ELEC 391 Project Report

Reliable Transfer of Wireless Information

Group 8

Yifeng Liu (28177384):

A / D and D / A Conversion, AWGN Channel

Lynn Kelly (14711071):

Error encoding / decoding, modulation / demodulation

Sidharth Sudhir (11255635):

Transmitter / receiver, Gilbert Fading Model

Table of Content

Objectives, Requirements & Constraints

The communication scenario our team is implementing is a system for the reliable transfer of wireless information over a noisy channel that meets the following specifications:

Message Transmission	Reliable Transmission	Processing Delay (mic to speaker)	Simulation tool	Prototype
8 kHz bandwidth	Bit error rate of 10^{-4}	25 ms	Matlab / Simulink / ModelSim	Altera DE1-SoC FPGA board

Average transmit power	Channel bandwidth	Audio source	Audio sink
1 W	100 kHz spectral mask	• Audio file in a WAV format with 32 bits per sample • DE1 microphone input • Duration 5 sec – 10 sec	• Audio file in WAV format based on team requirements • DE1 speaker output

Channel Sample Frequency	Channel quantization / dimension	Channel amplitude range	Channel quantization interval	Gilbert channel model Name
1 MHz	16 bits	±4	2^{-13}	γ

We were fully able to meet all of the required specifications aside from those related to the implementation of the FPGA, which will be the next step of the project.

There were several significant design challenges including understanding the system as a whole and how each subsystem functioned within the system and researching the different options available for each design decision. However, the most significant challenges were related to determining optimal parameters for subsystems upon integration of each subsystem into the full system model and determining the appropriate delays for the system. We used various testing methods to tackle these challenges including using scopes and displays to better visualize our system and isolating subsystems to test them separately to ensure they were functioning as expected. Another major challenge was determining how to design our channel model according to the Gilbert channel model specifications of switching between good and bad channel states based on probability. We discussed implementation ideas as a team and then used various methods to debug and test our final channel model.

System Design

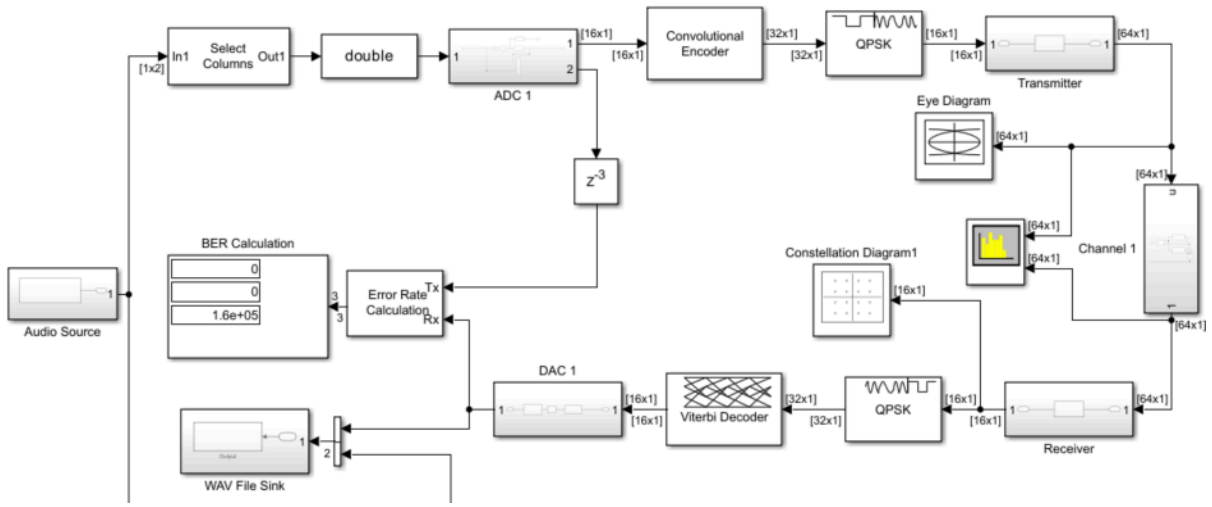


Figure 1: System block diagram, left channel. The right channel has a duplicate structure so it was omitted from the above figure.

Our system consists of the subsystem blocks shown in the figure above. The audio source uses a stereo 32-bit WAV file, and the Sink outputs a 16-bit WAV file. The ADC subsystem uses the *Rate Transition* and *Integer to Bit Converter* to convert the signal from analog to digital. The Nyquist rate was used to determine a sampling rate by doubling the message bandwidth. The 32-bit source was quantized to 16-bit frames. The DAC block uses the *Bit to Integer Converter* and *Data Type Converter* blocks to convert the digital signal back to an analog signal. A convolutional encoder and Viterbi decoder is used for error correction due to its robust error correction abilities in a moderately noisy channel. For modulation/ demodulation, QPSK was chosen because it has a low theoretical BER. The transmitter/ receiver subsystem uses a raised cosine filter for pulse shaping to transmit and receive the signal. The raised cosine filter was chosen because it can reduce inter-symbol interference and the bandwidth can be controlled with the roll-off factor. The channel uses the γ Gilbert channel model to model a noisy AWGN channel. The block was designed to switch between a good and bad channel using Markov chain probabilities of state switching.

The results of our system simulation are shown in the following table:

Project Conditions	Specifications	Simulation Results
Message Transmission	8 kHz bandwidth	8 kHz bandwidth
Reliable Transmission	BER of 10^{-4}	BER of 0
Processing Delay	25 ms	3 ms
Average Transmit Power	1W	1W
Channel Bandwidth	100 kHz	10 kHz

Subsystem Design

1 Source / Sink

Block Diagram:

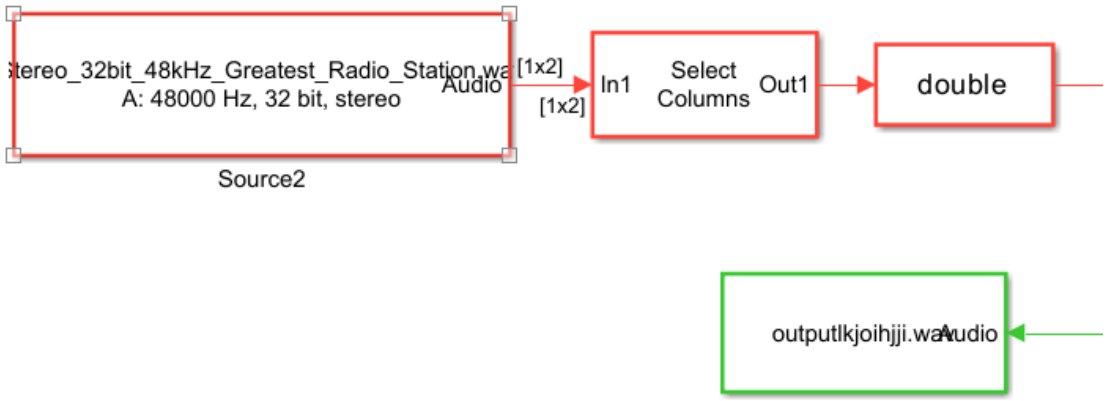


Figure 2: Block Diagram for Source and Sink

Description and justification:

The *From Multimedia File* is chosen as the source of our system as shown in figure . The output type is set to int16 with 1 sample per channel. A 32-bit stereo .WAV file is chosen as the source. Due to the stereo audio of the 32-bit source, the variable selector block is added to select one channel out of the source. For the sink, the *To Multimedia File* is implemented with the output data type of 16-bit integer to get better clarity.

Simulink Blocks & Parameters:

Simulink Block	Parameters Used
From Multimedia File	**Number of times to play file**: 1 **Read range**: [1 inf] **Samples per audio channel**: 1
Variable Selector	**Select:** Columns **Selector mode:** Fixed **Elements:** [1]
Data Type Converter	**Output data type:** double
To Multimedia File	**File type:** WAV **Audio data type:** 16-bit integer

2 A/D and D/A Converters

Block Diagram:

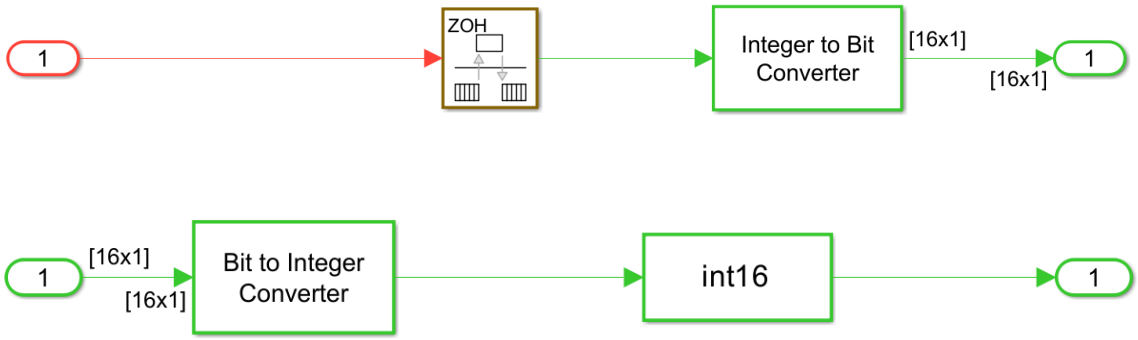


Figure 3: Block Diagram for A/D and D/A Converters

Description and justification:

For the A/D converter, the *Rate Transition* and *Integer to Bit Converter* are used to convert the signal from analog to digital. According to the Nyquist rate, the sampling rate should be at least two times the bandwidth. The target bandwidth for our system is 8kHz. Therefore, the Output port sample time is set to 1/160000 to avoid aliasing. To match the requirement of later encoding requirement, the *Integer to Bits Converter* is implemented with a setting of 16 bits per integer to downsample the signal to 16-bit.

To convert it back to the analog signal, *Bit to Integer Converter* and *Data Type Converter* are used to convert the signal from digital to analog. The number of bits per integer is also set to 16 to match the setting in the A/D converter. The *Data Type Converter* is implemented to ensure the correctness of the data type.

Simulink Blocks & Parameters:

Simulink Block	Parameters Used
Rate Transitions	**Initial conditions:** 0 **Output port sample time options:** Specify **Output port sample time:** 1/16000
Integer to Bit Converter	**Number of bits per integer(M):** 16 **Treat input values as:** Signed **Output bit order:** MSB first **Output data type:** Inherit via internal rule
Bit to Integer Converter	**Number of bits per integer(M):** 16 **Input bit order:** MSB first

	After bit packing, treat resulting integer values as: Signed **Output data type:** Inherit via internal rule
Data Type Converter	**Output data type:** int 16

3 Error Correction Encoder / Decoder

Block Diagram:

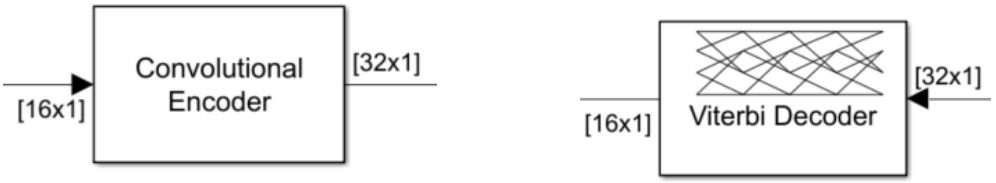


Figure 4: Block Diagram for Error Correction Encoder / Decoder

Description and justification:

The error correction encoder and decoder are an integral part of our system. We chose to use a ½ rate (7, [171, 133]) convolutional encoder with a Viterbi decoder due to its robust error correction abilities when used in a moderately noisy AWGN channel. Additionally, with a constraint length of 7 and traceback depth of 32, the complexity and latency of the decoder remains relatively low compared to convolutional codes with a larger constraint length or larger traceback depth [1]. A trade-off for using this convolutional code and the Viterbi decoder for error correction is that the decoder introduces a delay of 2 cycles into the system, however this is very small, and given that our implementation is significantly under our requirements for processing delay, this is acceptable. Another trade-off is that compared to a convolutional code with a higher constraint length and traceback depth, the error correcting capabilities of this code are reduced, but despite this, the error correcting abilities of the chosen encoder/decoder combination are more than adequate for our communication scenario. In particular with QPSK modulation, which was used in our system, the theoretical BER of the (7, [171, 133]) convolutional code when used on a channel with a SNR of 9dB is significantly lower than our specified maximum BER of 10^{-4} as shown in the following figure:

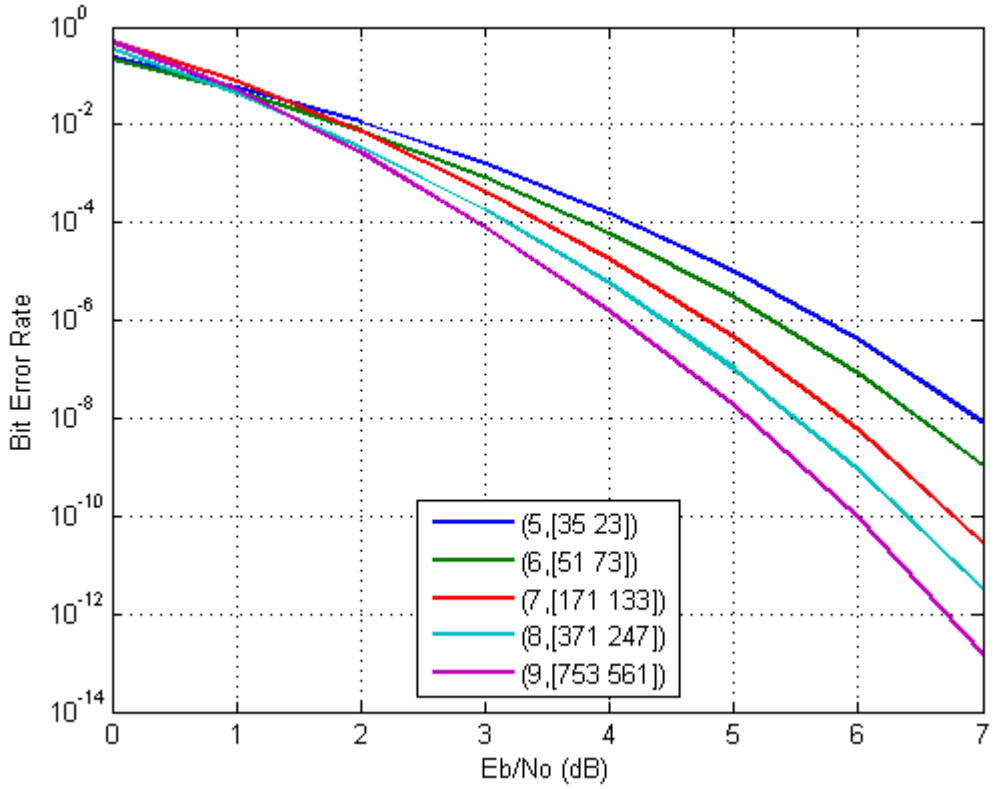


Figure 5: Theoretical bit error rate curves for QPSK modulated signals encoded with convolutional codes of various constraint lengths through an AWGN channel. [2]

The figure shows that a theoretical BER of 10^{-4} occurs when the SNR is approximately 3.5, which is significantly lower than the worst SNR of our channel model. Thus this error correction code in combination with QPSK modulation is more than sufficient for the needs of our system.

Simulink Blocks & Parameters:

Simulink Block	Parameters Used
Convolutional Encoder	**Trellis Structure:** poly2trellis(7, [171 133]) **Operation mode:** Continuous Did not use Puncture code option
Viterbi Decoder	**Trellis Structure:** poly2trellis(7, [171 133]) **Decision type:** Hard decision **Traceback Depth:** 32 **Operation mode:** Continuous **State metric word length:** 16 **Output data type:** Inherit via internal rule Did not use Puncture code option

4 Modulation / Demodulation

Block Diagram:

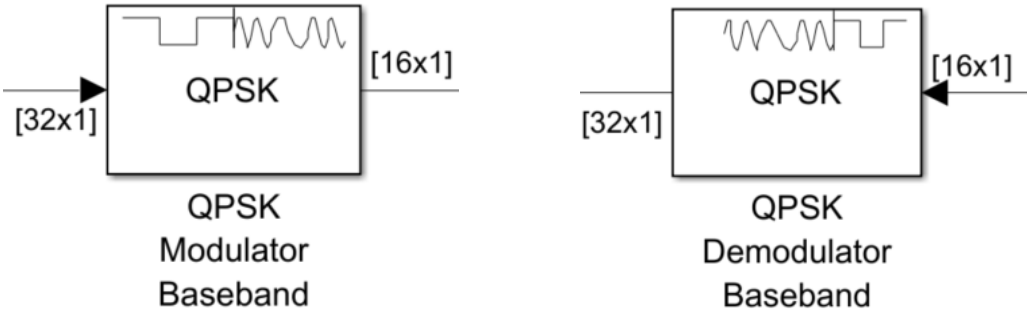


Figure 6: Block Diagram for Modulator/Demodulator

Description and justification:

QPSK modulation and demodulation was chosen for our system because it has a low theoretical BER. In particular it has the same BER as that of BPSK when Gray coding is used with a phase offset of pi/4 since this can be implemented as two BPSK signals orthogonally oriented to one another [3]. The trade-off of using QPSK is that it has a relatively low bit to symbol ratio of 2 bits/symbol, so the symbol rate is higher than that of a higher order modulation scheme. Thus the bandwidth usage is greater than that of a higher order modulation scheme. However, QPSK is still sufficient to offset the ½ rate of the error correction code, which causes the system to have a symbol rate that is equal to the sampling rate set by the ADC. The greater bandwidth usage due to using QPSK modulation is effectively limited with the pulse shaping filter so our system remains within our spectral mask bandwidth requirements.

Simulink Blocks & Parameters:

Simulink Block	Parameters Used
QPSK Modulator Baseband	**Input type:** Bit **Constellation ordering:** Gray **Phase offset (rad):** pi/4 **Output data type:** double
QPSK Demodulator Baseband	**Output type:** Bit **Decision type:** Hard decision **Constellation ordering:** Gray **Phase offset (rad):** pi/4 **Output data type:** Inherit via internal rule **Derotate factor:** Same word length as input **Rounding Mode:** Nearest **Overflow mode:** Saturate

5 Transmitter / Receiver

Block Diagram:

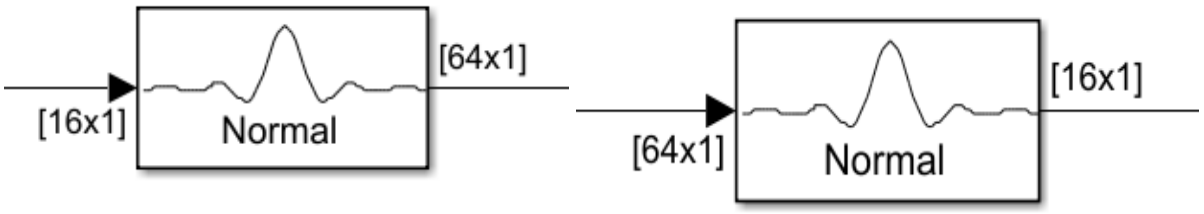


Figure 7: Block Diagram for Transmitter/Receiver

Description and justification:

The Raised Cosine Filter was chosen for our transmitter and its matched version was chosen for our receiver. The parameters were chosen so as to balance efficient spectral usage and signal clarity. The filter's roll-off factor of 0.25 ensures minimal inter-symbol interference (ISI) by controlling the bandwidth and smoothing the transitions between symbols. The filter span of 16 symbols aligns with the 16-bit data chunks, simplifying delay calculations. Additionally, using 4 samples per symbol enhances the accuracy of the signal representation, leading to better performance in subsequent processing stages and more accurate signal reconstruction. The matched filter on the receiver end optimizes the signal-to-noise ratio (SNR) and further minimizes ISI. Implementing these filters on an FPGA using Finite Impulse Response (FIR) filter approximations leverages the FPGA's capabilities and resources. The filter coefficients, which are designed to match the transmitter filter, ensure that the FPGA processes the signal accurately, maintaining the benefits of the Raised Cosine filtering in real-time applications.

Simulink Blocks & Parameters:

Simulink Block	Parameters Used
Raised Cosine Transmit Filter	**Filter shape:** Normal **Rolloff factor:** 0.25 **Filter span in symbols:** 16 **Output samples per symbol:** 4 **Linear amplitude filter gain:** 1 **Input processing:** Columns as channels (frame based) **Rate options:** Enforce single-rate processing **Export filter coefficients to workspace:** Checked **Coefficient variable name:** rcTxFilt
Raised Cosine Receive Filter	**Filter shape:** Normal

	Rolloff factor: 0.25 **Filter span in symbols:** 16 **Input samples per symbol:** 4 **Decimation factor:** 4 **Decimation offset:** 0 **Linear amplitude filter gain:** 1 **Input processing:** Columns as channels (frame based) **Rate options:** Enforce single-rate processing **Export filter coefficients to workspace:** Unchecked

6 Channel

Block Diagram:

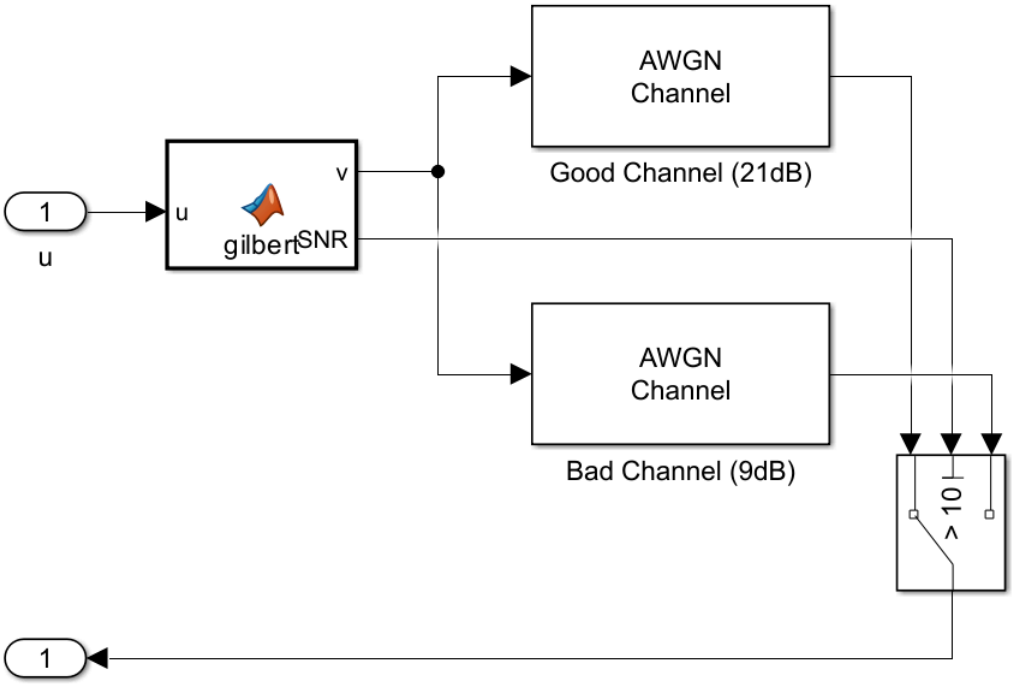


Figure 8: Block Diagram for Channel

Description and justification:

The channel model follows a Gilbert model to simulate the effects of fading and varying channel conditions in a communication system. A MATLAB function block ('gilbert') determines the state of the channel, routing the input signal to two Additive White Gaussian Noise (AWGN) channels (Good and Bad Channel). The "Good Channel" has a high

signal-to-noise ratio (SNR) of 21 dB, while the "Bad Channel" has a lower SNR of 9 dB. The switch block uses the output of the gilbert function to direct the output signal coming out of either the good or bad channel based on the probabilistic state of the Gilbert model of channel γ. The input signal power is set to 1 watt (W) to maintain consistent power levels.

Simulink Blocks & Parameters:

Simulink Block	Parameters Used
MATLAB Function	**Coding:** Appendix A 1
AWGN Channel (GOOD)	**Mode:** Signal to noise ratio (SNR) **SNR(dB):** 21 **Input signal power:** 1 **Random number source:** Global stream **Simulate using:** Code generation
AWGN Channel (BAD)	**Mode:** Signal to noise ratio (SNR) **SNR(dB):** 9 **Input signal power:** 1 **Random number source:** Global stream **Simulate using:** Code generation
Switch	**Criteria for passing first input:** u2>Threshold **Threshold:** 10

Verification

1 Source and Sink

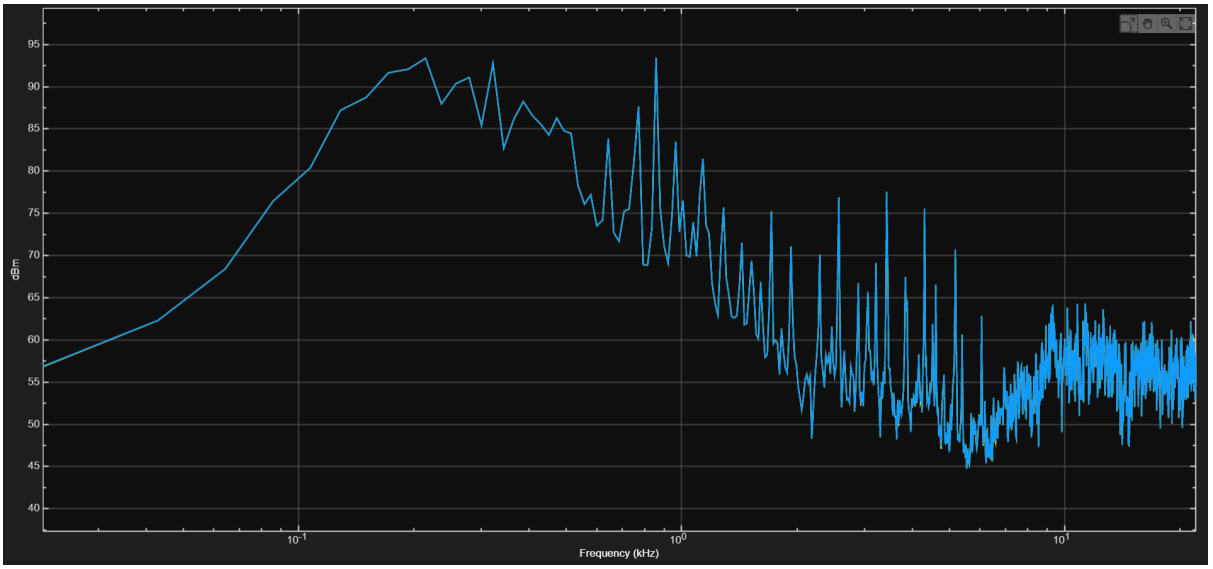


Figure 9: Frequency spectrum comparing Source (yellow) and Sink (blue) at 100 dB SNR

As the Figure shows above, the source frequency spectrum is shown in yellow and the sink frequency spectrum is shown in blue. In analyzing the frequency spectrum of the source and sink waveforms, it is evident that both spectra are nearly identical when observed under conditions of a high signal-to-noise ratio.

2 A/D and D/A Converters

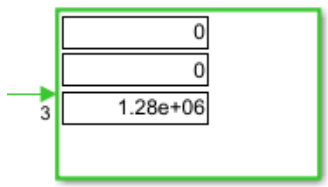


Figure 10: Converter BER at 9dB

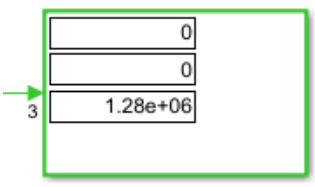


Figure 11: Converter BER at 100dB

The Bit Error Rate (BER) for both 9dB and 100dB performed above expectation for our system. Their BERs are both zero. It was expected that the system with 9dB would exhibit a higher BER, however due to our encoder's exceptional performance in combination with QPSK modulation, there is no difference between the BER for 9dB compared to 21dB.

3 Error Correction Encoder/Decoder

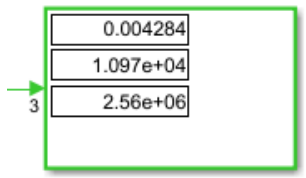


Figure 12: Error Correction Encoder/Decoder BER at 9dB without encoder/decoder

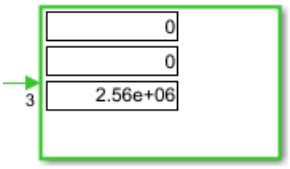


Figure 13: Error Correction Encoder/Decoder BER at 100dB without encoder/decoder

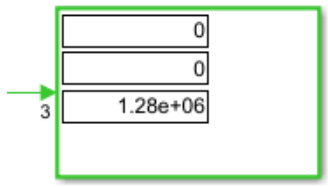


Figure 14: Error Correction Encoder/Decoder BER at 9dB with encoder/decoder

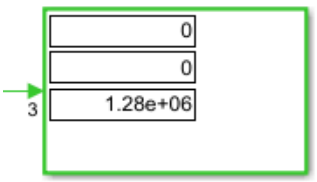


Figure 15: Error Correction Encoder/Decoder BER at 100dB with encoder/decoder

As expected, the BER with the error correction encoder and decoder implemented is significantly lower than without the encoder and decoder enabled when the SNR is 9dB. It also meets expectations that the BER is higher at 9dB compared to 100dB since a SNR of 100dB has minimal noise.

4 Modulation/Demodulation

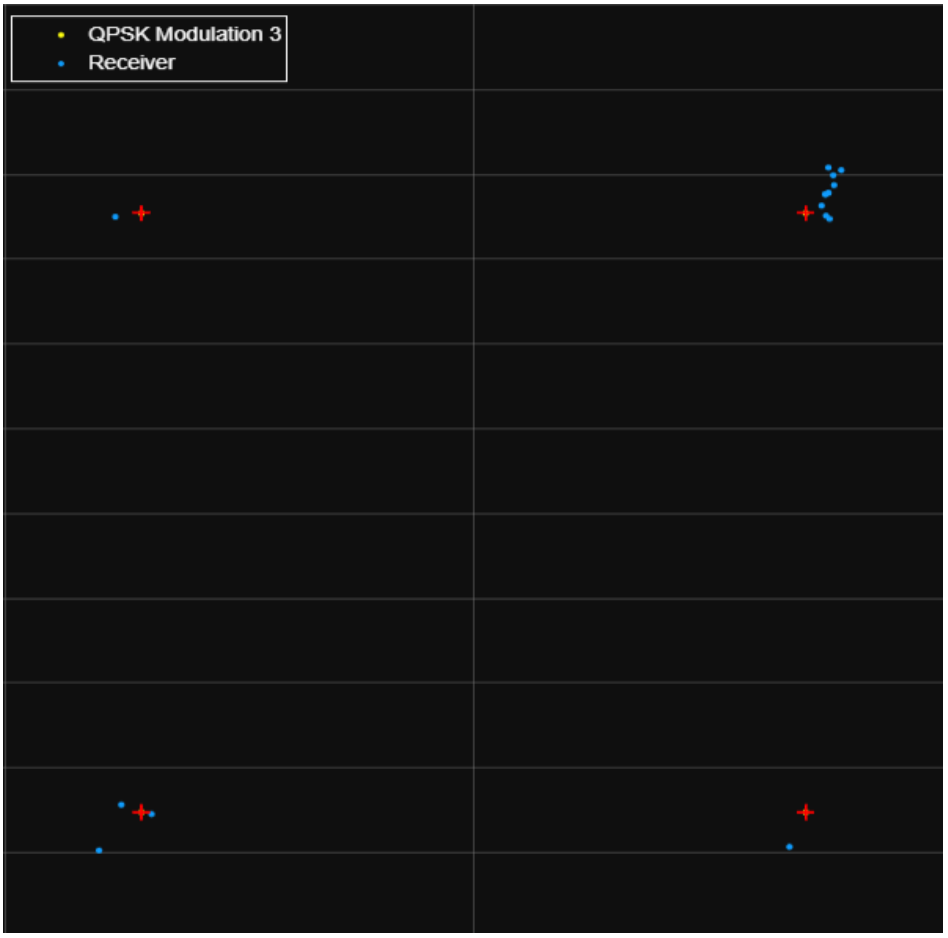


Figure 16: Constellation scatter plot comparing modulator output & demodulator input for 100dB SNR

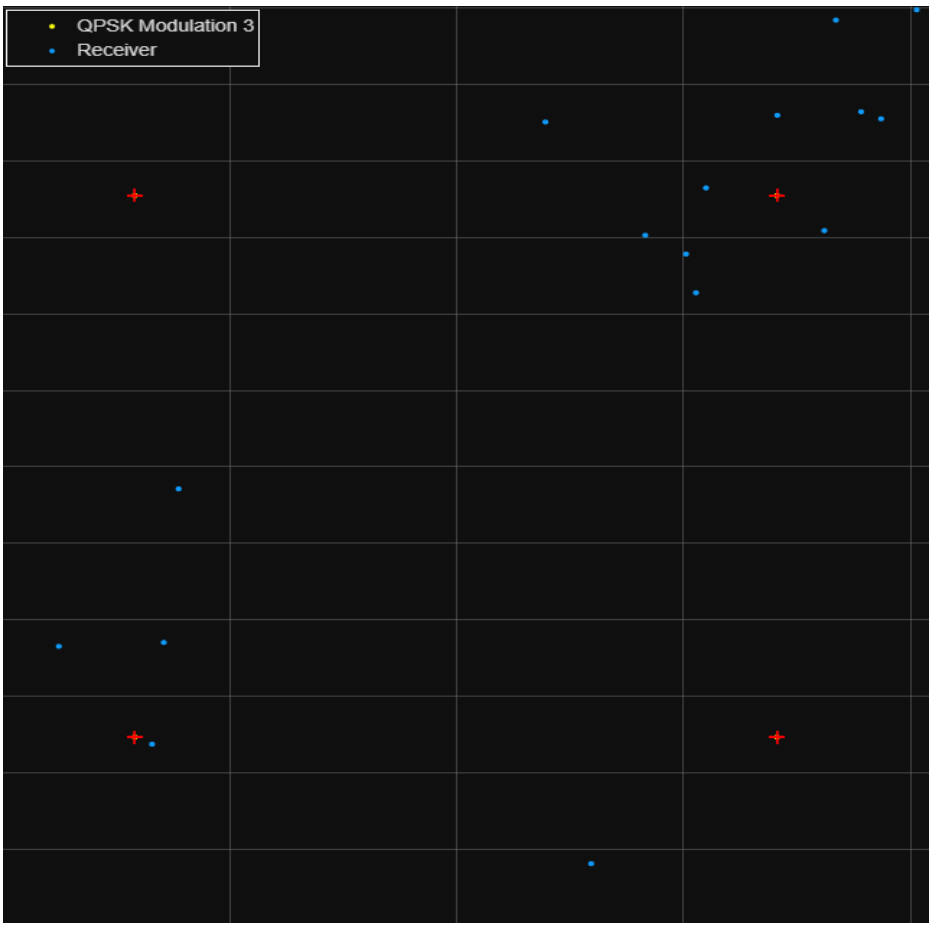


Figure 17: Constellation scatter plot comparing modulator output and demodulator input for 9dB SNR

As expected, there is greater scatter and spread of the points around the symbols for a SNR of 9dB than 100dB. The points are very close to the symbols for 100dB, which is consistent with a very low noise channel, but they are more spread out and further from the expected values of the symbols for 9dB because the noise would add to the signal's symbols and move them further from their expected values.

Of note, the modulator output points for both graphs cannot easily be seen on the scatter plot because they all lie on the red '+' indicating the location of the symbols. This is expected since all the bits should have been mapped to symbols at these locations by the modulator.

5 Transmitter/Receiver

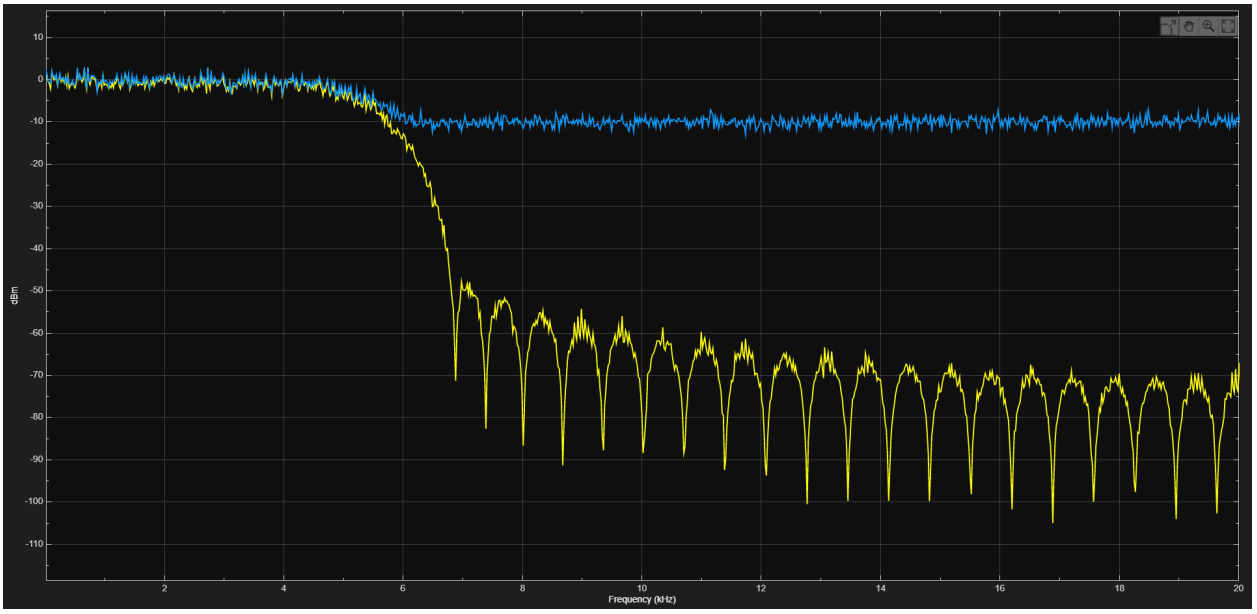


Figure 18: Frequency domain signals from the output of the transmitter to the input of the receiver at 9dB

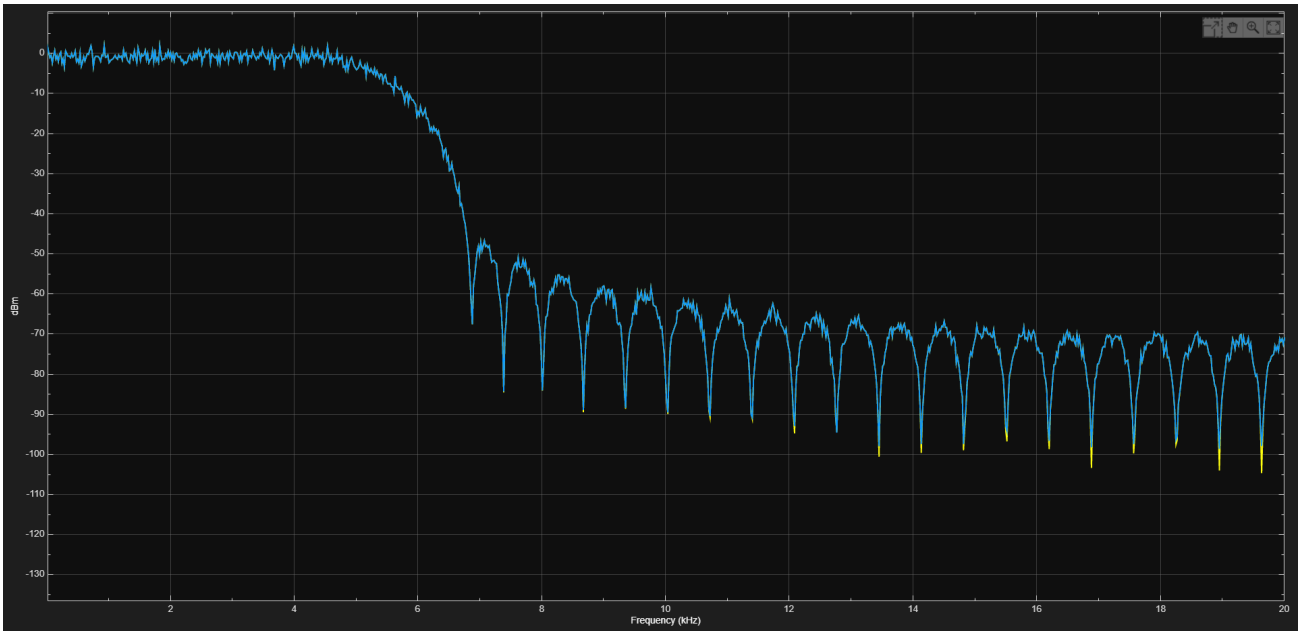
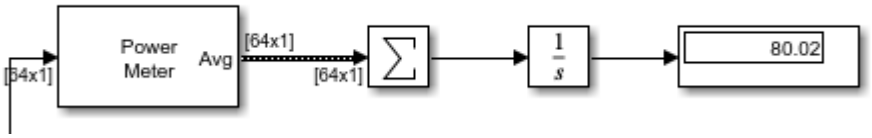


Figure 19: Frequency domain signals from the output of the transmitter to the input of the receiver at 100dB

The two graphs compare the frequency domain signals from the output of the transmitter to the input of the receiver at SNR levels of 9 dB and 100 dB respectively. In the first graph, the transmitted waveform (yellow) shows significant attenuation and distortion due to the low SNR condition. The received waveform (blue) is heavily affected by noise, resulting in high

fluctuations and reduced amplitude, indicating a degraded signal quality. In contrast, the second graph shows that the transmitted signal maintains its amplitude and consistency across the frequency spectrum with minimal distortion. The received waveform closely matches the transmitted signal, indicating maximum signal integrity with minimal noise interference.



We use the following configuration above to calculate the total energy over a given time frame. In the above example we send the transmitted waveform into the power meter. The sum of elements block sums the 64 elements per frame and feeds it to the integrator giving us the total energy. As we can see the total energy over a duration of 5 seconds gives us 80.02 J of energy.

6 Channel

The Variance for the AWGN Channel is calculated as follows:

$$10^{-SNR/10} = 10^{-9/10} = 0.126$$

$$10^{-SNR/10} = 10^{-21/10} = 0.00794$$

With solving the following equations, we are able to get that at steady state, the average amount of time the channel is in the good state is 80%, and the average time it is in the bad state is 20%:

x: fraction of time the channel is in the good state at steady state

y: fraction of time the channel is in the bad state at steady state

x + y = 1

$$\begin{bmatrix} 0.95 & 0.8 \\ 0.05 & 0.2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

References

[1] Information on convolutional codes from:

https://en.wikipedia.org/wiki/Convolutional_code

[2] Graph for BER for Convolutional codes taken from Wikimedia Commons:

https://commons.wikimedia.org/wiki/File:Lenss.png

3] Information on the implementation and BER of QPSK found from:

https://www.gaussianwaves.com/2010/10/qpsk-modulation-and-demodulation-2/

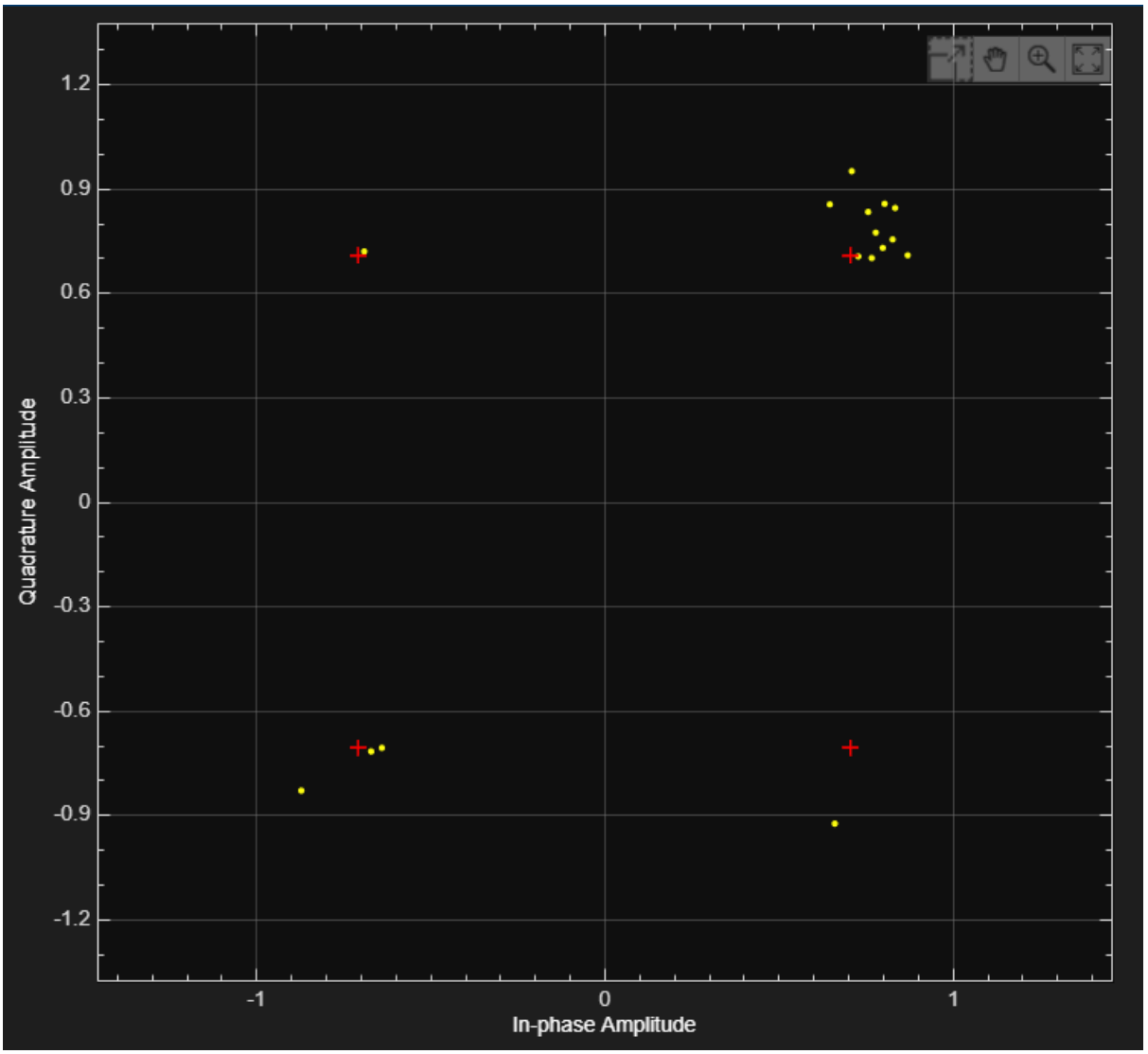
Appendix A.

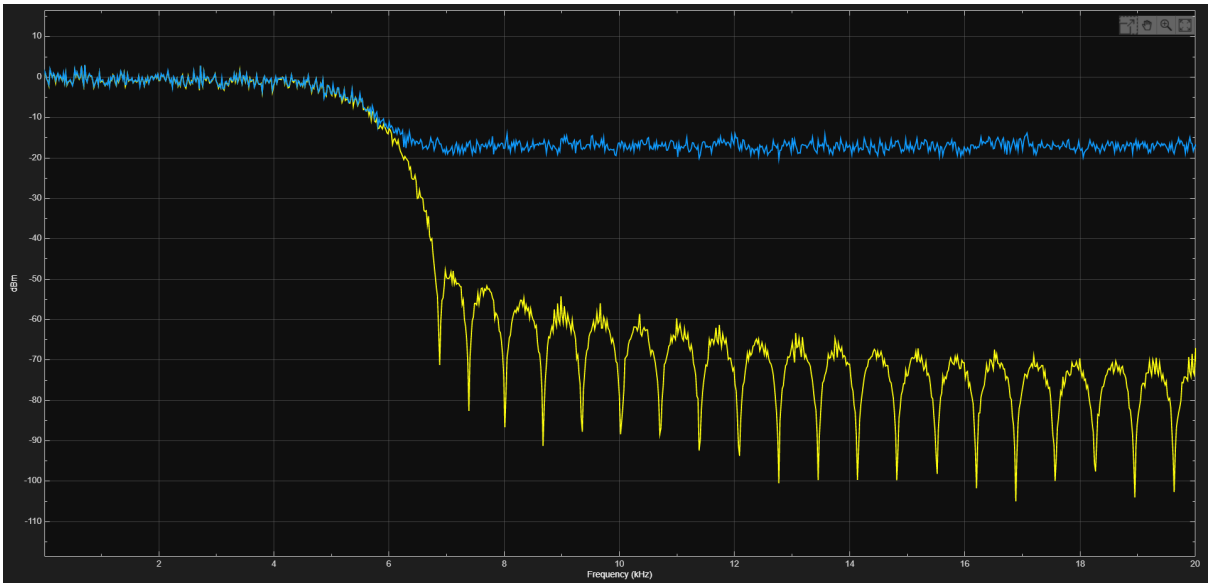
1. The Gilbert fading model code is generated by ChatGPT

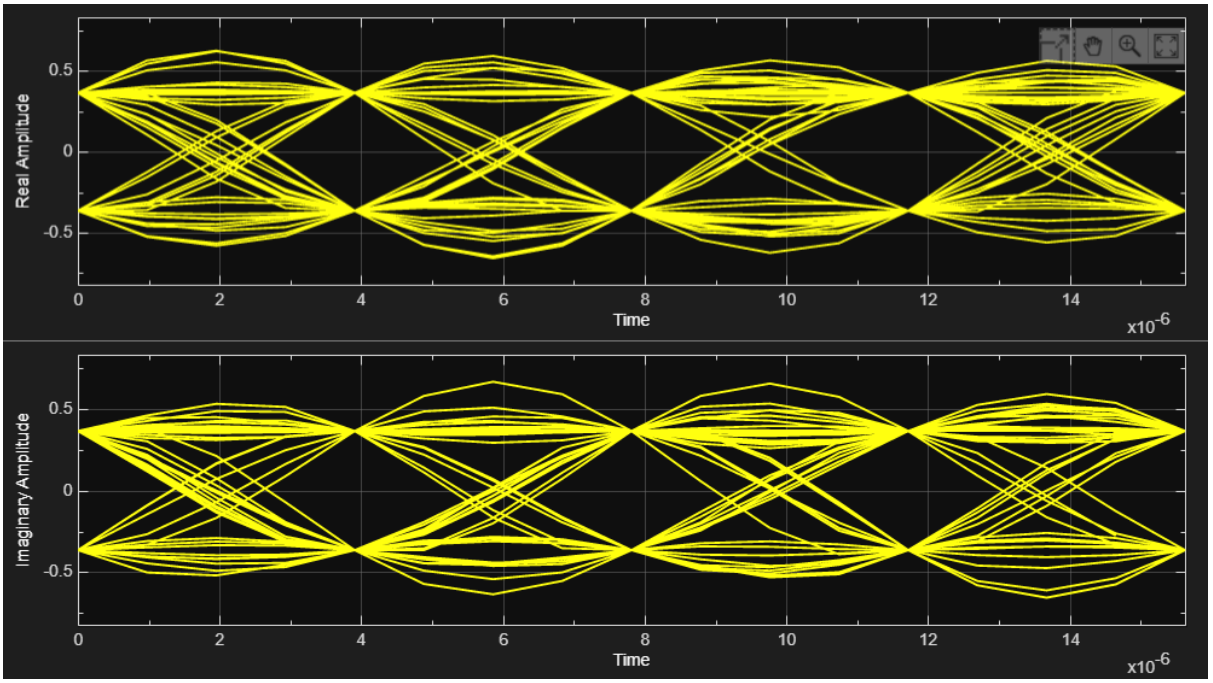
```
function [v, SNR] = gilbert(u)

% Define Transition Matrix
P = [0.95, 0.05; 0.2, 0.8];

% Persistent variable to hold the current state
persistent presentstate;

% Initialize the state if it is the first run
if isempty(presentstate)
    presentstate = 1; % Start in state 1 (Good)
    SNR = 21;
end

% Determine the next state based on the current state and input
if presentstate == 1
    if rand < P(1,2)
        nextstate = 2; % Transition to bad state
    else
        nextstate = 1; % Remain in good state
    end
else
    if rand < P(2,1)
        nextstate = 1; % Transition to good state
    else
        nextstate = 2; % Remain in bad state
    end
end

% Update the present state
presentstate = nextstate;

% Set the SNR value based on the current state
if presentstate == 1
    SNR = 21; % Good state
else
    SNR = 10; % Bad state
end

v = u;
```

-

Appendix B.


1.2
0.9
0.6
0.3
0
-0.3
-0.6
-0.9
-1.2
Quadrature Amplitude
-1
0
1
In-phase Amplitude


dBm
Frequency (kHz)


Real Amplitude
Time
x10^-6
Imaginary Amplitude
Time
x10^-6