Lab 3 - OFDM Simulation

Pranavi Boyalakuntla
Principles of Wireless
Spring 2021

I. Introduction

One of the downsides to a single carrier communication system is that the symbol period must be high in order to avoid inter-symbol interference (ISI). When the symbol period is higher, the data rate of the system is lower.

One way of increasing data rate is using a multiple carrier communication system. Using more than one carrier allows for information to be transmitted at multiple frequency intervals at the same time. In order to ensure there is no intercarrier interference (ICI), guard bands, or unused bandwidth between channels, are used. Note that guard bands are not unique to multicarrier systems. This partitioning scheme is called Frequency Division Multiplexing (FDM). Fig. 1 shows a multiple carrier communication system using OFDM.

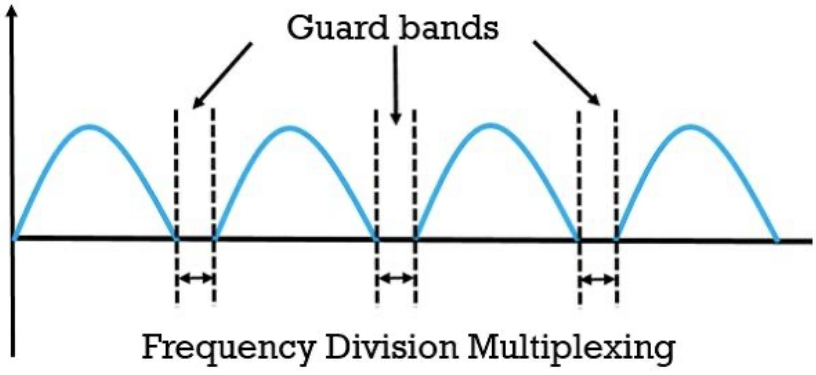

Fig. 1: A multiple carrier communications system that utilizes guard bands.

The use of guard bands requires that extra bandwidth be available. In the case where bandwidth is more limited, orthogonal frequency division multiplexing (OFDM) can be used.

OFDM utilizes the concept of orthogonality to allow channels to overlap while minimizing ICI. When two channels are orthogonal to one another, their zero crossings align and the peak of one channel does not interfere with the peak of another channel. In Fig. 2, each channel is represented by a sinc function in the frequency domain. Each zero crossing aligns. This

means that the channels are orthogonal. The peak of one channel also exists over lower values of other channels. Although these channels overlap, the ICI is minimized by maintaining orthogonality.

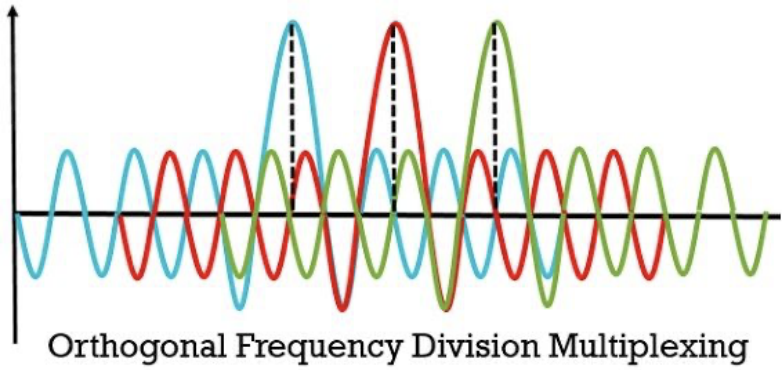

Fig. 2: Orthogonal channels that overlap.

OFDM uses the available bandwidth more efficiently than FDM as there are no unused portions of the spectrum. As a result, OFDM is used in popular communication systems like WiFi and LTE.

The goal of this project is to implement an OFDM system on a Universal Serial Radio Protocol (USRP). This paper will cover the three steps that were taken to reach this goal:

1. Simulation of a timing synchronized OFDM communication system.
2. Simulation of an OFDM system with timing errors.
3. Implemented OFDM system using the USRP.

II. Overview of OFDM

A. Flat-Fading Assumption

The frequency flat-fading assumption means that the channel's transfer function can be assumed to be constant over a frequency band. Assuming that all frequencies in a region fade in the same way is advantageous because all frequencies can be demodulated in the same way; it is known how to adjust each distinct region.

Most channels do not allow the flat-fading assumption to be made across the entire channel. However, when looking at a narrower frequency band, the channel effects vary in smaller degrees . This allows the flat-fading assumption to hold true. In Fig. 3, it is shown how given any channel response, flat-fading can be assumed if it is divided into smaller portions. The x axis is frequency and the y axis is signal amplitude.

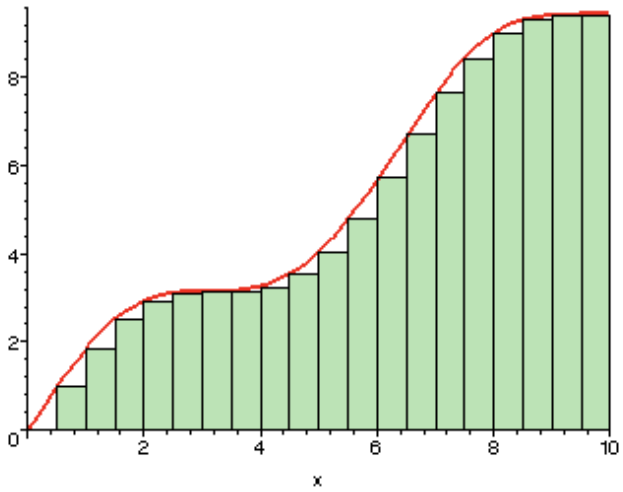

Fig. 3: The frequency response of a channel chunked into smaller portions to uphold the flat-fading assumption.

B. The Discrete Fourier Transform

Orthogonality in the frequency domain is used to send more information. Since data is sent across multiple frequencies at the same time, each block starts in the frequency domain. In order to be sent, this data is converted to the time domain using the Fourier transform.

The Fourier transform is a mathematical operation that transforms a time domain signal to the frequency domain. This is done by breaking a signal down into its frequency components. The Fourier transform is closely related to the Fourier series which is the expansion of a periodic function into multiple frequencies.

The definition of the Fourier series is shown in Eq. 0 where ω_0 is the fundamental frequency and t is time. The definition of c_n is shown in Eq. 1.

$$x(t) = \sum_{n=-\infty}^{+\infty} c_n e^{jn\omega_o t} \tag{0}$$

$$\text{where } c_n = \frac{1}{T}\int_T x(t)e^{-jn\omega_0 t}dt \tag{1}$$

Eq. 0 can be rearranged into Eq. 2 as shown below.

$$Tc_n = \int_T x(t)e^{-jn\omega_0 t}dt \tag{2}$$

In fact, the Fourier transform is the limit of the Fourier series as $T \to \infty$. As $T \to \infty, \omega_0 \to 0$ and $n\omega_0$ becomes a constant quantity equal to ω. By substitution, the definition of the Fourier transform can be derived as shown in Eq. 3. It can be seen that the Fourier transform of a continuous signal in time will be continous in frequency as well.

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt \tag{3}$$

Conversely, the Inverse Fourier transform of a continuous signal in the frequency domain will be continuous in time. However, when implemented in hardware, continuous signals are sampled and transmitted discretely in time. The Discrete Time Fourier transform (DTFT) of a discrete-time signal is continuous in the frequency domain. The definition of the DTFT is shown in Eq. 4. Comparing Eq. 4 with Eq. 3 shows that the DTFT is a time discrete version of the Fourier Transform.

$$X\left(e^{j\Omega}\right) = \sum_{\ell=-\infty}^{\infty} x[\ell]e^{-j\Omega} = \sum_{\ell=0}^{L-1} x[\ell]e^{-j\Omega} \tag{4}$$

In the case of OFDM, the frequency domain needs to be divided to uphold the flat-fading assumption. As a result, the signal will be discrete in the frequency domain. The Inverse Discrete Fourier Transform (IDFT) is a mathematical operation that takes a discrete signal in the frequency domain and returns a discrete signal in the time domain. The L-point IDFT is defined in Eq. 5.

$$x[m] = \frac{1}{L}\sum_{k=0}^{L-1} X_k e^{2\pi j \frac{mk}{L}} \tag{5}$$

Conversely, the Discrete Fourier Transform (DFT) takes a discrete signal in the time domain and converts it to a discrete signal in the frequency domain. The definition of the DFT is shown in Eq. 6. Comparing Eq. 6 to Eq. 4 shows that the DFT is a sampled version of the DTFT.

$$X_k = \sum_{\ell=0}^{L-1} x[\ell]e^{-2\pi j \frac{\ell k}{L}} \tag{6}$$

In using the DFT and IDFT in the OFDM algorithm, there are several properties that will be helpful.

- **Linearity of the DFT:** The additive and scaling relationships between signals are maintained after the DFT is taken.
- **Periodicity of the DFT:** The L-point DFT is periodic with period L.
- **Circular Convolution in Time Become Multiplication in Frequency: T**he relationship between circular convolution and multiplication forms the basis of OFDM.

C. Mechanics Behind OFDM

The OFDM algorithm uses orthogonality to transmit more information across one unit of time. Therefore, the data in the OFDM algorithm begins in the frequency domain. It is important to note that since this information represents a snapshot of what is sent across multiple channels in one unit of time, time acts like a third dimension.

In order to send this information, it needs to be converted to the time domain using the IDFT. The DFT and IDFT hold the property that circular convolution in the time domain becomes multiplication in the frequency domain. However, when transmitting a signal through air, it is convolved with the channel impulse response. In order to maintain that transmitting a signal over the air is equivalent to multiplication in the frequency domain with the channel frequency response, the system needs to perform circular convolution.

Circular convolution is a special case of periodic convolution where two periodic functions with the same period are convolved. To imitate periodicity, the signal can be prepended and postpended with itself ad infinitum. This will then "trick" the air into performing circular convolution instead of regular convolution.

To convert the signal back into the frequency domain, the DFT is taken.

Now, the signal is processed back into an estimate what was originally sent, $\hat{X_k}$. Eq. 7 describes the relationship between the received signal in the frequency domain, Y_k, the channel response, H_k, and the sent signal in the frequency domain, X_k.

$$Y_k = H_k X_k \tag{7}$$

Rearranging Eq. 7 results in Eq. 8 which solves for the guess of what was originally sent, X_k.

$$\hat{X_k} = \frac{Y_k}{H_k} \tag{8}$$

The data rate of the method described is equal to $\frac{blockSize}{\infty}$ because ∞ bits need to be sent to transmit one block. This issue can be circumvented using the properties of convolution. Prepending each block with a cyclic prefix 25% the size of it doesn't include the portions of the signal where the estimated channel convolution has diminished or negligible effects. Note that the

25% cyclic prefix length is a rule of thumb and is not always the case. The cyclic prefix length depends on the channel impulse response length.

The cyclic prefix also helps eliminate ISI as it acts as a guard period. This is because any ISI that would have affected the beginning of the next symbol is now affecting the cyclic prefix. In addition, Fig. 4 shows how the cyclic prefix acts as a guard against time offsets of when the receiver will start capturing the block. This positive attribute to the cyclic prefix is less relevant to this simulation, but is something that needs to be thought about in hardware.

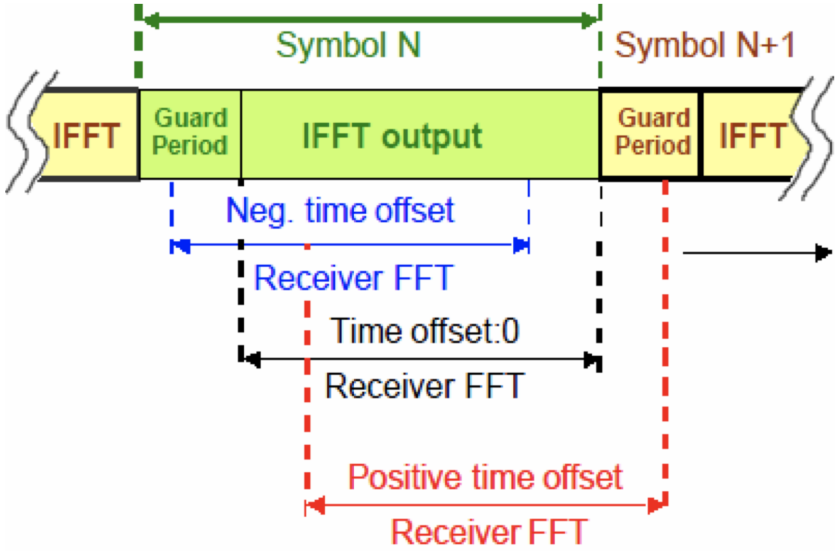

Fig. 4: The cyclic prefix guards helps handle uncertainty of when the receiver will capture the starting point of the block.

III. Implementation: Timing Synchronized Simulation

The implementation for the timing synchronized simulation uses a block length of 64 channels. As discussed above, the cyclic prefix length will be 25% of the block length: 16 bits. The high level overview of the software implementation is shown in Fig. 5.

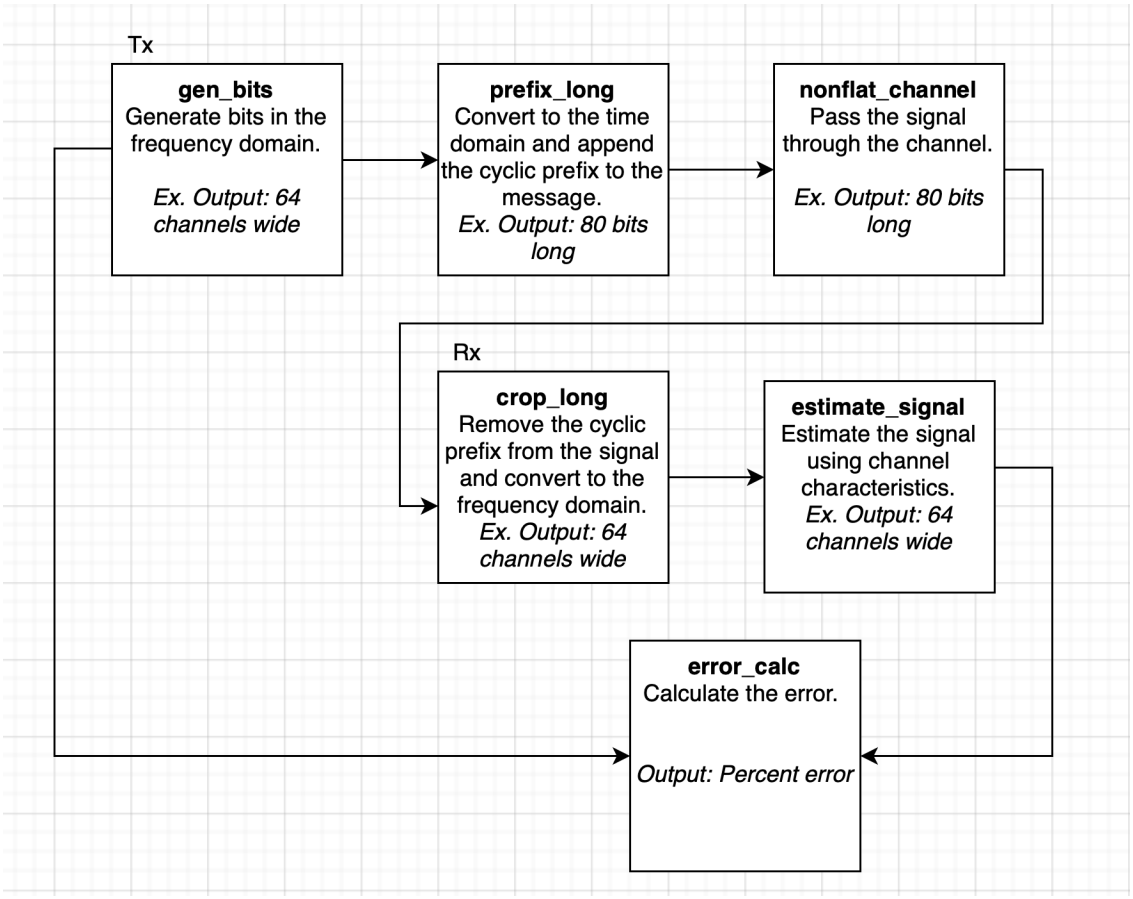

Fig. 5: High level software implementation for the timing synchronized simulation

First, the data needs to be generated in the frequency domain. The block size is 64. This means that there are 64 channels to transmit data across. An example of three generated blocks is shown in Fig. 6.

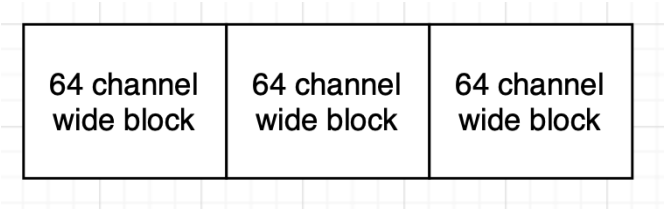

Fig. 6: Generation structure of three 64 channel wide blocks.

This information is then converted to the time domain using the IDFT and the cyclic prefix is prepended to each block. Fig. 7 shows the structure of the data after the cyclic prefixes are prepended. Each prefix is unique to each block and is a copy of the last 16 bits. Therefore, the combined length of the prefix and data is 80 bits long.

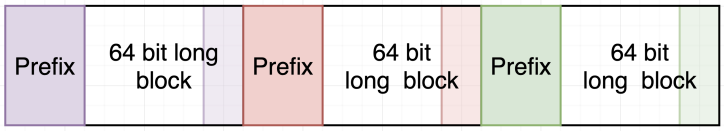

Fig. 7: Data structure after the cyclic prefixes are prepended.

The data is then is sent through the channel and the cyclic prefixes are removed. Fig. 8 shows how the data might look at this point.

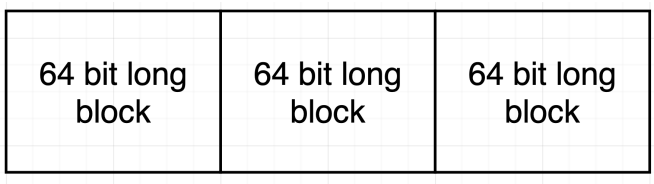

Fig. 8: Example of three blocks of data once cyclic prefixes are removed.

To convert the data back into the frequency domain, the DFT of the data is taken. Although the data is in the frequency domain and resembled Fig. 6, it is not an estimate of what was originally sent. The frequency response of the channel still needs to be accounted for.

To factor the channel in, the blocks are divided by the estimated channel response to receive $\hat{X}_k$ in accordance with Eq. 7. Fig. 9 illustrates how this division process is implemented with a 64-channel wide H matrix across multiple blocks.

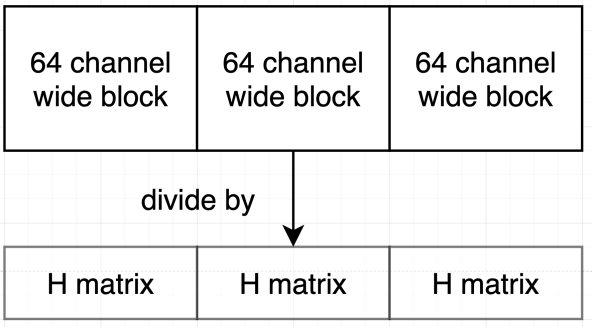

Fig. 9: Dividing a 3 block wide signal with the estimated channel matrix repeated 3 times.

A. Channel Estimation

In order to implement the steps explained above, H_k, or the channel estimation matrix needs to be calculated. This can be done by:

- Sending multiple blocks of a known signals across the channel.
- Dividing the received signal by the sent signal using Eqn. 7.
- Averaging the channel estimation responses for each of the blocks.

The signals are sent across the air using the same OFDM method described above.

IV. Results: Timing Synchronized Simulation

Fig. 10 shows the estimated channel impulse response using the steps illustrated above. Fig. 11 shows the actual channel impulse response. The estimated impulse response looks very similar to the actual impulse response. This allows the effects of the channel to be undone accurately.

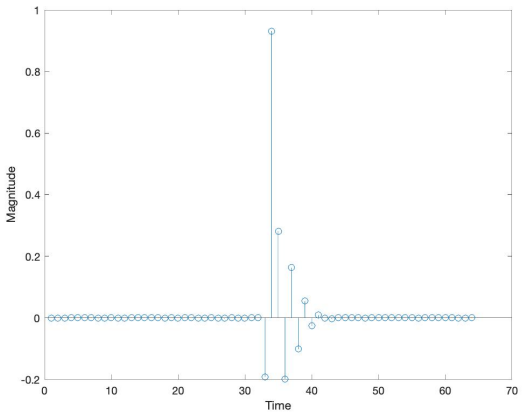

Fig. 10: Estimated channel impulse response for the timing synchronized simulated channel

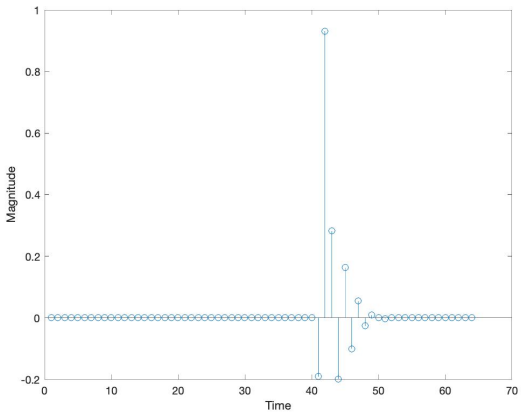

Fig. 11: Actual channel impulse response for the timing synchronized simulated channel

Fig. 12 shows the received signal in the frequency domain after the cyclic prefixes were removed and the DFT was computed. This graph only displays one block. The channel effects can be seen here as not flat-fading, meaning some frequencies experience greater fading than others.

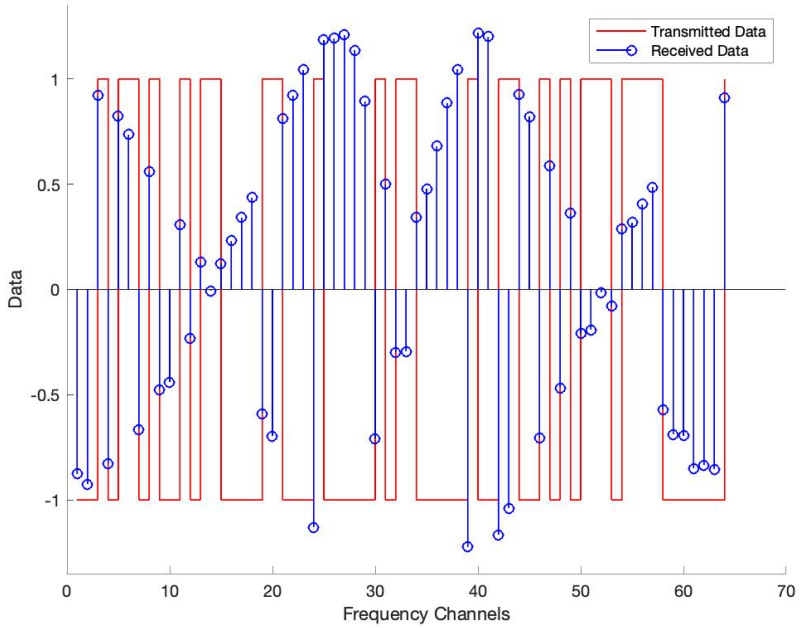

Fig. 12: Received data once cyclic prefixes were removed and the transmitted data plotted in the frequency domain

This implementation of the timing synchronized OFDM algorithm resulted in a BER of 0. This can be seen in Fig. 13 and Fig. 14. Fig. 13 is a constellation plot of the received estimate after accounting for channel effects. All of these points are distributed around either 1 or -1. These point clouds are relatively tight vertically and have no cross over. This allows there to be confidence in the value each bit represents.

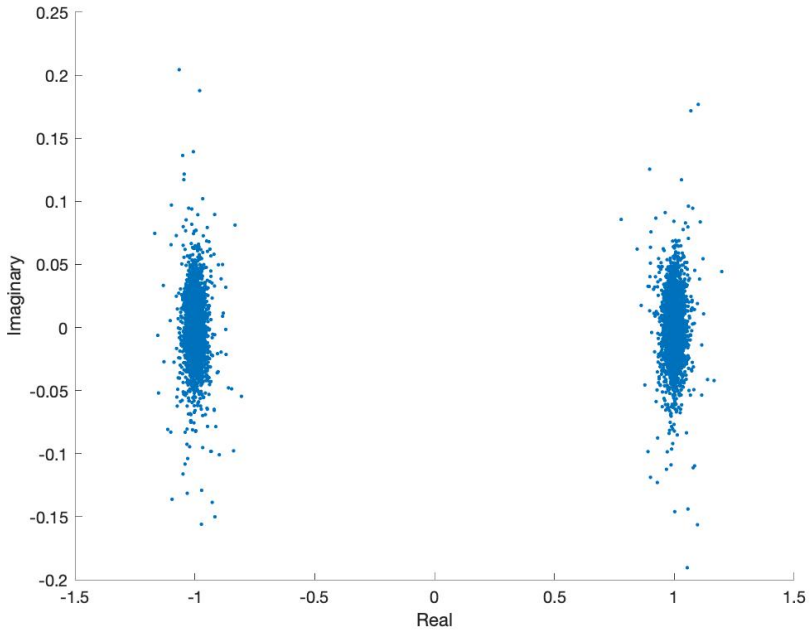

Fig. 13: Constellation plot of the final processed points using OFDM

Fig. 14 shows one block of the same data shown in Fig. 13. The received data varies, but there is high confidence in the bit estimate. This supports the information presented in Fig. 13.

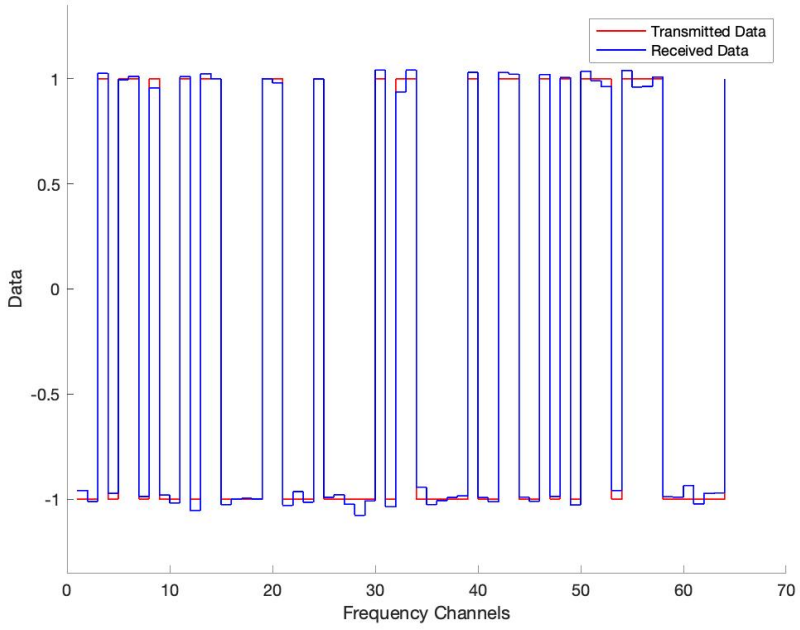

Fig. 14: Received data once cyclic prefixes were removed and the transmitted data plotted in the frequency domain

V. Discussion: Timing Synchronized Simulation

As discussed earlier, the BER for the timing synchronized OFDM simulation was 0. This is due to the fact that the SNR of this channel is also 30 dB which indicates great link quality.

When implementing this simulation never take the Hermitian transpose, or the complex conjugate transpose. Once data is sent through the channel it will be returned as complex values. These complex components cannot be disregarded until the very end of the simulation and, therefore, are used in channel effects correction. If implementing this system in MATLAB, be careful to use the `matrix.'` transpose method rather than the `matrix'` transpose method which will take the Hermitian transpose.

VI. Schmidl-Cox Algorithm

In hardware, it is unlikely that a transmitter and receiver will be perfectly synchronized. This is due to differences in hardware.

Carrier synchonization is the process where the frequency and phase of the oscillator in the receiver are tuned to match the frequency and phase of the received signal. The next step in implementing an OFDM system over USRP is to account for this frequency offset. Eq. 9 shows how this frequency offset can be mathematically included into the received signal equation. The definition of the frequency offset is shown in Eq. 10.

$$y[k] = h * x[k] \cdot f_{offset} + n[k] \tag{9}$$

$$\text{where } f_{offset} = e^{jf_\Delta k} \tag{10}$$

The effects of the channel impulse response can be undone through techniques discussed earlier. Assuming noise is negligible, this leaves Eq. 11. The received signal is equation to the sent signal multiplied by this frequency offset component.

$$y[k] = x[k] \cdot e^{jf_\Delta k} \tag{11}$$

In order to correct for the frequency offset, both sides can be divided by this component leaving Eq. 12.

$$\hat{x}[k] = y[k] \cdot e^{-jf_\Delta k} \tag{11}$$

The f_Δ variable is still unknown, and can be solved for using the Schmidl-Cox algorithm for frequency offset. This algorithm uses a specific type of preamble whose construction is detailed below.

A Long Traning Sequence (LTS), or a block of random data, is generated. This LTS is then copied twice and cyclicly prefixed to create a preamble. This is illustrated in Fig. 15.

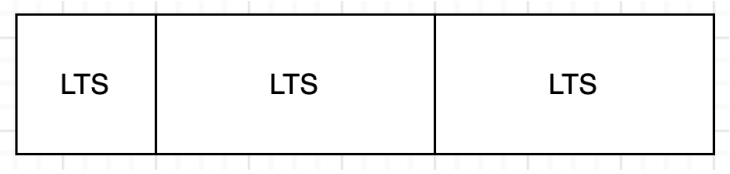

Fig. 15: Preamble constructing using LTS'

There are sustained effects between the end of one block and the beginning of the next blockd. The two LTS blocks are prefixed in order to ensure that any effects of prior bits is maintained the same across the two full LTS blocks. This concept is illustrated in Fig. 16.

Fig. 16: Illustration of sustained effects between blocks

Any sustained effects have been made uniform across the last two LTS blocks. It can be said that the first bit in the last two blocks are equal in the absense of any frequency offsets. This can also be said for the rest of the bits in the two full LTS blocks. This is shown in Fig. 17.

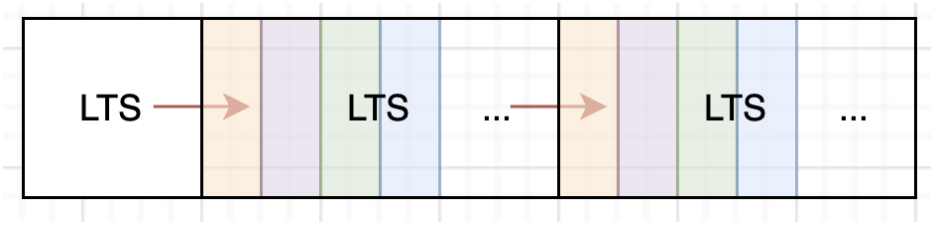

Fig. 17: Illustration of equation bits in the last two LTS blocks.

We can derive f_Δ by calculating the frequency offset between each LTS block. For any given LTS pair, the angle difference between a corresponding pair of bits can be used to calculate the frequency offset of the signal. This is represented by Eq. 12 where blockSize is the length of one LTS and i is an index of the preamble located inside the first full LTS.

$$\frac{y_k[i+blockSize]}{y_k[i]} = \frac{x_k[i+blockSize]e^{jf_\Delta k+blockSize} + n[k+blockSize]}{x_k[i]e^{jf_\Delta k} + n[k]} \tag{12}$$

Assuming noise is negligible, Eq. 12 can be simplified to Eq. 13.

$$\frac{y_k[i+blockSize]}{y_k[i]} = \frac{x_k[i+blockSize]}{x_k[i]}e^{jf_\Delta blockSize} \tag{13}$$

Since the two orange bits are equal once frequency offsets are removed, Eq. 13 can be simplified even further as shown in Eq. 14.

$$\frac{y_k[i+blockSize]}{y_k[i]} = e^{jf_\Delta blockSize} \tag{14}$$

Originally, the goal was to solve for f_Δ to counteract any frequency offsets are shown in Eq. 11. The angle of both sides can be taken, and f_Δ can be solved for as shown in Eq. 15.

$$f_\Delta = \frac{1}{blockSize}\angle\left(\frac{y_k[i+blockSize]}{y_k[i]}\right) \tag{15}$$

Now that the f_Δ can be computed at each pair of bits, an average frequency shift can be calculated using the sum shown in Eq. 16.

$$f_\Delta = \frac{1}{blockSize}\sum_{i=1}^{blockSize}\frac{1}{blockSize}\angle\left(\frac{y_k[i+blockSize]}{y_k[i]}\right) \tag{15}$$

This frequency offset can now be corrected for using Eq. 11. Note that this equation includes k which is the index. This means that the frequency offset component will change for each signal bit sent.

VII. Implementation: Simulation with Timing Errors

The overall software structure for this implementation of an OFDM system with timing errors is described in Fig. 18. The preamble is added to the signal right before it is sent across the channel. Once the signal is received, the preamble is removed from the signal. It is then used to calculate the frequency offsets and is applied to the entire signal before it is converted back to the frequency domain.

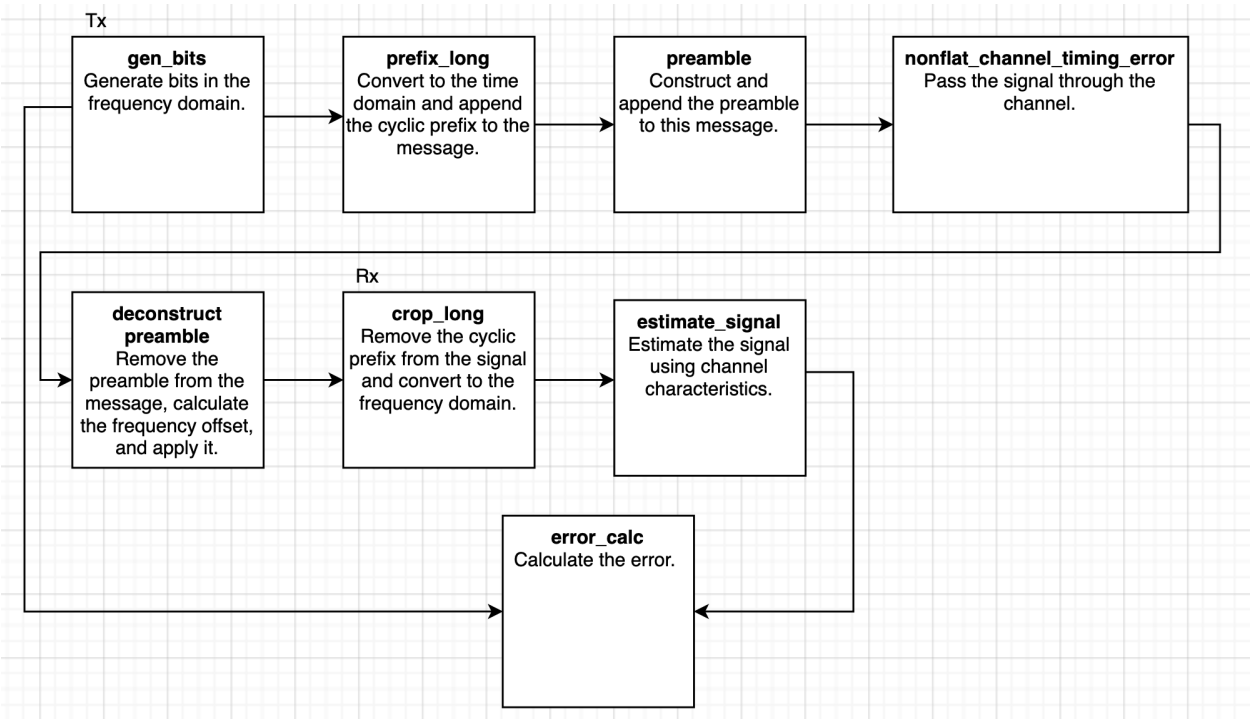

Fig. 18: High level software implementation for the simulation with timing errors.

The LTS used in this implementation is 64 bits long. In addition, three LTS (rather than 2.5 LTS) were used in the preamble for simplicity. So, the entire preamble added 192 bits to the signal. In order to account for any delay that results across the channel, the received signal can be cross correlated with the preamble to find this delay. However, when implementing this, it seemed that the random bits of the preamble didn't prove unique enough to find the delay consistently. Therefore, an alternating sequence of length 60 was appended to the beginning of the preamble. An alternating sequence of length 60 is highly unlikely to happen randomly. Since the rest of the data is generated randomly, when the received signal is cross correlated with this alternating sequence, the delay is able to be found with higher accuracy.

VIII: Results: Simulation with Timing Errors

This implementation resulted in a BER of 0. Fig. 19 shows a constellation diagram of the received data before and after the frequency correction is applied. Before the frequency correction is applied, the data mainly centers around 0 and is not distinctly either 1 or -1. After the frequency correction is applied, the data is clearly seperated and there is no overlap between the two point clouds. This is ideal to maintain low BER.

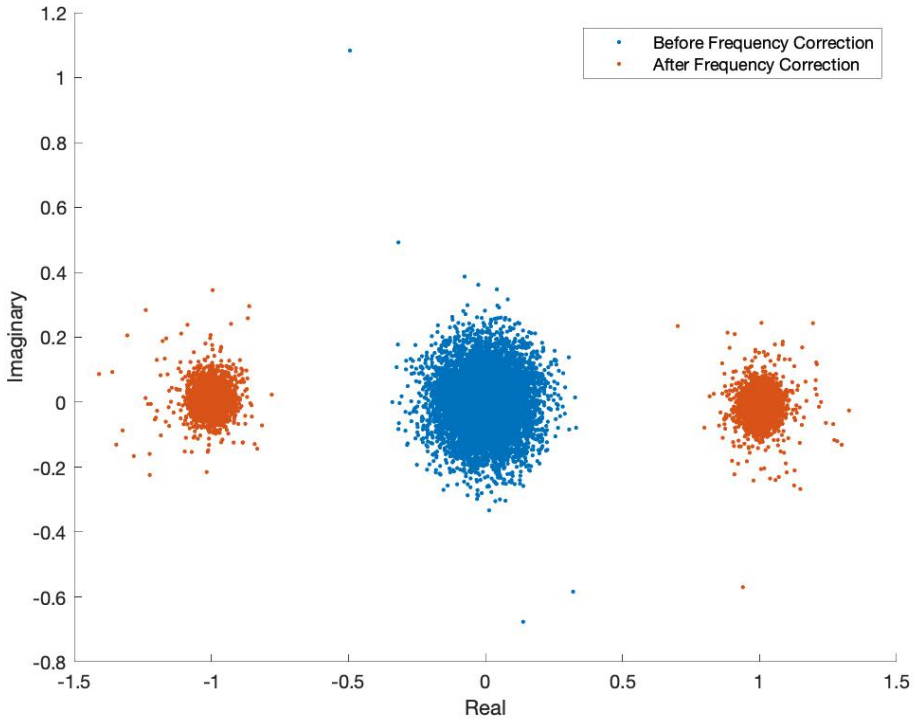

Fig. 19: Constellation diagram of the received signal before and after frequency correction was applied

Fig. 20 zooms in on one block of data shown in Fig. 19 and compares the frequency corrected received data with the original transmitted data. It can be seen that the final processed data has no ambiguity on which value the bit holds. This is corroborated by Fig. 19.

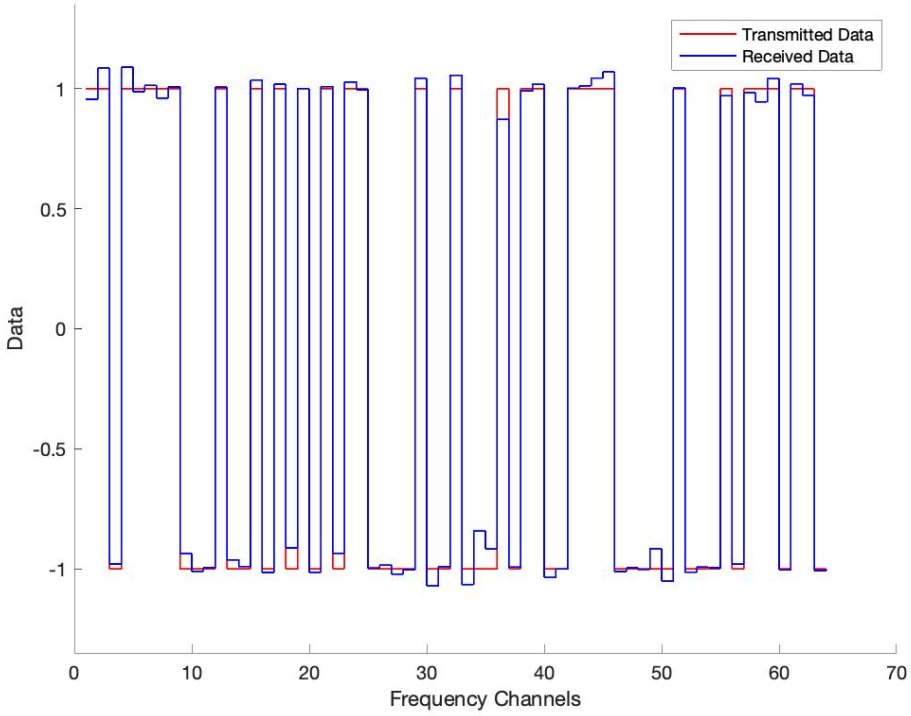

Fig. 20: One block of frequency corrected data from the signal shown in Fig. 19 compared with the original sent data

Althought the constellation plot shown in Fig. 19 has very neat point clusters, many times when running this simulation constelation diagrams resembling Fig. 21 and Fig. 22 are produced. the BER in these cases is still 0 as there are still 2 distinct point clusters after frequency correction. This is due to a phase offset that is produced during transmission. Accounting for such phase offsets will be discussed further in this paper.

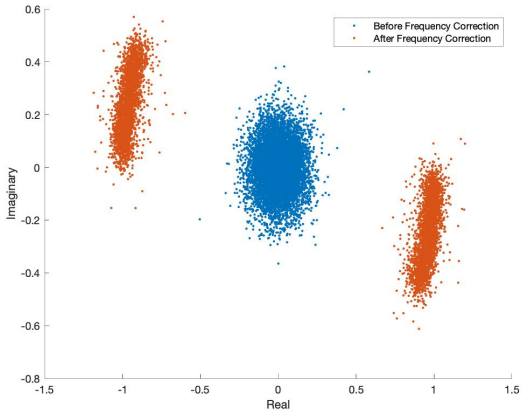

Fig 21: An example of a constellation diagram with phase offset

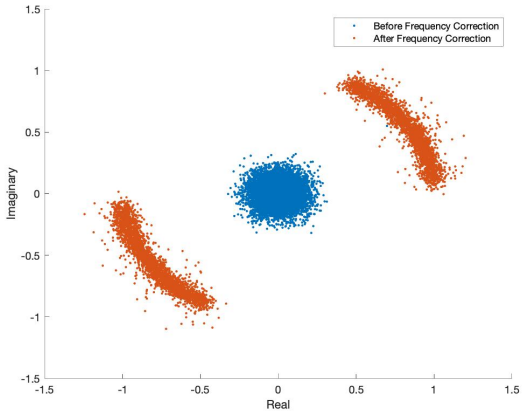

Fig 22: An example of a constellation diagram with phase offset

Very rarely, the delay isn't calculated correctly even though an incredibly specific signal is used to calculate the delay. This is due to the random nature of the block generation which allows it to also create an alternating signal. When this happens, constellation diagrams like Fig. 23 are produced by the simulation. As it can be seen in Fig. 24, the value of the bits matches 1

and -1 less clearly than in Fig. 20. The BER in situations like these is no longer 0 as there are not distinct point clusters in the constellation diagram. It is usually around 0.5 which signifies no correlation, or randomness.

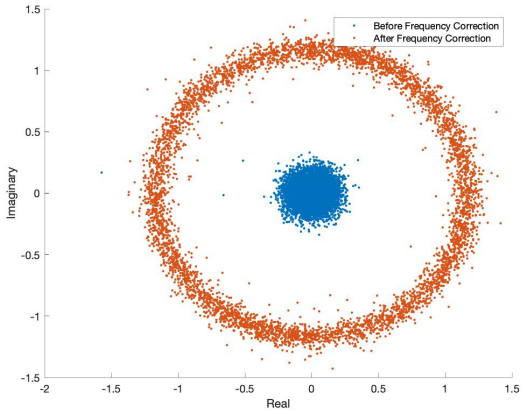

Fig. 23: Constellation diagram of an ambiguous received signal before and after frequency correction was applied

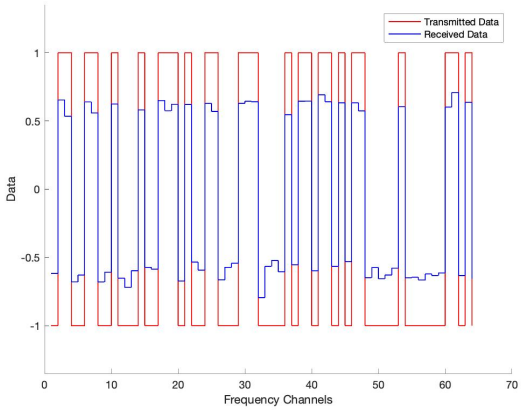

Fig. 24: One block of frequency corrected data from the signal shown in Fig. 23 compared with the original sent data

IX: Accounting for Hardware

When transmitting a signal in hardware, there are several extra steps that need to be taken to ensure a clean signal. When a signal is modulated with a carrier frequency, it is convolved with two impulses in the frequency domain. In hardware, imperfections in oscilattors cause these impluses to "bleed through" and effect that signal at the point where they convolve. This is called the DC offset. In order to take this into account, the 0 frequency component of each signal block needs to be 0.

In addition, low pass filters (LPFs) are not perfect in hardware either. LPFs with ideal characteristics like in Fig. 25 are extremely expensive and aren't commonly used. LPFs with a factor called roll off are more commonly used. Roll off is the steepness of the transfer function of the LPF against frequency. This means, that as long as the LPF doesn't have a sudden cutoff, it has roll off. An example is shown in Fig. 26.

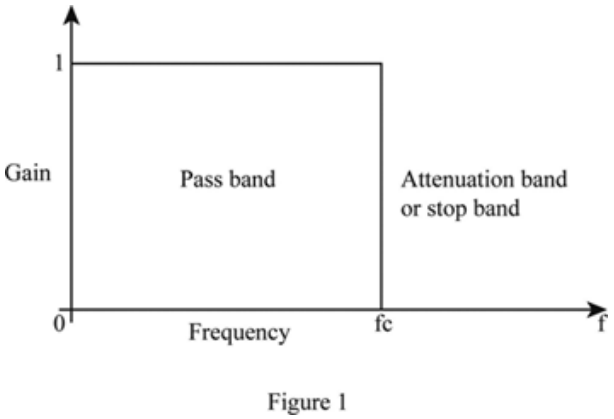

Fig. 25: Ideal low pass filter frequency characteristics

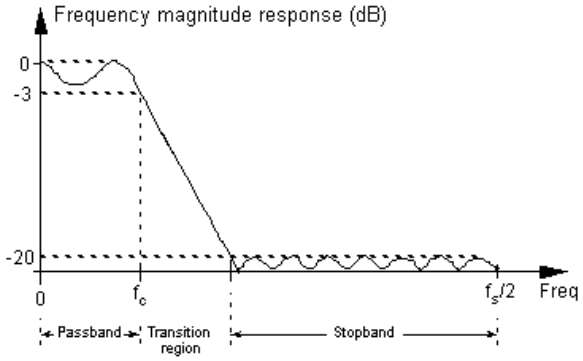

Fig. 26: LPF with roll off

Since there is loss in the transition region, it is not ideal to include information here. Therefore, guard bands are used to fill these regions of frequency in each block.

A. Phase Offset

There is jitter in the oscillators in hardware. This jitter induces a nonuniform phase offset across each block of data. This phase offset can be represented as $\theta_k$as shown in Eq. 16.

$$y[k] = h * x[k]e^{jf_\Delta k+\theta_k} + n[k] \tag{16}$$

Assuming noise is negligible, and the channel effects and frequency offsets have already been corrected for, Eq. 17 results. And same as the Schmidl-Cox algorithm, the reciprocal of the phase offset component can be multiplied on either side to result in the best guess of what was sent.

$$y[k] = x[k]e^{j\theta_k} \tag{17}$$

In order to estimate θ_k, pilot signals are used. Pilot signals are known signals interspersed throughout data that provide reference for how much phase offset is left to account for in each block. It is necessary to make the assumption that the phase offset is constant throughout one block of data. Then, the received pilot signals and sent pilot signals can be compared to find the phase offset using Eq. 18.

$$\theta_k = \angle\left(\frac{y[pilot]}{x[pilot]}\right) \tag{18}$$

If mulitiple pilot signals are sent across one block, they can be averaged to result in a general phase offset for the block. The effects of this phase offset can be reversed by using Eq. 19.

$$\hat{x}[k] = y[k]e^{-j\theta_k} \tag{19}$$

X: Implementation: OFDM in Hardware

The overall software structure for the OFDM implementation in hardware is shown in Fig. 27.

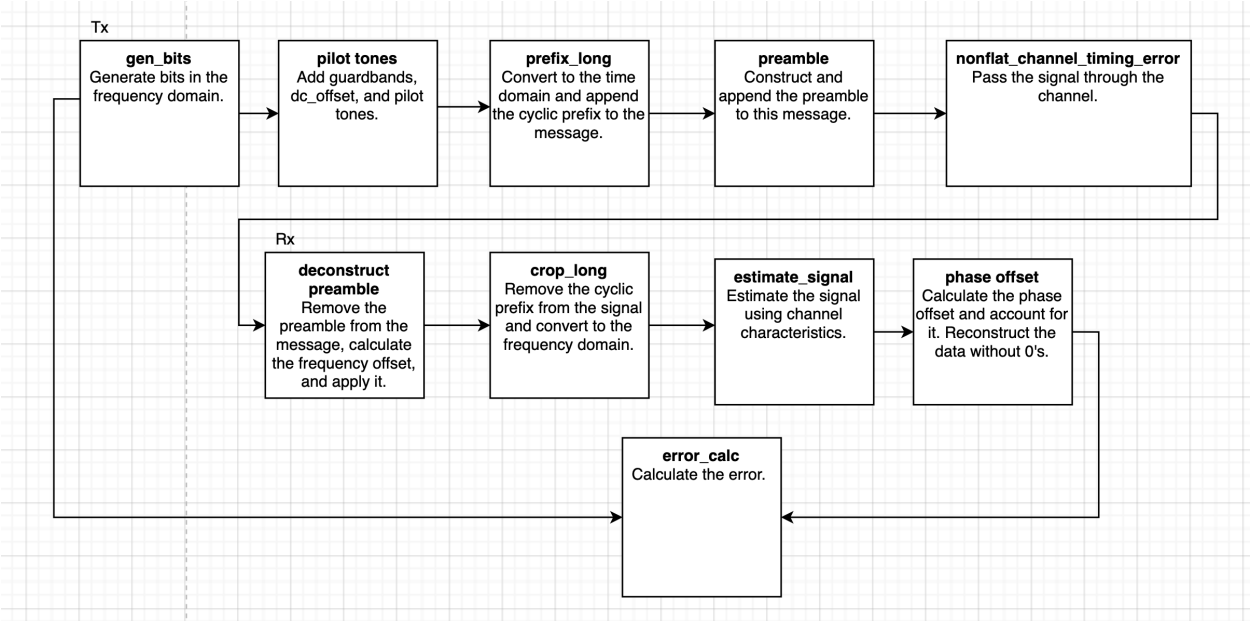

Fig. 27: Software structure for OFDM implementation in hardware

To prepare the data for transmission, guard bands are appended to both sides of each block in the frequency domain: length 6 on the left and length 5 on the right. These guard bands are constructed from a string of 0's. To account for the DC component,

the signal at the 0 frequency component is made to be 0. Lastly, 4 pilot signals of value 1 are interspersed through this data. In order to send this data, it needs to be fftshifted to due to how MATLAB calculates the ifft. Fig. 28 shows how the guard bands, DC offset, and pilot tones were distributed in frequencies, the MATLAB array, and after the fftshift.

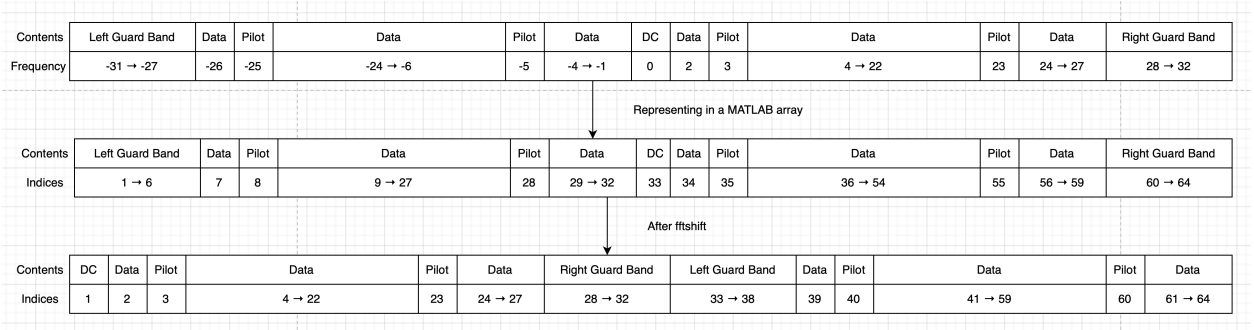

Fig. 28: Representation of the guard bands and pilot tones being added

When the signal is received, the first step is to calculate the delay and crop the signal. Then, the frequency is corrected for and the channel effects are used to estimate the signal. Each block is fftshifted back and the guard bands and DC offset are removed. The pilot tones are acquired and the phase offset is calculated using the process described earlier. This phase offset is applied to the data in the current block. It is important to note that now there are 11 bits of guard band, 1 bit of DC offset, and 4 bits of pilot signals. This means that only 48 bits of data are sent in each block. This means that when calculating the error, 48 bit long blocks are used.

XI. Results: OFDM In Hardware

A BER of 50% was achieved. This BER implies that the implementation is not currently working. This could be due to a bug on either the transmission or receiving side.

Fig. 29 shows the complete received signal. The actual signal is visible and the noise surrounding it is much lower.

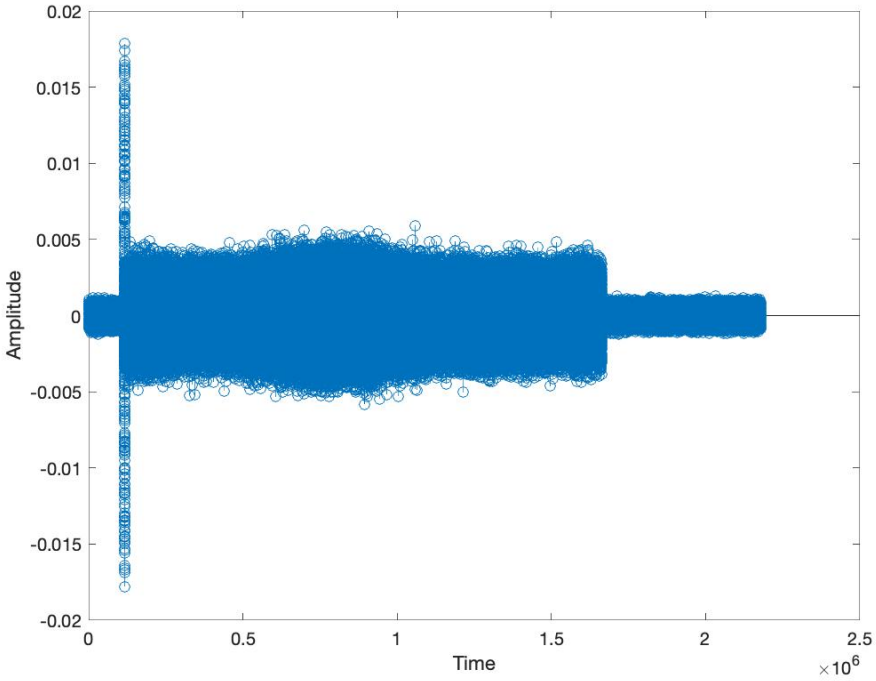

Fig. 29: Complete received signal

The constellation plot of the complete received signal is shown in Fig. 30. Since there aren't any clear point clouds, no bit is clearly any value.

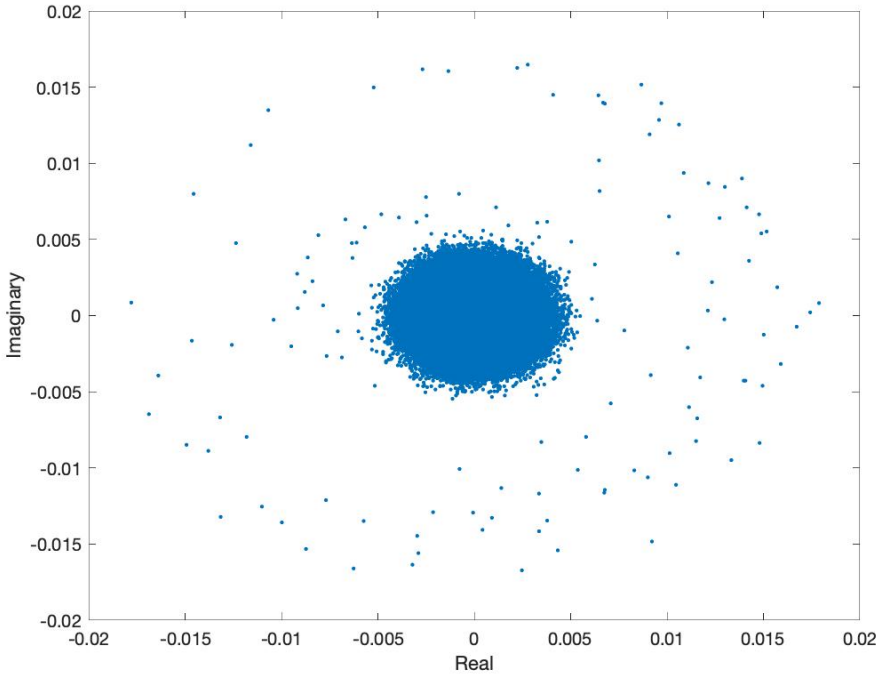

Fig. 30: Constellation plot of the complete received signal

The first step when receiving the data is to determine which portions are actual data and which portions are not. Fig. 31 shows that the delay was accounted for and much of the actual signal was not lost.

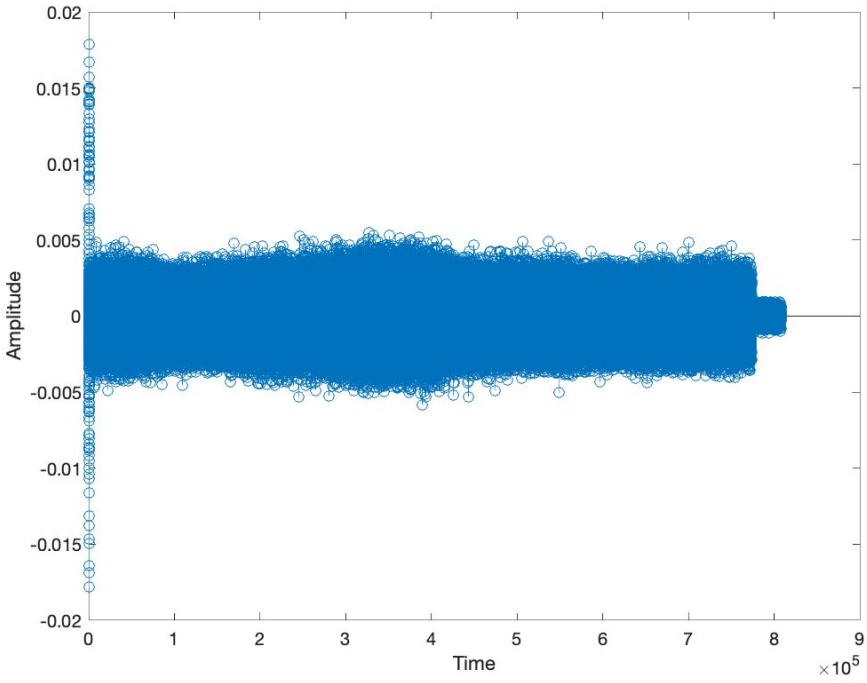

Fig. 31: Cropped signal

Fig. 32 shows that there are still not two distinct point clouds once the frequency offset, phase offset, and channel effects were accounted for. This corroborates the 50% BER.

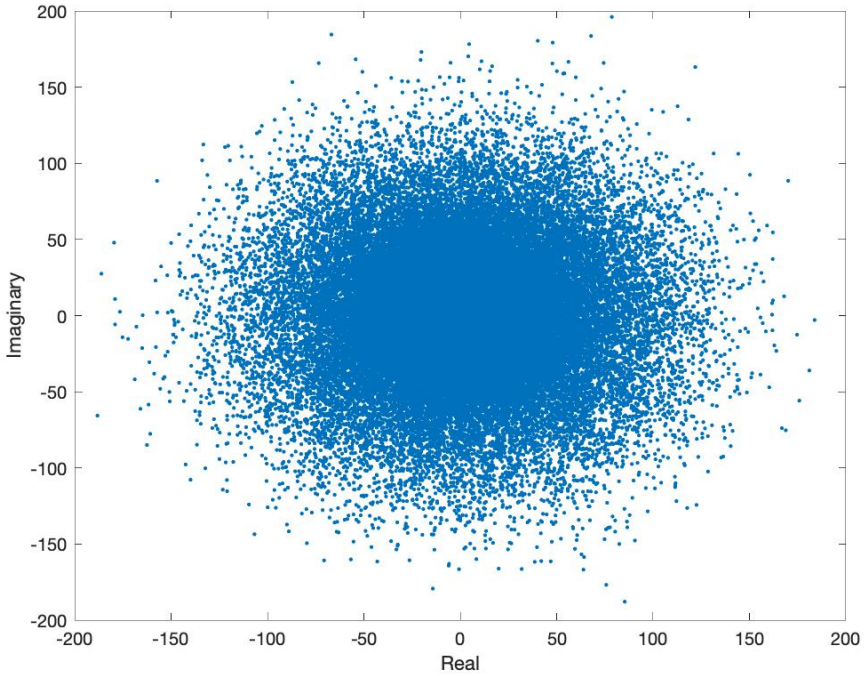

Fig. 32: Constellation plot of the processed received signal

Fig. 33 shows one block of received and sent data. Each bit does not remotely align with what is expected. This means that the implementation had a bug in it.

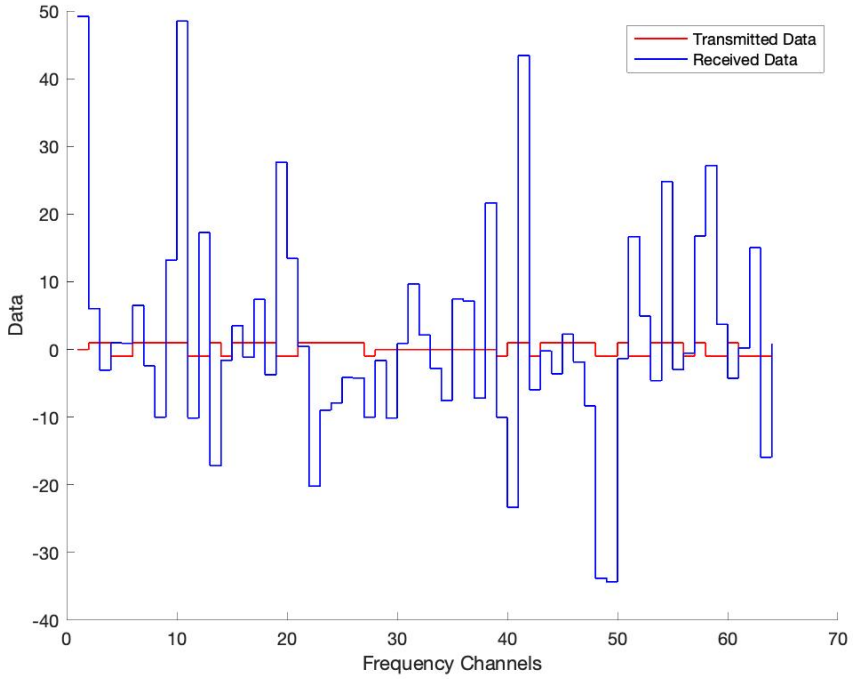

Fig. 33: One block of sent and received data

XII. Hardware Implementation Discussion

In an attempt to get this implementation working, the simulated channel used earlier was altered to add a phase offset. When this was employed, the BER was 0. This could be due to the higher SNR in the simulated channel when compared with transmitting over air. The results from this simulation are included below. Fig. 34 shows the

constellation plots of the data before and after processing. The final constellation diagram has two tight point clouds which means that the processing was successful. Fig. 35 shows one block of data in this result. The received bits closely batch the sent bits.

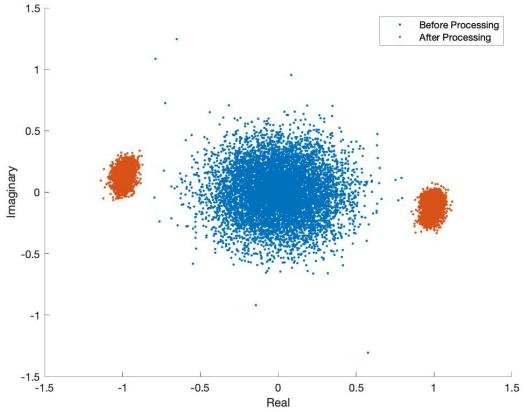

Fig. 34: Constellation plot of the simulated implementation of phase offset

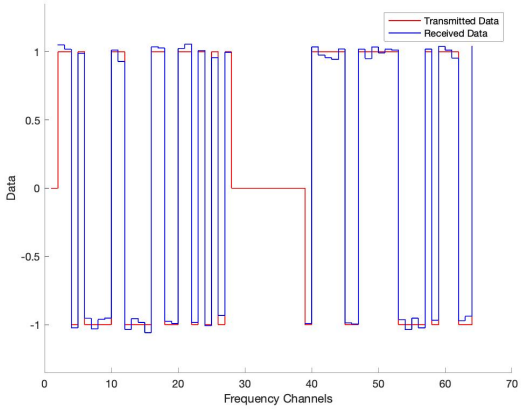

Fig. 35: One block of the simulated implementation of phase offset

XII. Resources

https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1063&context=eesp

https://techdifferences.com/difference-between-fdm-and-ofdm.html

https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.chegg.com%2Fhomework-help%2Fdefinitions%2Fideal-low-pass-filter-4&psig=AOvVaw0xr_3MeLAXAGN6Bus-nwh3&ust=1617792985056000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCMi1tNW66e8CFQAAAAAdAAAAABAD

XIII. Appendix

The code is located at: https://github.com/naviatolin/PoW-Labs/tree/master/Lab_3