Alexis Emmanuelle Pascual A16193469
Uriberto Lopez A16251534

Mini-Project 3: Demodulating Real Life Signals

In this module, you will demodulate a simple real-life signal using the concepts taught in the course. We have provided an I/Q sample file for the project. Alternatively, you can choose a signal to try and decode (see notes). You have a good SDR with you, so have fun trying to see and decode signals!

At the end of this module, you will:

1. Analyze properties of the signals and identify them
2. Demodulate the signal using MATLAB or GNURadio

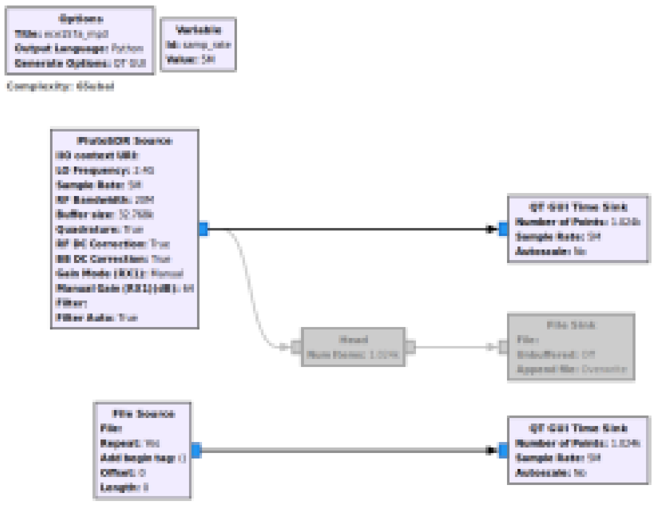
Overall Description:

1. Three Tasks (Signal Analysis, Demodulation Method, Demodulation Implementation)
2. Some tasks require you to take a screenshot and/or write a very short answer to a question.
3. Note on open-ended questions: There may be many correct answers to the same question, it is up to you to make assumptions and choose one.
4. Submission format: upload the following as a .zip to Gradescope
    a. PDF containing screenshots and short answers
    b. GNURadio .grc file that you create, any MATLAB scripts
5. Tips on working as a team:
    a. You can choose to divide the tasks among yourselves. Note that they have to be done in the order they are described.
    b. Include a note on each person's contribution on the pdf.
6. You may need the SDR for this mini-project.

Walkthrough:

Jump Off Point

1. You will start with the following reference flowgraph:



Options
Title: ece157a_mp3
Output Language: Python
Generate Options: QT GUI
Complexity: 6Global

Variable
Id: samp_rate
Value: 5M

PlutoSDR Source
IIO context URI:
LO Frequency: 2.4G
Sample Rate: 5M
RF Bandwidth: 20M
Buffer size: 32.768k
Quadrature: True
RF DC Correction: True
BB DC Correction: True
Gain Mode (RX1): Manual
Manual Gain (RX1)(dB): 64
Filter:
Filter Auto: True

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 5M
Autoscale: No

Head
Num Items: 1.024k

File Sink
File:
Unbuffered: Off
Append file: Overwrite

File Source
File:
Repeat: Yes
Add begin tag: ()
Offset: 0
Length: 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 5M
Autoscale: No

   a.
   b. Make sure you have cloned the latest version of the course git repository.
   c. You can open this flowgraph by:

cd ece157A/grc/miniproj/
gnuradio-companion ece157a_mp3_0_reference.grc

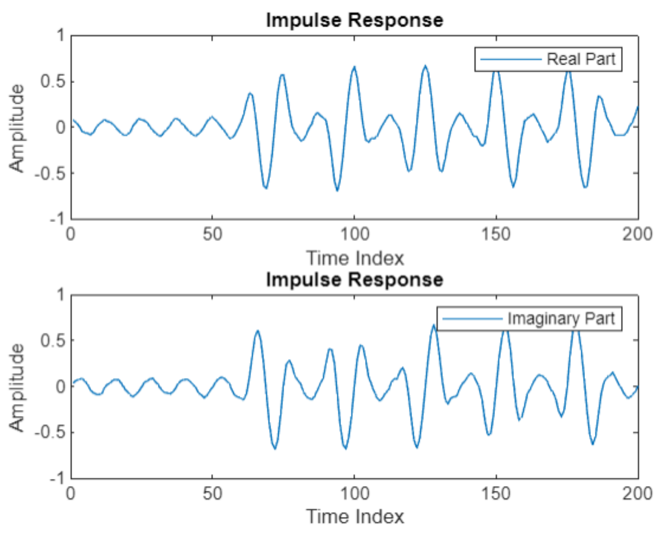
   d. Use the “Save As” option to save your modified flowgraph in a different location, name it **ece157a_w21_mp3_<teamidx>.grc**
   e. Some of the modules appear gray because they are disabled. To enable a block, you need to highlight it and press ‘E’. To disable a block, press ‘D’.
2. The flowgraph has two parts:
   a. **SDR -> Time/File Sink:** This allows you to tune the SDR to a particular frequency and visualize the samples on the **Time Sink**, or save them into a **File Sink** for analysis in MATLAB.
   b. File Source -> Time Sink: This allows you to visualize the samples already stored in a file.
3. You may use this flowgraph to help demodulate the signals you are looking for.

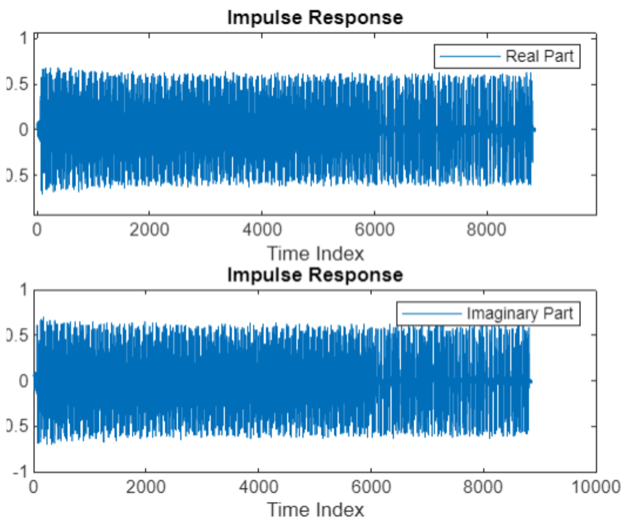
Signal Identification

1. We have provided an IQ sample file with this writeup:
    a. Use the **read_complex_binary.m** and **the write_complex_binary.m** functions attached with this write-up to read the samples to MATLAB or write them out into a file respectively.
    b. The info.txt file provides some rudimentary information about the IQ sample capture.

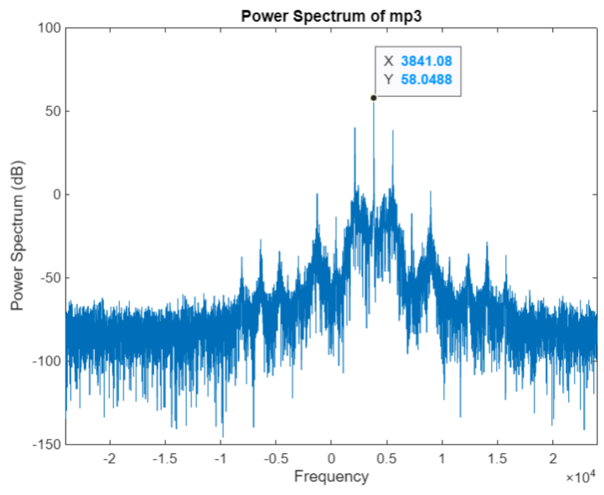
2. You may alternatively choose other signals. Here are some examples:
    a. Car Key fob, Garage Key fob (OOK or FSK)
    b. Bluetooth Low Energy (GFSK)
    c. Automatic Identification System (GMSK)
    d. RC Toy Devices (Usually FSK or OOK)
    e. More resources: https://www.sigidwiki.com/wiki/Signal_Identification_Guide f. To capture real-signals, you will have to tune the SDR to the correct center frequency and then run the example flowgraph. This process is similar to Lab 1. **g. If you are choosing this option, please send a note to the instructors, we will approve/suggest alternatives.**

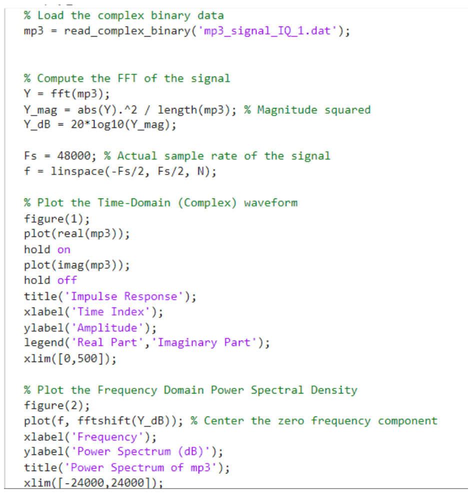
3. **Task 1: Signal Properties**
    a. Identify basic properties: Center Frequency, Bandwidth, Modulation Scheme
    i. Plot/Screenshot of the Time-Domain (Complex) waveform



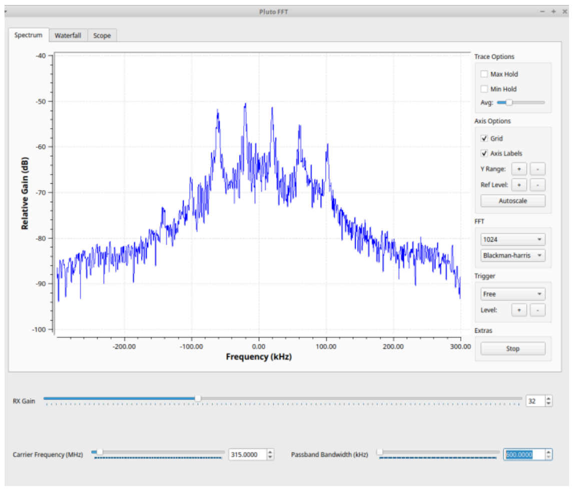


ii. Plot/Screenshot of the Frequency Domain Power Spectral Density




```
% Load the complex binary data
mp3 = read_complex_binary('mp3_signal_IQ_1.dat');


% Compute the FFT of the signal
Y = fft(mp3);
Y_mag = abs(Y).^2 / length(mp3); % Magnitude squared
Y_dB = 20*log10(Y_mag);

Fs = 48000; % Actual sample rate of the signal
f = linspace(-Fs/2, Fs/2, N);

% Plot the Time-Domain (Complex) waveform
figure(1);
plot(real(mp3));
hold on
plot(imag(mp3));
hold off
title('Impulse Response');
xlabel('Time Index');
ylabel('Amplitude');
legend('Real Part','Imaginary Part');
xlim([0,500]);

% Plot the Frequency Domain Power Spectral Density
figure(2);
plot(f, fftshift(Y_dB)); % Center the zero frequency component
xlabel('Frequency');
ylabel('Power Spectrum (dB)');
title('Power Spectrum of mp3');
xlim([-24000,24000]);
```

From Lab 1:



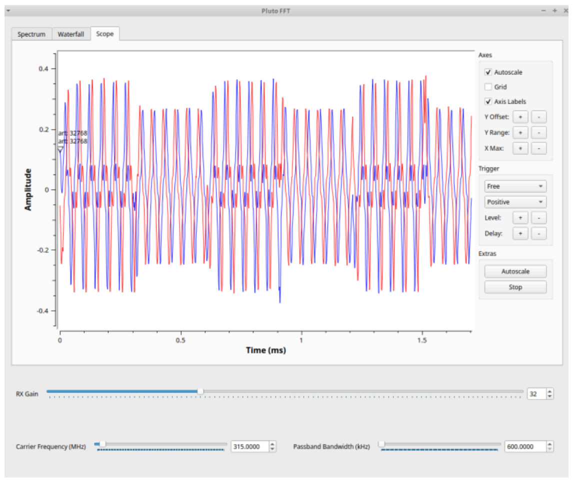
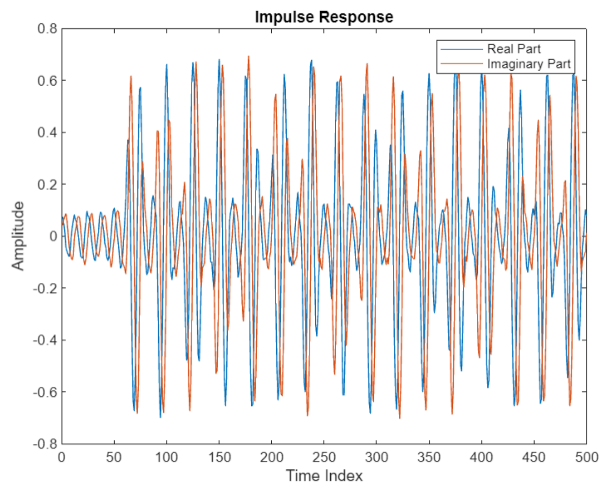
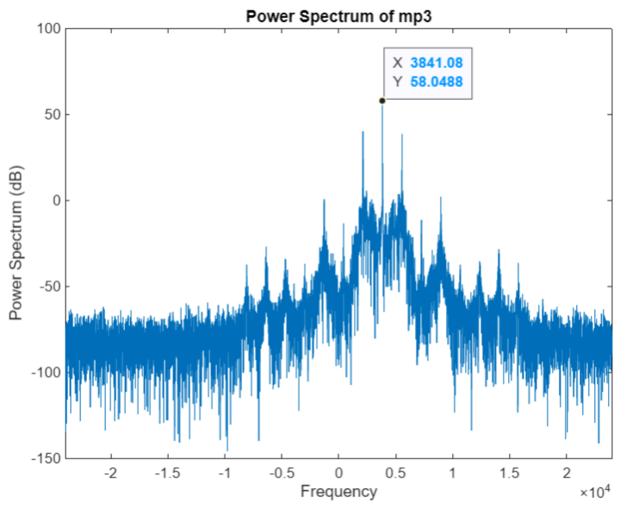


Figure 5: Top: Spectrum of remote wireless keyless entry. Bottom: Amplitude-shift-keyed time waveform using the “scope” tab on the front panel.

These figures are similar to our simulations:





Center Frequency: 3841 Hz
Bandwidth: limited to 17000 Hz

The presence of multiple peaks leads us to believe that this is an Amplitude Shift Keying (ASK).

An ASK is a type of amplitude modulation which represents the binary data in the form of variations in the amplitude of a signal.

b. Write a short note on why the properties of the signal may be a good fit for an application it is used for.

RFID Systems: ASK is effective in RFID systems because of its simplicity and cost-effectiveness. RFID tags have limited power and computational resources, making the straightforward implementation of ASK beneficial.

Keyless Entry Systems: ASK is used in remote keyless entry systems for automobiles, where a simple and reliable modulation scheme is necessary to communicate between the key fob and the vehicle. ASK is excellent for this application because it provides a simple and reliable way to transmit data over short distances with low power consumption, which is ideal for battery-operated key fobs.

Remote Controls: Infrared (IR) remote controls for televisions, air conditioners, and other appliances use ASK for data transmission. The IR signal's intensity is modulated to represent the binary data. ASK modulation in IR remote controls is straightforward and cost-effective.

Medical Implants: Some medical implant communication systems use ASK for transmitting data between the implant and external devices. ASK's simplicity and low power requirements make it suitable for medical implants, where battery life is critical. Additionally, ASK can operate efficiently over the short distances typically involved in these applications.

Demodulation Method

1. **Task 2:**
    a. Based on your analysis of the signal from the previous task, propose a suitable demodulation technique.

To demodulate an ASK signal, you typically follow these steps:

b. Clearly Identify:

i. How to resample the signal to the correct rate
- GNUradio and Matlab both have functions that will resample
    - Matlab has resample()

ii. How you will detect the start of the packet
- Cross-correlation
    - Pre-lab 3 correlate()

iii. Whether frequency offset needs to be removed, and if you need a PLL
- We needed to downconvert the signal to baseband
- We were able to accurately estimate the carrier frequency, so we can downconvert the signal to baseband without the need for a Phase-Locked Loop (PLL) for carrier recovery.

iv. How the signal symbols are identified in the signal space and how you will demodulate them.
- Downconverted to baseband (remove frequency offset)
- Chebyshev II (Lowpass Filter)
- Resample from 48k to 17k
- Decision-making
    - Assign each sample to the closest symbol based on threshold comparison
        - Compare the amplitude of each sample with predefined thresholds (bit is either 1 or -1)
- Envelope Detection
    - Cross-correlation w/preamble
- Manchester Decoding (Optional)

c. Finally, provide a block diagram showing the receiver structure. You may use any resource on the Internet and material from Papen/Blahut.

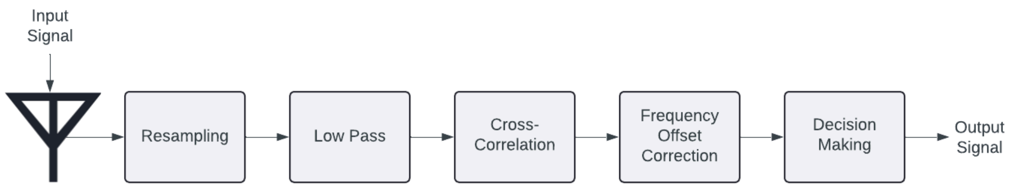


Figure: Initial Block Diagram

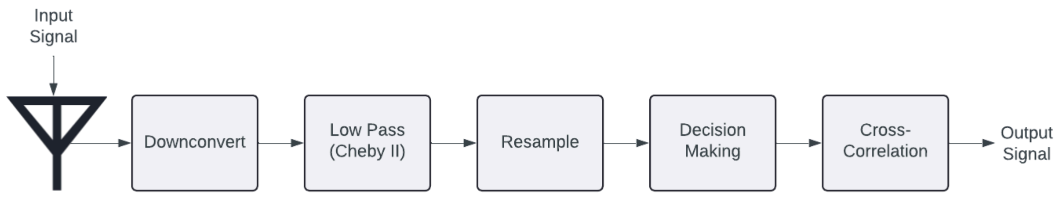


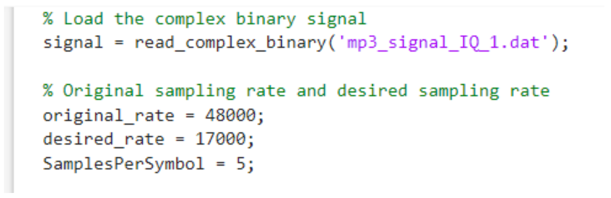
Figure: Updated Block Diagram

Demodulation Implementation

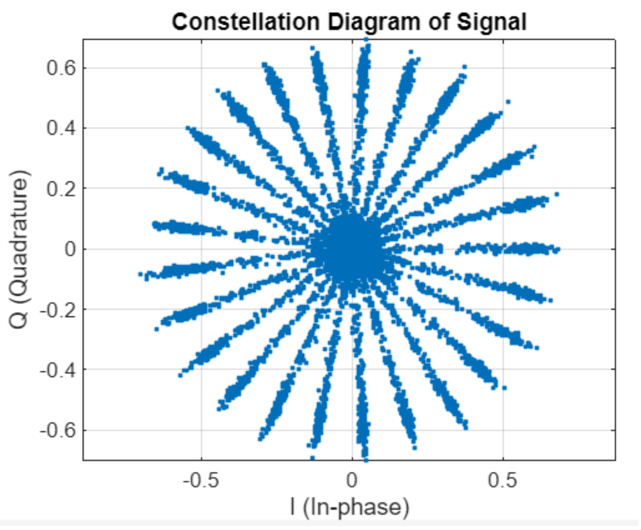
1. **Task 3:**
    a. Implement the technique you have identified in Task 2
    b. You may choose to do some parts in MATLAB and others in GNURadio
    c. You have to provide:
        i. A screenshot or write-up on how you identified the packet start
        ii. A before/after screenshot of frequency/timing offset removal
        iii. A scatter plot of the symbols in signal space (for example, BPSK would be +/-1)
        iv. The decoded bits
    d. Since it may be sometimes difficult to do all of the tasks mentioned in c., you will be graded on Task 3 for effort and understanding/application of what you have learnt.

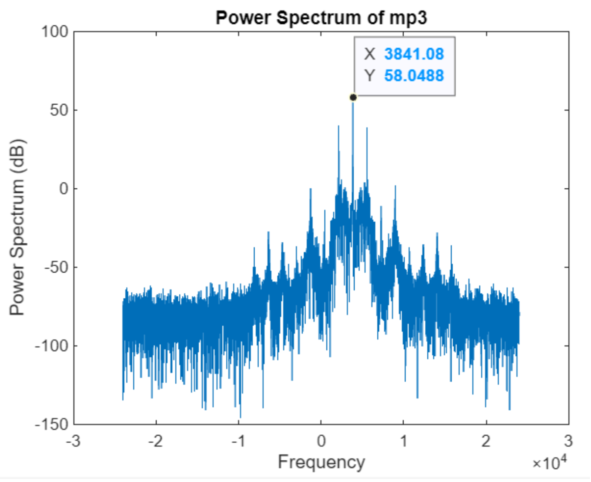
Demodulation Steps:

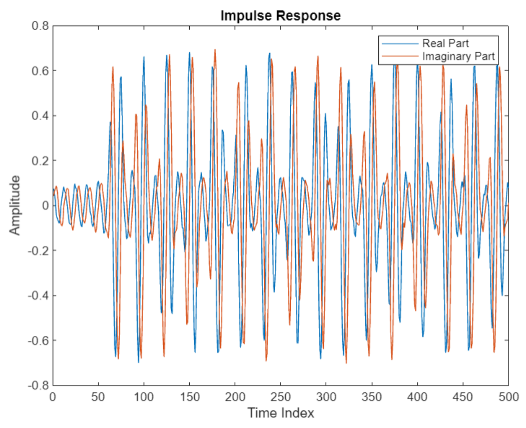
Original Signal



```
% Load the complex binary signal
signal = read_complex_binary('mp3_signal_IQ_1.dat');

% Original sampling rate and desired sampling rate
original_rate = 48000;
desired_rate = 17000;
SamplesPerSymbol = 5;
```

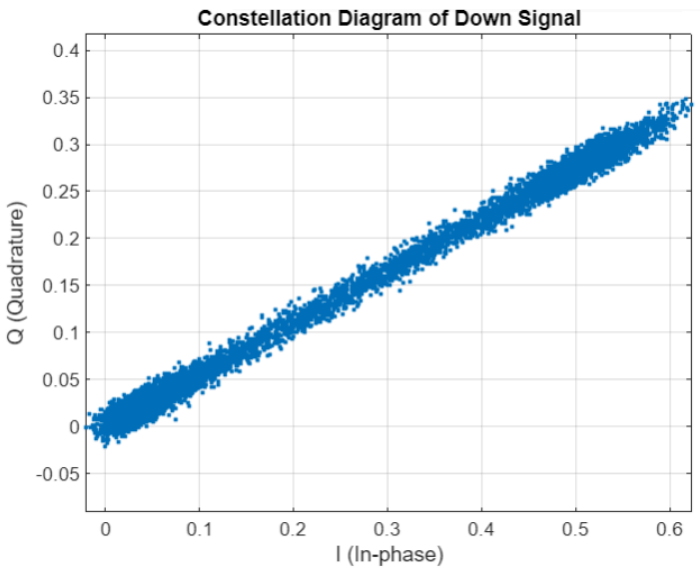


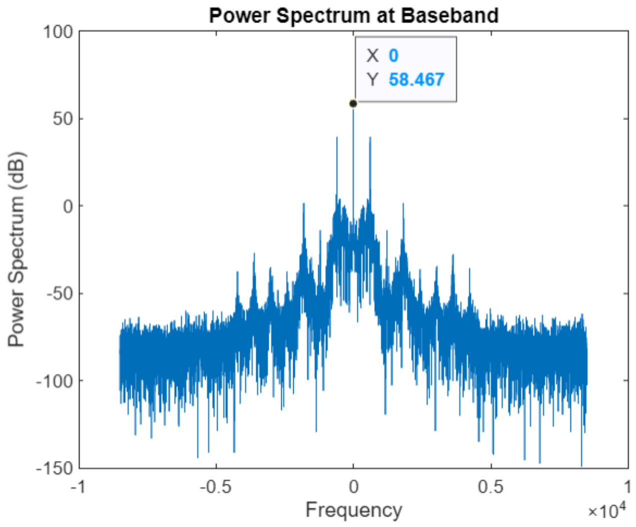


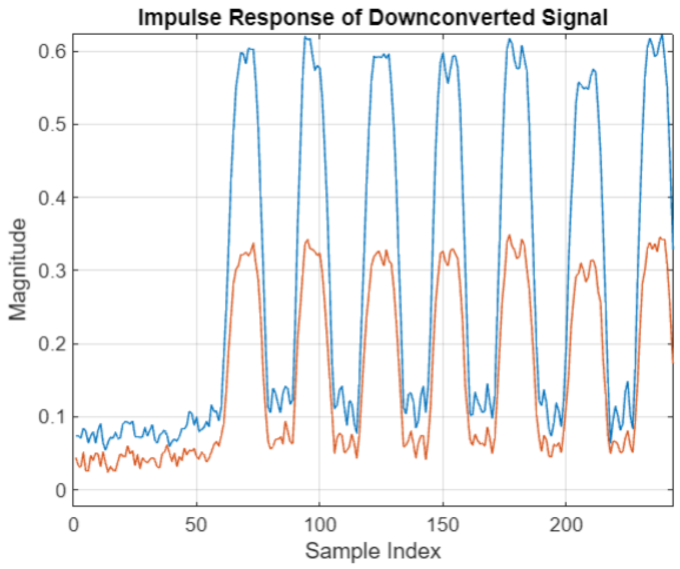


1) Downconverted to baseband

```
% Step 1: Downconvert the signal to baseband
fc = 3840; % Carrier frequency to downconvert to baseband
n = 0:length(signal)-1; % Create a time vector
downconverted_signal = signal .* exp(-1j*2*pi*fc/original_rate*n).';
```







A diagonal line constellation diagram suggests that the phase relationship between the I and Q components is changing as the symbols are transmitted.

We were able to accurately estimate the carrier frequency, so we can downconvert the signal to baseband without the need for a Phase-Locked Loop (PLL) for carrier recovery.

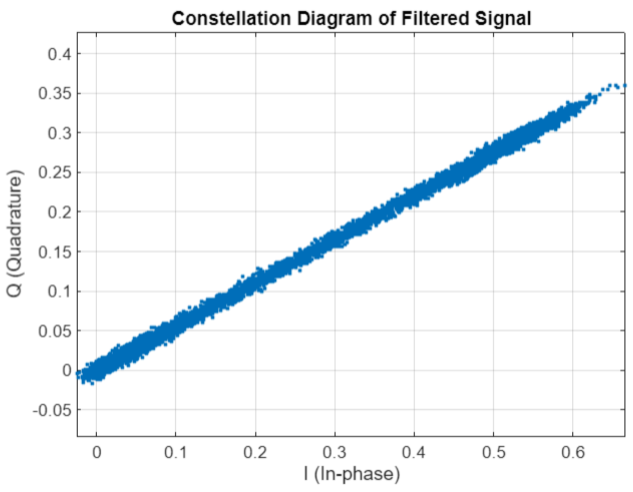
(We found this carrier frequency by starting at 5k and just adjusting it until we reached baseband)

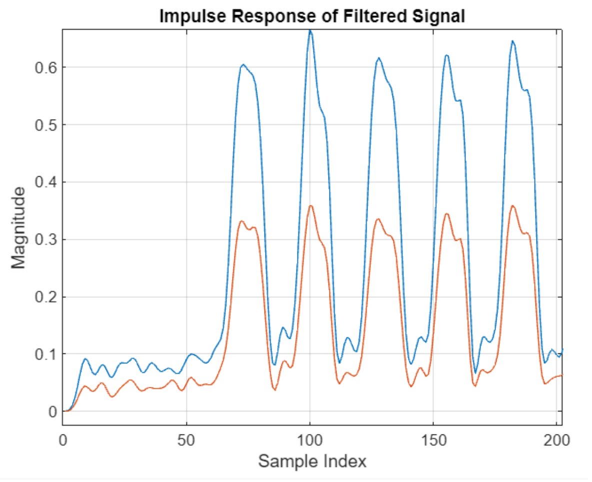
From there, we multiplied the signal by a complex exponential, which shifts these components to lower frequencies, bringing the signal down to baseband.

2) Filtered Signal

Filtered at half-sample rate using a Chebyshev II, IIR lowpass filter in order to reduce the noise in the signal, making it easier for further processing and detecting.

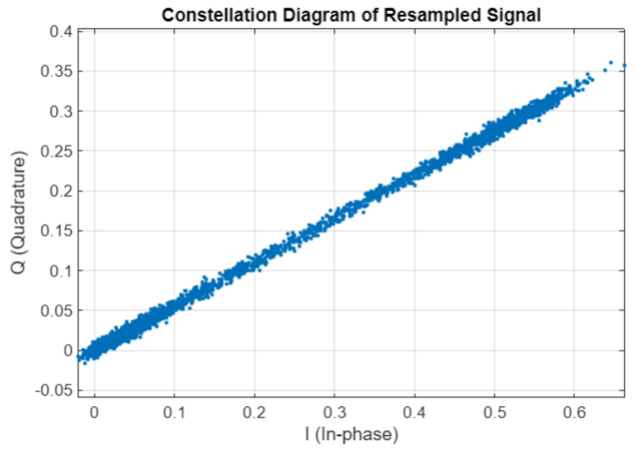
```
% Step 2: Apply Low-Pass Filter (Chebyshev Type II)
cutoff_freq = 8500; % Adjusted cutoff frequency for half-sample rate
stopband_ripple = 80; % Stopband attenuation in dB
normalized_cutoff = cutoff_freq / (original_rate / 2);

% Design the Chebyshev Type II filter
[filter_order, Wn] = cheb2ord(normalized_cutoff, normalized_cutoff * 1.1, 1, stopband_ripple);
[b, a] = cheby2(filter_order, stopband_ripple, Wn);

% Apply the Chebyshev Type II filter to the downconverted signal
filtered_signal = filter(b, a, downconverted_signal);
```

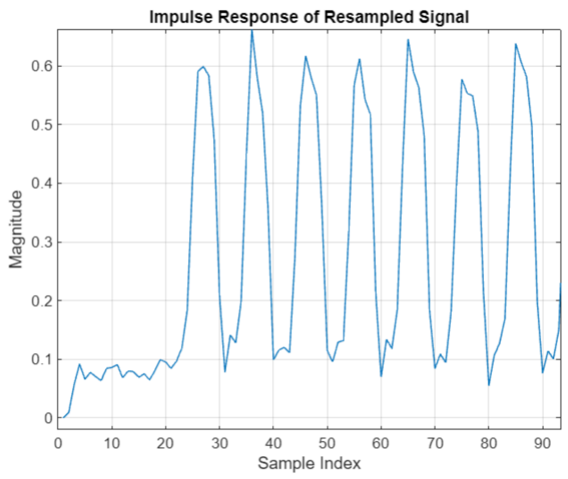




3) Resampled Signal

```
% Step 3: Resample the filtered signal
resampled_signal = resample(filtered_signal, desired_rate, original_rate);
```





We resampled the signal from 48k to 17k.

(From here and out, we only need to consider the real component)

4) Decision Making

Before we do cross-correlation, we need to convert the signal into bits so we can properly detect the packet.

Here we choose a threshold that will help map amplitudes to bits:

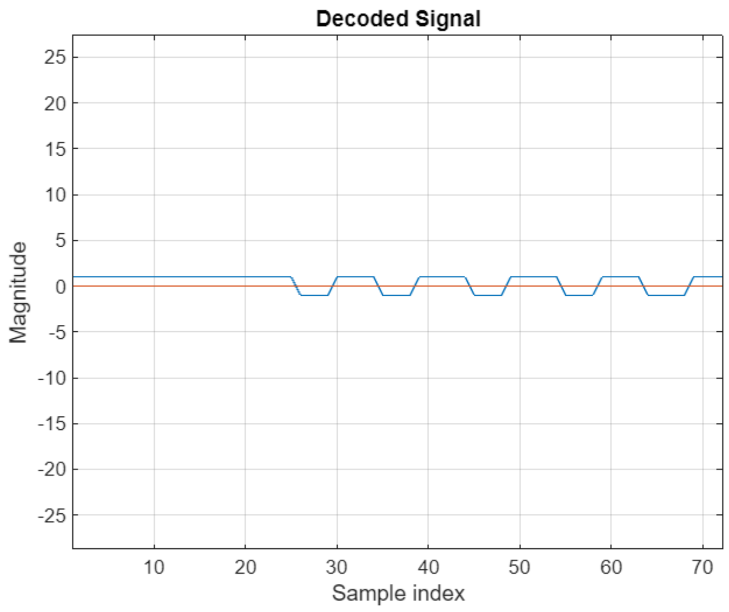
```
%Step 4: Decision making
threshold2 = 0.65 * max(real(resampled_signal));
decoded_signal = ask_demodulation(real(resampled_signal), threshold2);
```

```
function bits = ask_demodulation(signal_segment, threshold)
    bits = 2*(signal_segment < threshold) - 1;
end
```

To demodulate the signal, we decided to use the following convention:

1 → -1 and 0 → 1

Any convention would work but we found that this one yielded the most accurate packet detection.

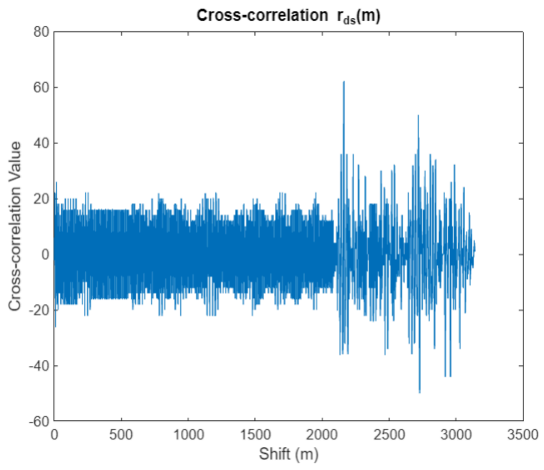


5) Cross-Correlation

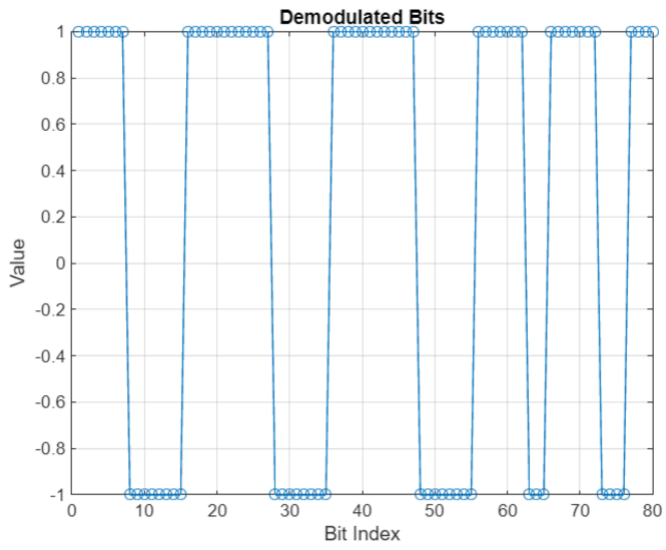
```
% Load the preamble
preamble_data = load('mp3_signal_IQ_1_encoded_preamble_signal_space.mat');
preamble = preamble_data.encoded_preamble_signal_space;
```

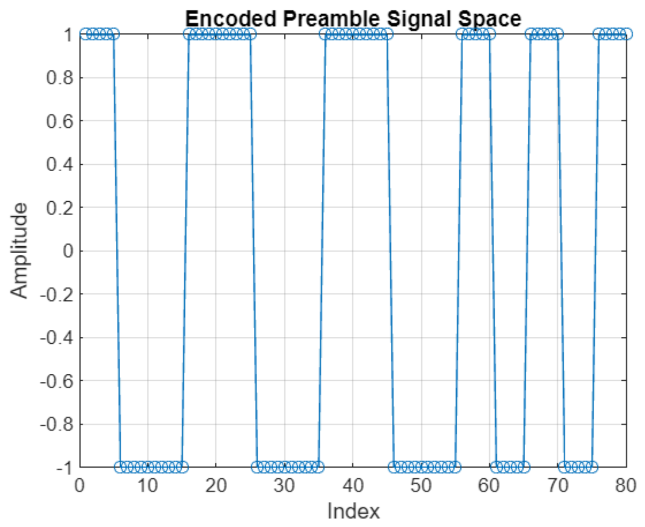
We used cross-correlation to find the most probable index to the start of the packet.

```
% Step 5: Cross-correlation to detect the preamble
r_ds = correlate(real(decoded_signal), preamble);

% Set the threshold for peak detection
threshold = 0.65 * max(r_ds);

% Find the peaks above the threshold
peaks = r_ds(r_ds > threshold);

% Find the peak indices
peak_indices = find(r_ds > threshold);

% Find the first and second peak indices
first_peak_index = peak_indices(1);

demodulated_bits = decoded_signal(first_peak_index+1.5:first_peak_index+1.5+79);
```

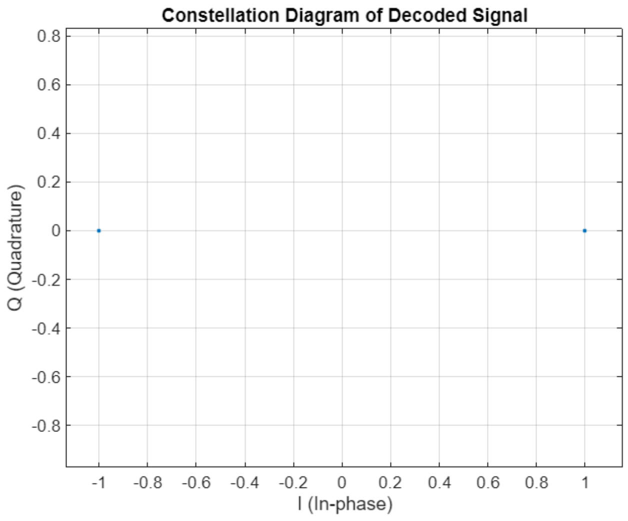


The first peak index was at 2161.





We were able to detect the closest interval that resembles the preamble from the signal.

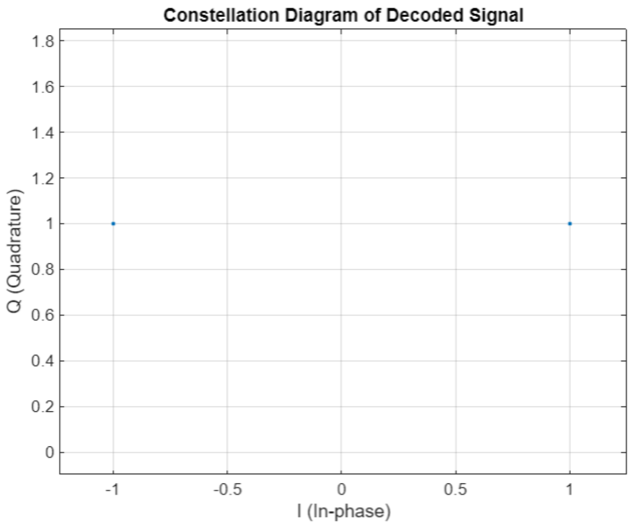


As you can see, now we only have two clusters at (-1,0) and (1,0) which is to be expected because the imaginary signal was zeroed out using the threshold.

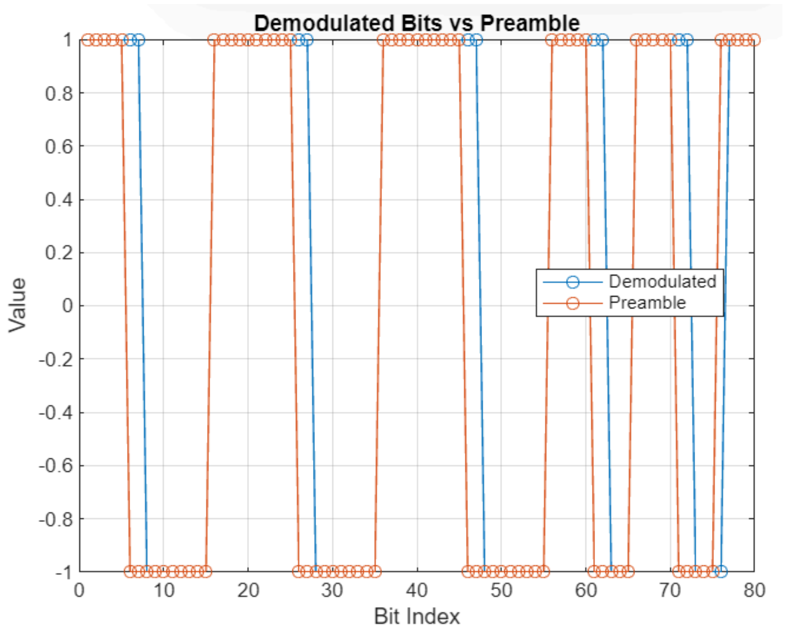
The threshold was set at about the midpoint between the two ASK amplitudes, therefore, the demodulated signal was effectively binary, with one level corresponding to one ASK amplitude and the other level corresponding to the other ASK amplitude. This binary signal is now represented as a BPSK constellation diagram, where the two points correspond to the two binary symbols (-1 and 1).

If we were to decode the imaginary and real signal separately with their own threshold values:

Either wouldn't matter because to detect the preamble we only needed to use the real signal. (Using the imaginary signal would yield similar results because the signals are identical besides the amplitude values).

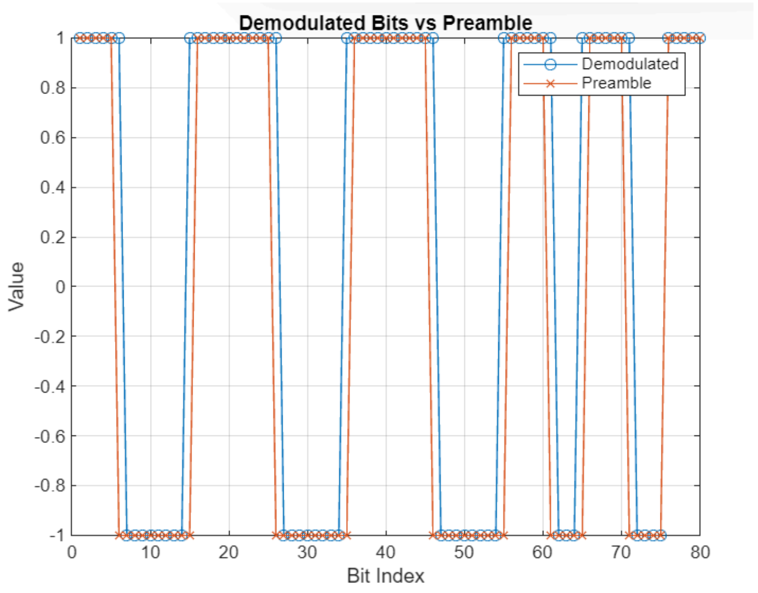


Error between Preamble and Detected Packet:



We were able to accurately detect 69/80 points.

This is a real-life signal where we have to deal with the doppler shift and other channel impairments; these factors can introduce delays in the received signal, leading to timing offsets between the transmitted and received bits. The demodulated bits seem to be two bits ahead of the preamble. This could be dealt with by using timing synchronization, fixing this misalignment.

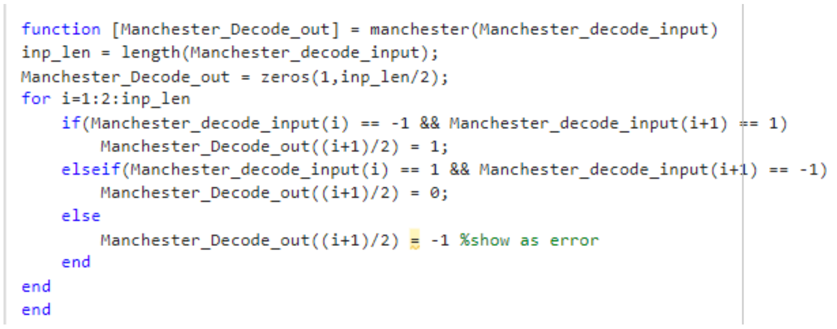


We could attempt to correct this by shifting the index by 1 but we'd need proper time correction to fix the error in bit detection.

This decreased the error from 11 to 9.

6) Manchester Decoding (Optional)

```
manchester_bits = manchester(demodulated_bits);
```

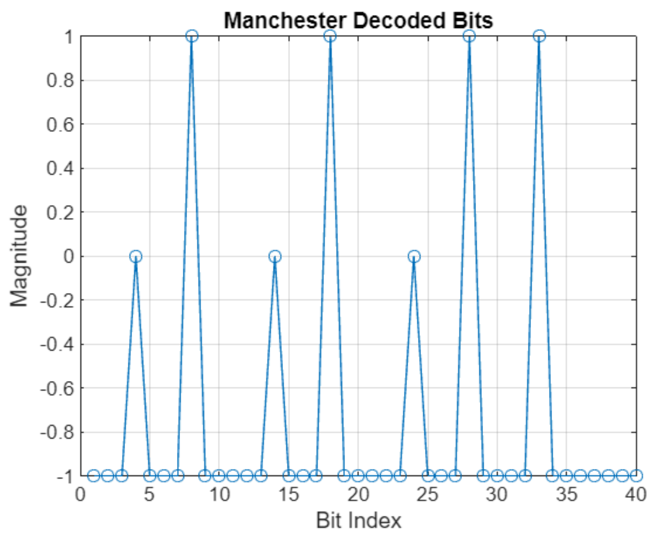


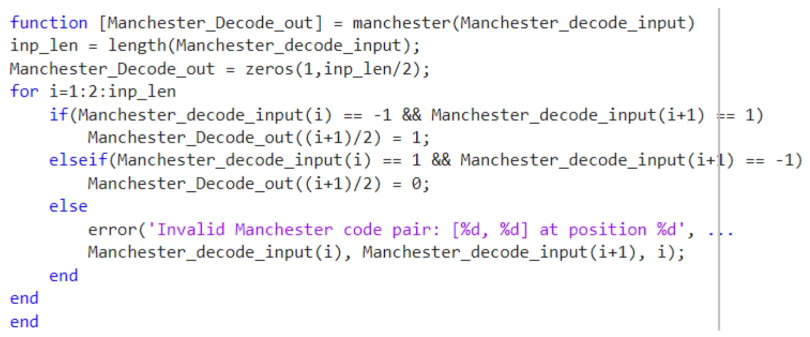
```
function [Manchester_Decode_out] = manchester(Manchester_decode_input)
inp_len = length(Manchester_decode_input);
Manchester_Decode_out = zeros(1,inp_len/2);
for i=1:2:inp_len
    if(Manchester_decode_input(i) == -1 && Manchester_decode_input(i+1) == 1)
        Manchester_Decode_out((i+1)/2) = 1;
    elseif(Manchester_decode_input(i) == 1 && Manchester_decode_input(i+1) == -1)
        Manchester_Decode_out((i+1)/2) = 0;
    else
        Manchester_Decode_out((i+1)/2) = -1 %show as error
    end
end
end
```

Manchester Mapping:
-1,1: 1
1, -1: 0
1, 1: error
-1, -1: error

For the sake of graphing, we represented the bit errors with -1.

However, it would just result in an error flag:



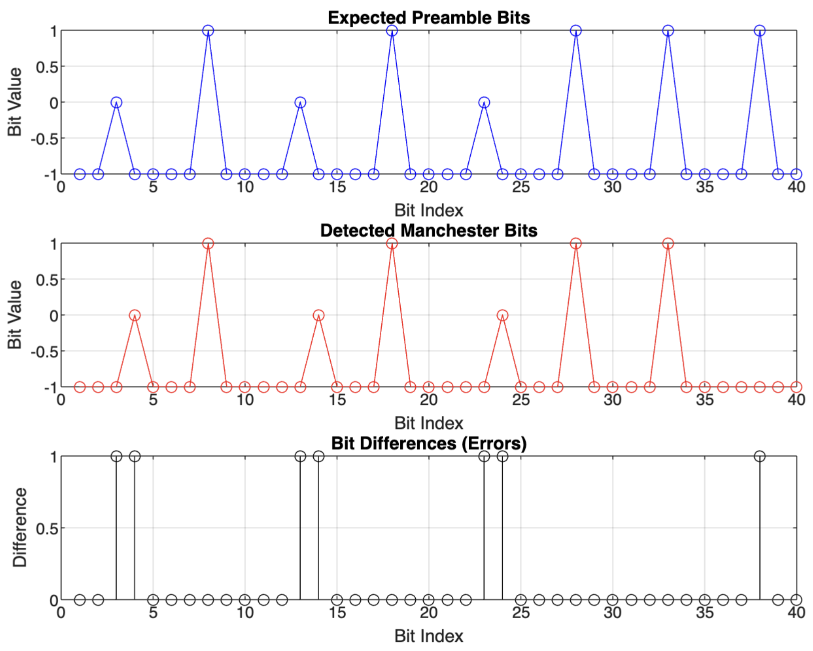


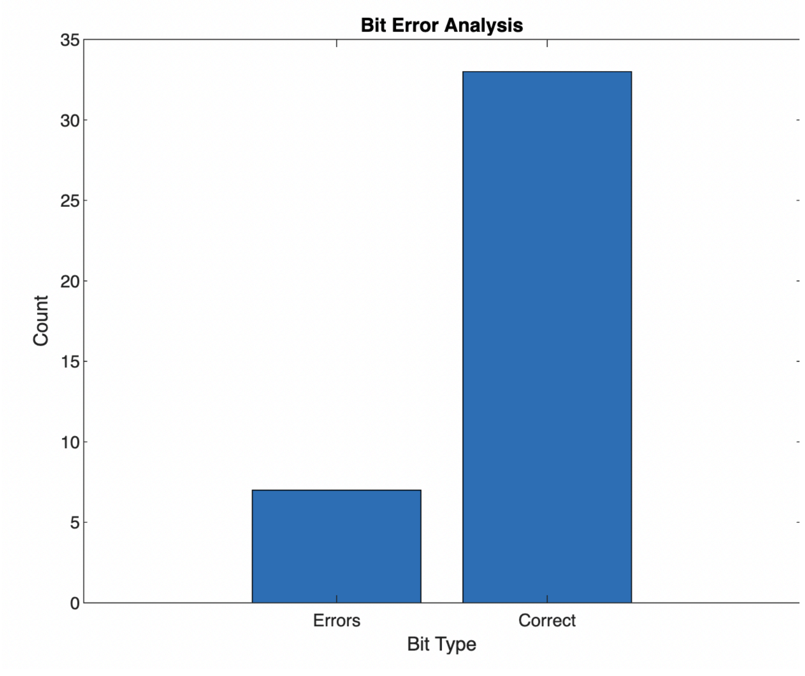
```
function [Manchester_Decode_out] = manchester(Manchester_decode_input)
inp_len = length(Manchester_decode_input);
Manchester_Decode_out = zeros(1,inp_len/2);
for i=1:2:inp_len
    if(Manchester_decode_input(i) == -1 && Manchester_decode_input(i+1) == 1)
        Manchester_Decode_out((i+1)/2) = 1;
    elseif(Manchester_decode_input(i) == 1 && Manchester_decode_input(i+1) == -1)
        Manchester_Decode_out((i+1)/2) = 0;
    else
        error('Invalid Manchester code pair: [%d, %d] at position %d', ...
        Manchester_decode_input(i), Manchester_decode_input(i+1), i);
    end
end
end
```

<u>This is invalid Manchester Code.</u>

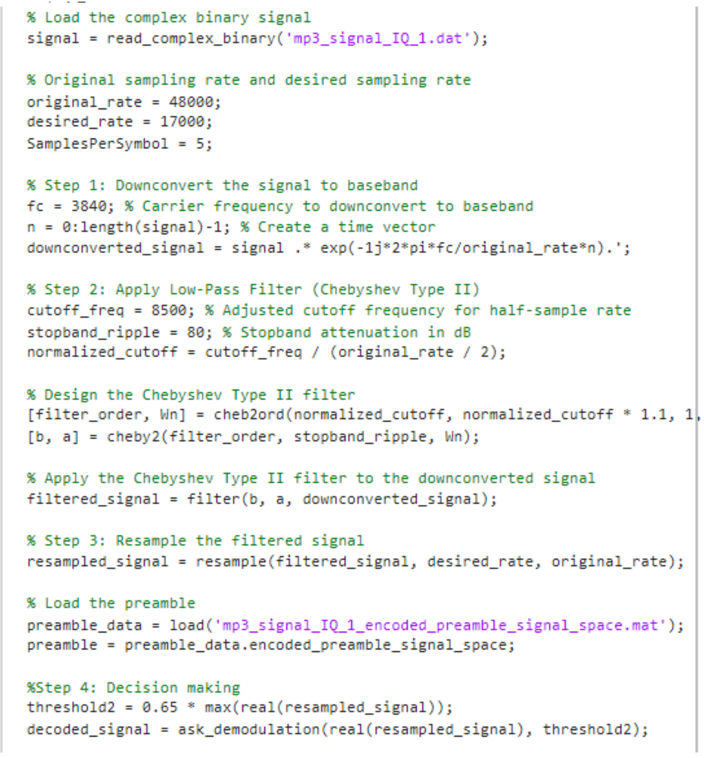
BER:

```
Bit Error Rate (BER): 0.175
```

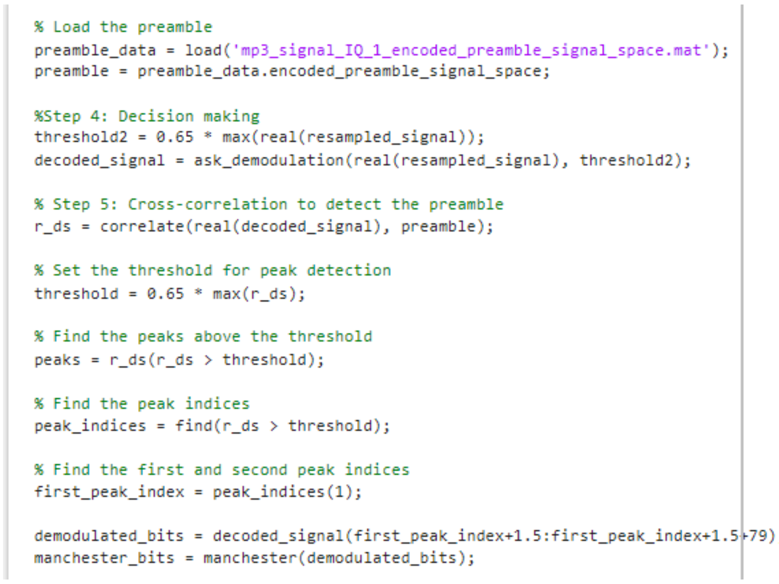




Full code:

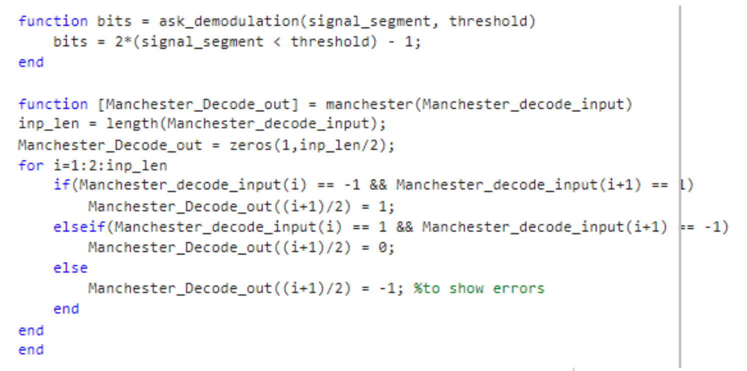


```
% Load the complex binary signal
signal = read_complex_binary('mp3_signal_IQ_1.dat');

% Original sampling rate and desired sampling rate
original_rate = 48000;
desired_rate = 17000;
SamplesPerSymbol = 5;

% Step 1: Downconvert the signal to baseband
fc = 3840; % Carrier frequency to downconvert to baseband
n = 0:length(signal)-1; % Create a time vector
downconverted_signal = signal .* exp(-1j*2*pi*fc/original_rate*n).';

% Step 2: Apply Low-Pass Filter (Chebyshev Type II)
cutoff_freq = 8500; % Adjusted cutoff frequency for half-sample rate
stopband_ripple = 80; % Stopband attenuation in dB
normalized_cutoff = cutoff_freq / (original_rate / 2);

% Design the Chebyshev Type II filter
[filter_order, Wn] = cheb2ord(normalized_cutoff, normalized_cutoff * 1.1, 1, stopband_ripple)
[b, a] = cheby2(filter_order, stopband_ripple, Wn);

% Apply the Chebyshev Type II filter to the downconverted signal
filtered_signal = filter(b, a, downconverted_signal);

% Step 3: Resample the filtered signal
resampled_signal = resample(filtered_signal, desired_rate, original_rate);

% Load the preamble
preamble_data = load('mp3_signal_IQ_1_encoded_preamble_signal_space.mat');
preamble = preamble_data.encoded_preamble_signal_space;

%Step 4: Decision making
threshold2 = 0.65 * max(real(resampled_signal));
decoded_signal = ask_demodulation(real(resampled_signal), threshold2);
```

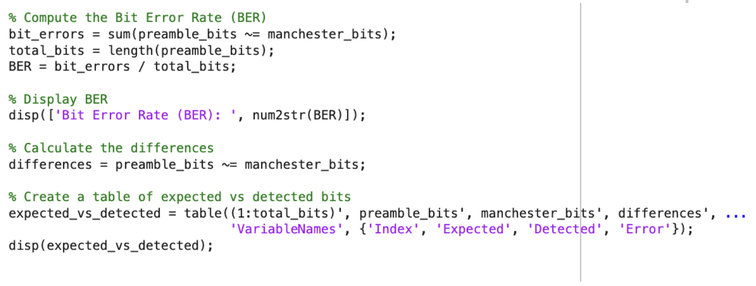


```
% Load the preamble
preamble_data = load('mp3_signal_IQ_1_encoded_preamble_signal_space.mat');
preamble = preamble_data.encoded_preamble_signal_space;

%Step 4: Decision making
threshold2 = 0.65 * max(real(resampled_signal));
decoded_signal = ask_demodulation(real(resampled_signal), threshold2);
```

```
% Step 5: Cross-correlation to detect the preamble
r_ds = correlate(real(decoded_signal), preamble);

% Set the threshold for peak detection
threshold = 0.65 * max(r_ds);

% Find the peaks above the threshold
peaks = r_ds(r_ds > threshold);

% Find the peak indices
peak_indices = find(r_ds > threshold);

% Find the first and second peak indices
first_peak_index = peak_indices(1);

demodulated_bits = decoded_signal(first_peak_index+1.5:first_peak_index+1.5+79)
manchester_bits = manchester(demodulated_bits);
```



```
function bits = ask_demodulation(signal_segment, threshold)
    bits = 2*(signal_segment < threshold) - 1;
end

function [Manchester_Decode_out] = manchester(Manchester_decode_input)
inp_len = length(Manchester_decode_input);
Manchester_Decode_out = zeros(1,inp_len/2);
for i=1:2:inp_len
    if(Manchester_decode_input(i) == -1 && Manchester_decode_input(i+1) == 1)
        Manchester_Decode_out((i+1)/2) = 1;
    elseif(Manchester_decode_input(i) == 1 && Manchester_decode_input(i+1) == -1)
        Manchester_Decode_out((i+1)/2) = 0;
    else
        Manchester_Decode_out((i+1)/2) = -1; %to show errors
    end
end
end
```



```
% Compute the Bit Error Rate (BER)
bit_errors = sum(preamble_bits ~= manchester_bits);
total_bits = length(preamble_bits);
BER = bit_errors / total_bits;

% Display BER
disp(['Bit Error Rate (BER): ', num2str(BER)]);

% Calculate the differences
differences = preamble_bits ~= manchester_bits;

% Create a table of expected vs detected bits
expected_vs_detected = table((1:total_bits)', preamble_bits', manchester_bits', differences', ...
                             'VariableNames', {'Index', 'Expected', 'Detected', 'Error'});
disp(expected_vs_detected);
```