to the system design of transmitting 16 bits at a time.

After the bits were generated, they were separated by groups of four into the bits that would be transmitted on each carrier. An array with four rows and N/4 columns was used to store this data. Each row of the array represented the bits that would be transmitted on each carrier. Figure 2 below depicts how the bits were rearranged from the vector **`bits_tx`** to the array **`bits_carrier_tx`**, for the case where N=32. The below pattern would continue for N=48, N=64, etc.

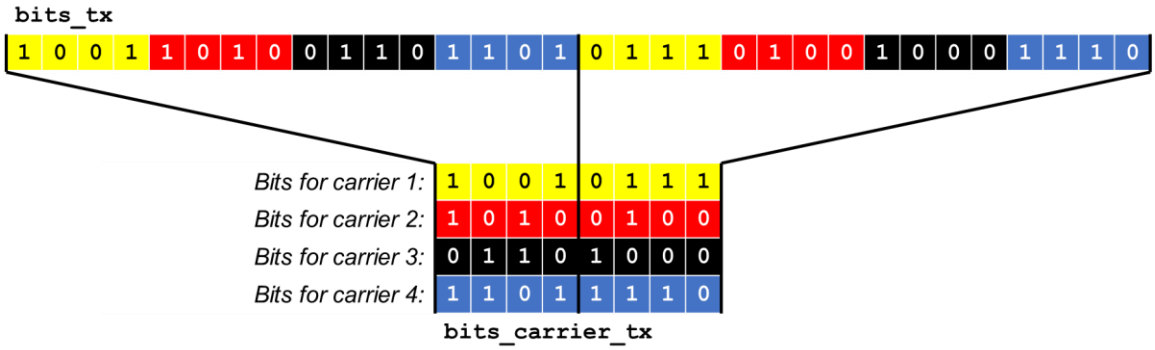

Figure 2: Rearrangement of bits

Generation of Symbols from Bits

Each group of four bits in the **`bits_carrier_tx`** array, shown above in figure 2, were then used to generate a symbol. Each symbol consisted of two values, an "I" value and a "Q" value. The I values were used to modulate a cosine wave at the carrier frequency, and the Q values were used to modulate a sine wave at the carrier frequency.

Groups of four bits were mapped to I and Q values as shown in figure 3 below [5].

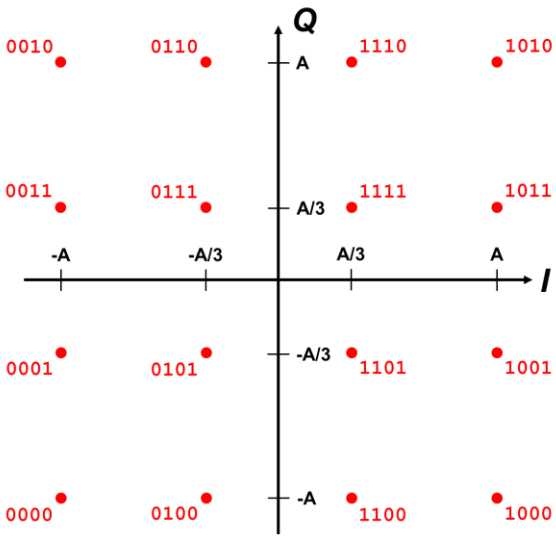

Figure 3: 16-QAM Constellation

The arrays **`carrier_i_tx`** and **`carrier_q_tx`** were used to store the I and Q values, respectively. Each of these arrays had four rows (one for each carrier) and N/16 columns. Figure 4 below depicts how the **`bits_carrier_tx`** array was mapped to **`carrier_i_tx`** and **`carrier_q_tx`** for N=32.

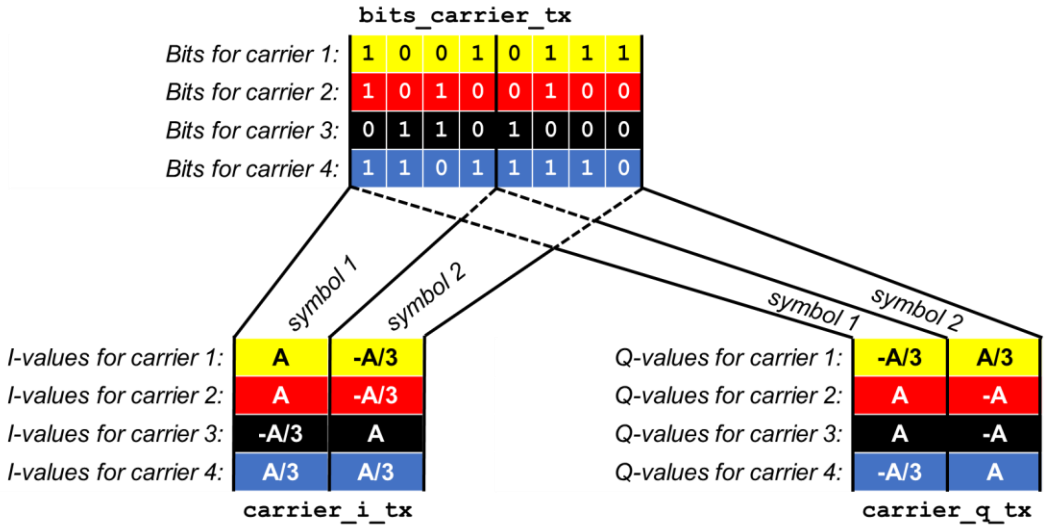

Figure 4: Mapping of bits to symbols

Synthesis of Baseband I-Q Signals

Baseband I and Q signals were then synthesized from the I and Q values by summing scaled, time-shifted root RCRO pulses. The following summations were performed for each carrier (the subscript c indicates the carrier):

$$b_{c,I}(t) = \sum_{n=1}^{N} I_c[n]h\big(t - T_{sym}[n-1]\big)$$

$$b_{c,Q}(t) = \sum_{n=1}^{N} Q_c[n]h\big(t - T_{sym}[n-1]\big)$$

These baseband I-Q signals were then plotted for each carrier to ensure the summation was being performed properly.

Modulation

The baseband I and Q signals for each carrier were then modulated according to the following equation. This modulation was performed four times, once for each carrier, c.

$$s_c(t) = b_{c,I}(t)\cos(2\pi f_c t) - b_{c,Q}(t)\sin(2\pi f_c t)$$

The signal that would hypothetically be transmitted was created by summing each of the modulated carriers. Note that each carrier frequency and the symbol period were selected such that there would be no spectral overlap between modulated carriers.

$$s(t) = \sum_{c=1}^{4} s_c(t)$$

Each modulated carrier and the sum of the modulated carriers were then plotted for visualization purposes.

Channel Model

The channel the signal passed through from transmitter to receiver was modeled by additive white gaussian noise. In continuous time, the received signal $r(t)$ would be equal to the transmitted signal $s(t)$:

$$r(t) = s(t) + n(t)$$

However, since this MATLAB simulation is in discrete time, the following line of code was used to add noise to the transmitted signal **s** to represent the received signal **r**:

```
r = s + sqrt(noiseVar)*randn(1, length(s));
```

`randn(1, length(s))` generates a vector that is the same length as the signal **`s`** and contains normally distributed random values with mean zero and standard deviation 1. Each element of this vector was then multiplied by the square root of the noise variance **`noiseVar`**. The **`noiseVar`** variable allowed the intensity of the noise to be changed easily for exploratory purposes.

The received signal was then plotted with various noise variances to visually verify that noise was being added.

I-Q Downconversion of Each Carrier

The recovery of the individual carriers from the received signal began with I-Q downconversion. For each of the four carrier frequencies, the following operations were performed in MATLAB. A plot was also generated for each downconverted carrier.

$$d_{c,I}(t) = 2r(t)\cos(2\pi f_c t)$$

$$d_{c,Q}(t) = -2r(t)\sin(2\pi f_c t)$$

Filtering of I-Q Downconverted Signals

The I-Q downconverted signals were then convolved with the root RCRO pulse to create RCRO pulses representing the initial transmitted symbols. Once again, the below calculations were performed for each of the four carriers, i.e., c = [1, 2, 3, 4]. A plot was also generated for each filtered signal.

$$y_{c,I}(t) = d_{c,I}(t) * h(t)$$

$$y_{c,Q}(t) = d_{c,Q}(t) * h(t)$$

Sampling Filtered Signals at the Symbol Times

The filtered signals were then sampled at each of the symbol times:

$$\tilde{I}_c[n] = y_{c,I}\big(T_{sym}[n-1]\big)$$

$$\tilde{Q}_c[n] = y_{c,I}\big(T_{sym}[n-1]\big)$$

A plot of each carrier's I-Q samples, with I on the horizontal axis and Q on the vertical axis, was then generated to visualize the constellations of the received signal.

Rounding Sampled I-Q Values

The sampled I-Q values for each received carrier were then rounded to the nearest expected value, i.e. -A, -A/3, A/3, or A/3. This was accomplished using the interp1 function for 1-D data interpolation [6].

First, rounding targets were defined:

```
roundTargets = [(-A) (-A/3) (A/3) (A)]
```

Then, the received I-Q values were interpolated to these targets:

```
carrier_i_rx = interp1(roundTargets, roundTargets, carrier_i_rx_raw,'nearest','extrap');
carrier_q_rx = interp1(roundTargets, roundTargets, carrier_q_rx_raw,'nearest','extrap');
```

Determining Bits for Each Carrier

The rounded I-Q values were then used to determine the bits for each symbol in each carrier, according to the constellation diagram in figure 3. These bits were placed in the array **`bits_carrier_rx`**, where each row represented one of the four carriers. This process for the example with N=32 is depicted in figure 5 below.

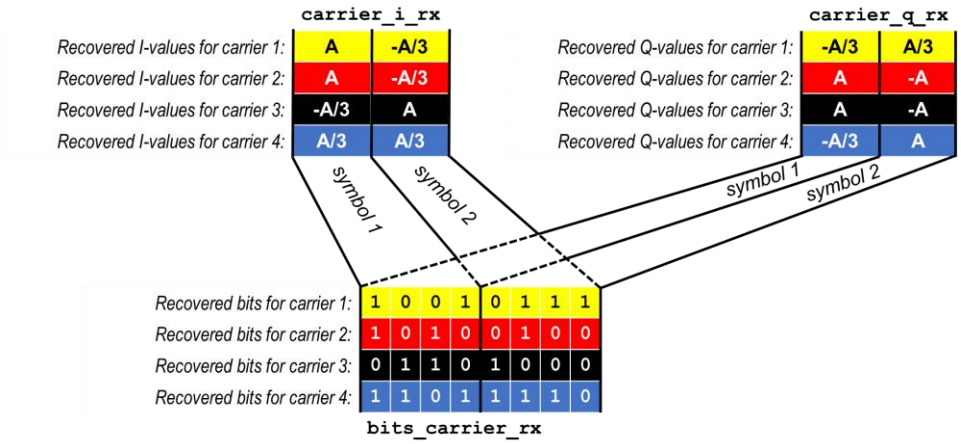

Figure 5: Mapping of rounded received I-Q values to bits

Combining Bits from Each Carrier Back into Single Bitstream

Lastly, the bits from each carrier were combined back to a single bitstream, as depicted in figure 6 below. A loop was used to synthesize the **`bits_rx`** vector from the **`bits_carrier_rx`** array.

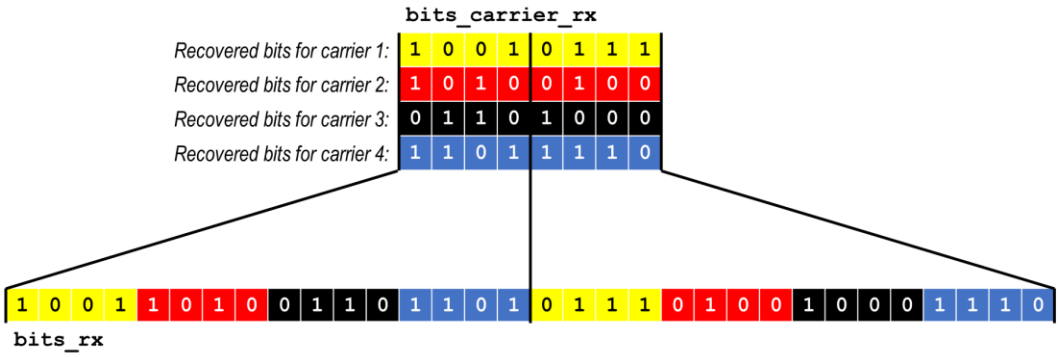

Figure 6: Recovering single bitstream

Plotting the Power Spectral Density and Ensuring Power Requirements Were Met

To calculate the power spectral density of the transmitted signal, initially the discrete Fourier transform (DFT) **S** of the transmitted signal **s** was found using MATLAB's **fft** function:

```
S = t_step .* fftshift( fft(s) );
```

The **fftshift** function shifts the DFT appropriately and multiplying each element by **t_step** scales the DFT appropriately.

The frequencies corresponding to this DFT are spaced **f_step = 1 / (length(S) * t_step);** apart, where **t_step** is the time between samples of the signal. The array of frequencies corresponding to the spectrum S was then defined as:

```
f = -f_step*(length(S)/2) : f_step : f_step*(length(S)/2-1);
```

An approximation to the power spectral density was then found by taking the absolute value and squaring each element of the DFT, and then dividing by the time in which data is transmitted.

```
P_s = (abs(S)).^2 / (N/16 * T_sym);
```

The frequencies **f** corresponding to this power spectral density array **P_s** are the same as those that correspond to the spectrum **S**.

This power spectral density was computed over many trails of random bits and averaged to generate a plot for power calculation purposes. The flowchart in figure 7 summarizes how this average PSD was computed.

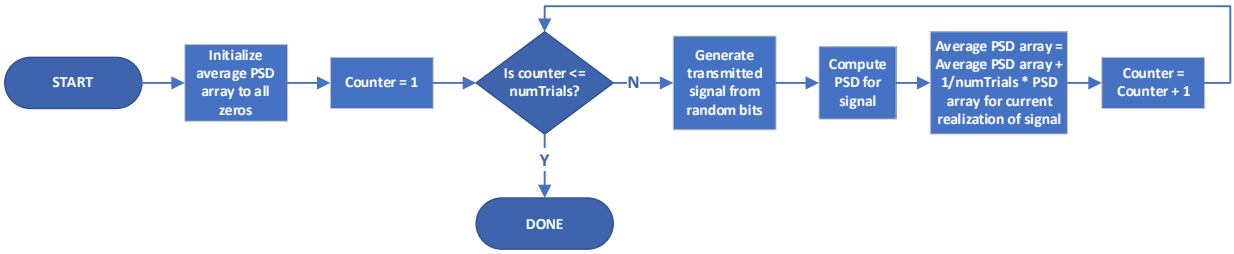

Figure 7: Flowchart for computation of average PSD

To verify the out-of-band emissions requirements of this simulation were met (20 dB below the in-band PSD level at the band edges and 55 dB below the in-band PSD level 3 kHz above and below the band edges), the power spectral density was calculated in decibels and then plotted:

```
P_s_dB = 10.*log10(P_s);
```

To verify that the in-band power was below one watt, as required for this simulation, a Riemann sum was used in MATLAB to approximate the integral of the PSD across the band (5 kHz to 15 kHz). Since only positive frequencies were considered for this calculation, and the PSD for any real signal is even symmetric, this integral was multiplied by 2 to find the total in-band power.

$$P_{in-band} = 2\int_{5000}^{15000} P_s(f)\,df$$

In MATLAB, this integral was implemented as a Riemann sum using the following code:

```
P_inband = 0;
for k = 1:1:length(P_s) % calcuate in-band power using Riemann sum
    if (f(k) >= 5000 && f(k) <= 15000)
        P_inband = P_inband + 2*P_s(k) * f_step;
    end
end
```

The value A, which scales the root RCRO pulses, was then adjusted experimentally so that the transmitted in-band power was just under one watt.

Bit Error Rate Calculation

A common measure for signal-to-noise ratio is $E_b \,/\, N_0$. E_b represents the average energy per bit, and N_0 represents the variance of the noise samples. The probability of a bit error, P_e can be found as a function of $E_b \,/\, N_0$ in literature for various modulation schemes.

To calculate E_b, the total energy of the signal was calculated and then divided by N, the number of bits. In continuous time, this would be calculated by the integral:

$$E_b = \frac{\int_{-\infty}^{\infty} s^2(t)\, dt}{N}$$

In MATLAB, this energy calculation was implemented with the following code:

```
% DETERMINE AVERAGE ENERGY PER BIT IN SIGNAL
        E = 0;
        for k = 1:1:length(s) % calcuate total energy in the signal using Riemann sum
            E = E + (s(k))^2*(t_step);
        end
        Eb = E/N; % obtain energy per bit by dividing the total energy by the number
of bits
```

To generate an experimental $E_b \,/\, N_0$ versus P_e curve, a range of $E_b \,/\, N_0$ values were predefined in an array.

```
Eb_N0_array_dB = linspace(Eb_N0_dB_min, Eb_N0_dB_max, numPoints); % Define Eb/N0
array in decibels

Eb_N0_array = 10.^(Eb_N0_array_dB./10); % Convert Eb/N0 array to absolute units
```

An array to store the corresponding values of bit error rates (i.e., the experimental realization of P_e) was also defined.

```
BER_array = zeros(1, length(Eb_N0_array)); % create vector for bit error rates, same
length as Eb/N0 vector
```

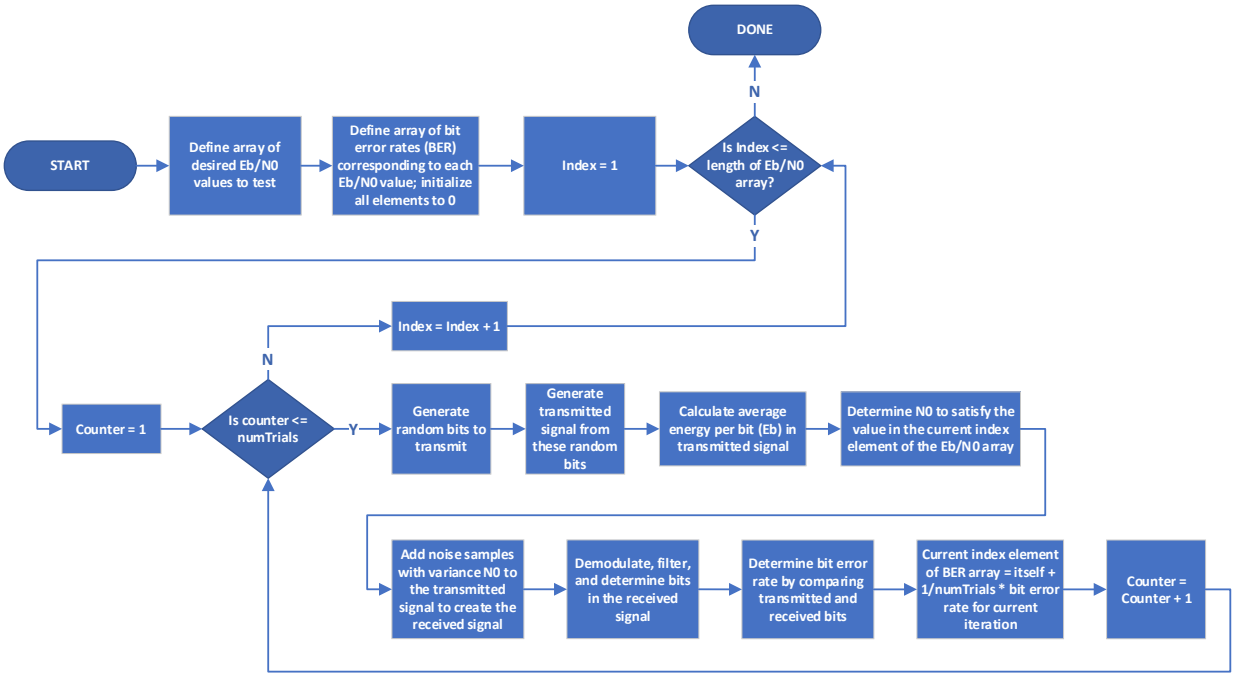

Figure 8: Flowchart for calculation of bit error rates

Nested loops were then used to run the simulation repeatedly, as shown in figure 8, using the value of each element of the **`Eb_N0_array`** multiple times (with the exact amount defined by the variable **`numTrials`**). For each transmitted signal, after **`Eb`** was calculated using the Riemann sum, the variance **`N0`** to use for the noise samples was calculated by **`N0 = 1/(Eb_N0)*Eb`**, where **`Eb_N0`** was the current element of the **`Eb_N0_array`**.

The bit error rate for each trial was calculated with the code **`mean(bits_tx ~= bits_rx)`** [7]. **`bits_tx ~= bits_rx`** generates an array in which the corresponding elements are 1 where **`bits_tx`** and **`bits_rx`** differ and 0 where they are the same. The mean of this array would thus be the bit error rate.

For each element in the **`Eb_N0_array`**, 200 simulations were carried out and the bit error rates were averaged and then stored to the **`BER_array`** of the same length. The **`Eb_N0_array`** was plotted against the **`BER_array`**, and this plot was compared to the theoretical E_b / N_0 versus P_e curve for 16-QAM using a Gray-coded constellation [8]:

$$P_e = \frac{3}{8}\mathrm{erfc}\left(\sqrt{0.4E_b / N_0}\right) + \frac{1}{4}\mathrm{erfc}\left(3\sqrt{0.4E_b / N_0}\right) - \frac{1}{8}\mathrm{erfc}\left(5\sqrt{0.4E_b / N_0}\right)$$

Equivalently, written in terms of the Q function:

$$P_e = \frac{3}{4}Q\left(\sqrt{0.8E_b / N_0}\right) + \frac{1}{2}Q\left(3\sqrt{0.8E_b / N_0}\right) - \frac{1}{4}Q\left(5\sqrt{0.8E_b / N_0}\right)$$

Results

Root-Raised Cosine Pulse Generation

The root RCRO pulse was plotted to ensure it was being generated correctly. Examples of this pulse for four different values of r with $T_{sym} = 1 \cdot 10^{-3}$, $k_T = 5$, and $s = 64$ are shown in figure 9.

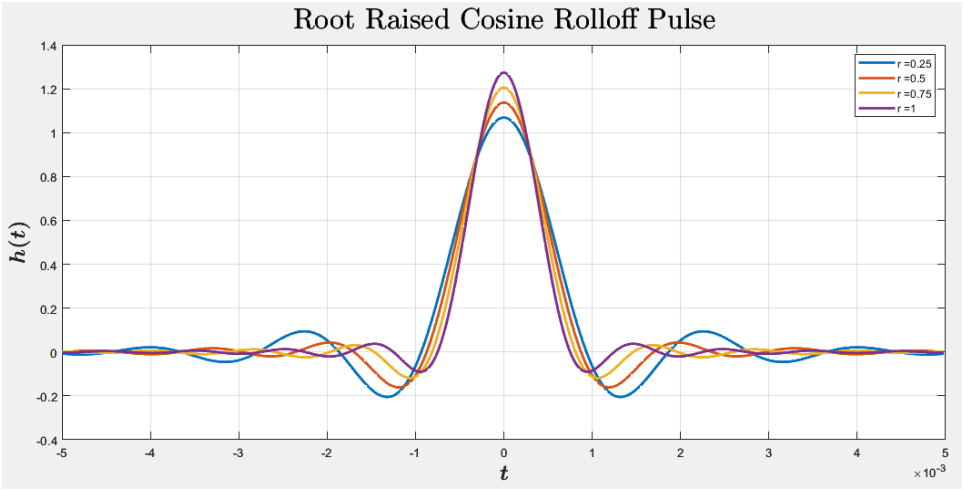

Figure 9: Root RCRO Pulses

Synthesis of Baseband I-Q Signals

The baseband I and Q signals, before transmission, representing the example data discussed in the procedure section are shown below in figure 10. For this calculation, $r = 0.5$, $A = 0.67$, and $T_{sym} = 6 \cdot 10^{-4}$ s.

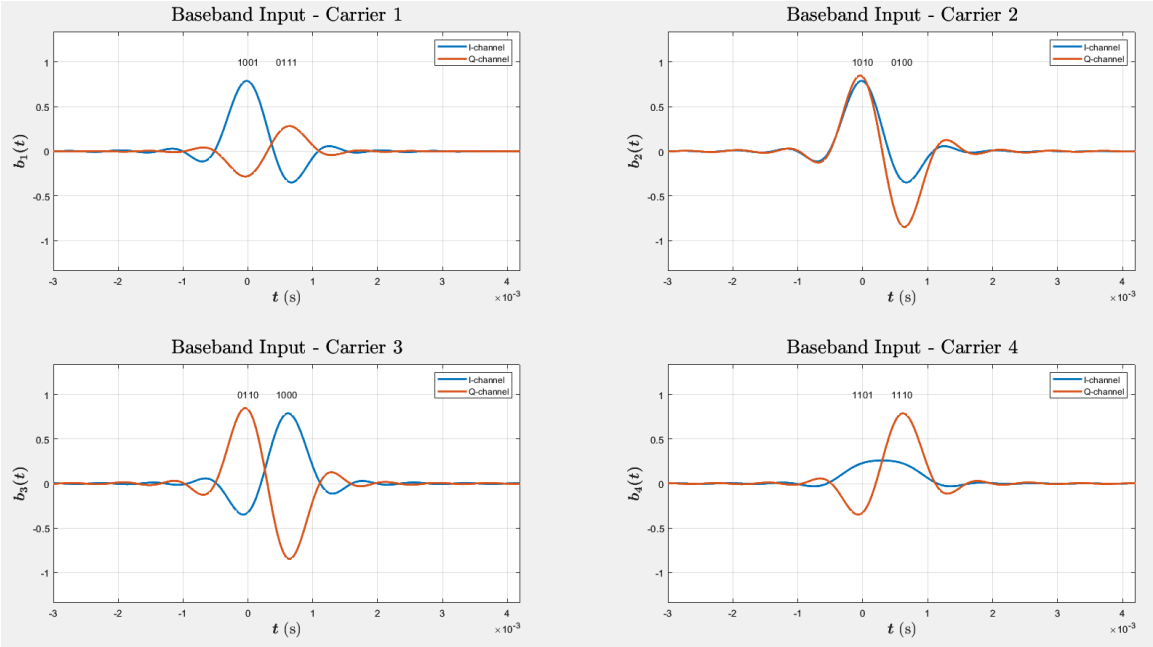

Figure 10: Baseband I and Q signals for Example with N=32 bits

Modulation

Figure 11 below depicts each modulated carrier for the example discussed in the procedure section of this document. Figure 12 depicts the sum of these carriers, representing the transmitted signal.

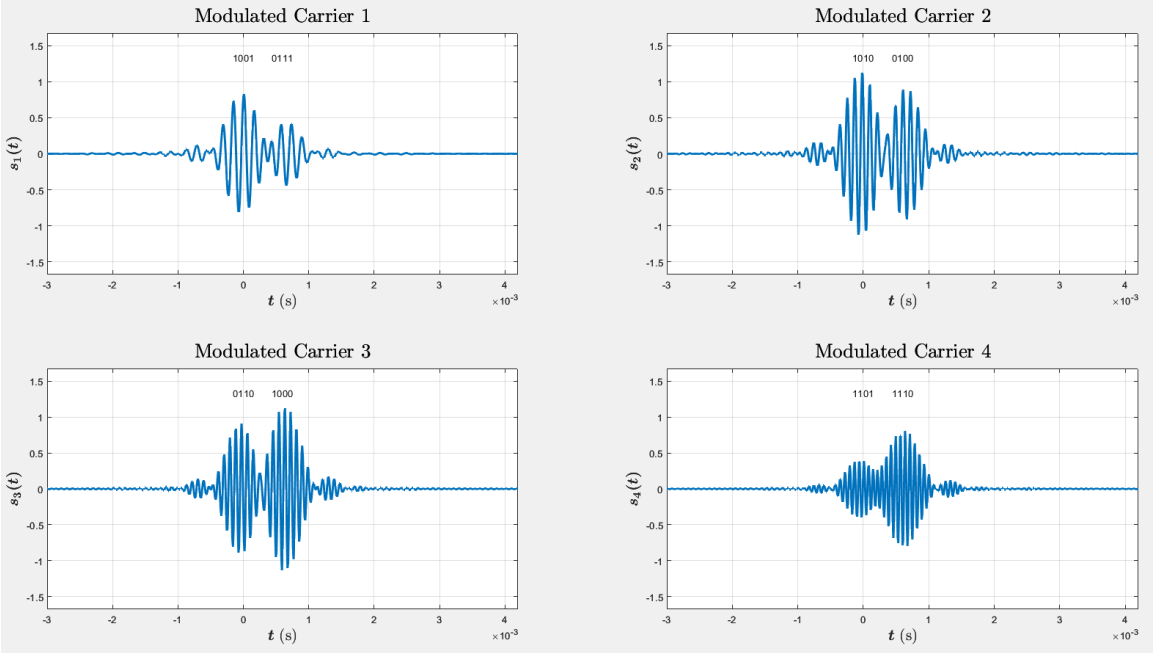

Figure 11: Modulated Carriers for Example with N=32 bits

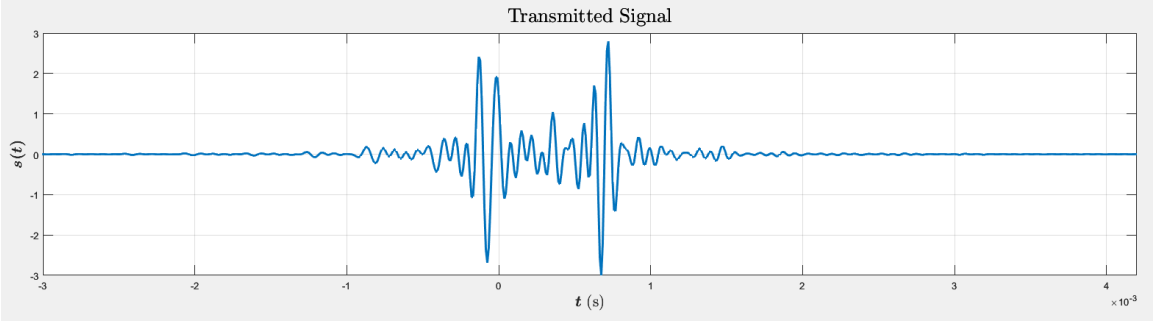

Figure 12: Transmitted Signal (Sum of Modulated Carriers)

Channel Model

Figures 13 through 15 depict the received signal as increasing levels of noise were added. When there was no noise, as in figure 13, the received signal was identical to the transmitted signal. However, as the noise was increased the received signal resembled the transmitted signal less and less.

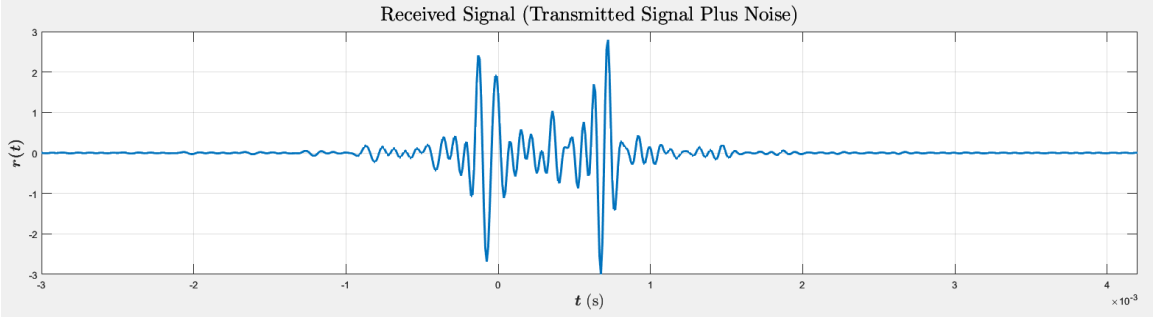

Figure 13: Received Signal when ***`noiseVar = 0`***

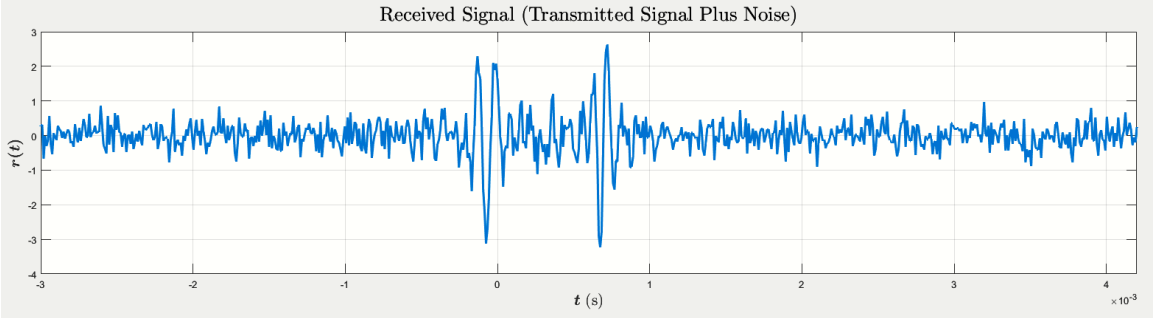

Figure 14: Received Signal when ***`noiseVar = 0.1`***

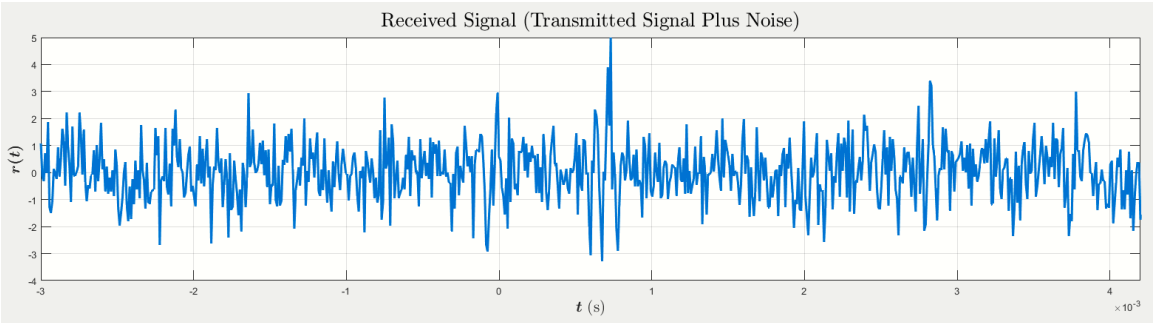

Figure 15: Received Signal when ***`noiseVar = 1`***

I-Q Downconversion of Each Carrier

Figures 16-18 below depict the result of I-Q downconversion for each carrier for increasing amounts of noise in the received signal.

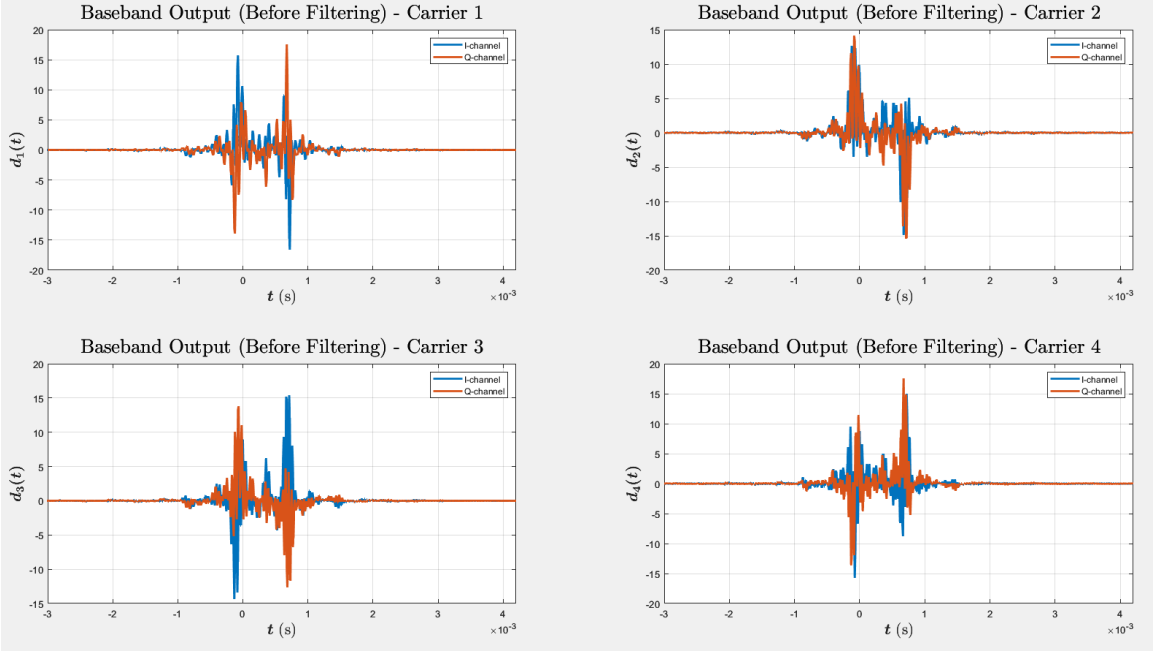

Figure 16: I-Q Downconverted Carriers Before Filtering when ***`noiseVar = 0`***

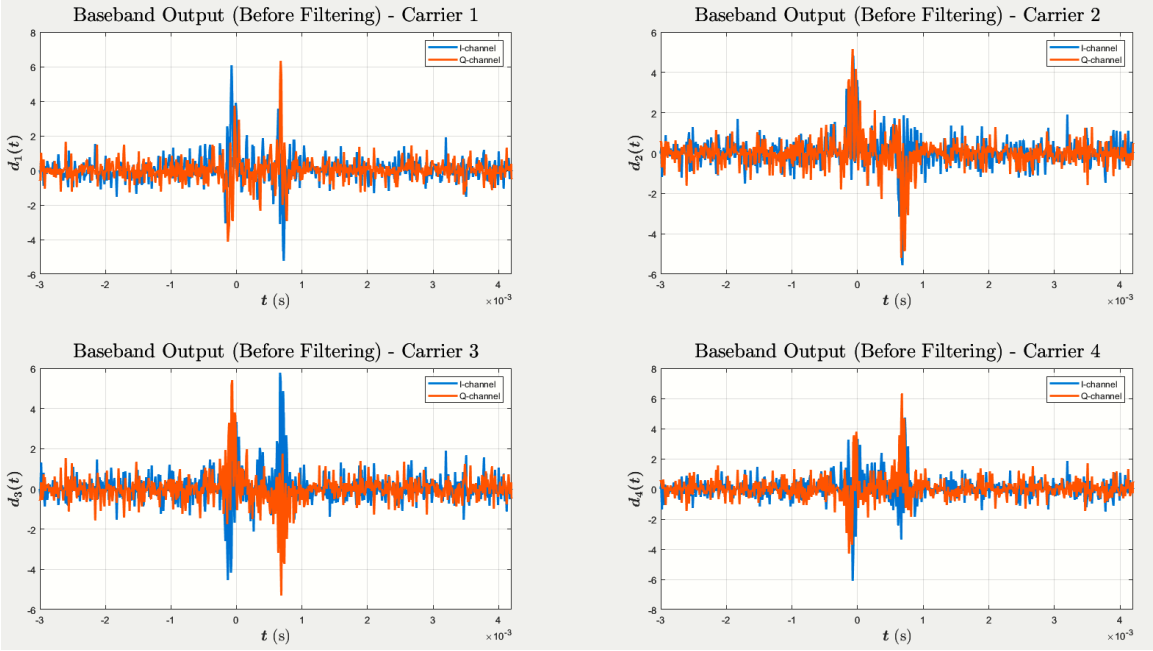

Figure 17: I-Q Downconverted Carriers Before Filtering when ***`noiseVar = 0.1`***

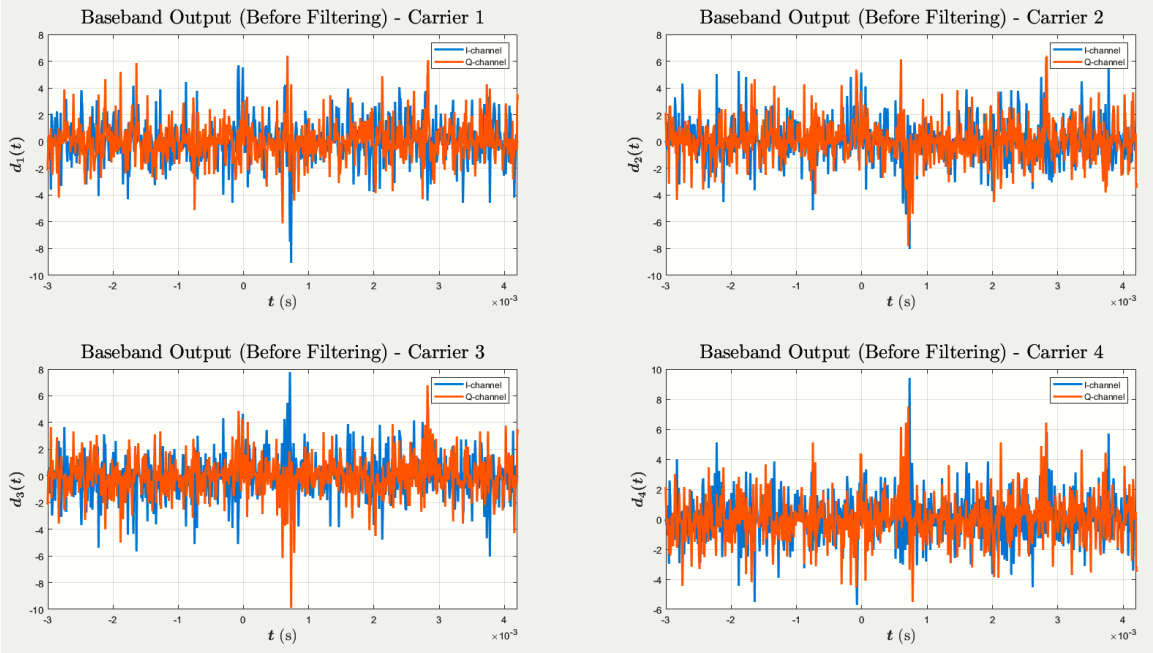

Figure 18: I-Q Downconverted Carriers Before Filtering when ***noiseVar = 1***

Filtering of I-Q Downconverted Signals

Figure 19 below depicts the I-Q downconverted signals after filtering. Because no noise was added for this simulation example, these were RCRO pulses representing the transmitted data.

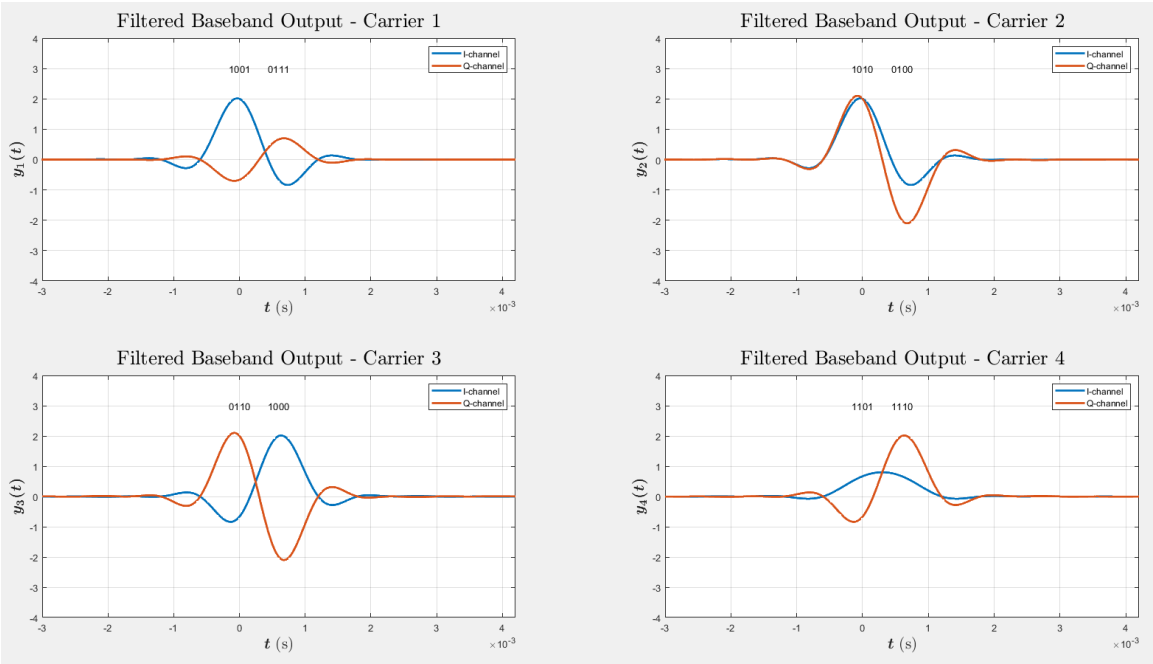

Figure 19: I-Q Downconverted Carriers After Filtering when ***noiseVar = 0***

When a small amount of noise was present in the received signal, as in figure 20, the filtered baseband output no longer consisted of perfect RCRO pulses. However, in this case there was not enough noise to cause any symbol errors.

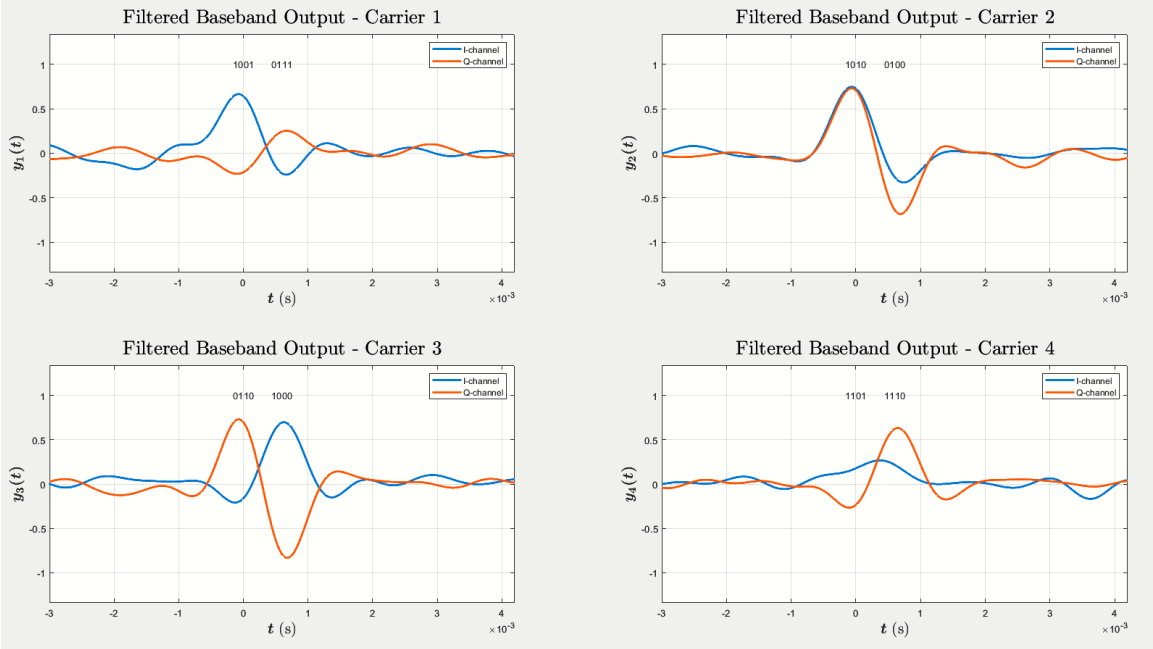

Figure 20: I-Q Downconverted Carriers After Filtering when ***`noiseVar = 0.1`***

When the noise in the received signal was increased further, the filtered baseband outputs resembled RCRO pulses even less, as shown in figure 21. Additionally, symbol errors (and thus bit errors) began to occur. Incorrectly received symbols are marked in red in figure 21. In this particular realization, three symbol errors and four bit errors were made. Thus, there was a symbol error rate of 3/8 (37.5%), but a bit error rate of only 4/32 (12.5%).

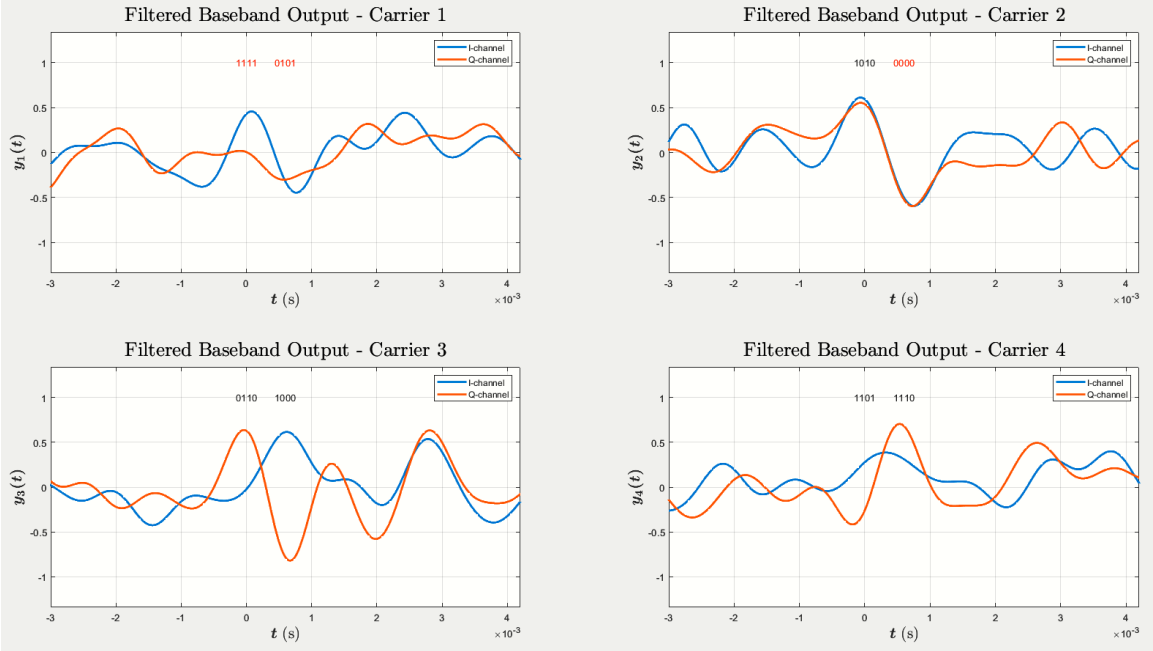

Figure 21: I-Q Downconverted Carriers After Filtering when ***noiseVar = 1***

Sampling Filtered Signals at the Symbol Times

Figures 22-24 below show the received signal constellations with **A = 0.67** and increasing noise variances. 8000 randomly generated bits (500 symbols/carrier) were used to produce these constellations.

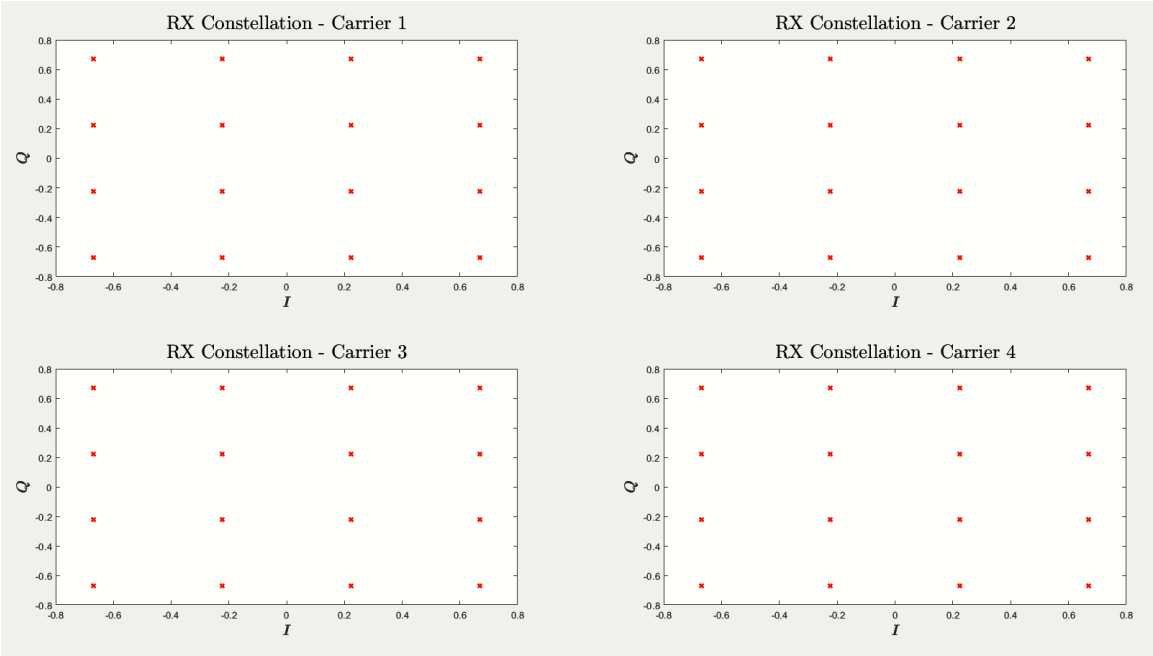

Figure 22: Received Signal Constellations with ***noiseVar = 0***

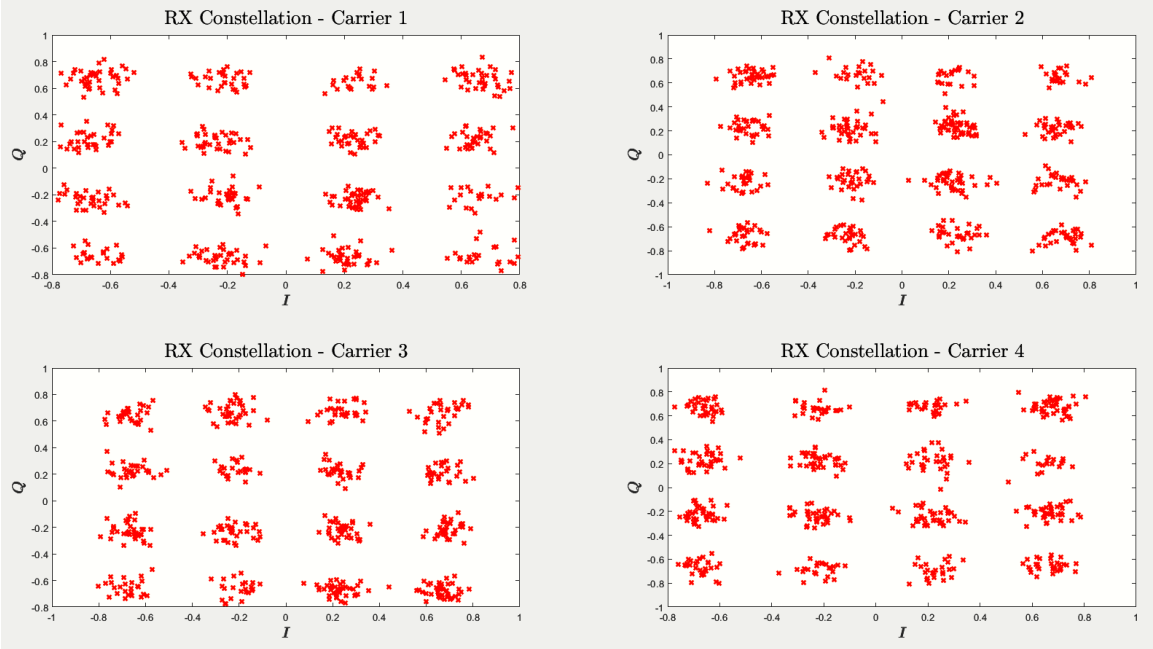

Figure 23: Received Signal Constellations with ***`noiseVar = 0.1`***

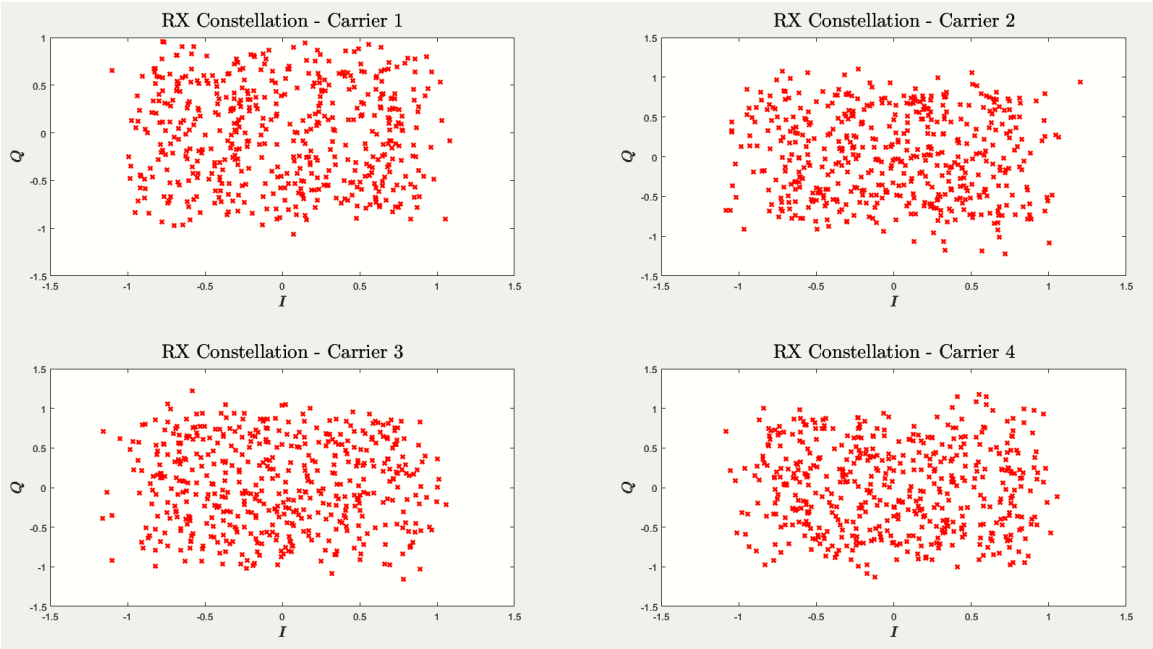

Figure 24: Received Signal Constellations with ***`noiseVar = 1`***

Plotting the Power Spectral Density and Ensuring Power Requirements Were Met

The power spectral density plots in figure 25 were calculated from an average of 10,000 realizations of the signal with 800 bits in each realization.

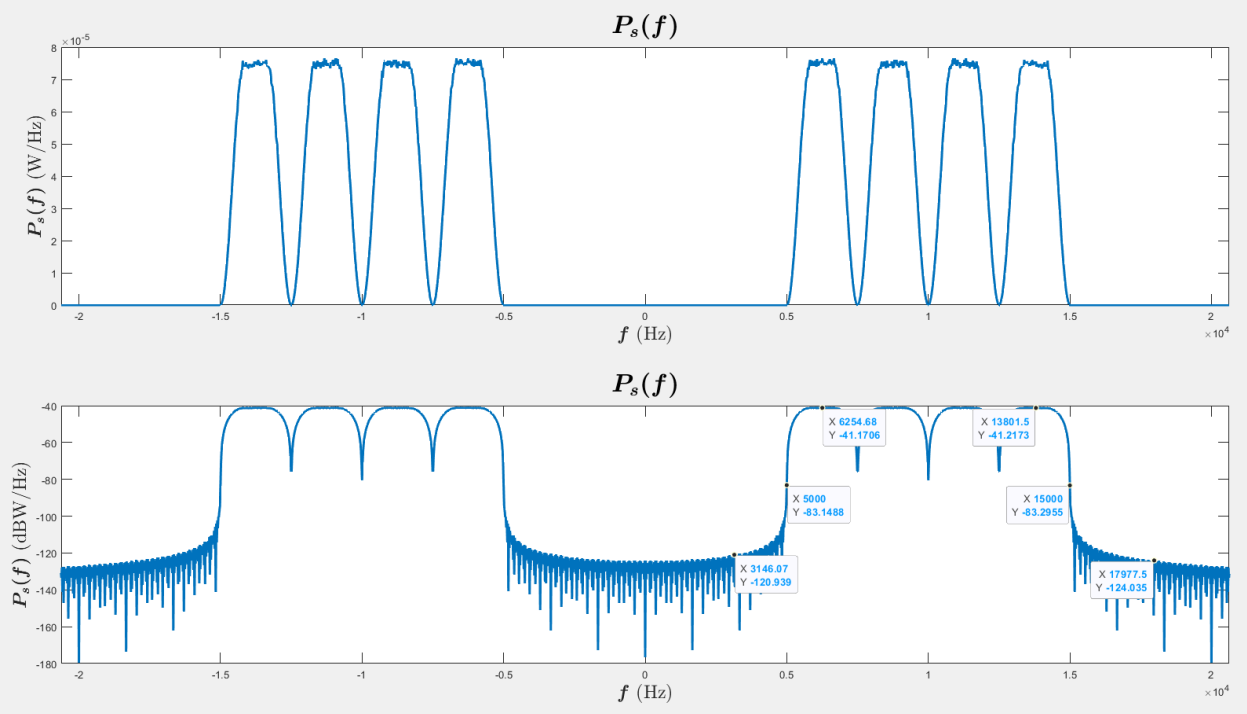

Figure 25: Average Power Spectral Density of Transmitted Signal

From the plot markers in figure 24, it was observed that the power spectral density (PSD) at the band edges (5 kHz and 15 kHz) was about -83 dBW/Hz, 42 dB below the peak PSD of about -41 dBW/Hz within the band. Furthermore, for frequencies more than 3 kHz outside the band, the PSD was more than 80 dB below the peak.

Additionally, it was found that an amplitude value for the pulses of **`A = 0.67`** produced an in-band signal power just under the 1 watt limit. The following values were calculated by the Riemann sum in the MATLAB script:

```
Total signal power (W) = 0.99843
Normalized in-band signal power (W) = 0.99843
```

Not that the total signal power is effectively equal to the in-band signal power, because there is so little power outside the band.

Bit Error Rate Calculation

Figure 26 shows the actual bit error rate versus the expected bit error rate (P_e) for a range of $E_b \,/\, N_0$. For the actual bit error rate, the average of 200 simulations was taken for each selected $E_b \,/\, N_0$ value. 800 bits were used per simulation.

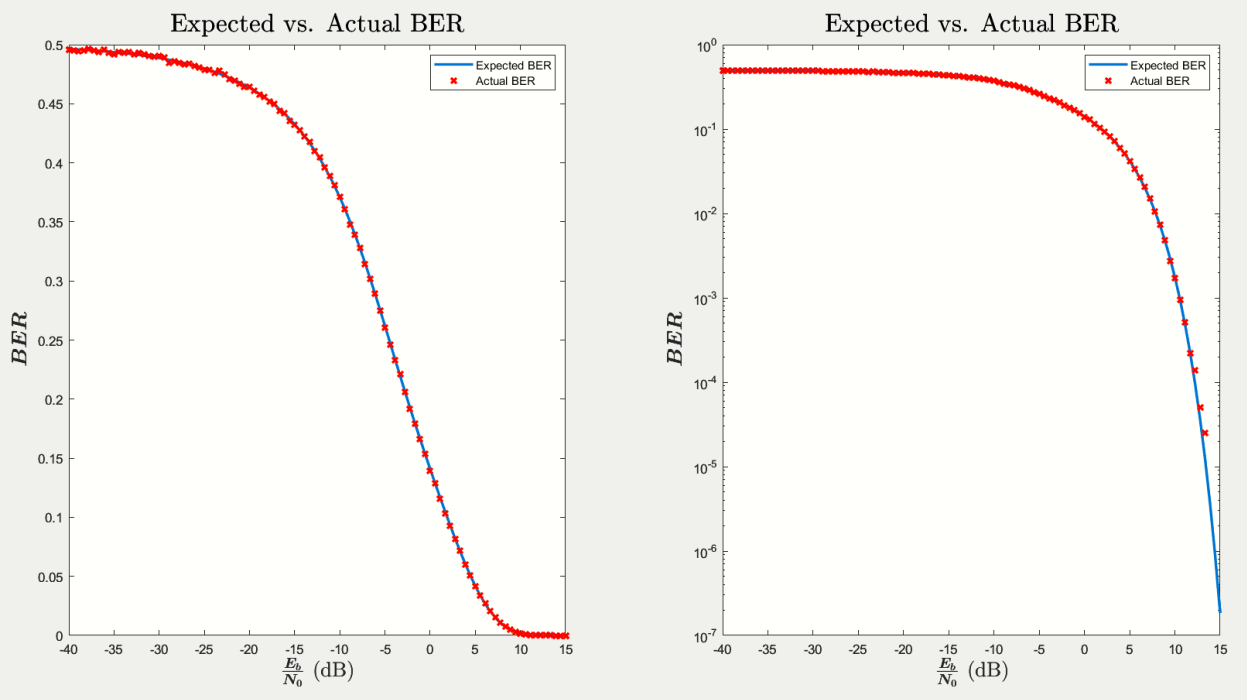

Figure 26: Actual Versus Expected Bit Error Rate

Discussion

Root-Raised Cosine Pulse Generation

The root raised cosine rolloff pulses plotted in figure 9 appeared to be correct. This data was obtained to ensure that all cases of the equation were being handled and the pulses were being plotted correctly, since root-RCRO pulses formed the foundation of this simulation.

Synthesis of Baseband I-Q Signals

The baseband I-Q signals in figure 10 also appeared to be synthesized correctly. The pulses were being time shifted and added correctly. These signals were plotted because generating them proved to be one of the most challenging parts of this simulation, and careful attention to array indexing was required to perfect this step.

Modulation

Each modulated carrier and the overall transmitted signal were plotted in figures 11 and 12 to ensure that they appeared to be correct. It was noted that the carrier frequencies appeared to be visibly different, with carrier 1 at the lowest frequency and carrier 4 at the highest frequency. The modulated carriers were not supposed to interfere with each other in the frequency domain, which can be deduced but not verified from these plots.

Channel Model & I-Q Downconversion of Each Carrier

In figures 13-18, the received signal and downconverted carriers with various levels of noise were plotted to verify that the noise was being added correctly. This appeared to be the case, since larger noise variances rendered received signals in which the transmitted signal was less recognizable.

Filtering of I-Q Downconverted Signals

To visualize the effects of matched filtering, the filtered baseband outputs were plotted in figures 19-21. What was surprising was how well the filtering worked to suppress the noise. Although the received signal had a visible amount of noise with a noise variance of 0.1, the filtered outputs looked decent, and no bit errors occurred. When the noise variance was increased to 1, however, the filtered outputs looked noticeably worse, and some errors did occur. Two of the three incorrect symbols had only one bit error, and the third had two bit errors. Since a gray-coded constellation was used for signaling, in which neighboring constellation points differed by one bit, one-bit symbol errors were expected to be the most common.

Sampling Filtered Signals at the Symbol Times

When the noise variance was set to 0, the constellations (see figure 22) looked exactly like the theoretical constellation for 16-QAM shown in figure 3. As the noise variance was increased in figures 23 and 24, the constellation points spread out from their theoretical values, as expected.

Plotting the Power Spectral Density and Ensuring Power Requirements Were Met

The power spectral density (PSD) averaged over a large number of trials, as shown in figure 25, matched what was expected. It was noted that the carriers did not appear to overlap, since a null was observed between each one on the PSD plot. Additionally, the power at the band edge and outside the band satisfied the requirements for this simulation (at least 20 dB below the in-band PSD at the band edge and 55 dB below the in-band PSD >3 kHz above or below the band edge). Furthermore, a pulse amplitude was set such that the total in-band power did not exceed the maximum of one watt, as calculated by a Riemann sum in MATLAB to approximate integrating over the PSD.

Bit Error Rate Calculation

The average observed bit error rate appeared to depend on $E_b \ / \ N_0$ and closely followed the curve for the theoretical relationship, as displayed in figure 26.

These results confirmed that it is possible to model a communication system in MATLAB that conforms to theoretical performance standards for additive white gaussian noise. My findings agreed with relationships found in literature for 16-QAM signaling and RCRO pulses. Additionally, I now have a better understanding of how communication systems may be implemented with DSP operations. The next step in this study would be to make the system reflect a real-world scenario more closely. One way to do this would be to add a time delay to the received signal and require the receiver to determine the bit times without knowing what this delay was. Multipath interference could also be modeled by making the received signal consist of a sum of a few transmitted signals shifted in time. Furthermore, frequency selective fading could be added to the channel model and the receiver could be improved to compensate for it.

Conclusions

The most important lesson learned in this experiment was how to model a digital communication system using MATLAB functions and operations. This study included visualization of power spectral density and bit error rate, which are both very important concepts in communications theory. The experiment also confirmed mathematical relationships found in literature for the type of digital signaling that was used.

A positive result that surprised me was how well the matched filtering process worked against noise. This should not have been too surprising, however, since a matched filter is an important part of an optimal receiver. There were no negative results from this experiment; all results were consistent with what was expected. Throughout this project, I learned a lot more about MATLAB, particularly regarding array and matrix operations. I concluded that MATLAB was a suitable software tool for this experiment and my design functioned as expected.

References

[1] Dennis, H. (2018, October 23). *Wi-Fi 6 fundamentals: What is 1024-QAM?* CommScope. Retrieved April 20, 2023, from https://www.commscope.com/blog/2018/wi-fi-6-fundamentals-what-is-1024-qam/

[2] Reis, K. (2023, January 3). *Modulation schemes, coding rates, and 4G/5G data speeds.* Waveform. Retrieved April 20, 2023, from https://www.waveform.com/a/b/guides/modulation-coding-speeds

[3] Al-Banna, A., & Cloonan, T. (2014). *The spectral efficiency of DOCSIS 3.1 systems.* CommScope. Retrieved April 20, 2023, from https://www.commscope.com/globalassets/digizuite/1650-arris-spectral-efficiency-of-docsis-wp.pdf

[4] Viswanathan, M. (2018, October 15). *Square-root raised-cosine pulse shaping*. GaussianWaves. Retrieved August 30, 2023, from https://www.gaussianwaves.com/2018/10/square-root-raised-cosine-pulse-shaping/

[5] Sankar, K. (2012, September 12). *Binary to Gray code for 16QAM*. DSP Log. Retrieved April 9, 2023, from http://www.dsplog.com/2008/06/01/binary-to-gray-code-for-16qam/

[6] *How do I round to the nearest arbitrary, enumerated value in Matlab?* MATLAB Answers - MATLAB Central. (n.d.). Retrieved April 9, 2023, from https://www.mathworks.com/matlabcentral/answers/9641-how-do-i-round-to-the-nearest-arbitrary-enumerated-value-in-matlab

[7] *How to count the number of different elements in two matrices.* MATLAB Answers - MATLAB Central. (n.d.). Retrieved April 9, 2023, from https://www.mathworks.com/matlabcentral/answers/429897-how-to-count-the-number-of-different-elements-in-two-matrices

[8] Bune, P. A. M. (2008, April 28). *Exact ber for gray-coded 2^2N-QAM modulation with AWGN*. MathWorks. Retrieved April 9, 2023, from https://www.mathworks.com/matlabcentral/fileexchange/19717-exact-ber-for-gray-coded-2-2n-qam-modulation-with-awgn