Department of Electrical & Electronics Engineering

Abdullah Gül University

EE 3001 Telecommunication System Design with DSP Capsule Project Report

---

Submitted on: (1/23/2021)

Submitted by:

Emre ASLAN (110 110 194)

Nuri CAN (110 110 236)

Yasar CANBOLAT (110 110 195)

Video and project files' link:

Click Here

Grade: / 100

OBJECTIVE

The objective of this project is to design a communication system that suppresses periodic noises and error correction on received audio signals using Universal Software Radio Peripheral (USRP). Firstly, the frequency Shift Keying (FSK) modulation technique was used to send data through the channel as a telecommunication part. Secondly, RLS adaptive filtering methodology was used to extract determined periodic noise on a signal including jet, fan, saw and drill background noises. In addition, (7,4) hamming code was used to encode binary data, and hard decision and soft-decision decoding techniques were implemented to observe the difference between these two error correction methodologies.

DESIGN AND TEST PROCEDURES

General overview of project

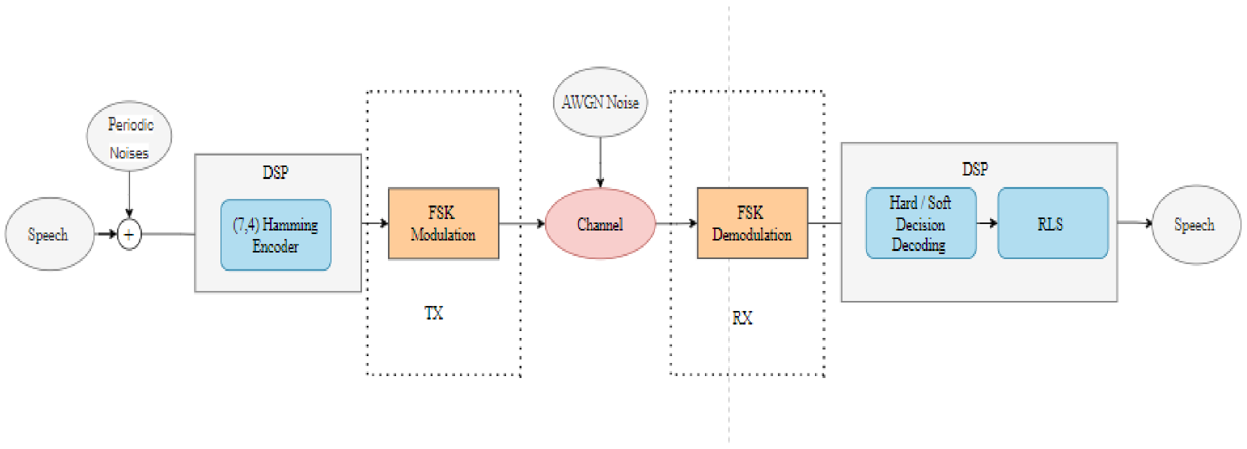
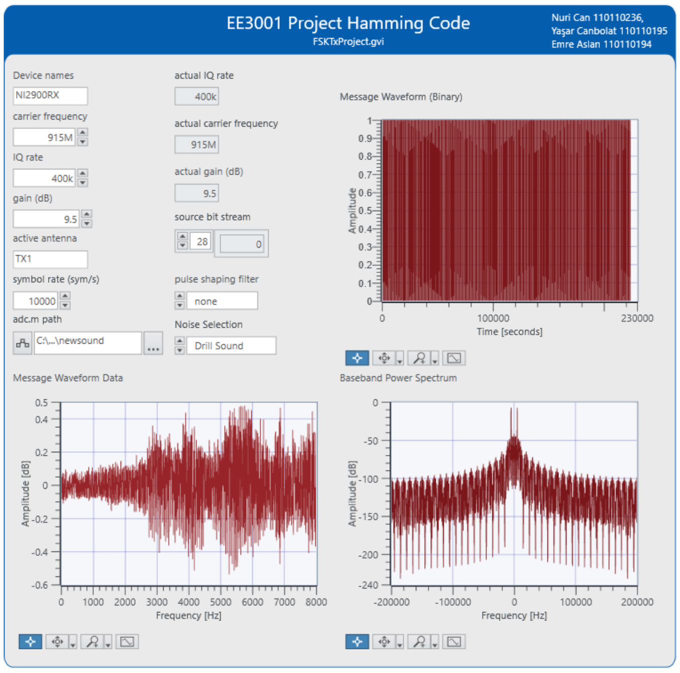


Figure 1: Block Diagram of the Project

Figure 1 shows the block diagram of the project to preserve a simplified view. As seen, the speech waveform is taken and a periodic noise is added to this signal. Then, Hamming encoder adds parity bits to the data. After that, FSK modulation takes place to transmit the data. RX part begins with the demodulation and Hamming code decoding the errored data to lessen the errors. To the error corrected waveform, RLS algorithm is applied to get rid of the periodic noise and the resulting data is played as sound in the end.

Frequency Shift Keying (FSK) TX Circuit


EE3001 Project Hamming Code
FSKTxProject.gvi
Nuri Can 110110236,
Yaşar Canbolat 110110195
Emre Aslan 110110194

Device names: NI2900RX
carrier frequency: 915M
IQ rate: 400k
gain (dB): 9.5
active antenna: TX1
symbol rate (sym/s): 10000
adc.m path: C:\...\newsound
actual IQ rate: 400k
actual carrier frequency: 915M
actual gain (dB): 9.5
source bit stream: 28, 0
pulse shaping filter: none
Noise Selection: Drill Sound

Figure 2: The panel of FSK TX design

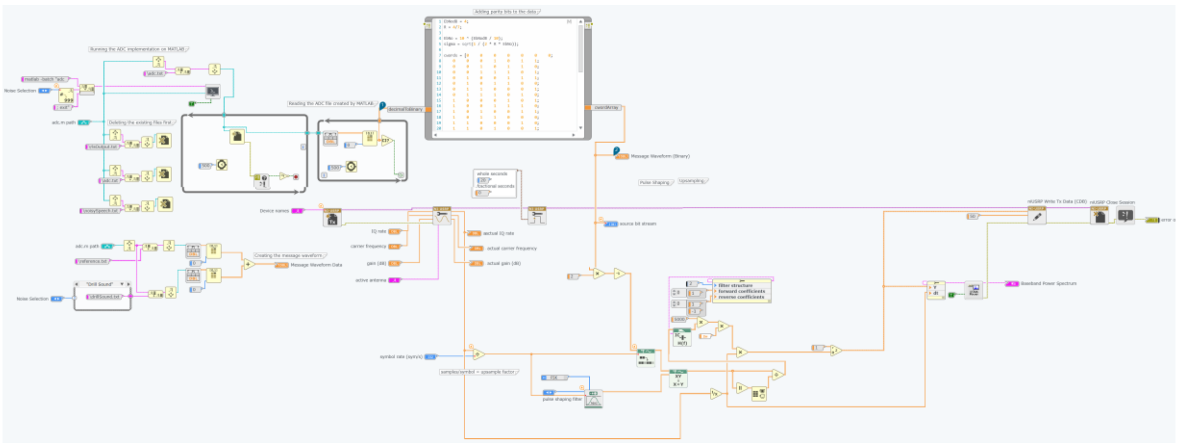
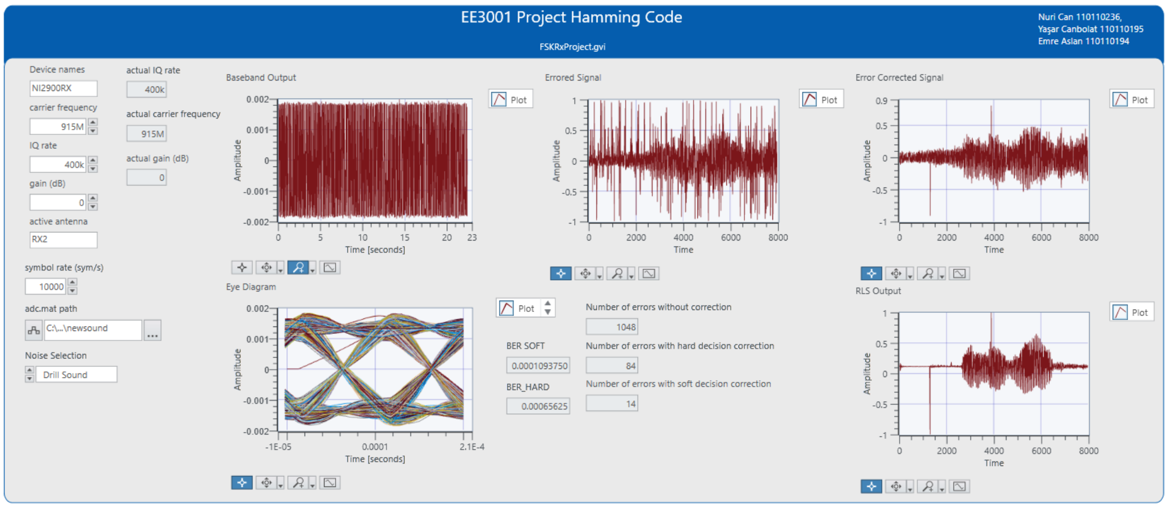


Figure 3: Full TX Schematic

The TX portion of this project is conducted according to the FSK TX lab methodology which was completed before. First, there's the output, which is obtained from floating data to binary. The bit value of 1 equates to 1 while the bit value of 0 corresponds to -1. The resulting data was then sampled using the L (Actual IQ rate /Symbol rate) up sampling factor. MT Generate Filter provided the output. The modulation type parameter has been changed as an FSK by coefficients. Furthermore, the pulse shaping filter parameter has been set to "none." The MT Generate Filter's output was rapidly analyzed. Finally, the baseband waveform was examined and the FFT spectrum for one channel signal was observed.

Frequency Shift Keying (FSK) RX Circuit



EE3001 Project Hamming Code

FSKRxProject.gvi

Nuri Can 110110236,
Yaşar Canbolat 110110195
Emre Aslan 110110194

Device names: NI2900RX
carrier frequency: 915M
IQ rate: 400k
gain (dB): 0
active antenna: RX2
symbol rate (sym/s): 10000
adc.mat path: C:\...\newsound
Noise Selection: Drill Sound
actual IQ rate: 400k
actual carrier frequency: 915M
actual gain (dB): 0
BER SOFT: 0.0001093750
BER_HARD: 0.00065625
Number of errors without correction: 1048
Number of errors with hard decision correction: 84
Number of errors with soft decision correction: 14

Figure 4: The panel of FSK RX design

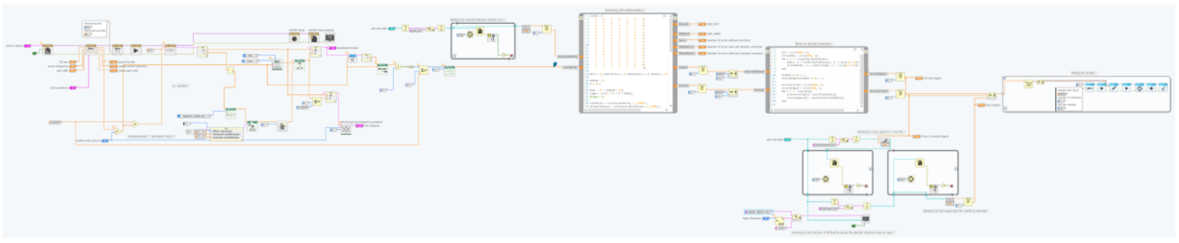


Figure 5: Full RX schematic

First, the number of samples in a frame is estimated by taking into account the message length, symbol rate, and the system's real IQ rate. The cluster attributes are used to connect the number of samples in a single frame output to the Unwrap phase's input. To finish all procedures, an FIR filter and a median filter are utilized. The MT Generate Filter was used with the identical parameters as the previous portion in the TX component. To better evaluate the signal resolution, the eye diagram of the Convolution result was generated. In addition, the Pulse Align function takes the output signal as an input. Finally, on FSK modulation, output bits are attached to the graphical representation to indicate the demodulated signal.

Synchronization

One of the serious problems that were encountered during the digital signal transmission is related to the synchronization of both the transmitter and receiver. Hence, we should come up with the solution of putting a certain amount of the delay in both TX and RX with the same durations. It was decided to use 30 secs delay before starting transmission. It was not easy to find how to implement such a delay into the system, but it was searched on Google and there were a couple of modules for such an application. And one of them suggests we use “niUSRP Configure Time Start Trigger” and it provides a certain amount of predefined delay. Finally, both TX and RX synchronized by putting a copy of the “niUSRP Configure Time Start Trigger”.

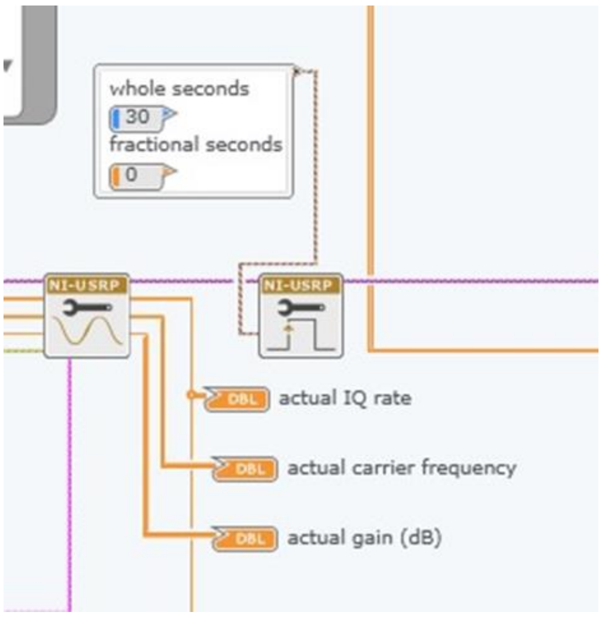


Figure 6: Synchronization Schematic

Analog to digital conversion

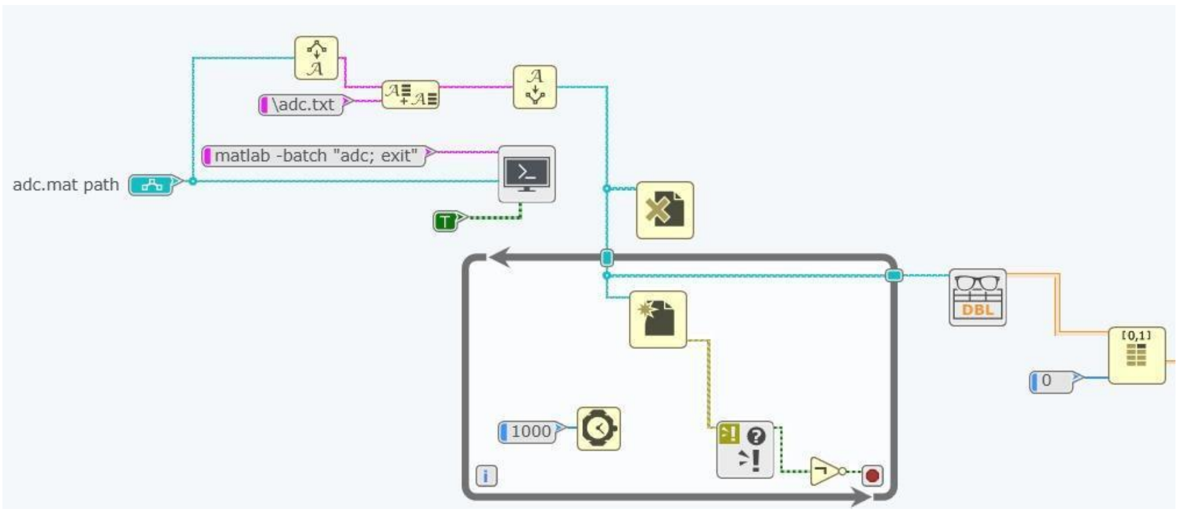


Figure 7: Analog to Digital Conservation on LabVIEW

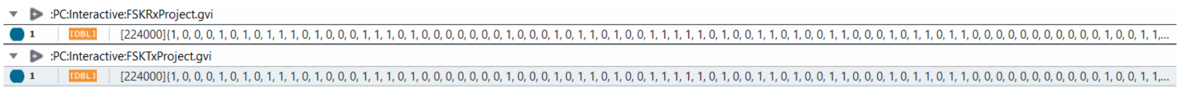
It was provided with an ADC.m file which contains our code to apply ADC to our data and save the output as a bitstream to adc.txt. The path of the ADC.m should be chosen from the panel for the program to operate. Before the adc.txt is created we are deleting the previous file and then use a while loop to check if the file is created again. When the file is created, the loop stops and passes the path to Read from Spreadsheet File VI. Then we are taking the first row of the bitstream and applying FSK afterwards, which we covered in the first report.

MATLAB code implementation of ADC can be seen below:

```
dt = 1 / 8000;
data = load("test.txt");

bits = 16;
partition = linspace(-1, 1, pow2(bits));
indx = quantiz(data, partition);

decimalToBinary = [];

d2b = dec2bin(indx, 16);
for i = 1 : length(d2b)
    for j = 1 : bits
        decimalToBinary(i,j) = str2double(d2b(i,j));
    end
end

decimalToBinary = reshape(decimalToBinary',1,128000)';
writematrix(decimalToBinary, "adc.txt");
```

The code divides the data into 2^16 parts as we are trying to represent the values in 16 bits. Then the code rounds the data values into their nearest partition values. Afterwards, these partition values are written and saved into a text file. Then we are reading this file from LabVIEW in the Tx part to send them. The path of the ADC.m should be chosen from the panel for the program to operate. Before the adc.txt is created we are deleting the previous file and then use a while loop to check if the file is created again. When the file is created, the loop stops and passes the path to Read from Spreadsheet File VI. Then we are taking the first row of the bitstream and applying FSK afterwards, which we covered in the first report.

For the RX part, with the changes we have done, now we can get the exact bitstream that is sent from TX in RX. The TX bitstream and RX bitstream can be seen below:

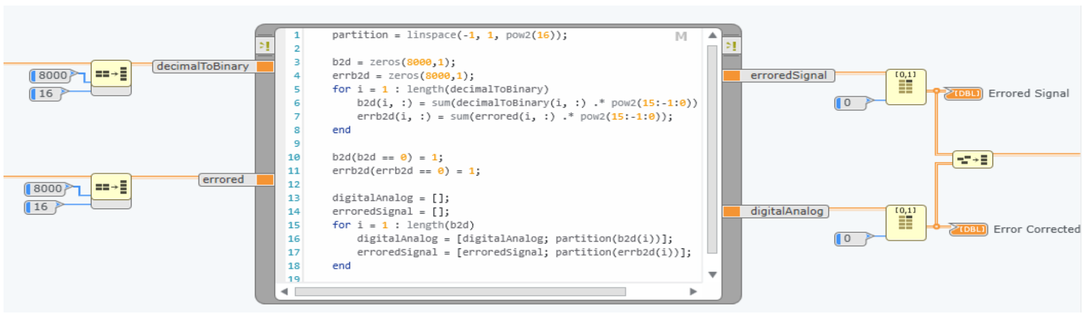


```
:PC:Interactive:FSKRxProject.gvi
1  [DBL]  [224000]{1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1,...
:PC:Interactive:FSKTxProject.gvi
1  [DBL]  [224000]{1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1,...
```

Figure 8:TX and RX bit Comparison

As seen, the bitstream in the Tx and Rx parts are now matching perfectly with the current adjustments. Lastly, we needed to convert the bitstream into their corresponding data values. To do so we created a Mathscript that takes the bits as an 8000x16 bit array, then finds the integer value of each row to find their indexes in the ADC part. Then, using these indexes we can find their corresponding values from the partition matrix and save them into an array (digital analogue). The component set can be seen below in Figure 2.

DAC



```
partition = linspace(-1, 1, pow2(16));

b2d = zeros(8000,1);
errb2d = zeros(8000,1);
for i = 1 : length(decimalToBinary)
    b2d(i, :) = sum(decimalToBinary(i, :) .* pow2(15:-1:0))
    errb2d(i, :) = sum(errored(i, :) .* pow2(15:-1:0));
end

b2d(b2d == 0) = 1;
errb2d(errb2d == 0) = 1;

digitalAnalog = [];
erroredSignal = [];
for i = 1 : length(b2d)
    digitalAnalog = [digitalAnalog; partition(b2d(i))];
    erroredSignal = [erroredSignal; partition(errb2d(i))];
end
```

Figure 9: Digital to Analog Conservation

The code takes each 16-bit row of data and error data and converts them into their decimal form again. These values were the partition values of the data. Therefore, to find their floating values again, the code finds the corresponding values in the partition array and saves them into matrices. At the output, we have the floating values of the error signal and corrected signal.

Playing Audio Circuit

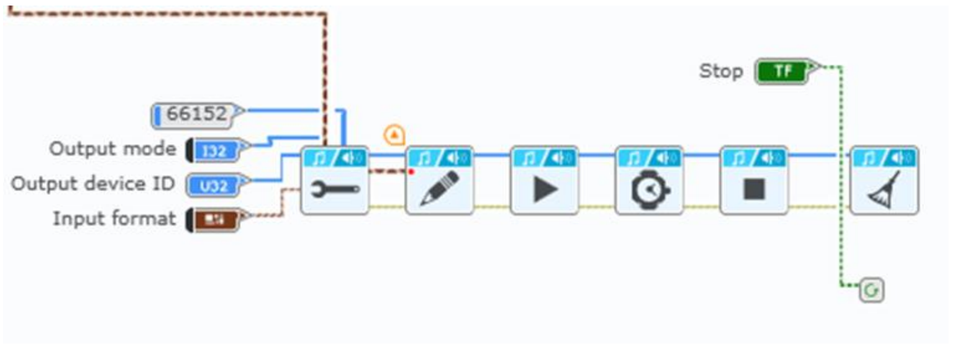


Figure 10: Voice data playing circuit.

The voice of the transmitted signal was listened to, and the signal was sampled more than the expected value but the audio was completely matched with the original version which was produced on the MATLAB. In conclusion, the signal was successfully modulated, demodulated, and turned into its previous form.

Eye Diagram

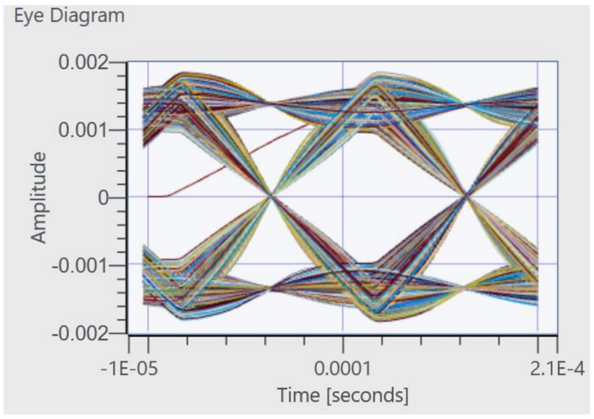


Figure 11:Eye Diagram of received signal

DSP Part

Hamming Code Test Simulation on LabVIEW

In this part, Simulation of (7,4) Hamming code (error correction) was implemented by using LabVIEW VIs. It was used as a testing procedure of Hamming code. After getting quite successful results, it was implemented with FSK modulation.

General Diagram of Encoding & Hard Decision and Soft Decision Decoding

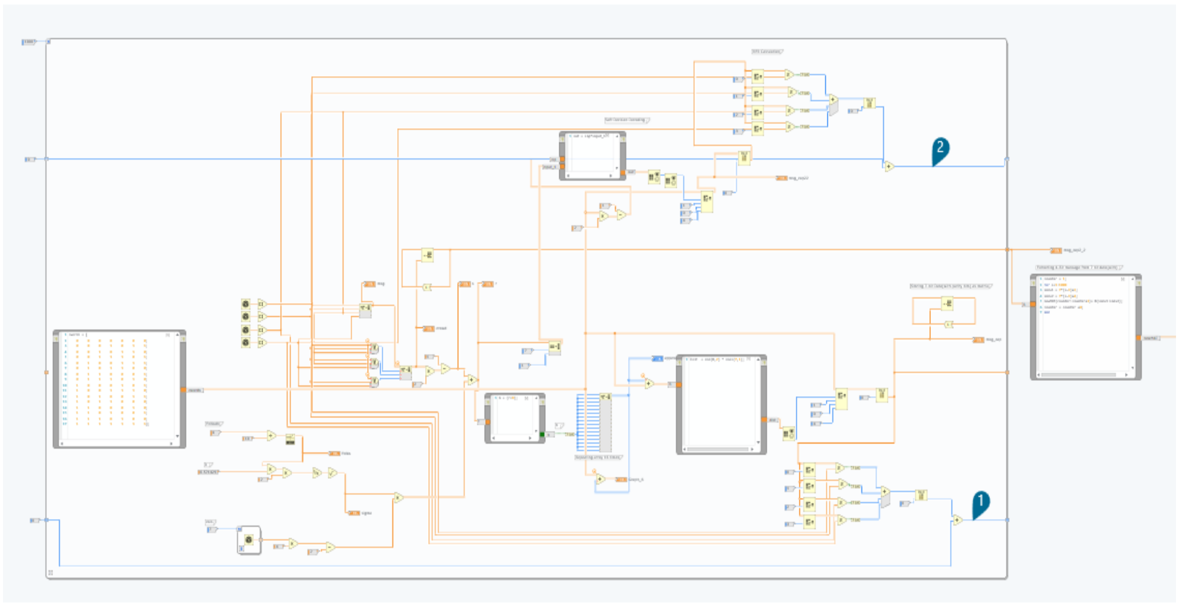


Figure 12: General Diagram of hamming encoding and decoding simulation

Generation of Codewords (ENCODING):

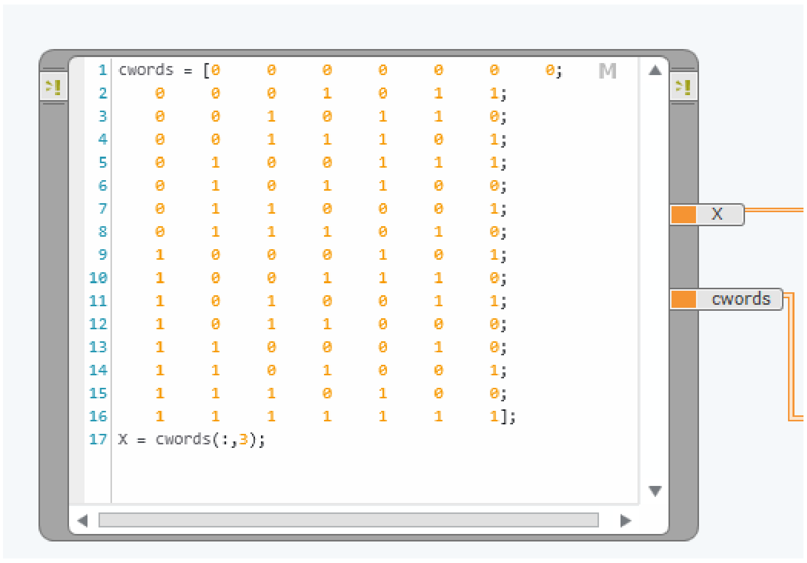

```
cwords = [0      0      0      0      0      0      0;
          0      0      0      1      0      1      1;
          0      0      1      0      1      1      0;
          0      0      1      1      1      0      1;
          0      1      0      0      1      1      1;
          0      1      0      1      1      0      0;
          0      1      1      0      0      0      1;
          0      1      1      1      0      1      0;
          1      0      0      0      1      0      1;
          1      0      0      1      1      1      0;
          1      0      1      0      0      1      1;
          1      0      1      1      0      0      0;
          1      1      0      0      0      1      0;
          1      1      0      1      0      0      1;
          1      1      1      0      1      0      0;
          1      1      1      1      1      1      1];
X = cwords(:,3);
```

Figure 13: Determination of codewords

In this part, codeword matrix was generated by using matrix mathscript. The codewords were produced with respect to 4/7 message transmission of the Hamming code.

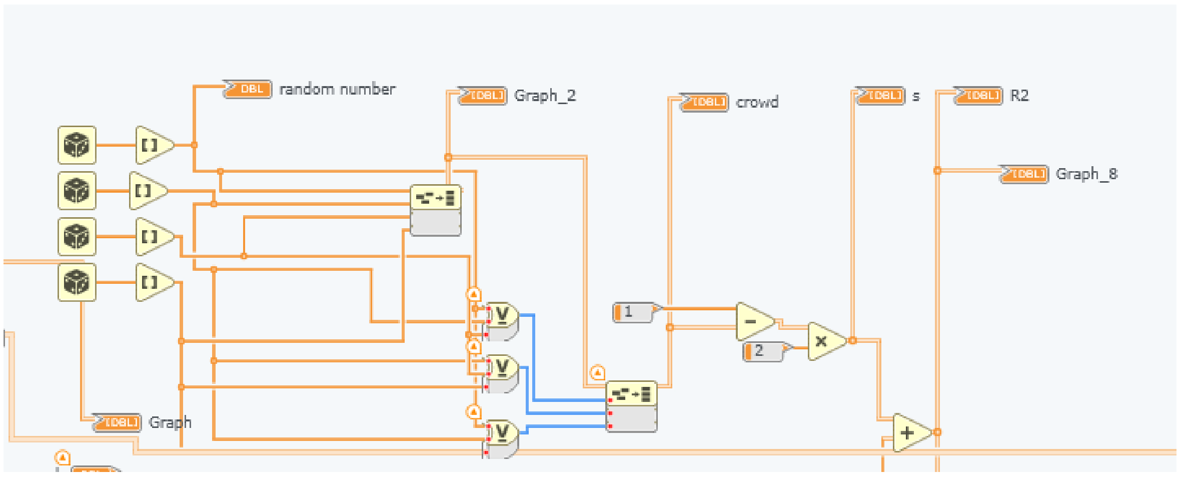


Figure 14: Encoder part of (7,4) Hamming code

As it can be seen from the figure, we generated ones and zeros by using random numbers and round to nearest as input for error correction. a 4-bit message signal is generated, and the Encoding process is made at that stage using gates.

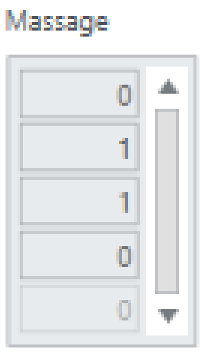


Figure 15: 4-bit generated massage signal

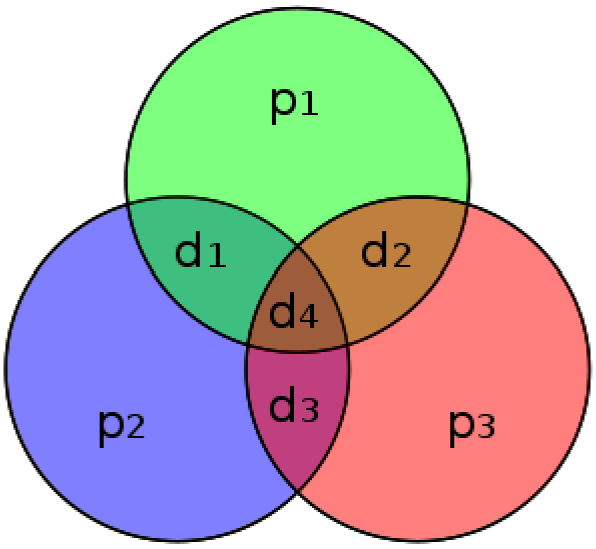


Figure 16: Determination of parity bits

Parity bit 1 was determined by using 3 bits of the message signal as d1, d4 and d3 with the XOR gate from LabVIEW with 3 inputs. Parity bit 2 was defined by using d1, d4 and d2 using XOR gate from LabVIEW with 3 inputs. Furthermore, parity bit 3 was calculated by using d2, d4 and d3 with the XOR gate from LabVIEW with 3 inputs.

Encoded Massage with parity bits

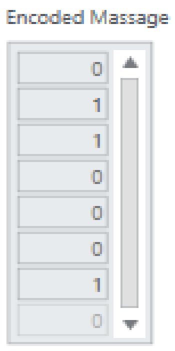


Figure 17: 3 parity bits are determined by using XOR gates.

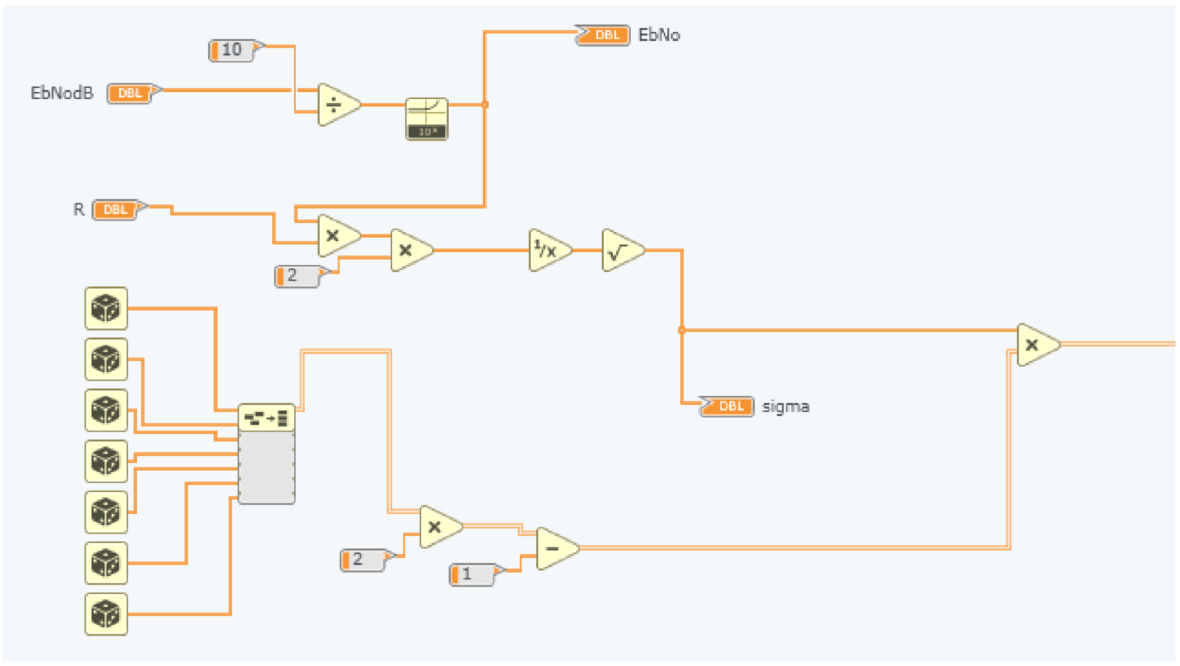


Figure 18: Random Noise Generation

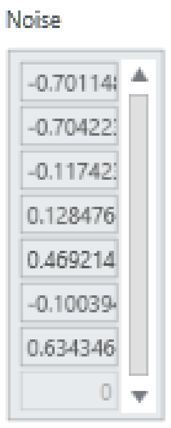


Figure 19: 7-bit Massage with Noise

In this part, 7-bit floating values between (-1 and 1) are generated using random data and a built-in array function as a noise for our signal. In addition, constant parameters are calculated according to features of predefined Hamming Code parameters including sigma, R, EbNoddB and EbNo to implement the (7,4) Hamming Code algorithm7 bit generated noise using random numbers and an array.

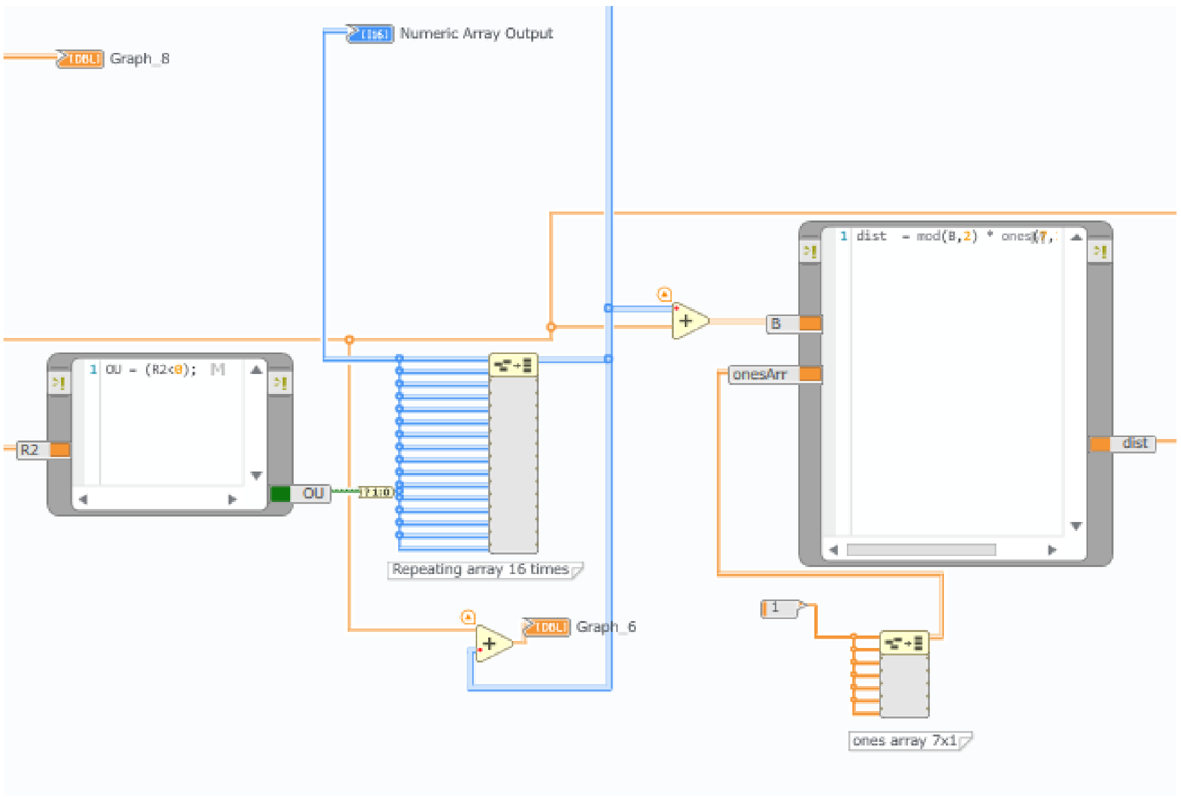
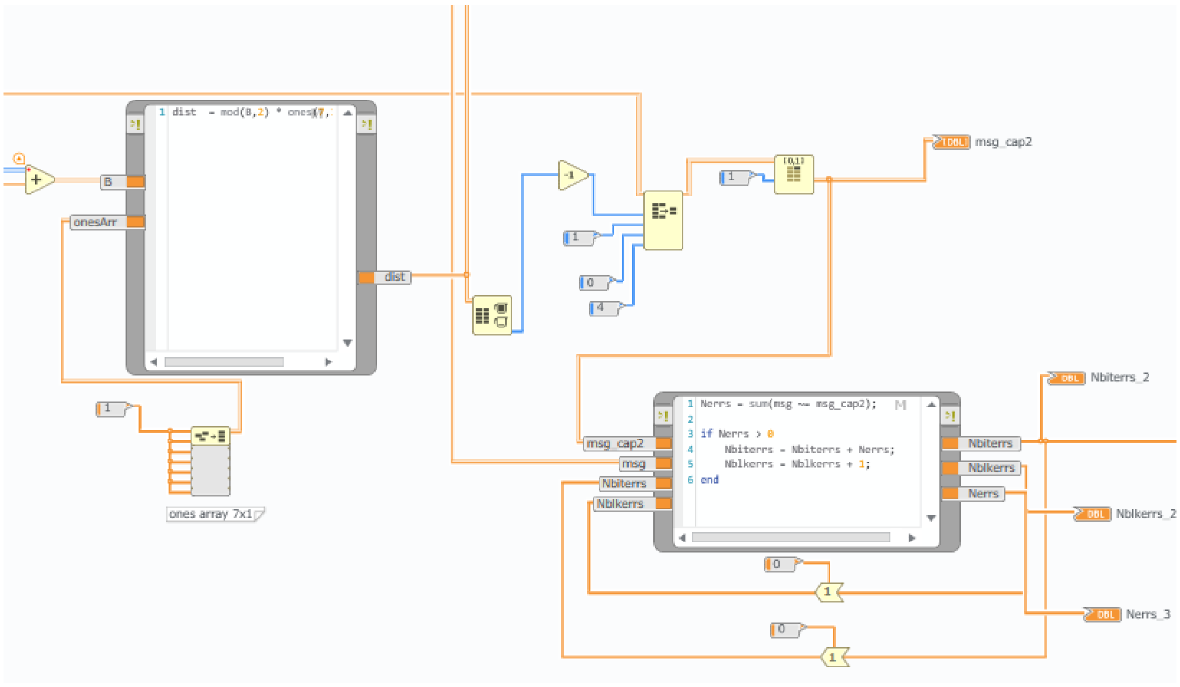


Figure 20: Hard Decision Decoding Circuit

Hard Decision Decoding

In this part, 7-bit encoded bits and 7-bit noise are summed up and observed arrays are compared with 0 threshold value. Streams of bits are compared with the threshold stage and decode each bit as either 1 or 0. It takes a sample of the incoming pulses and compares their voltages to predetermined thresholds. As a result of this comparison, obtained values are concatenated 16 times. After that stage, the least distance value was taken as an actual value of the transmitted message.



```
dist  = mod(B,2) * ones(7,
```

```
Nerrs = sum(msg ~= msg_cap2);

if Nerrs > 0
    Nbiterrs = Nbiterrs + Nerrs;
    Nblkerrs = Nblkerrs + 1;
end
```

Figure 21: Calculation of the number of error bits and BER

In this part, we tried to obtain the minimum distance between predefined 16 codewords and the actual message signal transmitted with AWGN noise. The message was matched with the minimum error codeword on the table. Finally, the number of the error bits were determined by looking at our message signal and the captured message signal after adding AWGN noise.

Soft Decision Decoding

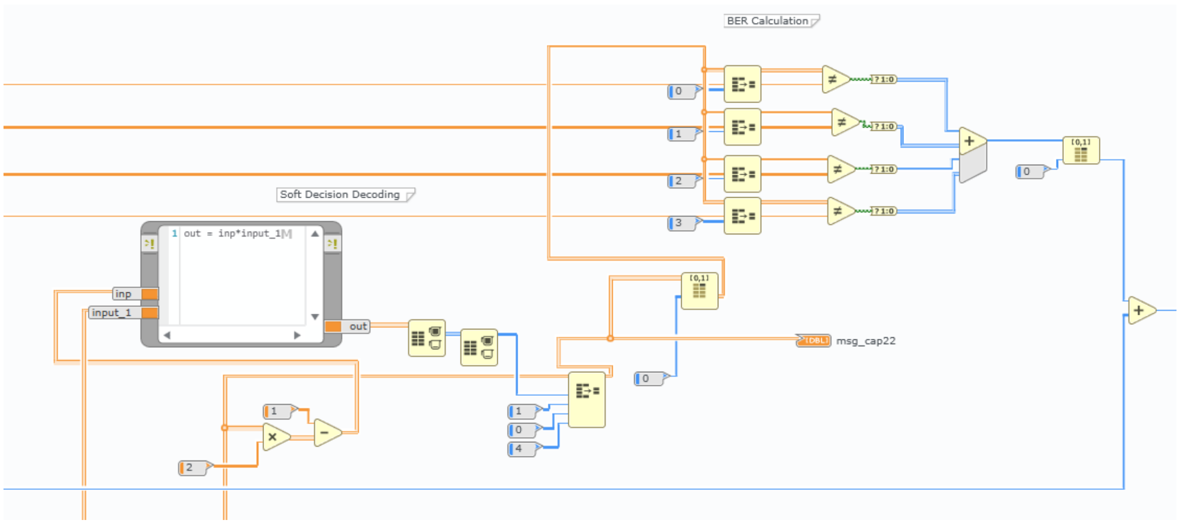


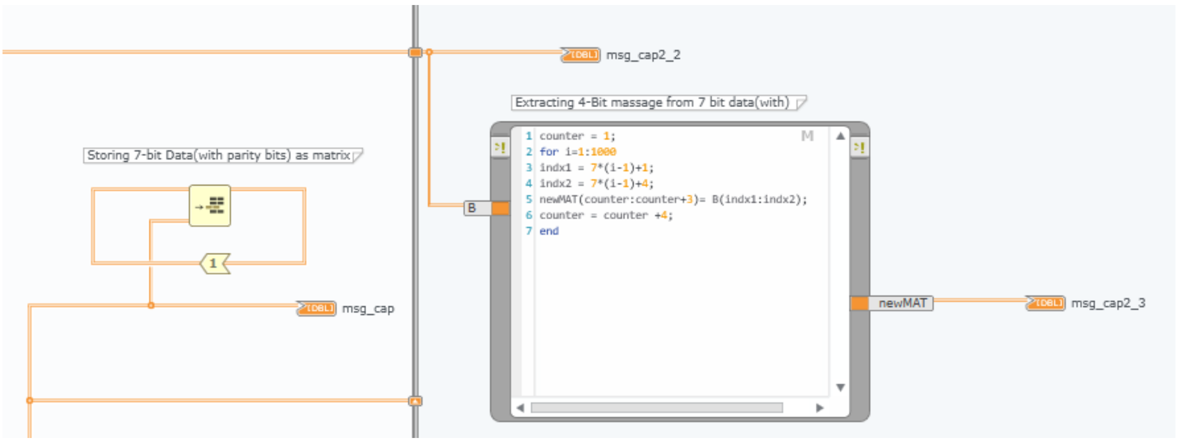
Figure 22: Soft Decision Decoding Circuit Diagram

What is meant by soft decoding?

Soft decision decoding is a type of algorithm that decodes a stream of bits by examining a range of potential values. To develop better estimations of input data, it examines the dependability of each incoming pulse. In our case, the correlation between (1-2*(codeword)) and the transpose of the r matrix is calculated, so the captured message signal is produced.

Bit ERROR Calculation:

Message bit coming from after soft-decision decoding is compared with the original message bits. After comparison, the number of errors is summed to learn the bit error rate for 16.000 bits. The shift register is used to calculate the total number of errors.

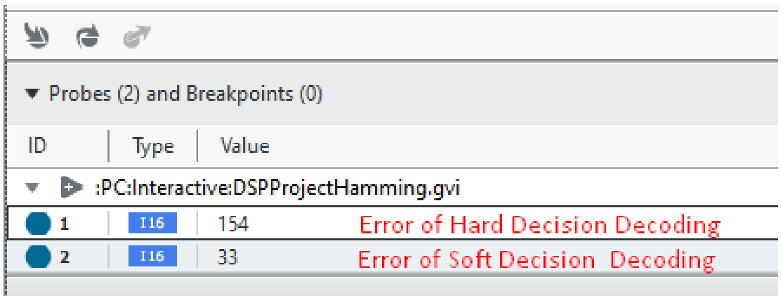

msg_cap2_2
Extracting 4-Bit massage from 7 bit data(with)
Storing 7-bit Data(with parity bits) as matrix
B
newMAT
msg_cap
msg_cap2_3
1

```
counter = 1;
for i=1:1000
indx1 = 7*(i-1)+1;
indx2 = 7*(i-1)+4;
newMAT(counter:counter+3)= B(indx1:indx2);
counter = counter +4;
end
```

Figure 23: Extracting 4-bit data from data which contains parity

In this part, 7-bit data (with parity bits) is saved into a new matrix with the use of a "feedback node" and "insert into array" components from LabVIEW. After that, 4-bit message data is extracted from 7-bit data for this case while doing that process with N=1000 iterations.

Comparison of the number of errors:

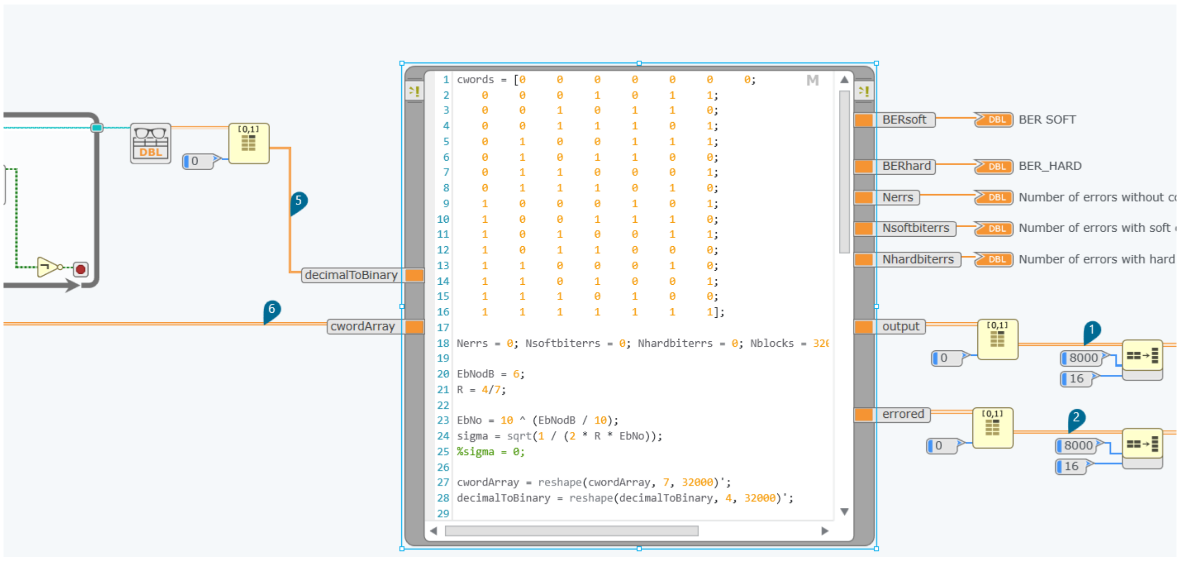

Probes (2) and Breakpoints (0)

ID	Type	Value	
:PC:Interactive:DSPProjectHamming.gvi			
1	I16	154	Error of Hard Decision Decoding
2	I16	33	Error of Soft Decision Decoding

Figure 24: The number of errors

The 2nd probe was put on the output of Hard Decision Decoding and Soft Decision Decoding to compare the number of bits after Hard Decision Decoding and Soft Decision Decoding. As it can be seen from the figures, soft-decision decoding is better than hard decision decoding for random noise between -2 and 2.

Implementation of Hamming Code



```
cwords = [0     0     0     0     0     0     0;
    0     0     0     1     0     1     1;
    0     0     1     0     1     1     0;
    0     0     1     1     1     0     1;
    0     1     0     0     1     1     1;
    0     1     0     1     1     0     0;
    0     1     1     0     0     0     1;
    0     1     1     1     0     1     0;
    1     0     0     0     1     0     1;
    1     0     0     1     1     1     0;
    1     0     1     0     0     1     1;
    1     0     1     1     0     0     0;
    1     1     0     0     0     1     0;
    1     1     0     1     0     0     1;
    1     1     1     0     1     0     0;
    1     1     1     1     1     1     1];

Nerrs = 0; Nsoftbiterrs = 0; Nhardbiterrs = 0; Nblocks = 32

EbNodB = 6;
R = 4/7;

EbNo = 10 ^ (EbNodB / 10);
sigma = sqrt(1 / (2 * R * EbNo));
%sigma = 0;

cwordArray = reshape(cwordArray, 7, 32000)';
decimalToBinary = reshape(decimalToBinary, 4, 32000)';
```

Figure 25: Implementation of hamming code

There were two error checking algorithms used to resolve transmitted bits (7 bits message at a time) which were named as soft decision and hard decision making in the system. There were 16 codewords and the transmitted message matched with the closest distance codeword on the table. The distance means the bitwise checking of the specific bits by looking at the 16 codewords table. The performance of the hard decision coding was weaker than the soft decoding methodology because the soft decision coding is taking floating-point distances into account. Finally, as it was seen on the code, the bit error rates were calculated by taking ratios of both errored bits and the number of bits sent on the channel.

Errored Signal and Error Corrected Signal

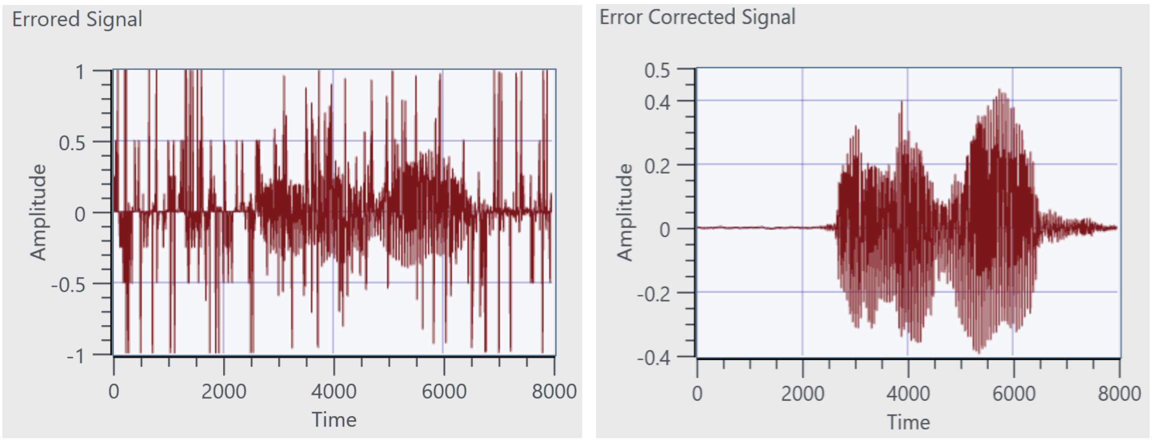
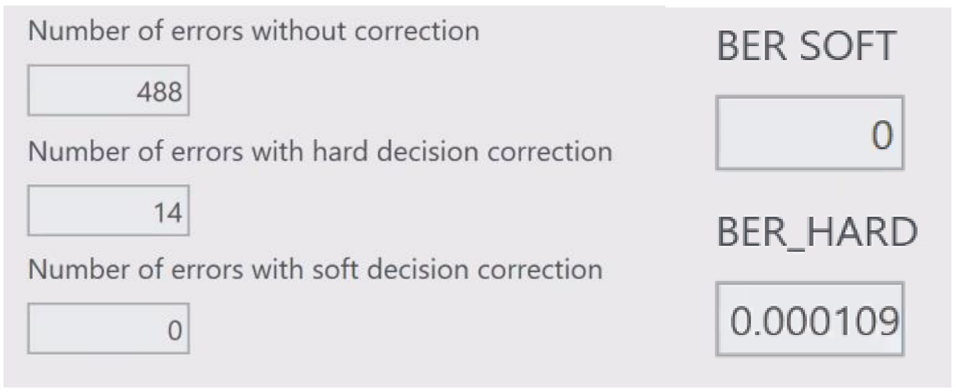


Figure 26: The errored signal and error corrected signal


Number of errors without correction

488

Number of errors with hard decision correction

14

Number of errors with soft decision correction

0

BER SOFT

0

BER_HARD

0.000109

Figure 27: The number or error's bits and BER for soft and hard decision decoding

Errored Signal and Error Corrected Signal (Increased Error Rate)

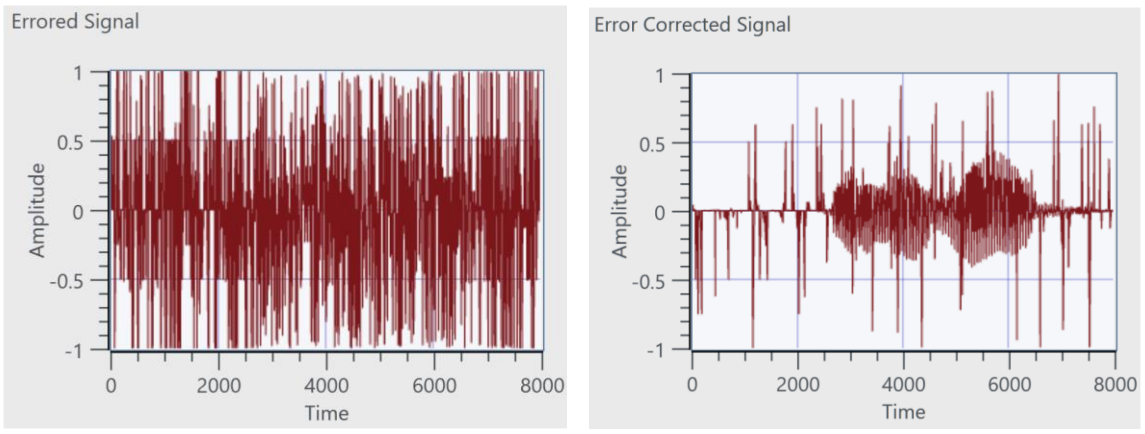
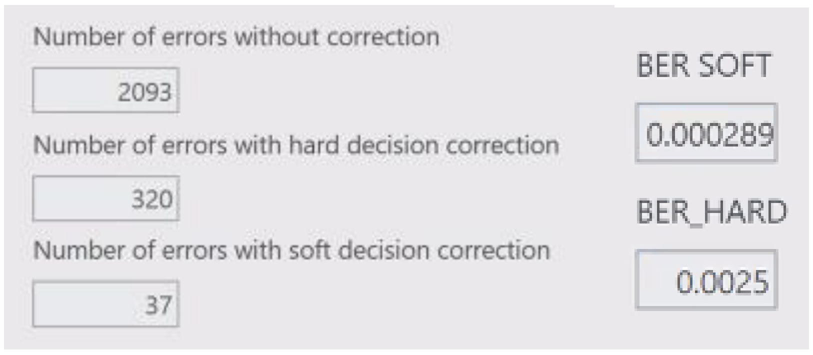


Figure 28: The errored signal and error corrected signal with increasing error rate


Number of errors without correction

2093

Number of errors with hard decision correction

320

Number of errors with soft decision correction

37

BER SOFT

0.000289

BER_HARD

0.0025

Figure 29:The number or error bits and BER for soft and hard decision decoding with increased error rate

The number of bits that were sent on the TX equals 128000 and it reaches 224000 bits after adding parity bits into the signal. There were two algorithms applied and they were hard decision making and soft decision making of the Hamming code. It was concluded that the soft decision methodology was performing better than the hard decision of the Hamming Code error correction. For instance, hard decision decoding was having 320 error bits whereas soft-decision decoding resulted in 37 error bits. However, if we did not use such an algorithm, there would be approximately 2093 errors in the transmitted signal. Thus, the algorithm avoids nearly 2050 error bits over 128000 transmitted bits.

The “Hello World” message was created on MATLAB, and it was converted into a digitized signal. After the conversion of the analogue signal, the parity bits were added with the ratio of 4/7 which means every 4 bits message has been controlled with extra parity bits. Hence, if AWGN noise was affected on the channel, it will be understood by checking those parity bits. Finally, bits were transformed on the channel to the receiver.

Moreover, it was seen that the power of the noise was so dependent on the performance of the error correction algorithm. For instance, when the EbNodB parameter was given as 7, there weren’t any errors captured during the transmission of the bits. It was an expected result because when noises of the environment are suppressed there will be less shifting during the transformation of the signal.

Adaptive Noise Cancelation

Figure 30: Adaptive noise cancellation on real life implementation

Adaptive noise cancelling (ANC) is a technique for removing additive sounds from corrupted signals that are particularly successful. It has a long history of application in telecommunications, radar, and sonar wave handling. Noise Cancellation is a form of optimum filtration that includes processing the baseline input to get a noise estimation and then removing this noise estimate from the primary input, which contains both message and noisy speech. It uses an extra or specific reference input that contains a correlated estimate of the disturbance that has to be cancelled. By installing one or even more sensors in the noise region where the message is missing or low enough, the baseline may be acquired. It would be feasible to make noise a near approximation of referenced noise by building constant filtering if the connection between n and n’ could be understood, or if the characteristics of the channels transferring distortion from the noise source to the main and reference inputs could be determined. Filtering and subtraction are regulated by an adaptive procedure since the properties of the propagation mechanisms are unknown and unexpected. As a result, adaptive filtering would be utilized, which may alter its impulse response to reduce an error signal that depends on the filter output.

Noise reduction may be achieved with adaptive control schemes with a low chance of signal distortion. The type of error signal to utilize is determined by the program. The minimizing of the mean square error, the temporal average of the least-squares error, and other parameters might be utilized. The Least Mean Squares (LMS) algorithm, the Recursive Least Squares (RLS) method, and other techniques could be employed according to the reduction requirements of a system. The algorithm that is utilized in our project was the Recursive Least Square method (RLS). As a result, the RLS method converges more quickly than the LMS approach.

Adaptive Filters

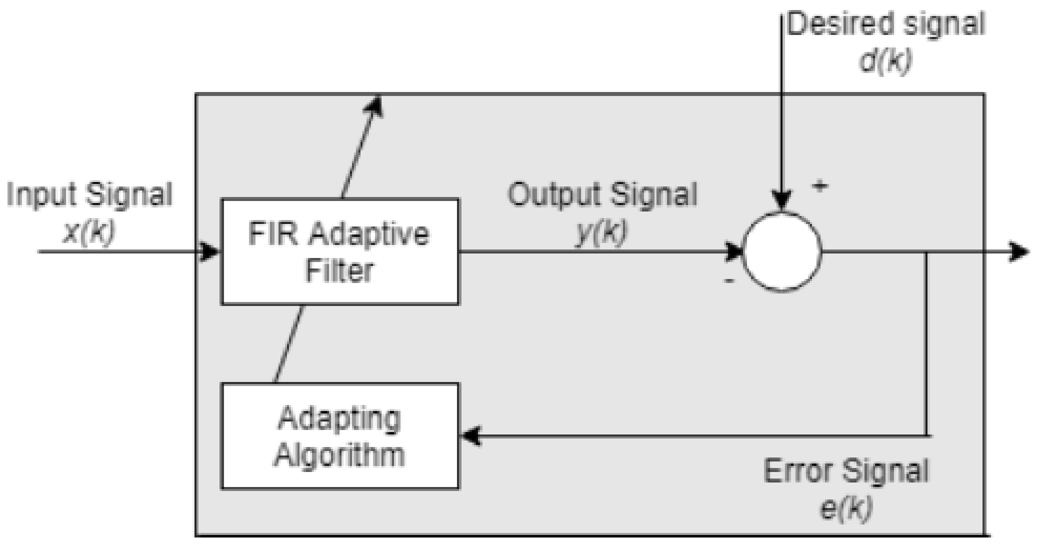


Figure 31: Adaptive filter schematic

When the input signal is non-stationary, an adaptive filter is a digital filter with auto-adjusting properties that set up instantly to changes in the signal. The adaptive filter with noise canceller is made up of two parts: a digital filter that could have changeable coefficients and an adaptive method for changing features of them. FIR filters are preferred over IIR filters because of their proven stability and simplicity.

Recursive Least Square

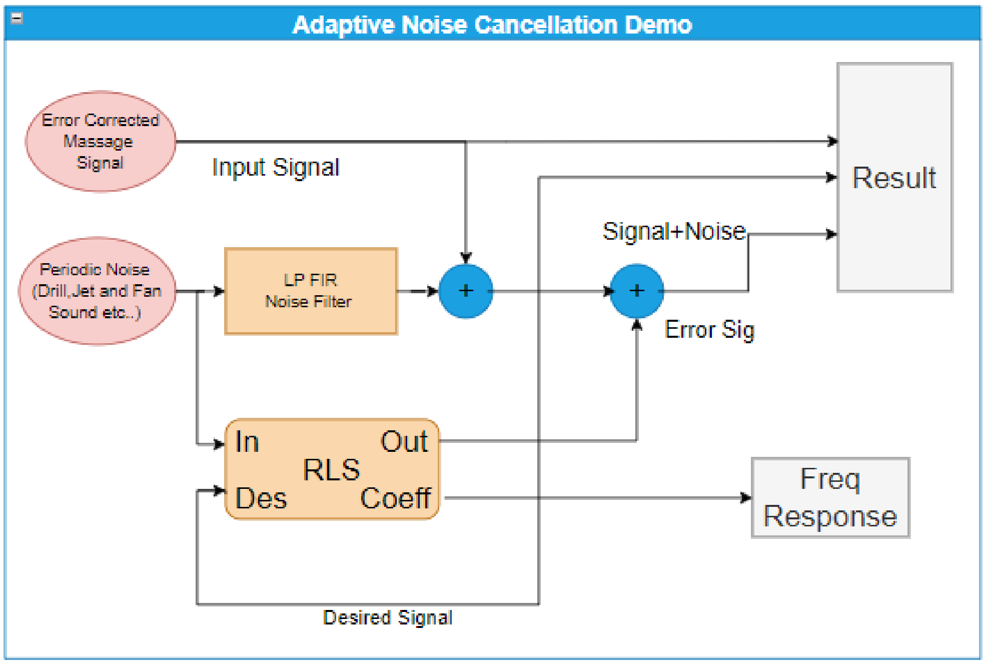

Adaptive Noise Cancellation Demo
Error Corrected Massage Signal
Input Signal
Periodic Noise (Drill,Jet and Fan Sound etc..)
LP FIR Noise Filter
+
+
Signal+Noise
Error Sig
Result
In
Out
RLS
Des
Coeff
Freq Response
Desired Signal

Figure 32:Adaptive noise cancellation demonstration

The recursive Least Square method is an iterative multiple regression approach in which the output signal is assessed at several points in time concerning the source signal, meanwhile, the input is supplied initially as.

Note: E represents an error signal in a signal.

$$Y_p = \sum_{i=0}^{Q-1} W(i) * X_p(i) + E\ _p \qquad (Equiation.1)$$

The RLS algorithm is using logic of a time averaging for each time.

The error equation could be as it follows.

$$zeta(i) = \sum_{i=0}^{n} lambda^{n-i} * |e(i)|^2 \qquad (Equation.2)$$

The Advantages of RLS Method Compared over LMS Technique

In a stationary environment, the Recursive Least Square method converges quicker than the Least Mean Square application, while in a non-stationary environment, the Least Mean Square method could be better than the Recursive Least Square method. The RLS algorithm makes use of previously accessible data. The step size, error signal, and tapping input vector all have a role in the correlations applied to the past approximation. However, there are two components in the RLS algorithm: estimating error and gain vector. The LMS method takes around 20 M iterations to converge in the mean square, but the RLS approach takes less than 2M cycles to reach in the mean square.

The MATLAB Implementation of Recursive Least Square Method

The code seen below is called by the LabVIEW and runs in the MATLAB. The function takes the selected noise's index from the RX panel and applies the RLS algorithm according to the noise input. Noisy speech is written to a text file called "noisySpeech.txt" by LabVIEW and the MATLAB function reads this file to get the received data. The function then writes the resulting output without the periodic noise to a text file. This text file called "rlsOutput.txt" then gets read by LabVIEW to get plotted and outputted as sound.

```matlab
function [] = rls(selectedNoiseIndex)

% Choosing the right noise reference file according to the input coming
% from LabVIEW.
switch selectedNoiseIndex
    case "0"
        selectedNoise = "drillSound.txt";
    case "1"
        selectedNoise = "fanNoise.txt";
    case "2"
        selectedNoise = "chainsawNoise.txt";
    otherwise
        selectedNoise = "drillSound.txt";
end

fs = 8000;

noisy_speech = load("noisySpeech.txt"); % Noisy signal data saved by LabVIEW
noise = load(selectedNoise); % Reference noise

Ts = 1/fs; % Sampling period
k = 1*fs-1; % Max sample for 1 second recording
t =(0:k)*Ts; % Time vector

M = 30; % Filter length

% If lamda is close to 1, there will be less fluctuations in the filter
% coefficients.
% When lamda is 1, we will have a growing window RLS algorithm

lambda = 1; % Forgetting factor

% Small positive constant. Smaller delta values result in lesser noise.
delta = 0.01;

lambdaInv = 1/lambda;
deltaInv = 1/delta;
```

```
N = length(noisy_speech);
w = zeros(M, 1); % Filter coefficients initialized
P = deltaInv*eye(M); % Inverse correlation matrix
output = zeros(N,1);

% The for loop below is to adapt the filter coefficients according to the
% noise reference and apply this filter to the input signal.

for i=M:N

    % Getting the M length noise reference
    y = noise(i:-1:i - M + 1);
    % Estimated output signal is subtracted from the noisy speech.
    output(i) = noisy_speech(i) - w' * y;
    % Filter gain is determined
    k = (P * y) / (lambda + y' * P * y);
    % Inverse correlation matrix is updated.
    P = (P - k * y' * P) * lambdaInv;
    % Updating filter coefficents.
    w = w + k * output(i);

end

output = normalize(output,'range',[-1 1]);
% Writing the output to a text file for the LabVIEW to read the file later.
writematrix(output,"rlsOutput.txt");

end
```

The speech signals were obtained on MATLAB with the sound of one of the group members. The drill noise, fan noise, saw, and jet engine noise was reduced. The adaptive noise filter assumes two inputs which are noisy speech and reference input that should be highly correlated with noise. The RLS algorithm uses adaptive weight to generate an output that is close to noise by minimizing error in the signal.

The length of a filter was chosen as 7. The forgetting factor that represents lambda in our code was defined as 1. If lambda is close to one, fewer fluctuations on the filter coefficients will occur (lambda = 1 called as growing window RLS algorithm). There is a parameter that is named delta and has a value of 5(small positive constant). Firstly, the reference input is obtained from the for a loop. Then, an error signal which denoted as a denoised signal was calculated and the filter gain coefficient vector was updated. The filter coefficient was saved on the matrix of w1. Finally, the denoised speech was observed and played on the LabVIEW.

LabVIEW Simulation

Drill Noise:

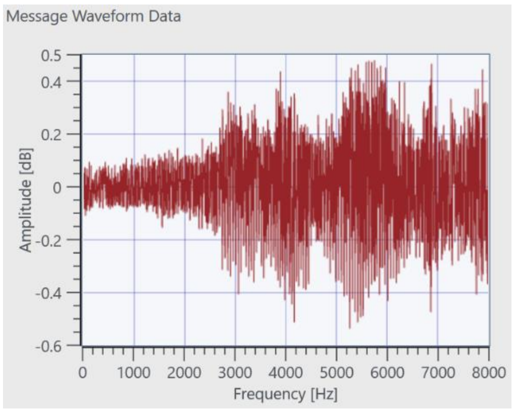


Figure 33: Drill noise added massage signal

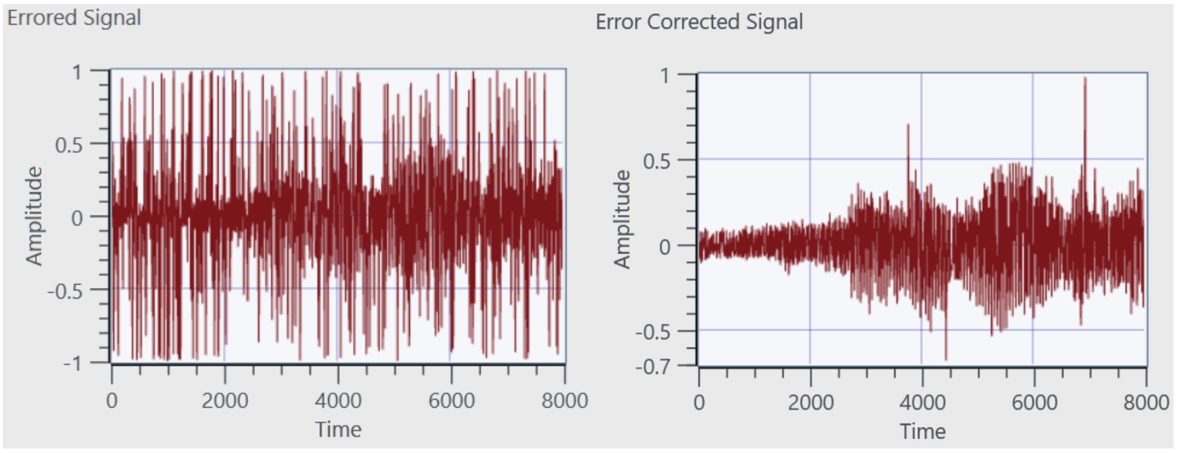


Figure 34: Errored and error corrected signal with Drill noise

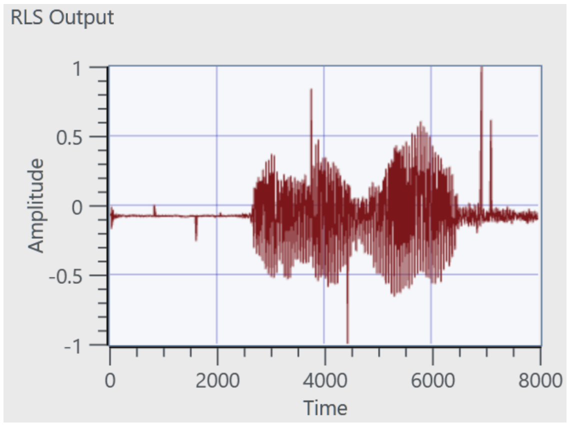


Figure 35: RLS algorithm applied signal with Drill noise

Chainsaw Noise:

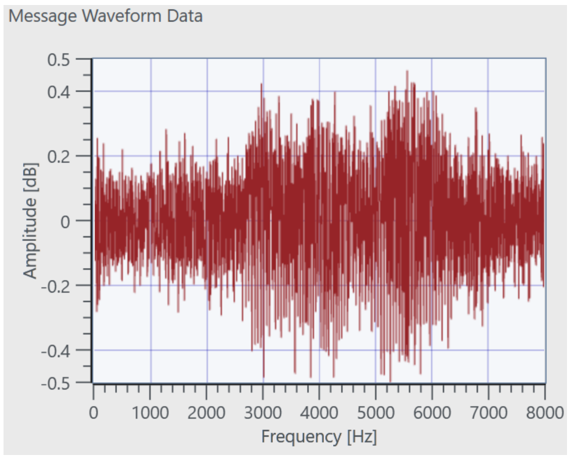


Figure 36: Massage waveform with Chainsaw noise

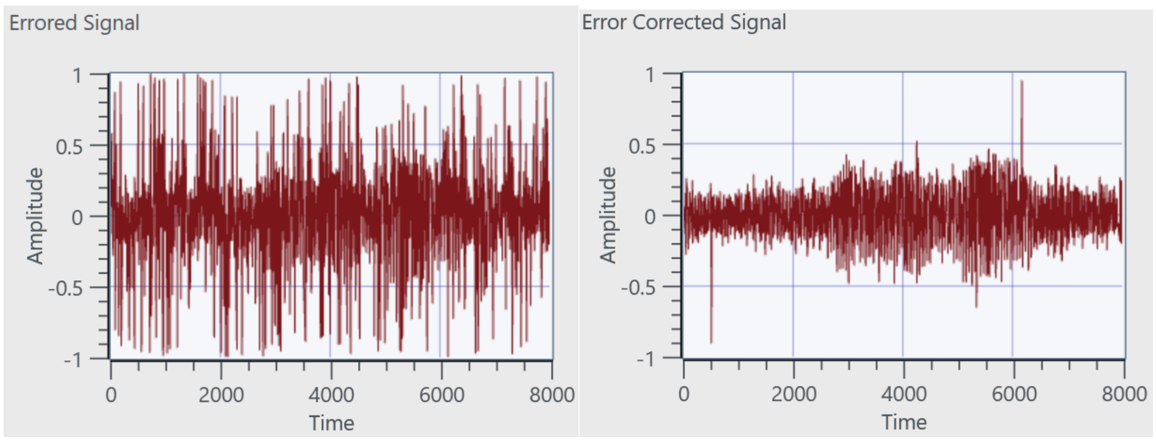


Figure 37: Errored and error corrected signal with Chainsaw noise

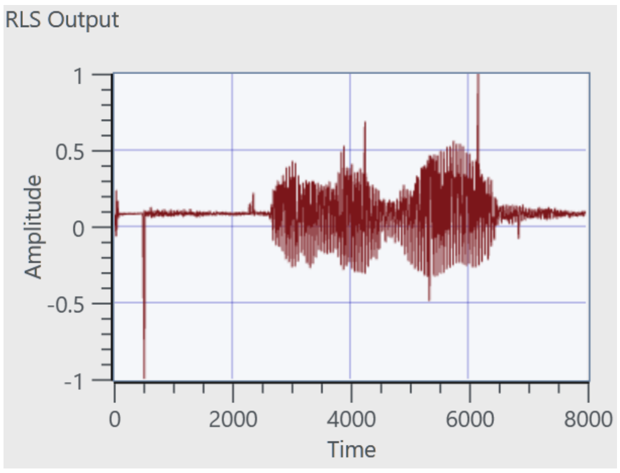


Figure 38: RLS algorithm applied signal with Chainsaw noise

Fan Noise:

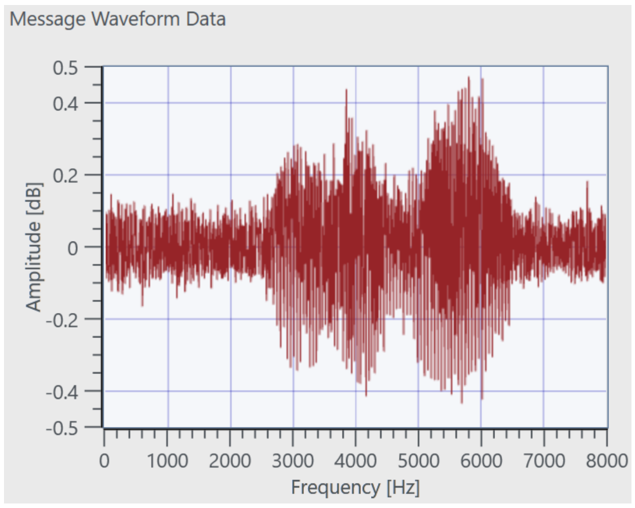


Figure 39:: Massage waveform with Fan noise

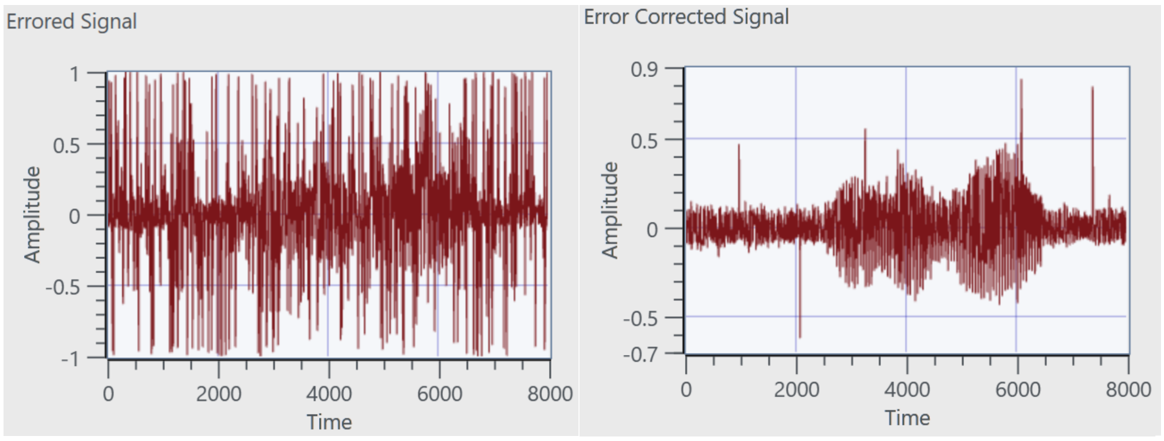


Figure 40:Errored and error corrected signal with Fan noise

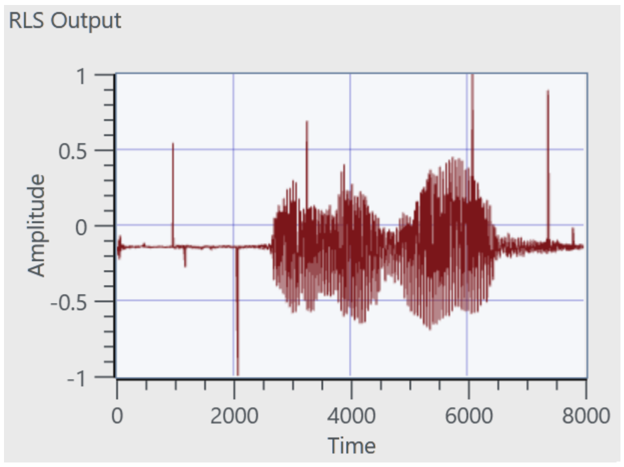


Figure 41:RLS algorithm applied signal with Fan noise

As it can be seen from output graphs of the RLS algorithm, unwanted periodical noises (drill, saw, fan, motor engine) were almost completely cleared out from the original speech signal. Hence, the RLS algorithm is working accurately. The promise of removing periodic noises from audio signals was provided thanks to the RLS adaptive noise cancellation method.

Moreover, hamming code implementation results can be checked by looking *Figure.34*, *Figure.37*, and *Figure.40*. Errored signals are having frequent noise distortions in many bits and error corrected signals was result of performing soft-decision decoding in the hamming code. It is significantly better than without error correction algorithm. Hence, the errors related to AWGN noise can be decreased by using error correction algorithms.

Conclusion

The FSK modulation and demodulation techniques were utilized in order to transmit messages over a channel. In summary, the aim of the project was successfully achieved by implementing both an adaptive noise cancellation algorithm and an error correction algorithm that is named Hamming code. In the part of the hamming code, the number of error bits was significantly lower after hard and soft decision algorithms worked and the resulting signal had way fewer errors than the originally received signal. Additionally, soft-decision decoding has been showing better results in terms of error correction bits. On the other hand, the adaptive noise cancellation algorithm was performed by using the Recursive Least Square method. However, there are many noise cancellation techniques but they are not as successful as LMS. Finally, both digital signal processing algorithms integrated and performed in the LabVIEW.

References

[1] *Error Correction methodology*. (2019, May 6). NPTEL-NOC IITM. Retrieved January 7, 2022, from **https://www.youtube.com/watch?v=UkdyGhle_Vc&t=1362s**

[2] *Hamming Code Error Correction*. (2021, November 17). Guru99. Retrieved January 17, 2022, from https://www.guru99.com/hamming-code-error-correction-example.html

[3] *Soft Decision Decoding*. (2019, January 12). Wikipedia. Retrieved January 14, 2022, from https://en.wikipedia.org/wiki/Soft-decision_decoder

[4] *Hard and Soft Decision Decoding*. (2019, December 1). Tutorialspoint. Retrieved January 14, 2022, from https://www.tutorialspoint.com/hard-and-soft-decision-decodinghttps://www.tutorialspoint.com/hard-and-soft-decision-decoding

[5] *(7,4) Hamming Code in MATLAB*. (2019, May 6). NPTEL-NOC IITM. Retrieved January 7, 2022, from https://www.youtube.com/watch?v=UkdyGhle_Vc