In the name of God

Sharif University of Technology

"Communication Systems" course

Mohammad Amin Ansari 400100 757

Aida Karimzadeh 400101797

Final project report

1.1

- Reading our music signal and plot it in time domain and frequency domain:

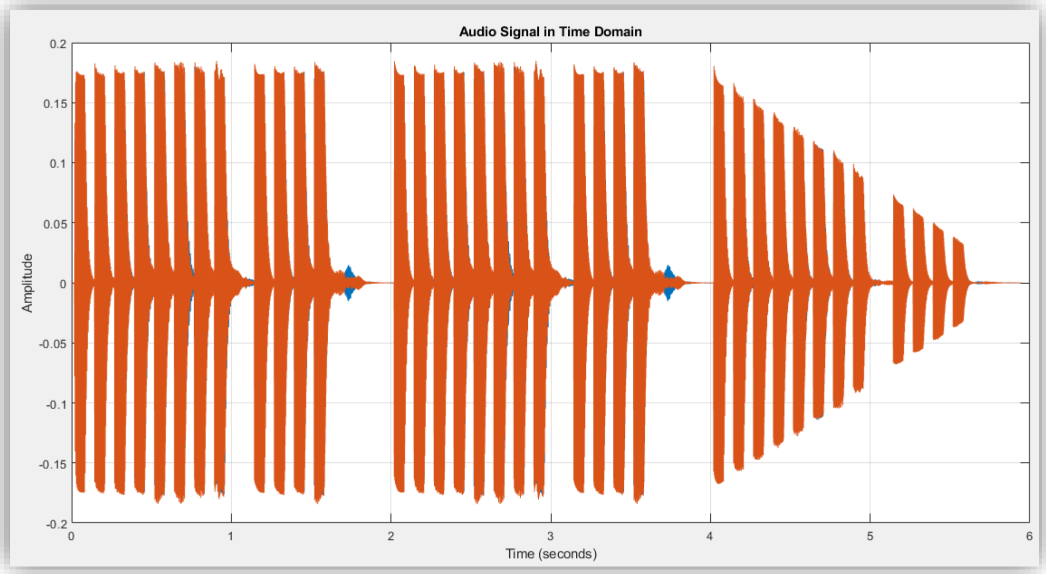

Figure 1 music.wav in time domain

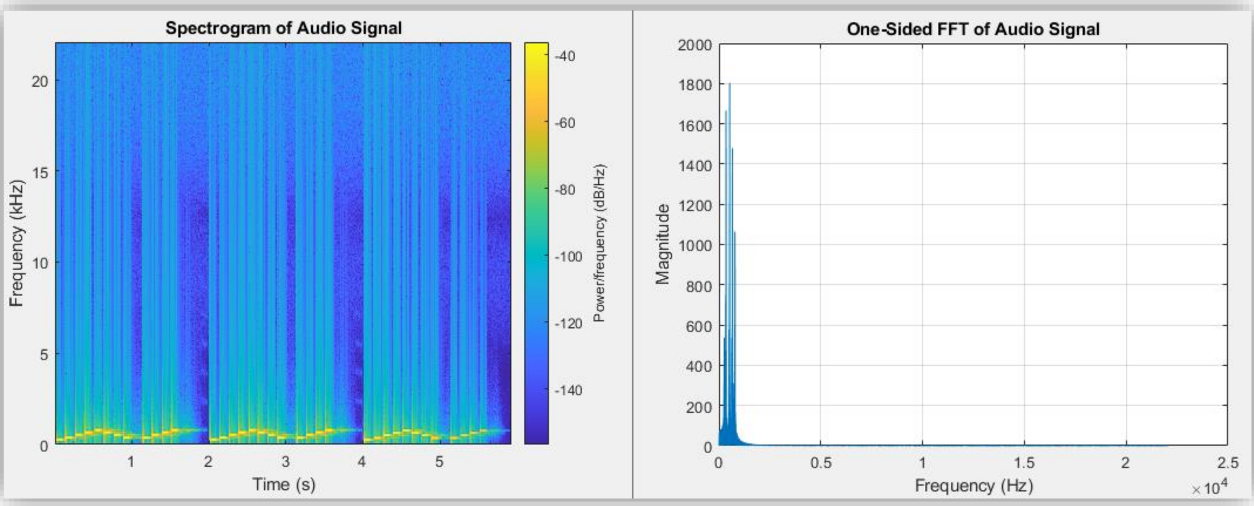

Figure 2 audio signal in freq domain

2.1 Filtering

1. Filtering audio signals in communication systems serves several important purposes:

   a) **Noise Reduction**:
      - Audio signals often contain unwanted noise due to interference, background sources, or transmission distortions.
      - Filters help suppress this noise, resulting in clearer and more intelligible audio.
      - Common noise sources include electrical interference, ambient sounds, and quantization noise.

   b) **Signal Enhancement**:
      - Filters can emphasize specific features of an audio signal.
      - Equalizers, for instance, boost or attenuate specific frequency bands to enhance bass, treble, or midrange tones.
      - Audio enhancement aims to improve the overall listening experience.

   - **Low-Pass Filters (LPF)**:
      - Allow low-frequency components to pass while attenuating higher frequencies.
      - Used for anti-aliasing, smoothing, and bass enhancement.

2. Designed filters:

```
designfilt('lowpassiir', 'FilterOrder', 5, 'HalfPowerFrequency', cutoff_freq_1k, 'SampleRate', Fs);
designfilt('lowpassiir', 'FilterOrder', 5, 'HalfPowerFrequency', cutoff_freq_2k, 'SampleRate', Fs);
```

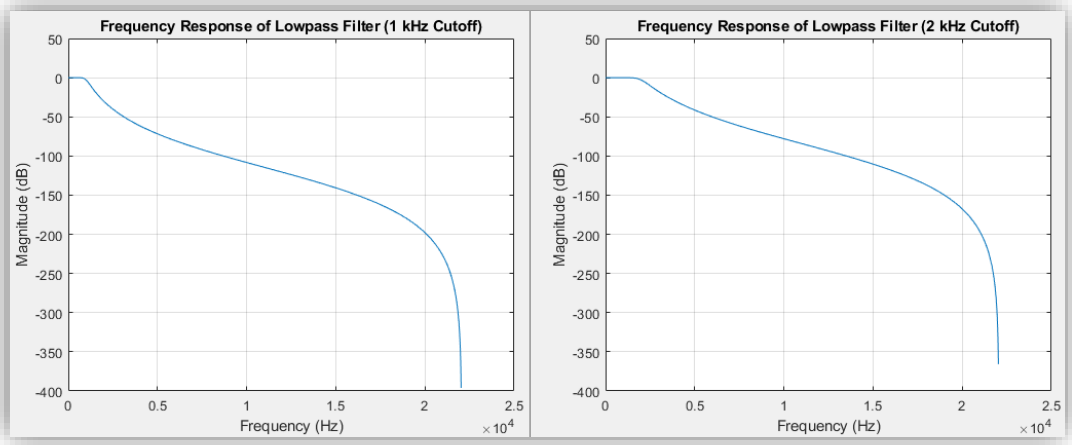

Figure 3 designed filters with 1k and 2k cut off frequencies

3. Audio signal after getting filtered using these two filters:

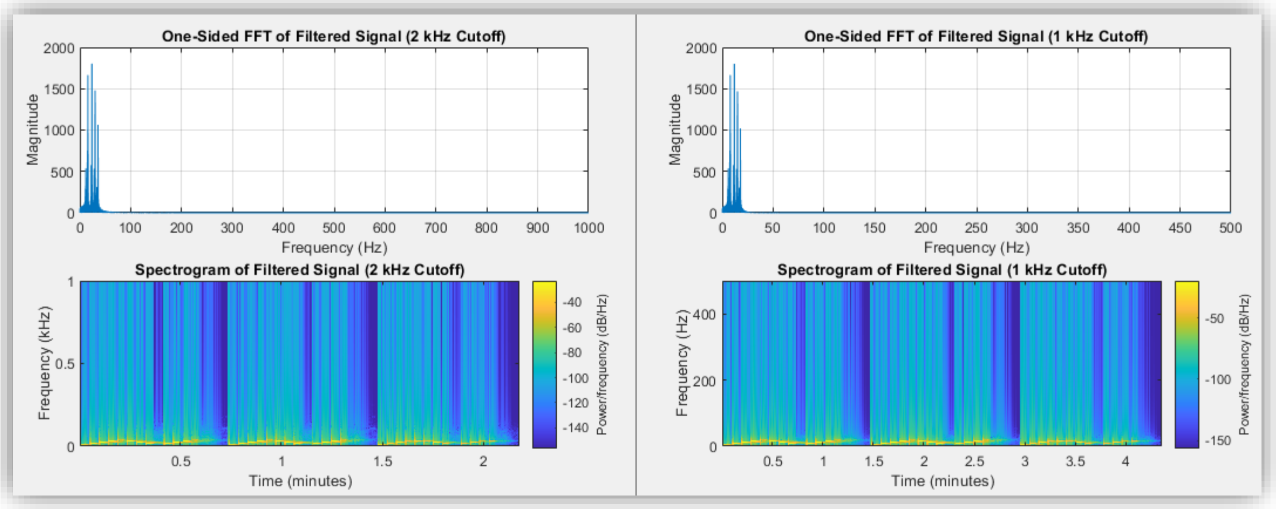

Figure 4 filtered audio signal

2.2 Sampling

Sampling signal in 2kHz and 4kHz and plotting them in time domain:

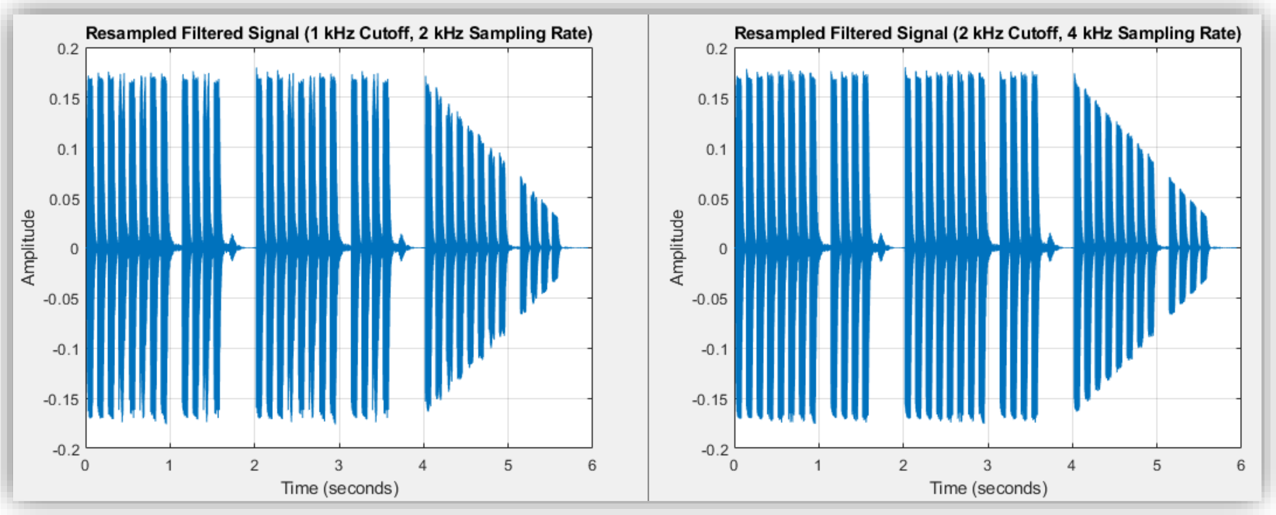

Figure 5 down sampled signals

2.3 Quantization

a) Quantization to 4-bits (16 levels):

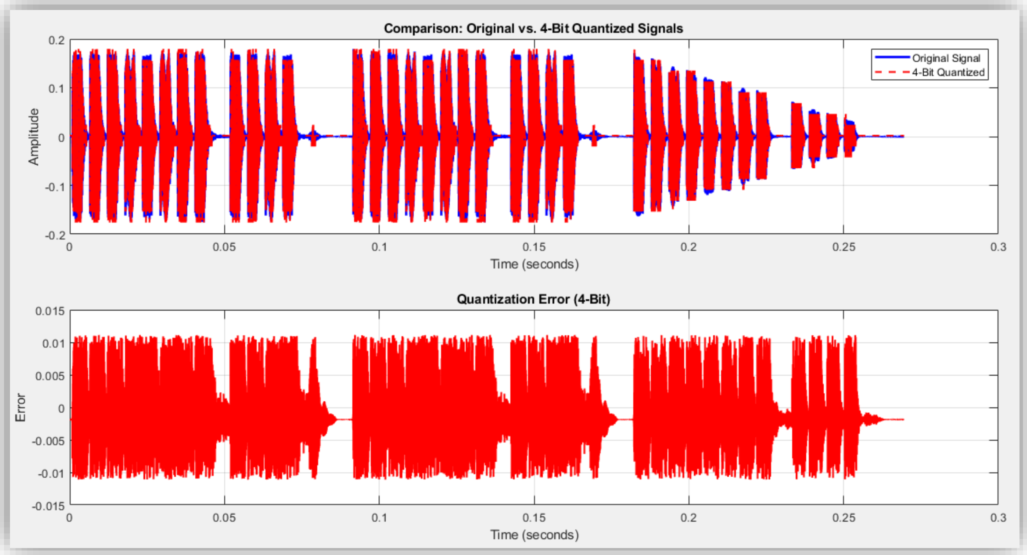

Figure 6 4-bit quantization

1. Quantization to 8-bits (32 levels):

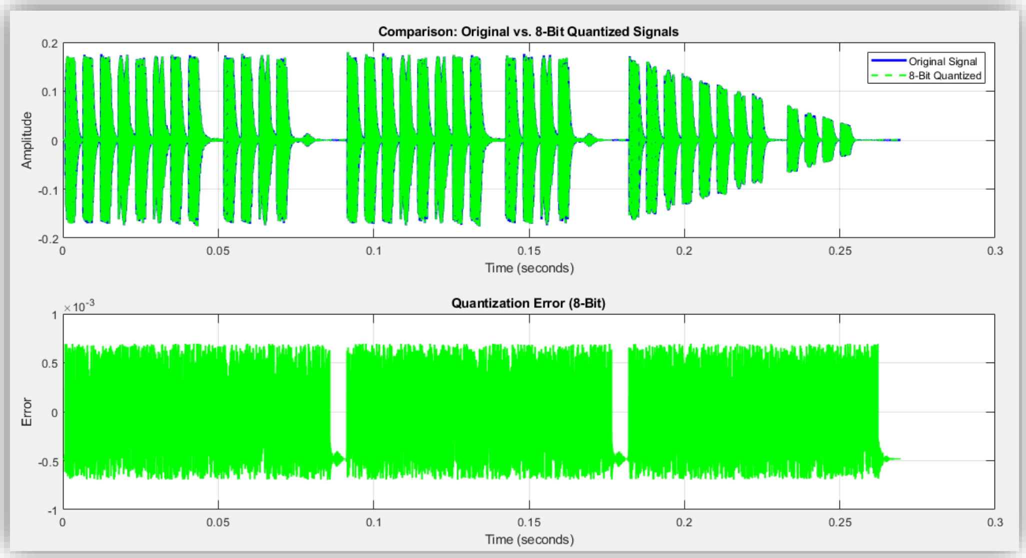

Figure 7 8-bit quantization

We can see a much softer and smaller error in 8-bit quantization.

2.4 Encoding

1. Gray Code, also known as **unit distance code** or **reflected binary code**, is a binary numeral system in which **two successive values differ in only one bit**. This unique property makes Gray Code highly valuable in **digital communication systems** and other applications. From its application we can note:

1. **Error Reduction**:
   - Gray Code ensures that adjacent values differ by only one bit. When transitioning between values, this minimizes the chance of errors or glitches.
   - In rotary encoders (used for position sensing) and other digital systems, Gray Code prevents ambiguous transitions.
2. **Mechanical Digital Conversions**:
   - In mechanical systems (such as gear-based encoders), Gray Code simplifies the conversion from analog to digital signals.
   - The single-bit change property ensures accurate tracking of position or rotation.
3. **Error Detection and Correction**:
   - Gray Code aids in error detection and repair.
   - When data is transmitted or stored, small errors can occur. Gray Code's unique structure helps identify and correct these errors.

2. Serial Communication:

- **Definition**: Serial communication involves sending data one bit at a time, sequentially, over a communication channel or computer bus. It contrasts with parallel communication, where several bits are sent simultaneously over multiple channels.
- **Key Points**:
  - **Data Transmission**: In serial communication, data is transmitted bit by bit.
  - **Medium**: It typically uses a single wire or a pair of wires.
  - **Examples**: Common interfaces include RS-232, RS-485, I2C, SPI, and more.
  - **Widely Used**: Serial communication is prevalent in devices like PCs, mobile phones, and embedded systems.

I2C (Inter-Integrated Circuit) Protocol:

- **Overview**:
    - Combines features of SPI and UART.
    - Allows multiple slaves to connect to a single master (or multiple masters controlling slaves).
    - Commonly used in projects with OLED displays, sensors, and modules.
- **Communication Lines**:
    - **SDA (Serial Data)**: Transmits data between master and slave.
    - **SCL (Serial Clock)**: Carries clock signal, synchronized between master and slave.
- **Message Structure**:
    - Each message has:
        - **Address Frame**: Unique slave address (7 or 10 bits).
        - **Data Frames**: Contain the transmitted data.
        - **Start and Stop Conditions**: Define message boundaries.
        - **Read/Write Bits**: Specify data direction.
        - **ACK/NACK Bits**: Acknowledge or no-acknowledge response from the receiving device.
- **Addressing**:
    - No slave select lines like SPI.
    - Master sends the slave's address to all connected slaves.
    - If the address matches, the slave responds with an ACK bit.

3&4. Generating serial RZ or NRZ binary sequences

NRZ pulse train generated for 4bit quantized 2k sampled signal:

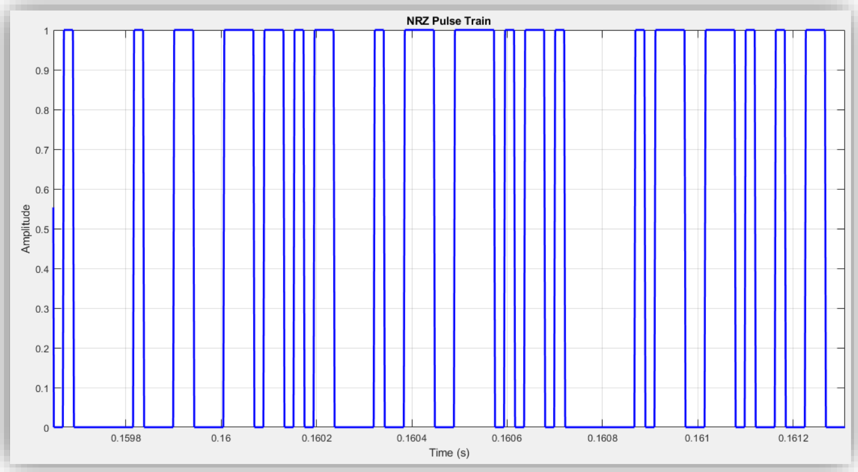

Figure 8 NRZ(2kHz , 4-Bit)

RZ pulse train generated for 4bit quantized 2k sampled signal:

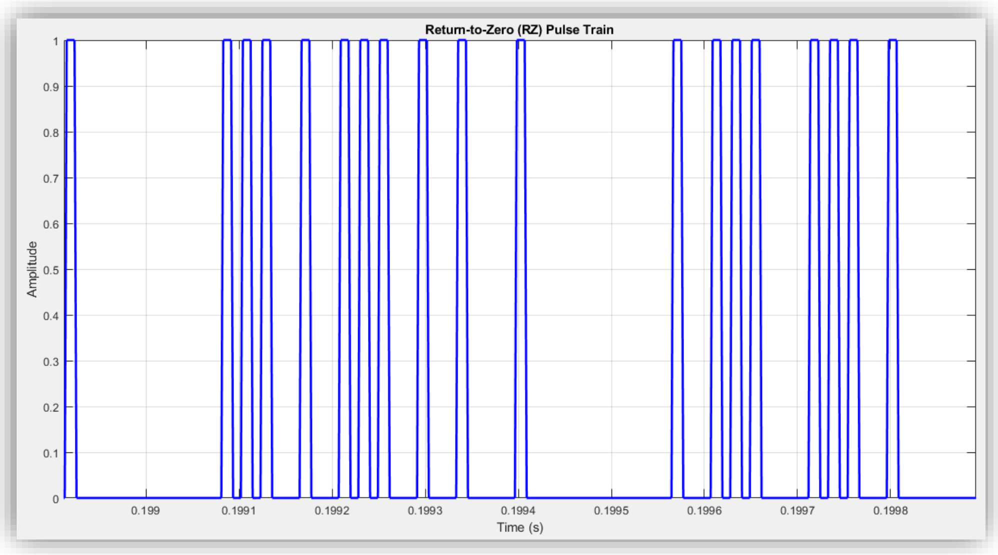

Figure 9 RZ(2kHz, 4-Bit)

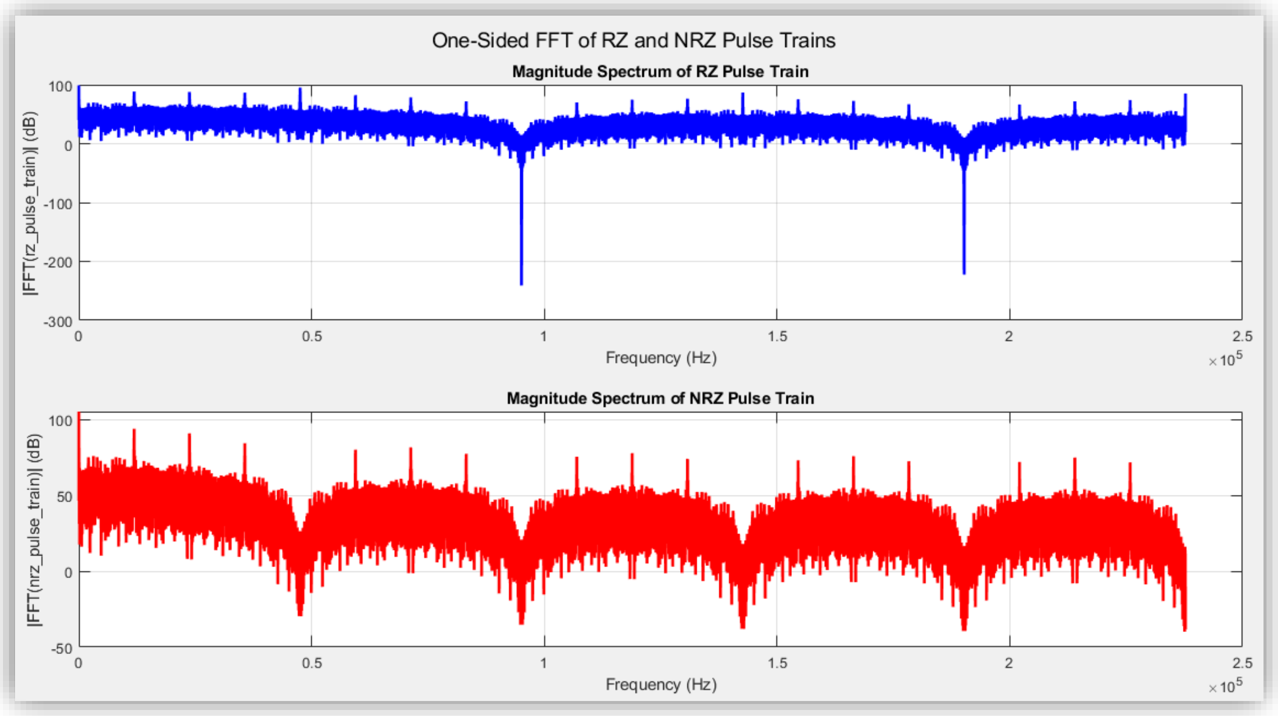

Figure 10 FFT of both RZ and NRZ signals

3 Communication Channel simulation

1.

- Power of RZ pulse train: 90680.1907 Watts
- Power of NRZ pulse train: 181360.3814 Watts

2. Generating noise and plotting it in time domain:

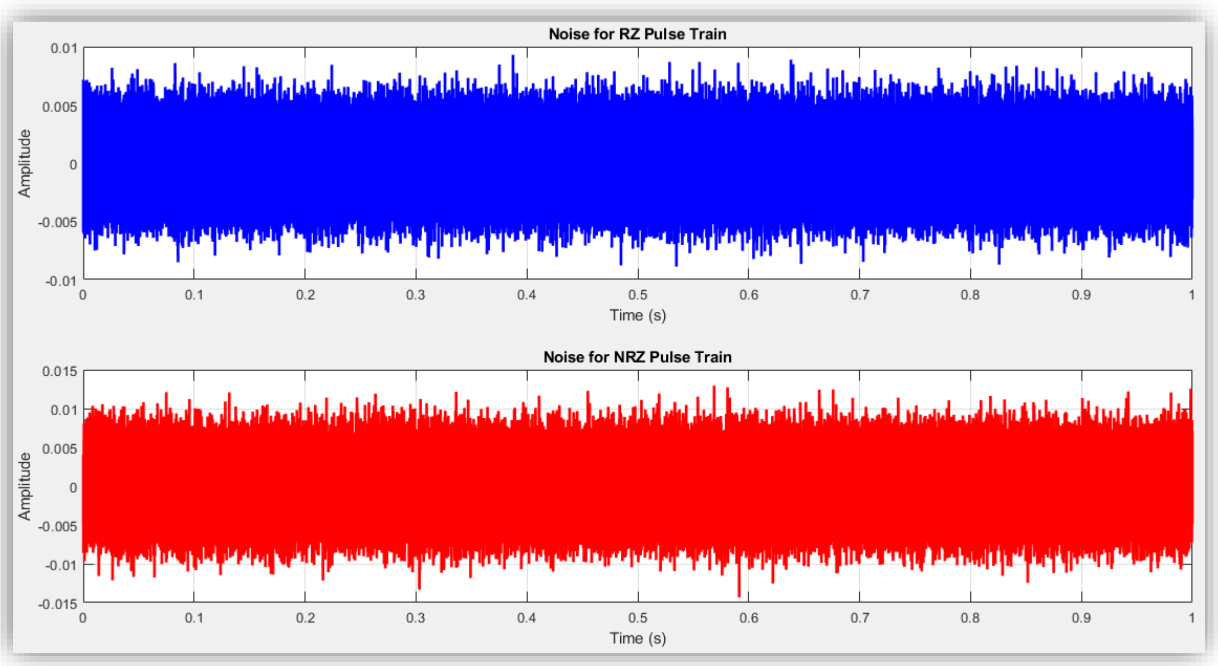

Figure 11 generated noises for RZ and NRZ channels

3. In SNR = 10 dB our received and decoded signal was quite understandable but not as clear as original one. Increasing the SNR (reducing noise power and amplitude) we achieved much clearer audio in receiver.

4. Received signal after going through channel (noise added):

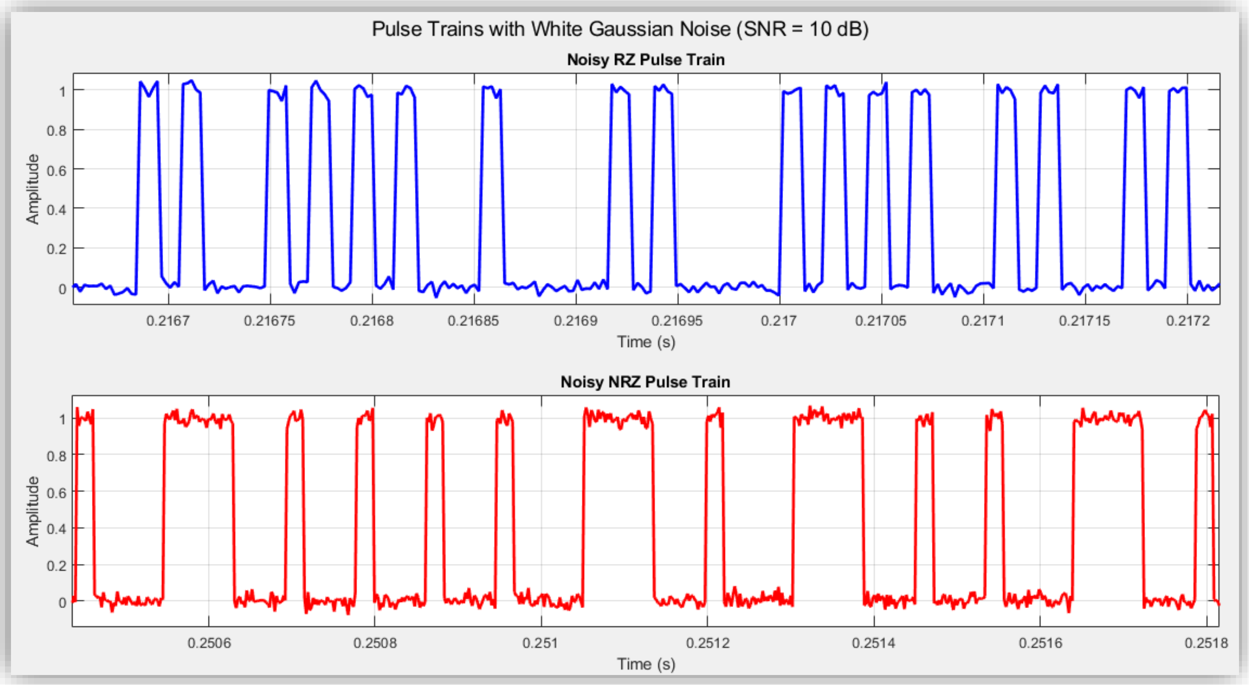

Figure 12 noisy signals

4 Receiver

```
% Parameters
threshold = 0.5;
sample_interval = 10;

% Sample the noisy pulse trains
sampled_rz = rz_pulse_train_noisy(3:sample_interval:end);
sampled_nrz = nrz_pulse_train_noisy(3:sample_interval:end);

% Apply threshold and generate binary sequences
rz_detected_sequence = (sampled_rz > threshold)';
nrz_detected_sequence = (sampled_nrz > threshold)';
```

In receiver we sampled our signal with contracted bitrate and then packed each 4-bits (or 8-bits in case of 8-bit quantization), then converted them from gray code format to binary. Then mapped each binary data to a voltage level assigned at the first stage in quantization part.

```
%% detecting digital signal

nrz_detected_signal_gray = reshape(nrz_detected_sequence, 4, [])';
rz_detected_signal_gray = reshape(rz_detected_sequence, 4, [])';

%% convert gray format into binary format

% Convert Gray code to binary
binary_rz_detected_signal = zeros(size(rz_detected_signal_gray));
for i = 1:size(rz_detected_signal_gray, 1)
    binary_rz_detected_signal(i, :) = bitxor(rz_detected_signal_gray(i, :), [0
rz_detected_signal_gray(i, 1:3)]);
end

% Convert Gray code to binary
binary_nrz_detected_signal = zeros(size(nrz_detected_signal_gray));
for i = 1:size(nrz_detected_signal_gray, 1)
    binary_nrz_detected_signal(i, :) = bitxor(nrz_detected_signal_gray(i, :), [0
nrz_detected_signal_gray(i, 1:3)]);
end

%% Decoding binary codes to voltage levels

% Normalize binary values to the range [0, 15]
normalized_values = bi2de(binary_rz_detected_signal, 'left-msb');

% Map normalized values to the desired voltage range
voltage_range_min = -0.5;
voltage_range_max = 0.5;
voltage_levels = voltage_range_min + (normalized_values / 16) * (voltage_range_max -
voltage_range_min);
```

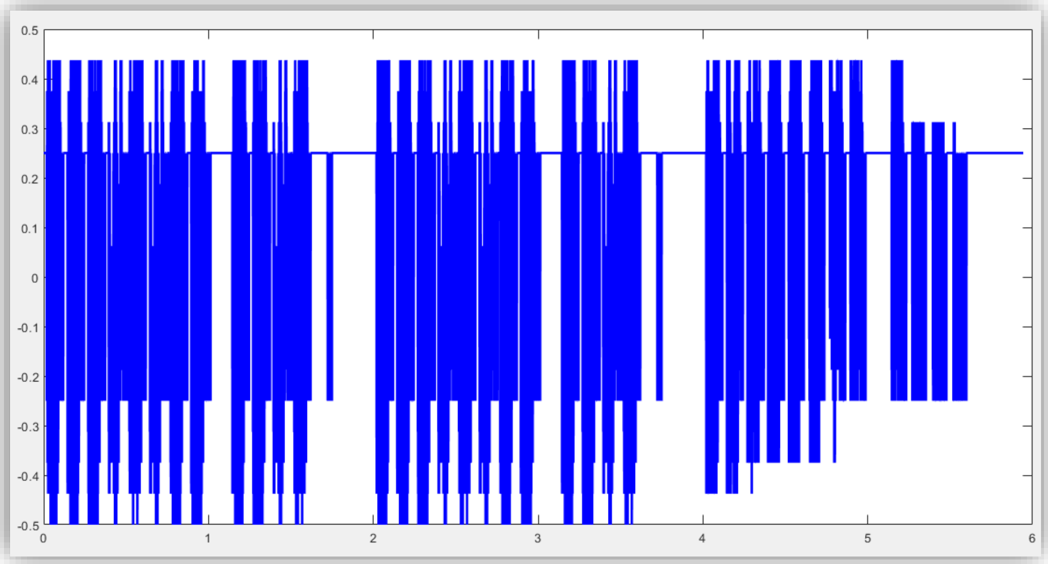

Figure 13 recovered signal from channel

BLOCK DIAGRAM

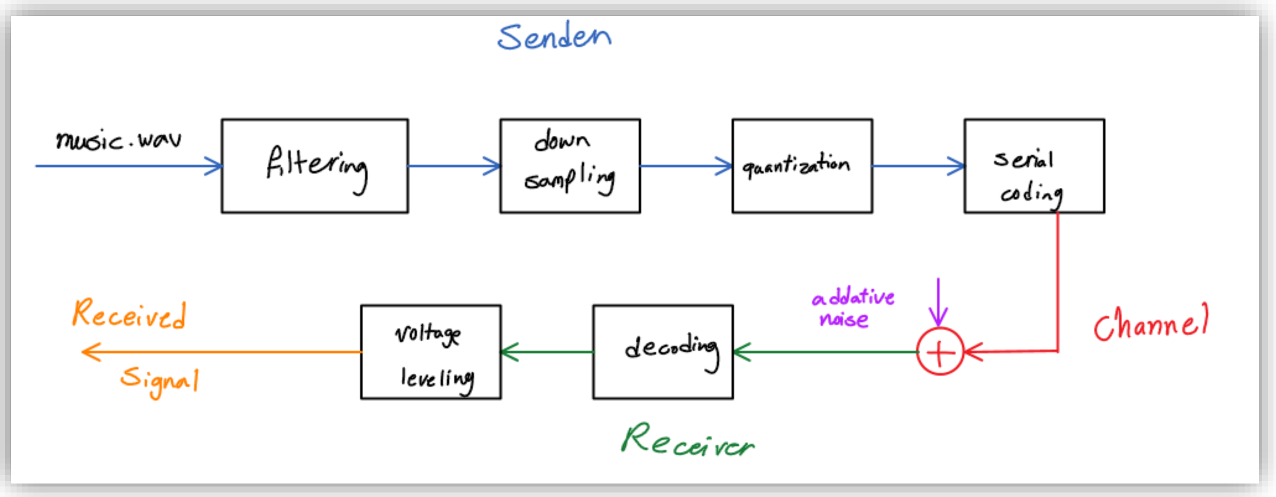

Figure 14 final block diagram of design

6 Extra bonus

Encrypting:

1. **Symmetric Encryption**:
    - You could use symmetric encryption to protect the 4-bit data during transmission.
    - **Application**:
        - Encrypt each 4-bit data point using a shared secret key.
        - The receiver decrypts using the same key.
    - **Considerations**:
        - Secure key distribution is essential.
        - Choose a strong symmetric encryption algorithm (e.g., AES).
2. **Asymmetric Encryption (Public-Key Encryption)**:
    - Use asymmetric encryption for secure key exchange.
    - **Application**:
        - Generate a key pair (public key and private key).

        - Share the public key openly.
        - Encrypt the 4-bit data using the recipient's public key.
        - The recipient decrypts using their private key.
    - **Considerations**:
        - Slower than symmetric encryption.
        - Ensures secure key distribution.
3. **End-to-End Encryption (E2EE)**:
    - Ensure that only the sender and receiver can read the 4-bit data.
    - **Application**:
        - Implement E2EE for your music sample transmission.
        - Encrypt the data at the sender's end.
        - Decrypt at the receiver's end.
    - **Considerations**:
        - Choose strong encryption algorithms.
        - Protect the private keys.
4. **Authentication and Integrity**:
    - Consider using digital signatures or message authentication codes (MACs) to verify data integrity.
    - Ensure that the received 4-bit data hasn't been tampered with during transmission.
5. **Secure Channels**:
    - Use secure communication channels (e.g., HTTPS, VPNs) to transmit the encrypted data.
    - Protect against eavesdropping and man-in-the-middle attacks.

Multiple serial sequences:

1. **Universal Asynchronous Receiver/Transmitter (UART)**:
    - UART is a common method for serial communication. It transmits data one bit at a time and is widely used for connecting microcontrollers, sensors, and other devices.
    - It's straightforward to implement and doesn't require a clock signal.
    - UART typically uses two lines: one for transmitting (TX) and one for receiving (RX).
    - Baud rate settings determine the data rate.
2. **RS-232 and RS-485**:
    - RS-232 (serial port) and RS-485 (multi-drop communication) are older standards but still used in some applications.
    - RS-232 is point-to-point, while RS-485 supports multi-point communication.
    - RS-485 allows multiple devices to share the same communication line, making it suitable for sending multiple sequences.
3. **I2C (Inter-Integrated Circuit)**:

- I2C is a synchronous serial communication protocol that allows multiple devices to communicate over a shared bus.
- It uses two lines: SDA (data) and SCL (clock).
- I2C is commonly used for connecting sensors, EEPROMs, and other peripherals.

4. **SPI (Serial Peripheral Interface)**:
    - SPI is another synchronous serial protocol.
    - It uses separate lines for data (MOSI/MISO), a clock (SCK), and a chip select (CS) line for each device.
    - SPI is often used for high-speed communication between microcontrollers and peripherals.
5. **CAN (Controller Area Network)**:
    - CAN is used in automotive and industrial applications.
    - It supports multi-node communication with high reliability.
    - CAN controllers handle message arbitration and collision detection.