ELEC 391

Product Document

Team 12

Team Member	Student ID
Dikshit Jain	14526552
Pratham Goel	90300815
Ayzar Farooqui	14647267

Objectives **2**
Requirements **2**
Constraints **2**
System Overview **3**
Subsystem Design **4**
Source/Sink 4
A/D Converters 4
D/A convertor 6
Error Encoder 7
Modulation 7
Transmitter 8
Channel 9
Receiver 10
The Raised Cosine Filter in the receiver end is used to complete the pulse shaping process and to create a matched filter. When the signal is received from the channel, it is distorted and not clear. The Raised Cosine Filter's job is to combat the distortion caused by the channel by minimizing the ICI 11
Demodulation 12
Error Decoder 12
D/A Conversion 13
Verifications **14**
Role Assignments **18**
References: 19

Objectives

The objective of this project is to transmit an audio signal through a noisy channel, with the objective of recovering it on the other side. While perfect noise-free recovery may not be feasible, the goal is to minimize the Bit Error Rate(BER) to a specified threshold.

Requirements

We were required to create a channel section that had a good channel and bad channel that had to follow SNR given in the pdf. We were required to determine the output of the channel section using the Gilbert Channel representation provided in the pdf.

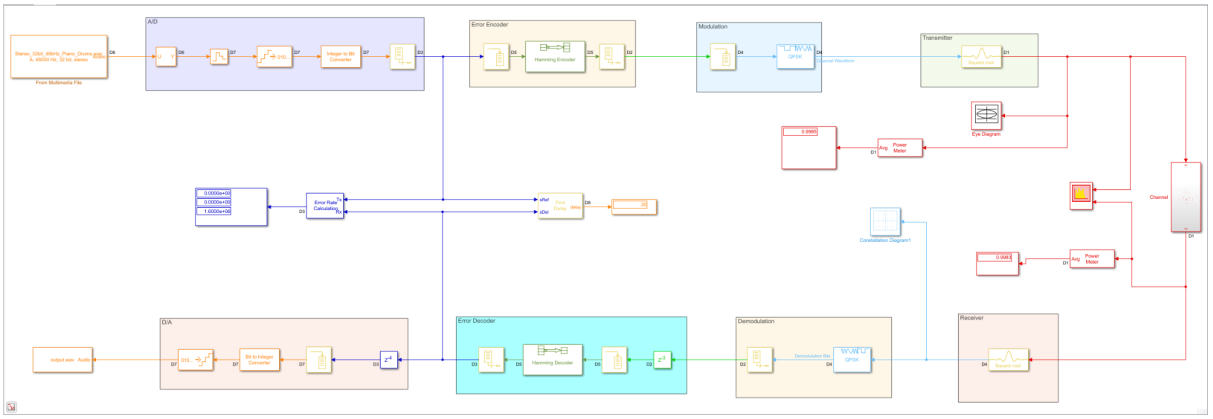
We are required to make a part of our system that includes a good channel and a bad channel. These channels must work with 21 dB and 9 dB that were given to us as part of our constraints. We need to figure out what comes out of the channel using a state machine that represents the Gilbert channel which is given in the pdf. It is crucial to include the exact SNR values from the constraints because they impact how well our system transmits and receives signals.

Constraints

We were assigned scenario F by the teaming team. We were required to have an audio bandwidth of 20 KHz, target BER of 10^-3, Spectral Mask of 200 kHz, use a beta channel for our target channel, and have a delay.

Scenario¶ (Teams)¤	High·Priority¤		¤		Low·Priority¤
	Audio·BW· (kHz)¤	Target·BER¤	Spectral· Mask·(kHz)¤	Target· Channel¤	Delay·(ms)¤
A¤	4¤	10^{-5}¤	100¤	δ¤	25¤
B¤	8¤	10^{-4}¤	100¤	γ¤	25¤
C¤	20¤	10^{-3}¤	150¤	β¤	35¤
D¤	4¤	10^{-5}¤	150¤	δ¤	15¤
E¤	8¤	10^{-4}¤	200¤	γ¤	15¤
F¤	20¤	10^{-3}¤	200¤	β¤	25¤

System Overview



Approach to my system design considering the trade-offs among the system performance criteria: transmit power, bandwidth, throughput (bits transmitted per second), and error rate. Here's a general guideline on how I have designed our system:

Transmit Power: The modulation scheme used (QPSK) is a power-efficient method, especially when compared to schemes like QAM, as it involves phase shifts rather than amplitude shifts.

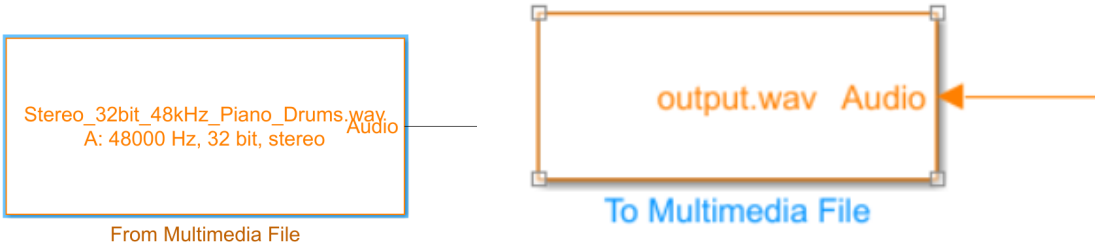
Bandwidth: The bandwidth constraint is defined as 20 kHz for the message and a spectral mask of 200 kHz. To ensure the signal stays within this bandwidth, the rate at which you sample the audio signal was set according to the Nyquist rate (twice the highest frequency). We visualized the spectral content using the Spectrum Analyzer block in Simulink.

Throughput (Bits Transmitted per Second): The throughput of the system is determined by the product of the number of bits per symbol (log2(M) for M-ary modulation) and the symbol rate (symbols per second). In this case, we are using QPSK modulation which means 2 bits per symbol, and given the bandwidth and the Nyquist rate, you can calculate the symbol rate. Multiplying these together gives you the throughput.

Error Rate: The system was designed to achieve a target bit error rate (BER) of 10^{-3}. Error Rate Calculation block in Simulink was used to measure the BER of the system. To achieve this target BER,we need to consider implementing techniques like error correction codes, which can reduce the BER at the cost of increased complexity and reduced throughput.

Subsystem Design

Source/Sink



1. Source Block:

The source block of your project is the "Audio_In_Deserializer" module, which takes input from the Altera DE1 Development Board's audio chip. This module serves to "deserialize" the serial input audio data and translate it into a format that is processable by the subsequent digital processing blocks of your system. It also serves to synchronize the received data with the internal clocking of your system.

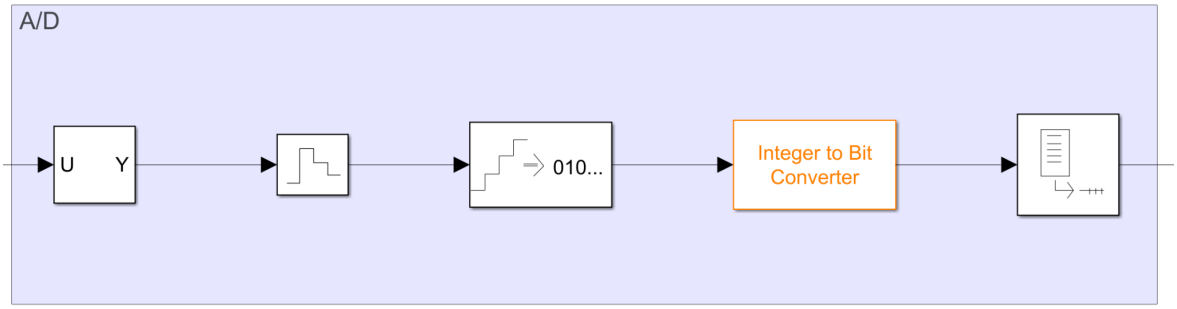
This source block takes in the audio input, clock signals, and control signals and produces audio data that is split into left and right channels. The module also provides signals indicating the amount of available data to read from each channel. The source block in this project is crucial as it's the interface between the real-world analog signals and the digital processing system.

2. Sink Block:

The sink block of your project is the "Audio_Out_Serializer" module, which takes the processed digital data and prepares it for output to the audio chip. It serializes the processed digital audio data from both the left and right channels, preparing it for the digital-to-analog conversion that the audio chip will perform.

The sink block also provides signals indicating the amount of available space to write data to each channel. This ensures that your digital processing system does not attempt to write data faster than it can be processed by the output stage.

A/D Converters



Analog to Digital Conversion (ADC) is a critical process in digital systems, particularly in digital audio applications. It enables the translation of analog signals like sound into digital data that can be processed by digital hardware like microprocessors and FPGAs. This report presents a design that uses both a software model (Simulink) and a hardware platform (FPGA) to validate the effectiveness and accuracy of the ADC system..

Simulink Design

The Simulink ADC system consists of five key blocks:

Selector Block: The Selector block, set with two input dimensions, handles multi-channel audio inputs and selects a specific channel for further processing.

Zero Order Hold Block: This block samples the selected input signal at a rate of 40 kHz, which is twice the maximum frequency of the input signal, satisfying the Nyquist-Shannon sampling theorem for accurate signal reconstruction.

Uniform Encoder: The Uniform Encoder, set with a peak value of 1 and 8 bits, quantizes the sampled signal into 256 discrete levels, transforming the analog signal into digital format.

Integer to Bit Converter: This block converts the quantized samples into a bit stream, creating the final digital output of the ADC process.

Unbuffer Block: The Unbuffer block ensures that the output data is correctly formatted for the next stage of processing, transmission, or storage.

FPGA Implementation

The DE1-SoC FPGA board from Altera provides the hardware platform for the ADC system. It includes several key features relevant to ADC systems, including a variety of digital and analog I/O options.

Input Signal Conditioning: The DE1-SoC board uses its audio inputs as the analog signal source. The board's analog front end conditions the signal before it enters the ADC pipeline.

Clock Generation: The FPGA fabric of the DE1-SoC board generates a 40 kHz clock signal using a Phase-Locked Loop (PLL) to control the rate of input signal sampling.

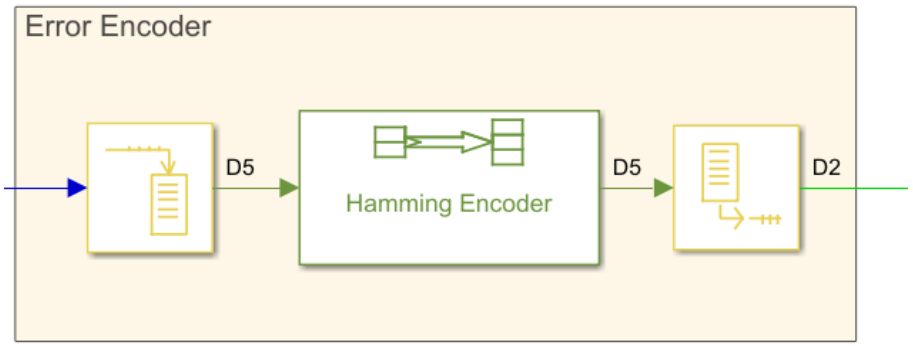
Discretization (Sampling): The FPGA fabric also implements the Zero Order Hold block, sampling the conditioned analog signal at intervals determined by the 40 kHz clock.

Quantization and Encoding: The Uniform Encoder block, implemented in the FPGA fabric, quantizes the sampled signal and transforms it into a digital format.

Digital Output: The Integer to Bit Converter block, also implemented in the FPGA fabric, converts the digital signal into a format suitable for transmission or storage. The DE1-SoC board's digital I/O options, including GPIO pins and several communication interfaces (UART, SPI, I2C), facilitate this process.

ADC Configuration: The FPGA fabric controls the entire ADC process. This is achieved using Verilog code, compiled and uploaded onto the FPGA using Altera's Quartus II software.

Error Encoder



This project focuses on the design and implementation of an error encoder system intended to add redundancy to data bits for error detection and correction. The system is first designed in MATLAB's Simulink environment and then implemented on the DE1-SoC FPGA board using a Hamming (7,4) code.

Simulink Design

The Simulink error encoder system comprises three main blocks:

Buffer Block: The Buffer block groups every 4 bits of the input data into a frame, preparing the data for the Hamming Encoder block.

Hamming Encoder (7, 4): This block takes each frame of 4 data bits and adds 3 redundant bits, creating a 7-bit codeword. The additional bits allow for the detection and correction of single-bit errors at the receiving end.

Unbuffer Block: This block converts the 7-bit frames from the Hamming Encoder back into a stream of bits, ready for transmission.

FPGA Implementation

The DE1-SoC FPGA board from Altera, used as the hardware platform, integrates several features suitable for the error encoding system:

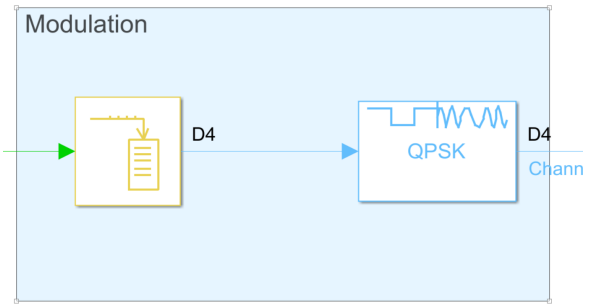
Input Buffering: The incoming data bits are grouped into frames of 4 bits using FPGA logic, mirroring the function of the Buffer block in the Simulink design.

Hamming (7, 4) Encoding: A custom Verilog module, Hamming74_Encoder, performs the Hamming (7,4) encoding operation on the FPGA. The module takes 4 bits of data as input and adds 3 redundant bits, resulting in a 7-bit codeword.

```
module Hamming74_Encoder(
    input logic [3:0] data_in,
    output logic [6:0] data_out
);

    logic p1, p2, p4;

    assign p1 = data_in[0] ^ data_in[1] ^ data_in[3];
    assign p2 = data_in[0] ^ data_in[2] ^ data_in[3];
    assign p4 = data_in[1] ^ data_in[2] ^ data_in[3];

    assign data_out = {data_in[3:1], p4, data_in[0], p2, p1};

endmodule
```

Unbuffering and Output: The encoded data frames are converted back into a stream of bits using additional FPGA logic, similar to the Unbuffer block in the Simulink model. This data stream is then ready for transmission through the FPGA board's digital I/O interfaces.

Modulation



The design involves utilizing a buffer block to group input data properly before modulation. The ultimate goal of the subsystem is to convert a digital bitstream into a modulated signal that is suitable for transmission over a communication channel. This project focuses on the QPSK modulation scheme, which is frequently used due to its balance of data rate and robustness against noise.

Simulink Design

The Simulink QPSK modulation system comprises two main blocks:

Buffer Block: The Buffer block groups the input data into frames of 2 bits, preparing the data for the QPSK Modulation block. This is because each QPSK symbol is represented using two bits.

QPSK Modulation Baseband: This block takes the 2-bit frames from the Buffer block and uses them to modulate a carrier signal. Each input pair of bits corresponds to one of the four possible phase shifts of the carrier signal. The output is a complex baseband representation of the modulated signal, ready for transmission.

FPGA Implementation
The DE1-SoC FPGA board from Altera, used as the hardware platform, integrates several features suitable for the modulation system:

Input Buffering: The incoming data bits are grouped into frames of 2 bits using FPGA logic, mirroring the function of the Buffer block in the Simulink design.

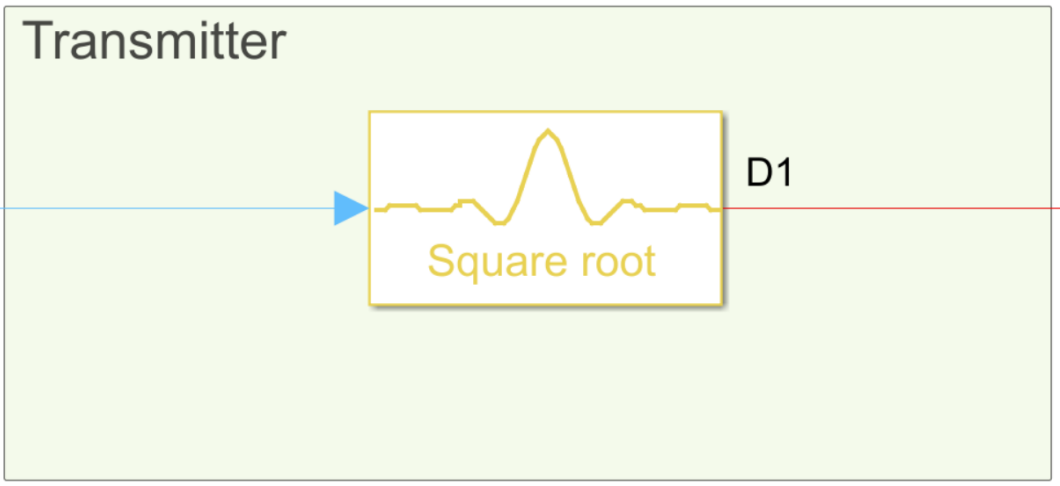
QPSK Modulation: A custom Verilog module, QPSK_modulator, performs the QPSK modulation operation on the FPGA. It takes a pair of bits as input and outputs two 16-bit values representing the in-phase (I) and quadrature (Q) components of the modulated signal.

Below is the Verilog module:

```
module Modulation(
    input reset,
    input [7:0] data_in,
    output signed [15:0] I_out1,
    output signed [15:0] I_out2,
    output signed [15:0] I_out3,
    output signed [15:0] I_out4,
    output signed [15:0] Q_out1,
    output signed [15:0] Q_out2,
    output signed [15:0] Q_out3,
    output signed [15:0] Q_out4
    );

    QPSK_modulator QPSK_mod1(.reset(reset), .data_in(data_in[1:0]), .I_out(I_out1), .Q_out(Q_out1));
    QPSK_modulator QPSK_mod2(.reset(reset), .data_in(data_in[3:2]), .I_out(I_out2), .Q_out(Q_out2));
    QPSK_modulator QPSK_mod3(.reset(reset), .data_in(data_in[5:4]), .I_out(I_out3), .Q_out(Q_out3));
    QPSK_modulator QPSK_mod4(.reset(reset), .data_in(data_in[7:6]), .I_out(I_out4), .Q_out(Q_out4));

endmodule


module QPSK_modulator(
    input logic reset,
    input logic [1:0] data_in,
    output logic signed [15:0] I_out,
    output logic signed [15:0] Q_out
);

    //Define Constants
    localparam signed [15:0] POS = 16'b0000_0000_0000_0001;  // Represents +1
    localparam signed [15:0] NEG = 16'b1000_0000_0000_0000;  // Represents -1

    always_comb begin
        if (reset) begin
            I_out <= 16'd0;
            Q_out <= 16'd0;
        end else begin
            case (data_in)
                2'b00: begin
                    I_out <= NEG;
                    Q_out <= NEG;
                end
                2'b01: begin
                    I_out <= POS;
                    Q_out <= NEG;
                end
                2'b10: begin
                    I_out <= NEG;
                    Q_out <= POS;
                end
                2'b11: begin
                    I_out <= POS;
                    Q_out <= POS;
                end
            endcase
        end
    end
endmodule
```

Output and Transmission: The modulated I and Q signals are then ready for further processing, such as digital-to-analog conversion, before being transmitted over a communication channel.

Transmitter



The subsystem design and implementation of a Raised Cosine Transmit Filter for a digital transmission system using MATLAB Simulink. The system employs specific parameter values for the rolloff factor, filter span in symbols, output samples per symbol, and filter gain, which are all critical to the filter's overall performance.

System Design:
The transmitter subsystem comprises the Raised Cosine Transmit Filter, designed with the following parameters:

1. Roll Off Factor:
The rolloff factor is a critical parameter that influences the bandwidth and ISI. In this project, a roll off factor of 0.4 is selected, which strikes a balance between spectral efficiency and ISI.

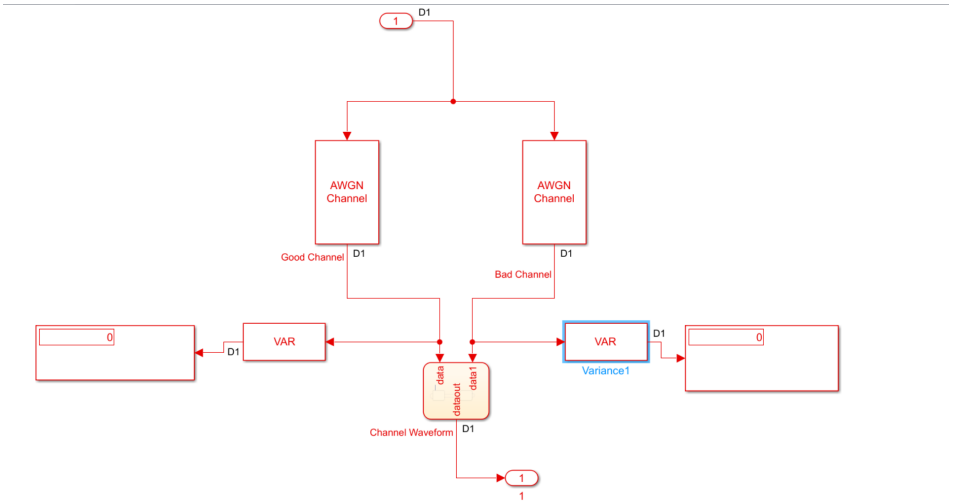
2. Filter Span in Symbols:
The filter span in symbols, set to 8, is the duration of the filter response in symbol periods. This influences the filter's complexity and its ability to minimize ISI.

3. Output Samples per Symbol:
This parameter, set to 4, controls the rate of oversampling or upsampling. It defines how many output samples are generated by the filter for each symbol period.

4. Filter Gain:
The filter gain, set to 2, controls the strength of the filter. This linear amplitude gain scales the output signal to achieve the desired power level.

In the transmitter section, it is receiving signals that have experienced Inter-symbol Interference(ISI) which causes the signals to overlap. Due to this, a symbol can be mistaken for another one leading to data errors. To fix such a problem, we used a Raised Cosine filer which allows us to shape the signal in a way so that they don't interfere with each other. The Raised Cosine filter allows the signal to be transmitted to the channel section without any errors or overlapping in the transmitted signal.

Channel



The AWGN Channel module provides an environment that closely resembles real-world communication, where transmitted signals are often affected by white Gaussian noise. The noise introduced by the AWGN Channel is completely random and has a constant power spectral density. The randomness of the noise means it has equal power at any frequency within a given band. It's "additive" because it adds this noise to the signal being transmitted.

Here, two AWGN modules are employed, each introducing a different level of noise, emulating "good" and "bad" channel conditions. The signal-to-noise ratio (SNR) plays a crucial role in distinguishing these conditions: a higher SNR (21dB) corresponds to the "good" channel, meaning the signal strength is much higher than the noise level. Conversely, a lower SNR (9dB) corresponds to the "bad" channel, signifying a higher level of noise relative to the signal strength.

In the FPGA, two separate memory modules are used to store pre-calculated Gaussian noise values for 21dB and 9dB SNR. The modules are ROMs (Read-Only Memory) and loaded with noise values saved in .mif (Memory Initialization File) format.

An address signal, generated elsewhere in the design, accesses the memory. On each clock cycle, a 16-bit wide noise value is read out from the memory and added to the transmitted signal. The system transitions between these modules ("good" and "bad" channels) based on a Gilbert State model, not shown here.

The Verilog code provided here only shows the AWGN Channel modules; it doesn't include the overall state machine that manages the transitions between the "good" and "bad" channels. Also, it's worth noting that an actual FPGA implementation should also include necessary adjustments for real-world applications, such as channel encoding and decoding blocks.

The Verilog code for each AWGN module is a standard template for accessing Altera's on-chip memory (RAM or ROM). The parameters at the bottom of the module instantiation ('defparam') define the memory characteristics, including the initialization file, memory size, operational mode (RAM or ROM), and others. This ROM module acts as a Look-Up Table (LUT) for AWGN noise values. On each clock cycle, it returns a 16-bit noise value corresponding to the provided address.

The clock signal is marked as a tri-state buffer (tri1). This is likely to prevent the synthesis tool from optimizing the clock signal in ways that could potentially disrupt the system's timing behavior.

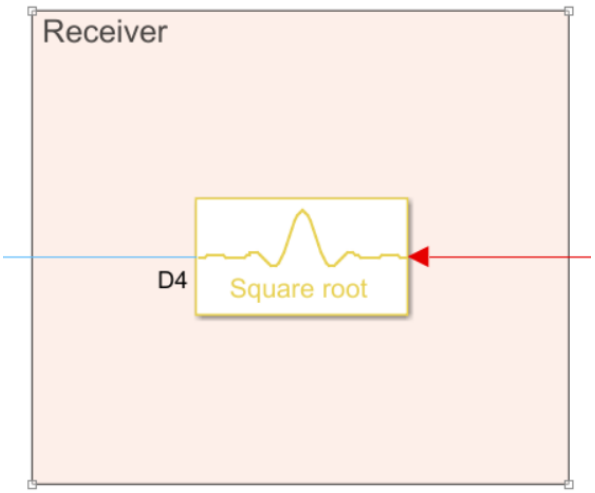
Finally, it's important to note that all code snippets here are isolated modules. In practice, these would be integrated into a larger system and connected appropriately. Consequently, the given modules may require slight modifications to fit a specific application or design framework.

Testing and Results

To validate our implementation, we tested both the "Good Channel" and "Bad Channel" using a set of known input data and verified the output. By examining the variance of the output signals, we were able to confirm that the correct SNR was being achieved in both

channels. As expected, the "Bad Channel" demonstrated a higher level of distortion in the transmitted signal compared to the "Good Channel".

Receiver



The Raised Cosine Filter in the receiver end is used to complete the pulse shaping process and to create a matched filter. When the signal is received from the channel, it is distorted and not clear. The Raised Cosine Filter's job is to combat the distortion caused by the channel by minimizing the ICI.

System Design:
The receiver subsystem consists of a Raised Cosine Receive Filter, defined by the following parameters:

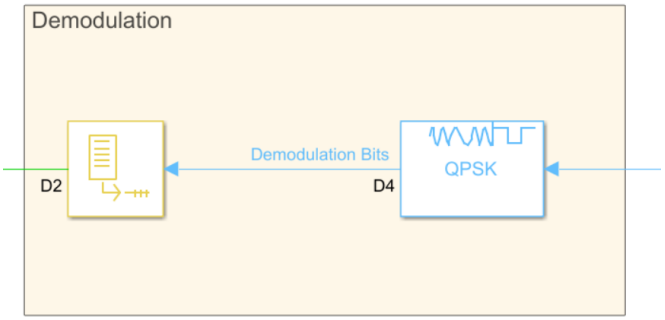
1. Rolloff Factor:
The rolloff factor, set to 0.4 for this project, influences the filter's bandwidth and its ability to minimize ISI. This value is usually set to match the rolloff factor used in the transmitter filter to ensure optimal performance.

2. Filter Span in Symbols:
This parameter, set to 8, represents the duration of the filter's impulse response in symbol periods. It directly impacts the complexity of the filter and its performance in minimizing ISI.

3. Input Samples per Symbol:
Set to 4, this parameter determines the rate of down-sampling. It defines the number of input samples to be processed by the filter for each symbol period.

4. Filter Gain:
Set to 2, the filter gain determines the scaling of the output signal. The gain ensures that the signal amplitude is suitable for the subsequent stages in the receiver.

Demodulation



The demodulation process in a digital communication system is critical to recovering the original transmitted information from the received signal. The subsystem primarily consists of the QPSK Demodulator and an Unbuffer block.

Simulink Design
The QPSK demodulation system in Simulink consists of two main blocks:

QPSK Demodulator Band: The primary component in the demodulation process. This block receives the QPSK modulated signal and reconverts it back into the original binary data by examining the phase of the incoming signal.

Unbuffer Block: After demodulation, the Unbuffer block restructures the output data back into its original format. The data is transformed from a column-based format, which is the output from the QPSK Demodulator Band, to a row-based format, suitable for further processing or display.

FPGA Implementation
The DE1-SoC FPGA board from Altera, utilized as the hardware platform, integrates multiple features suitable for the demodulation process:

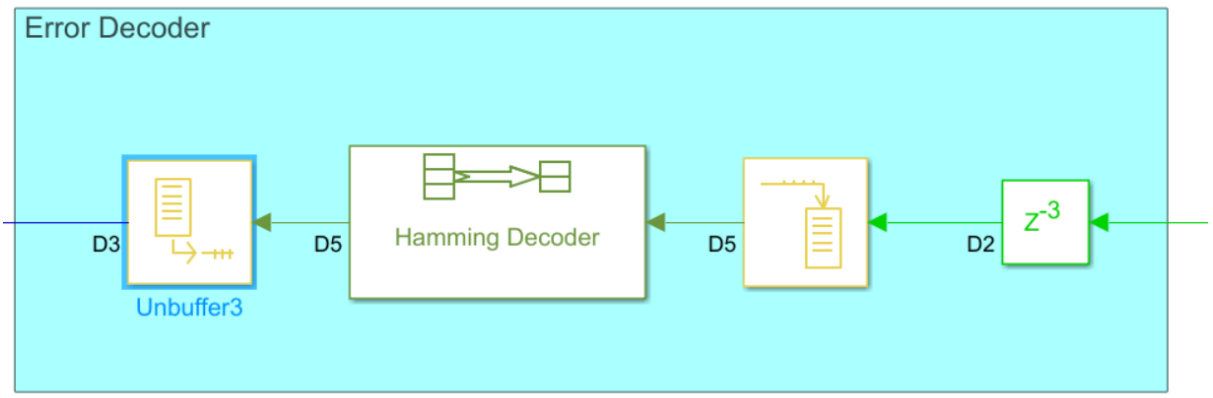
QPSK Demodulation: A custom Verilog module, QPSK_demodulator, performs the QPSK demodulation operation on the FPGA. It takes in-phase (I) and quadrature (Q) signals as inputs and outputs a pair of binary data corresponding to the phase of the input signal.

Below is the Verilog module:

```
module Demodulation(
    input logic reset,
    input logic signed [15:0] I_in1,
    input logic signed [15:0] I_in2,
    input logic signed [15:0] I_in3,
    input logic signed [15:0] I_in4,
    input logic signed [15:0] Q_in1,
    input logic signed [15:0] Q_in2,
    input logic signed [15:0] Q_in3,
    input logic signed [15:0] Q_in4,
    output [7:0] data_out
    );

    QPSK_demodulator QPSK_demod1(.reset(reset), .I_in(I_in1), .Q_in(Q_in1), .data_out(data_out[1:0]));
    QPSK_demodulator QPSK_demod2(.reset(reset), .I_in(I_in2), .Q_in(Q_in2), .data_out(data_out[3:2]));
    QPSK_demodulator QPSK_demod3(.reset(reset), .I_in(I_in3), .Q_in(Q_in3), .data_out(data_out[5:4]));
    QPSK_demodulator QPSK_demod4(.reset(reset), .I_in(I_in4), .Q_in(Q_in4), .data_out(data_out[7:6]));

endmodule


module QPSK_demodulator(
    input logic reset,
    input logic signed [15:0] I_in,
    input logic signed [15:0] Q_in,
    output logic [1:0] data_out
);

    //Define Constants
    localparam signed [15:0] POS = 16'b0000_0000_0000_0001;  // Represents +1
    localparam signed [15:0] NEG = 16'b1000_0000_0000_0000;  // Represents -1

    always_comb begin
        if (reset) begin
            data_out <= 2'b00;
        end else begin
            if({I_in, Q_in} == {NEG, NEG}) data_out = 2'b00;
            else if({I_in, Q_in} == {POS, NEG}) data_out = 2'b01;
            else if({I_in, Q_in} == {NEG, POS}) data_out = 2'b10;
            else if({I_in, Q_in} == {POS, POS}) data_out = 2'b11;
                        else data_out = 2'b00;
        end
    end
endmodule
```

Output: The demodulated data is now ready for further processing or display. Multiple instances of the QPSK_demodulator module can be used simultaneously for demodulation of multiple QPSK symbols, with the output data distributed among the instances.

Error Decoder



Error detection and correction are critical to digital communication systems. Hamming codes are especially prevalent due to their ability to detect and correct errors. This project explores a subsystem that employs a (7,4) Hamming decoder, capable of decoding a 7-bit codeword into 4-bit data, while detecting and correcting single-bit errors.It leverages both Simulink and Field Programmable Gate Arrays (FPGA) to transform a 7-bit Hamming codeword back into original 4-bit data while detecting and correcting single-bit errors.

Simulink Design

The (7,4) Hamming decoder subsystem in Simulink comprises four main blocks:

Delay Block: With a delay length of 3, this block synchronizes the data flow in the system.

Buffer Block: With an output buffer size of 7, this block transforms the incoming data stream into frames of the specified length. This transformation is essential for the Hamming decoder to function, as it operates on 7-bit codewords.

Hamming Decoder (7,4): This block accepts 7-bit codewords, detects and corrects single-bit errors, and outputs 4-bit data. Its error correction capability is key to maintaining the system's overall performance and ensuring data accuracy and integrity.

Unbuffer Block: This block reverts the output data back to its original stream format, completing the decoding process.

FPGA Implementation

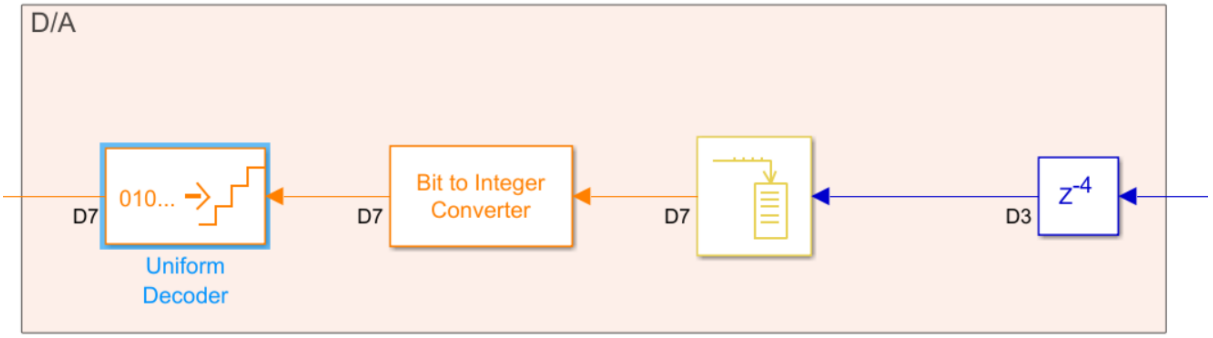
The DE1-SoC FPGA board from Altera, utilized as the hardware platform, integrates various features suitable for the decoding process:

Hamming (7,4) Decoder: A custom Verilog module, Hamming74_Decoder, performs the Hamming decoding operation on the FPGA. It takes a 7-bit codeword as input, detects and corrects single-bit errors, and outputs a 4-bit binary data.
Below is the Verilog module:

```
module Hamming74_Decoder(
    input logic [6:0] data_in,
    output logic [3:0] data_out,
    output logic error
  );

  logic p1, p2, p4;
  logic [6:0] syndrome; //one hot value for the bit to be corrected
  logic [6:0] decoded_data;
  logic parity;

  always_comb begin
    p1 = data_in[0] ^ data_in[2] ^ data_in[4] ^ data_in[6];
    p2 = data_in[1] ^ data_in[2] ^ data_in[5] ^ data_in[6];
    p4 = data_in[3] ^ data_in[4] ^ data_in[5] ^ data_in[6];
  end


  always_comb begin
    case({p4,p2,p1})
        3'd1: syndrome = 7'b0000_001;
        3'd2: syndrome = 7'b0000_010;
        3'd3: syndrome = 7'b0000_100;
        3'd4: syndrome = 7'b0001_000;
        3'd5: syndrome = 7'b0010_000;
        3'd6: syndrome = 7'b0100_000;
        3'd7: syndrome = 7'b1000_000;
        default: syndrome = 7'b0;
    endcase
  end

  // Error detection
  always_comb begin
    error = (syndrome != 3'b000);
  end

  assign decoded_data = syndrome ^ data_in; //correct the error if found

  assign data_out = {decoded_data[6:4], decoded_data[2]};

endmodule
```

Output: The decoded data is now ready for further processing or display. Multiple instances of the Hamming74_Decoder module can be used simultaneously for decoding multiple Hamming codewords, with the output data distributed among the instances.

D/A Conversion



The designed system is aimed at converting an 8-bit digital signal back into an analog signal. The key blocks involved in this process are a Delay block, Buffer block, Bit to Integer Converter, and a Uniform Decoder.This project centers on the design and implementation of a Digital to Analog Converter (DAC) system that takes an 8-bit digital signal and converts it

back to an analog signal. The DAC system is first designed in MATLAB's Simulink environment and then implemented on the DE1-SoC FPGA board from Altera.

Simulink Design
The Simulink DAC system comprises four main blocks:

Delay Block: The Delay block, set with a delay length of 4, introduces a delay of 4 samples to the input bitstream. This helps in ensuring the synchronization of data.

Buffer Block: This block groups the delayed bitstream into frames of 8 bits each, preparing the bitstream for the next stage, which expects integer inputs.

Bit to Integer Converter: The Bit to Integer Converter block converts each frame of 8 bits into an integer. This provides an integer stream ready to be decoded into the original signal values.

Uniform Decoder: This block, performing the inverse operation of the ADC's Uniform Encoder, converts the integer stream back into the original signal levels.

FPGA Implementation
The DE1-SoC FPGA board uses the hardware platform, integrates several features suitable for the DAC system.

Input Buffering: The digital bitstream is input to the FPGA through its various digital interfaces, such as GPIO, UART, SPI, and I2C. The FPGA logic groups the incoming data into frames of 8 bits, similar to the Buffer block in the Simulink design.

Bit to Integer Conversion: The FPGA implements a Bit to Integer Converter, transforming each frame of 8 bits into an integer using dedicated digital logic.

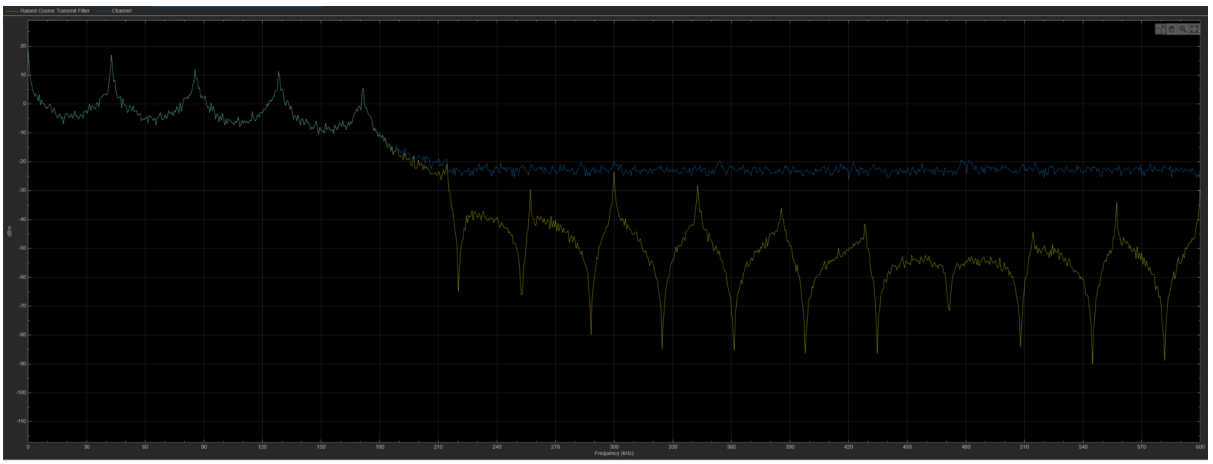
Decoding: The Uniform Decoder logic, mapped in the FPGA fabric, decodes the integer stream back into the original signal levels, effectively converting the digital signal back to an analog format.

DAC Configuration: The entire DAC process, including all blocks mentioned above, is controlled via the FPGA fabric on the DE1-SoC board. This control logic is written using Verilog, compiled and uploaded onto the FPGA using Quartus II, Altera's FPGA design software.

Analog Output Generation: The FPGA board's integrated DAC is used to generate the final analog output from the decoded digital signal. The board's audio output interfaces are used to output the reconstructed analog signal.

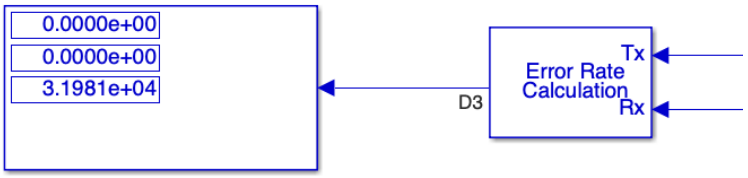
Verifications

1. Source/Sink: Spectral Analyzer output



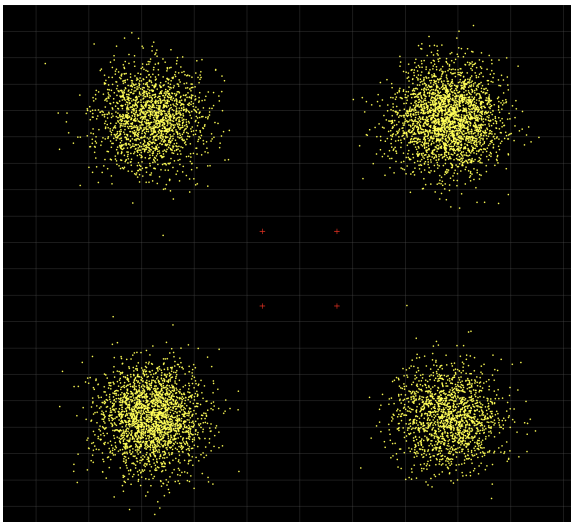
2. A/D and D/A converters:

BER for both 100dB and 9dB is same

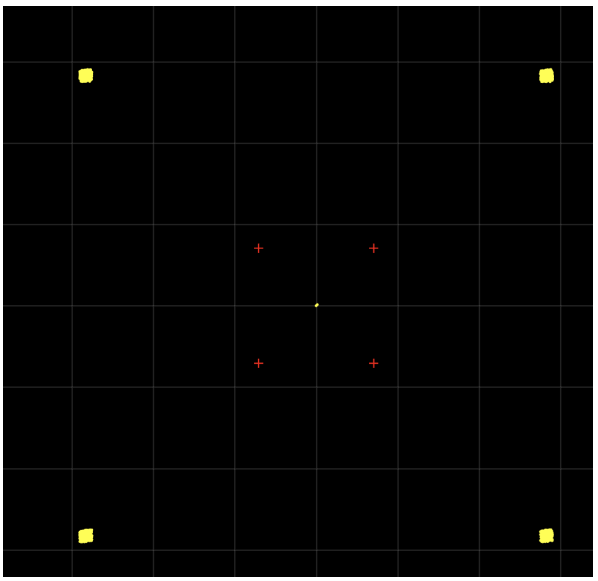


3. Modulator / Demodulator: Constellation scatter plot

9 dB

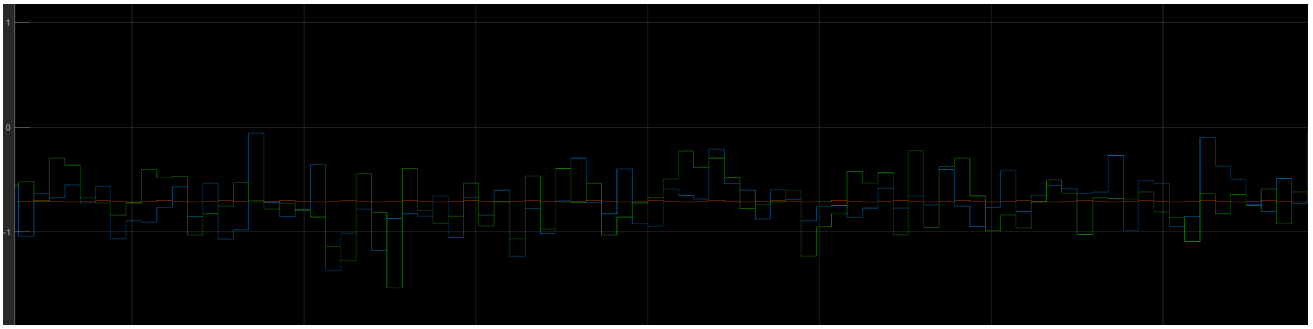


100 dB

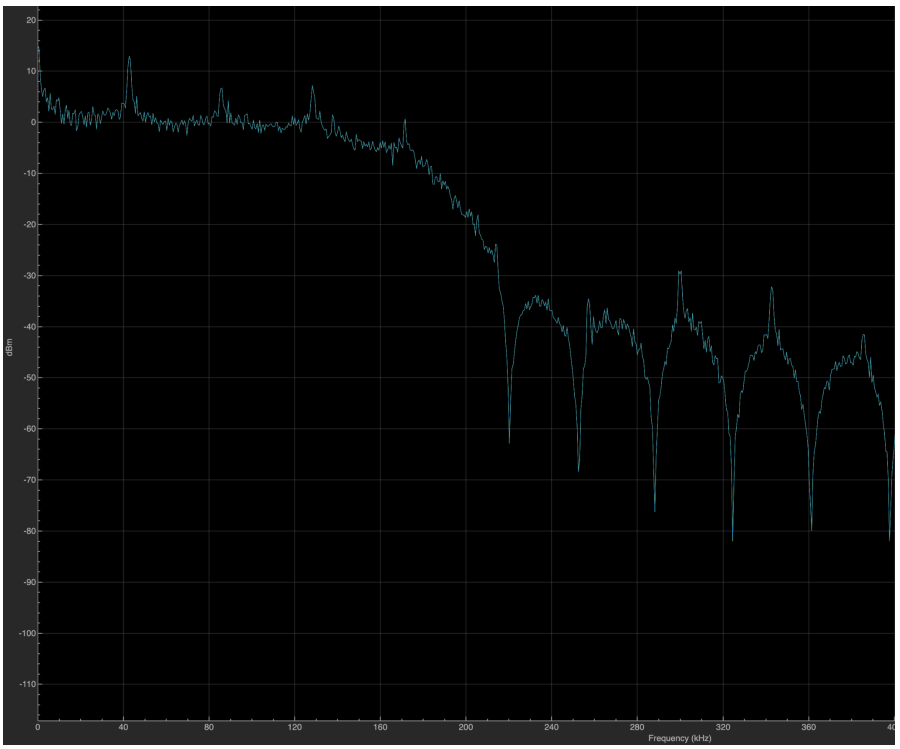


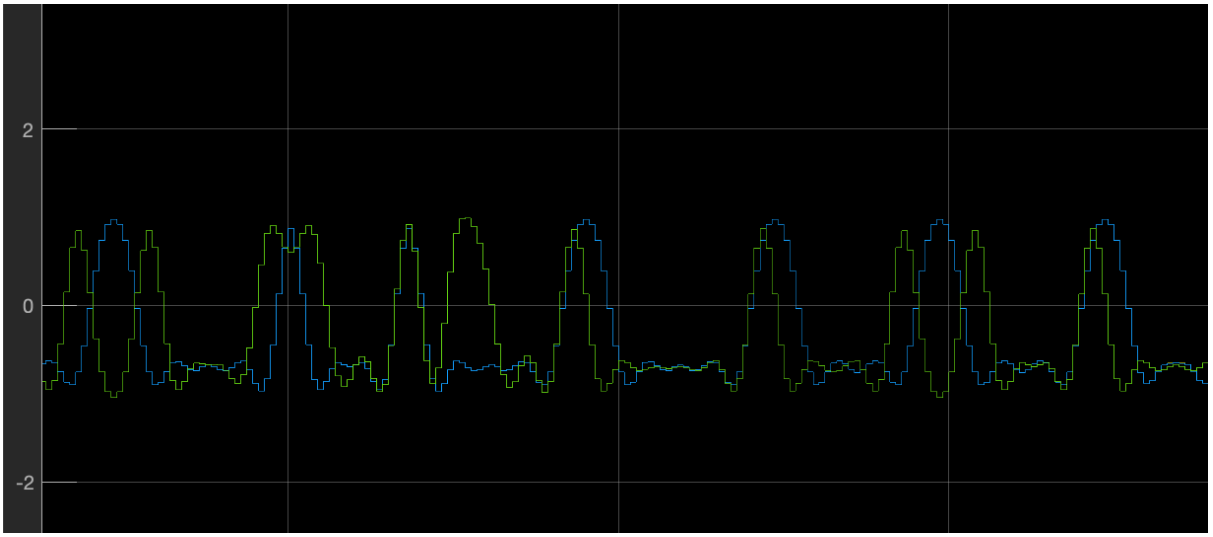
4. Transceiver / Receiver: Time and Frequency domain signals

9 dB

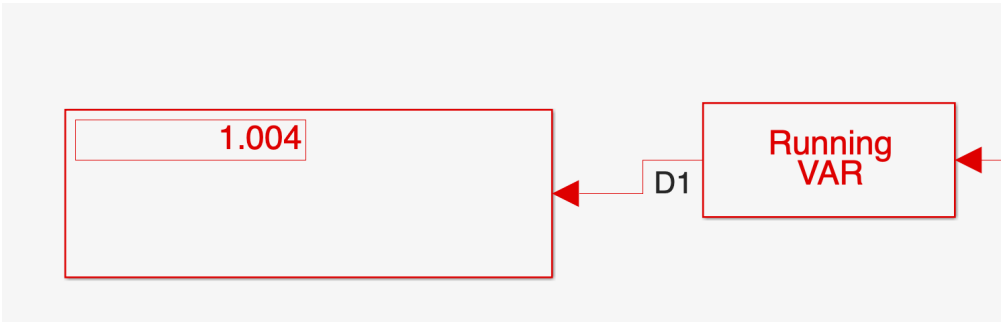


100 dB

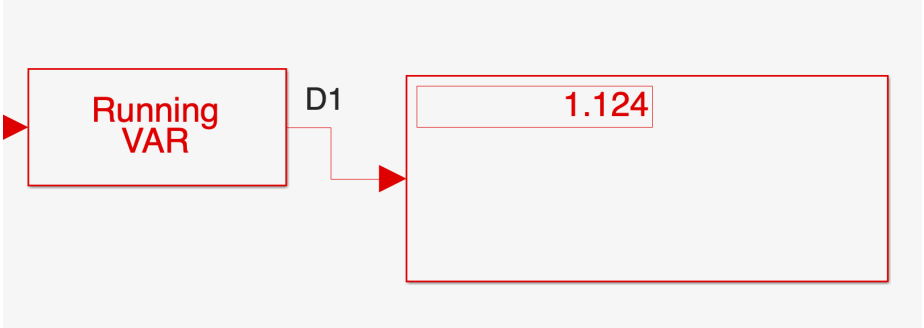




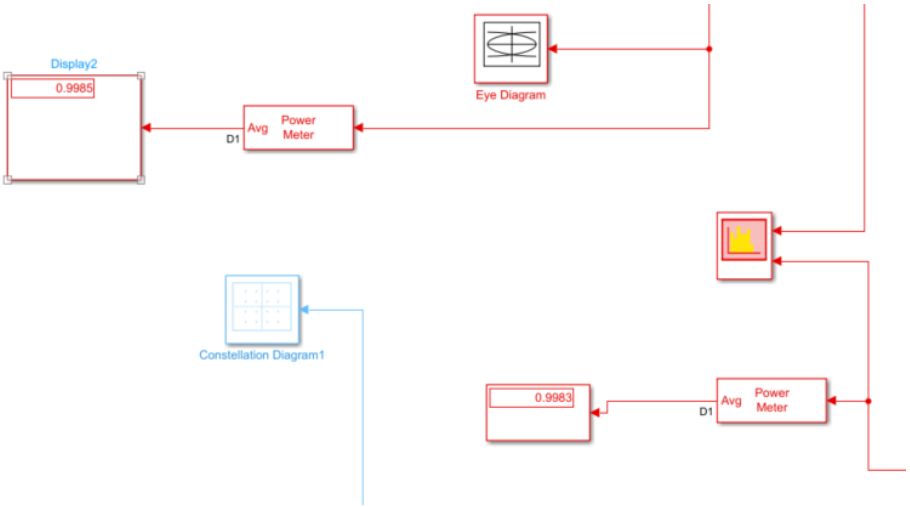
5. Channel: Variance for Good Channel with 21dB



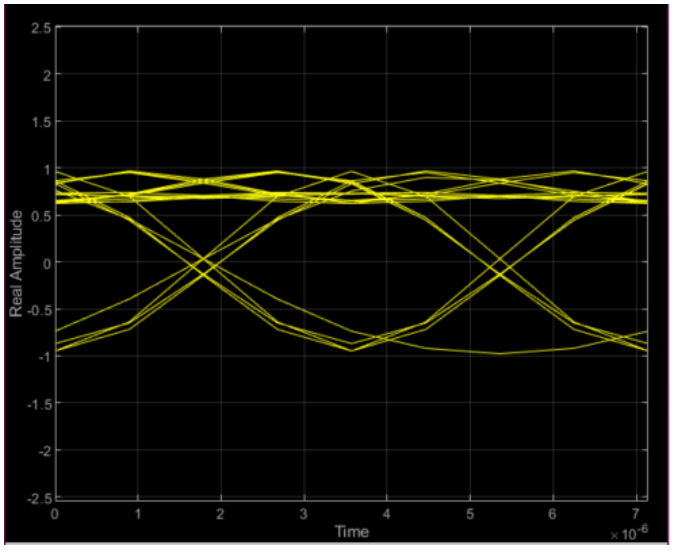
Variance for Bad Channel with 9 dB



Power:



Eye Diagram

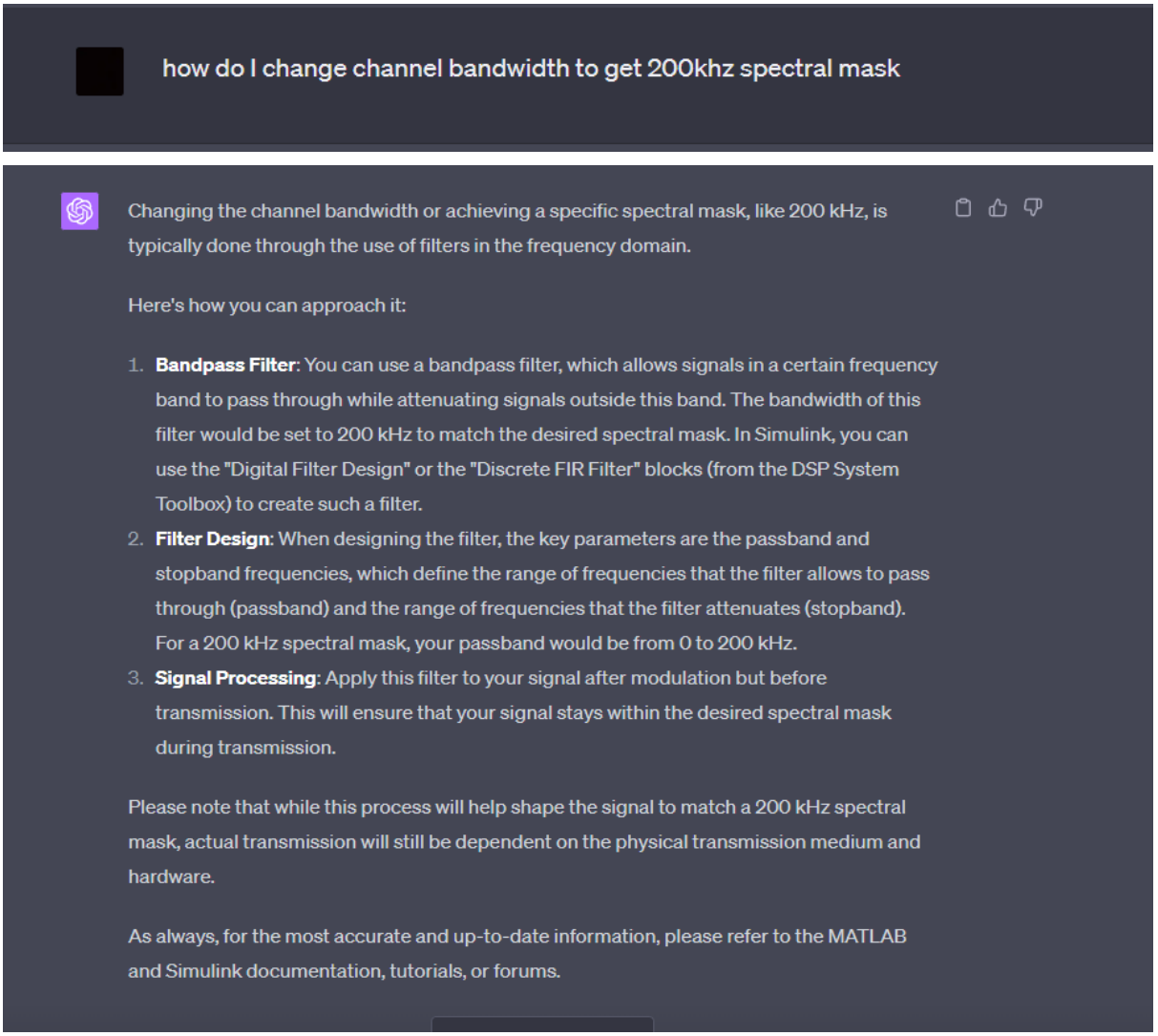


Role Assignments

Team Member	
Pratham Goel	A/D and D/A Conversion, Error encoding/decoding
Dikshit Jain	Transmitter/ Receiver, modulation/ Demodulation
Ayzar Farooqui	Noise Channel, Input/Output formats

References:

1. Proakis, John G. Digital communications. McGraw-Hill Companies, 2001.
2. Haykin, Simon S., and Michael Moher. "Communication systems." (2009).
3. MATLAB and Simulink Documentation. The MathWorks Inc., 2021. Web.
4. ChatGPT prompts:



what can be values for rolloff factor and filter span filter according to my contraints

The rolloff factor and filter span are parameters for a Raised Cosine Transmit and Receive Filter, often used in digital communication systems for pulse shaping to limit the signal bandwidth.

1. **Rolloff Factor (β)**: This parameter defines the excess bandwidth of the filter and typically varies between 0 and 1. A rolloff factor of 0 results in the narrowest bandwidth but with abrupt transitions, while a rolloff factor of 1 gives a much wider bandwidth but with smoother transitions. Given that you have a spectral mask of 200 kHz, you could start by setting the rolloff factor to a moderate value like 0.5, and then adjust it as needed to meet your system requirements.
2. **Filter Span in Symbols**: This parameter indicates the duration of the filter in symbol durations. A greater filter span gives better performance in terms of the signal-to-noise ratio but at the cost of increased complexity. A typical value to start with is 6, and then it can be adjusted according to your specific requirements.

Remember that these parameters are usually chosen based on trade-offs between bandwidth efficiency, complexity, and system performance (e.g., signal-to-noise ratio, bit error rate). Therefore, they might need to be fine-tuned according to your system requirements and constraints.

Also note that the system might need to consider the inter-symbol interference (ISI) caused by the multipath fading channels, where a matched filter or an equalizer might be required to mitigate the ISI effect. The parameter setting might also be adjusted according to this issue.

As always, refer to the MATLAB and Simulink documentation for more detailed information and instructions on how to use these blocks and optimize your model.

Regenerate response

Pros os using QPSk over QAM

Quadrature Phase Shift Keying (QPSK) and Quadrature Amplitude Modulation (QAM) are both digital modulation techniques used in wireless communications. Here are some advantages of using QPSK over QAM:

1. **Simplicity:** QPSK is simpler to implement than QAM as it only varies the phase of the carrier signal and not the amplitude. In contrast, QAM varies both the phase and amplitude, making it more complex to implement.
2. **Better Performance in Noisy Environments:** QPSK tends to be more resistant to noise than QAM because changing the amplitude (as in QAM) makes the signal more susceptible to variations due to noise. Hence, in a noisy channel environment, QPSK can provide a better bit error rate (BER) performance than QAM.
3. **Power Efficiency:** The power efficiency of QPSK is better than QAM. Since QPSK uses constant envelope modulation (the amplitude of the signal remains constant), it allows the use of non-linear power amplifiers to achieve high power efficiency. QAM requires linear amplifiers, which are less power efficient because the envelope of the signal changes.
4. **Constellation Points:** In QPSK, the constellation points are farther apart compared to QAM for the same energy. Therefore, QPSK is less prone to errors due to perturbations in the channel.

However, it's important to note that QAM can provide higher data rates for a given bandwidth and signal power, which can be an advantage in scenarios where high data rates are required and the channel conditions are good. The choice between QPSK and QAM will depend on the specific requirements of the communication system, such as data rate, power efficiency, complexity, and robustness to noise.