DCS Assignment

Ayush (2019156), Rishabh(2019193) , Rishi (2019194), Tathagat(2019211)

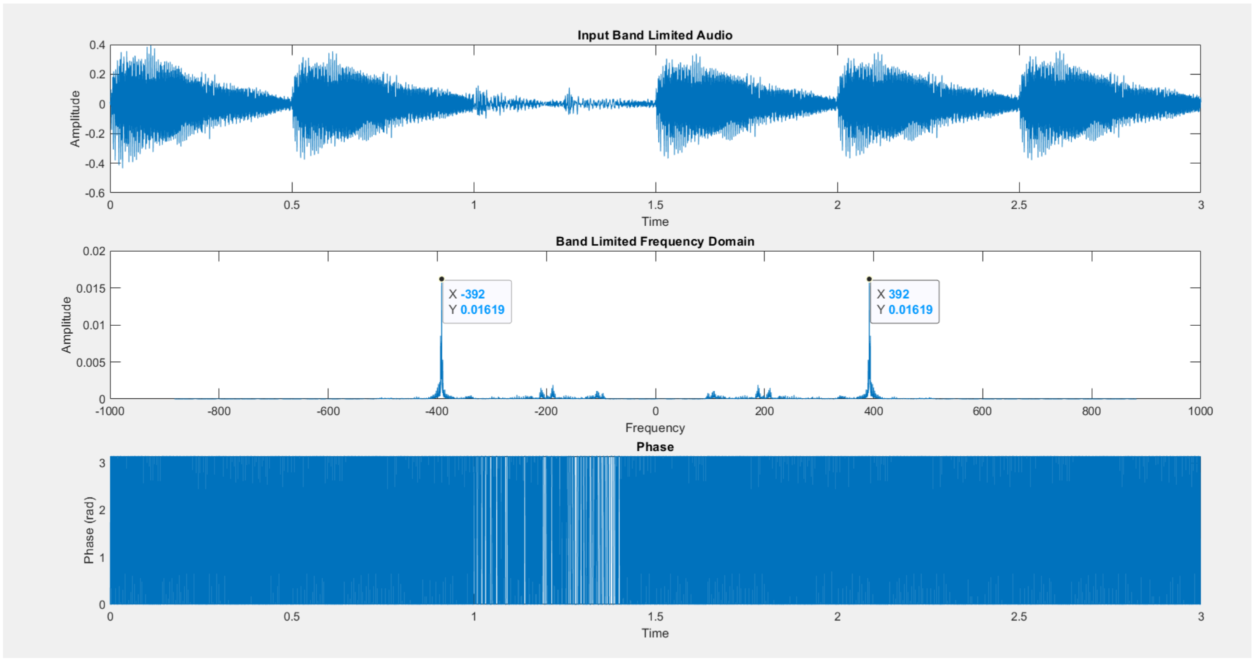
In this assignment, we have implemented a complete end-to-end communication system in MATLAB. To get the outputs for different sections, simply run the Main_Code.m file.

Step 1: Generate Input Signal

We generated our own custom audio file named “WID.wav” and then have done further operations on the same. Downsampling has been done to reduce data points to ease computation.

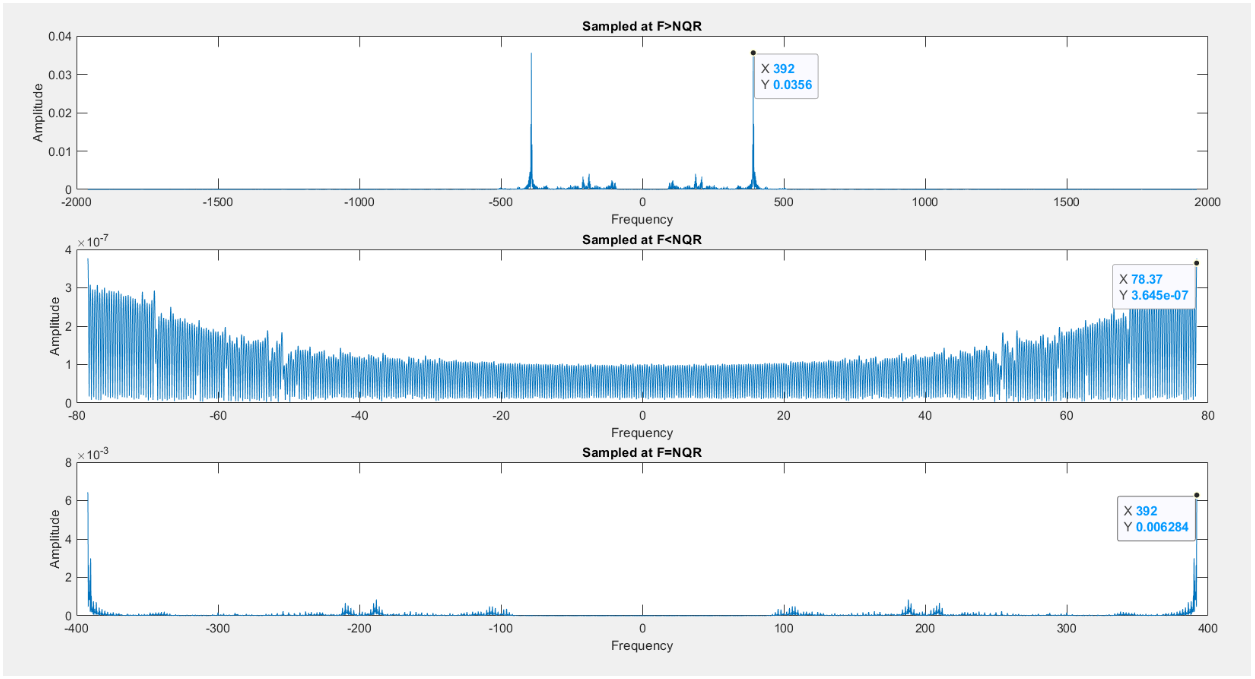
Step 2: Importing Input Signal & its characteristics

The audio file “WID.wav” is read using the inbuilt audioread() function in MATLAB. Initially we clipped the signal to 3 secs and the signal is further **bandlimited to 500Hz**. The time, frequency and phase domain representation are shown. All this is done in a function input()



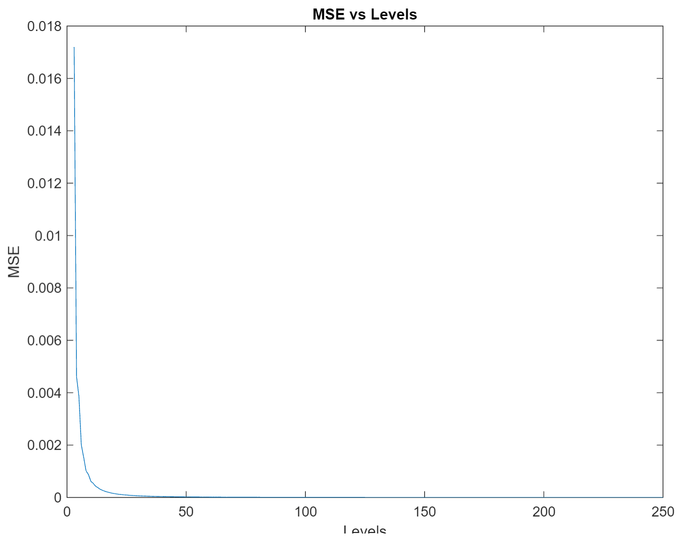
Step 3 : Sampling the i/p signal

Here we find that the maximum frequency component is **392 Hz**. Hence setting our Nyquist Rate appropriately, we sample the signal at frequencies less than(note the aliasing here), equal to and greater than Nyquist Rate. This is done in the source() function.



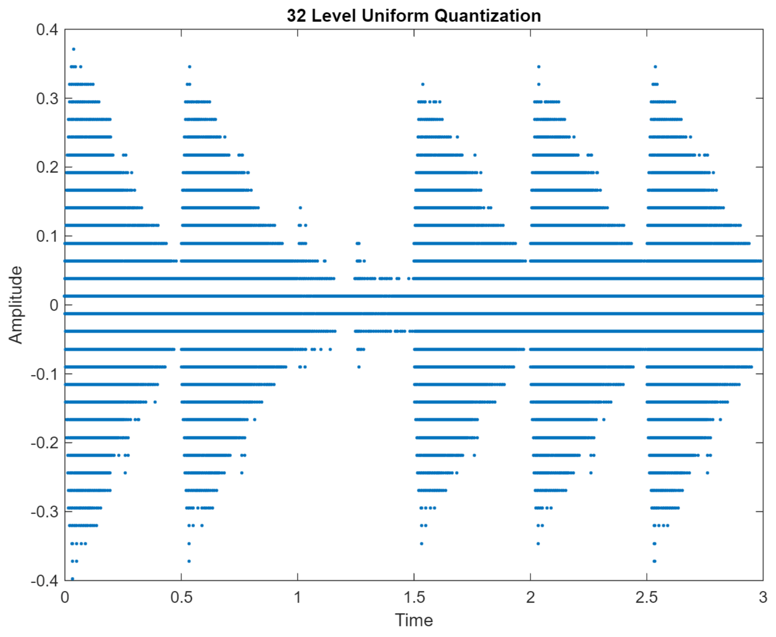
Step 4: Quantization : MSE v/s levels

First we take different levels from **3 to 250** and the MSE(mean squared error) vs Levels graph is plotted for uniform scalar quantization. This is done in the quantize() function.



Step 5: Quantization : Encoding samples to bitstream

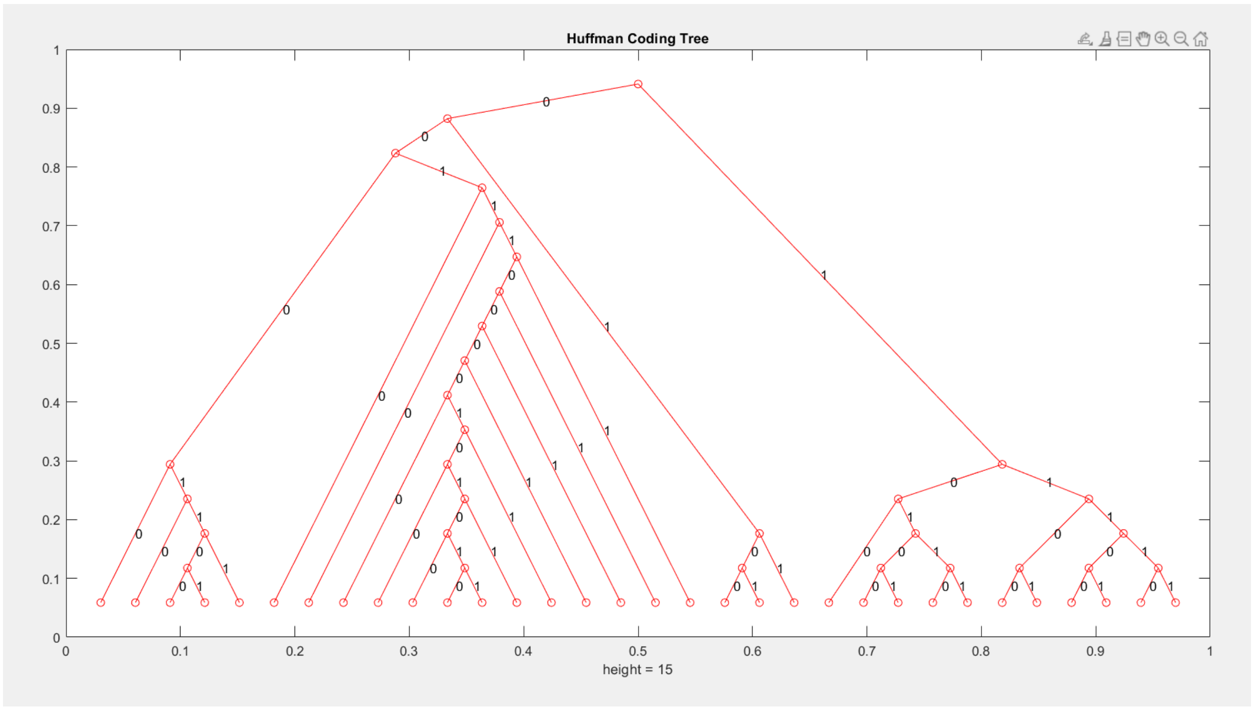
Then we quantize the signal using 32 levels of uniform scalar quantization, thus using 5 bits to represent each level. **661505** bits are used in total. (bitstream generated in the code itself). This is done in the quantize() function.



(Note: we are using “.” to represent a point, thus the observed discontinuity)

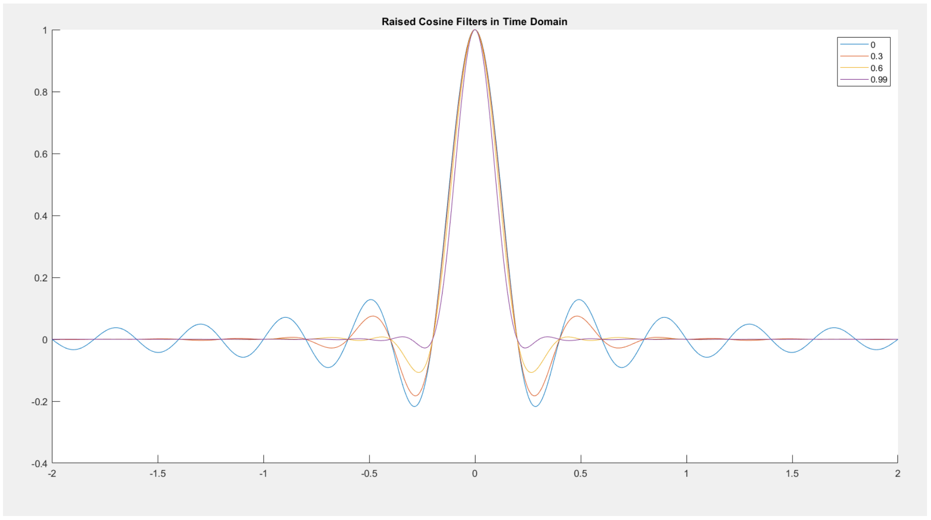
Step 6 : Huffman Encoding

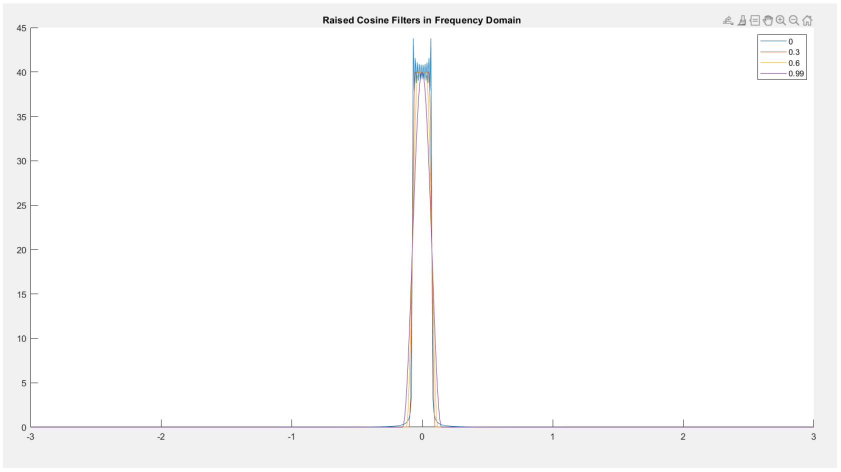
To perform Huffman encoding we have made a function called encode() in the encode.m file. This function takes input the levels array from the previous section of quantization and then we calculate the empirical probability distribution for the source symbols by simply calculating the frequency of each symbol and dividing it by total no. of symbols that have occurred. After getting this empirical probability distribution we have assigned each of the symbols with a unique code using the Huffman Coding Algorithm. The result of the Huffman coding is stored in a 2D matrix named final_codebook. The first column of the codebook corresponds to the symbol name and the 2nd column represents the code for the particular symbol. The number of bits after Huffman coding came out to be **563037** bits which was **much lesser than the 661505 bits** that we got after assigning each level with 5 bits (after quantization). Thus, we were successfully able to do huffman encoding by reducing down the length of the bitstream.(lossless compression). This is done in the encode() function.



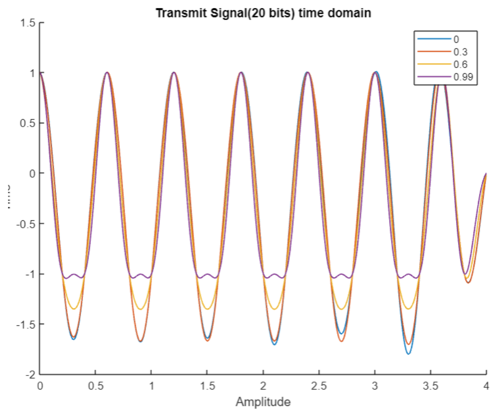
Step 7 : Pulse Shaping

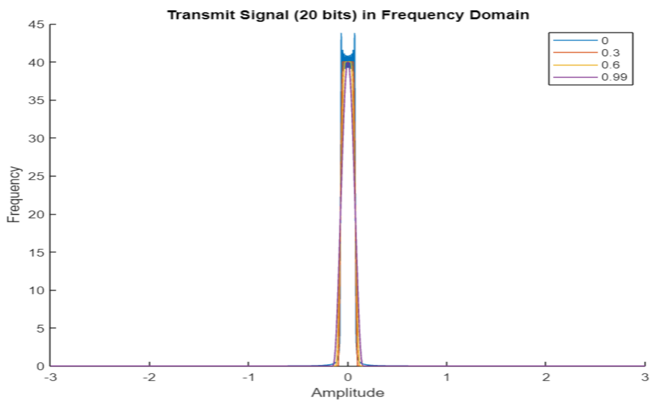
Huffman encoding gives us a bitstream which is modulated using PAM (+1 for 1 and -1 for 0). The modulated signal is then passed through an RC Filter to reduce transmission ISI. The roll off factor [0,0.3,0.6,0.99] is varied to visualize differences in time and frequency domain of the RC Filter. To visualize the same we did 2 cases - 1.) taking the first 3 bits and 2.) taking the first 20 bits. We noticed minimum ISI when roll off factor=0.99, and the ISI increases as we decrease the roll off factor. But we also need to take into account the bandwidth. This is done in the pulseshaping() function.



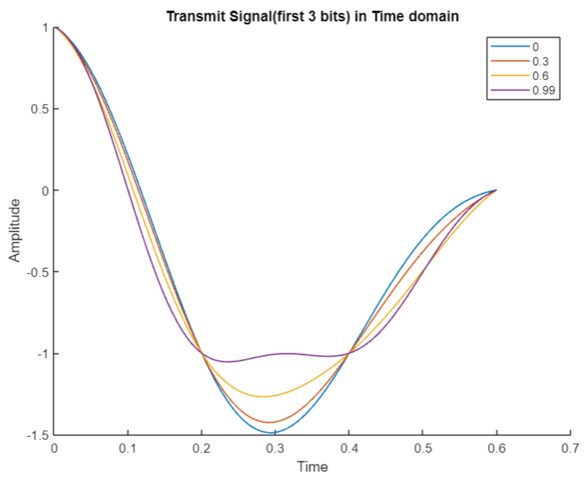


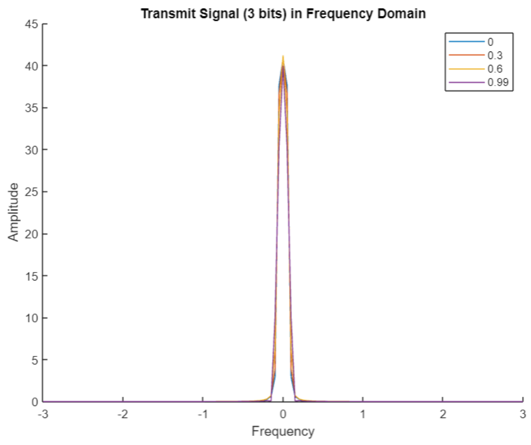
a.) For First 20 bits = 10010010010010010010





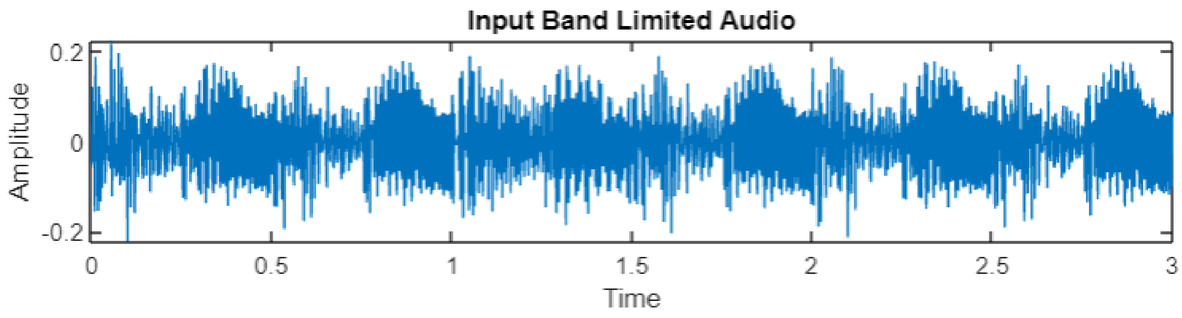
b.) First 3 bits - 100





Step 8 : Signal Reconstruction (BONUS)

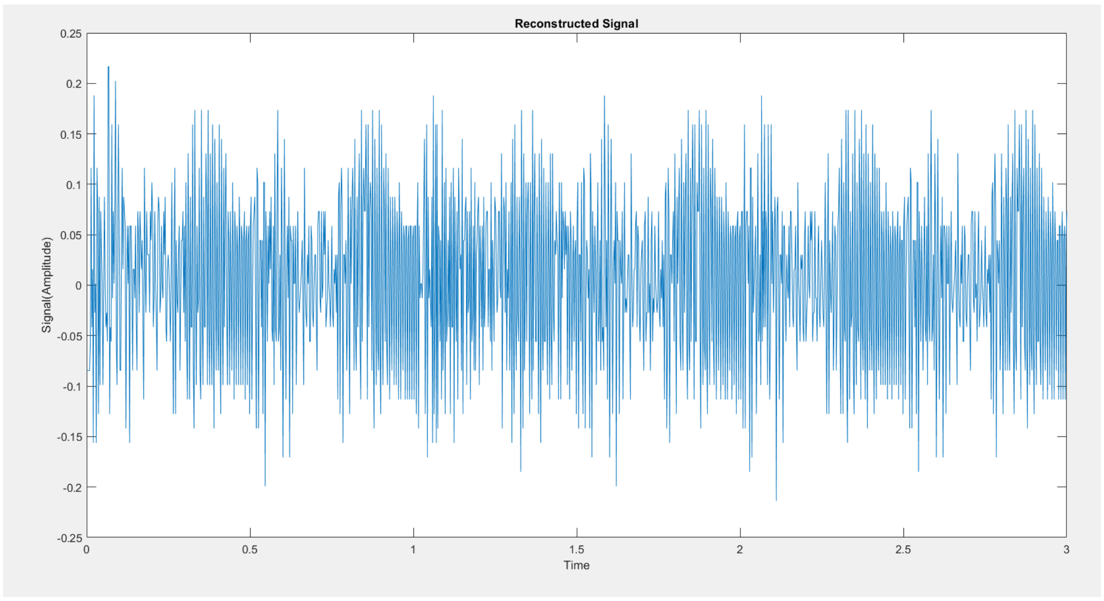
Firstly we downsampled our initial signal for this section of the assignment by a factor of 100. Then we sample the signal at the bit duration defined as Tb = 0.2, and check the value of the o/p waveform. Decoding is done in accordance with PAM i.e. if -1 is received then the signal is decoded as 0 else 1.



(Bandlimited signal after downsampling)

After getting back the transmitted bits after applying the decoding rule, we perform the huffman decoding on the decoded bits and convert them back to their symbols (A,B,C,D, etc). Now since we know from huffman encoding that which amplitude a particular symbol is mapped to. Thus we do inverse mapping and map each of the symbols that we got after huffman decoding. This provides us with the reconstructed signal which is shown below.
We have also stored the same signal in an out.wav file in our code.

Comparison with i/p signal : Since we had downsampled the i/p signal by a factor of 100 for reducing time complexity of our code for this assignment.Thus the downsampled i/p signal that we got was not a precise version of the original i/p. Thus, affecting the further sections of our code and so the o/p and initial i/p signal weren't totally same.


Reconstructed Signal
Signal(Amplitude)
Time
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
0
0.5
1
1.5
2
2.5
3