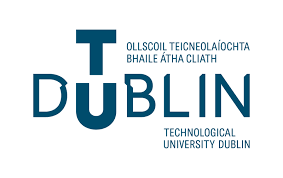
******

**SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING**

**Bachelor of Engineering (Hons) BE in Electrical & Electronic / Comp & Comm Eng**

**Programme Code: (DT021A)**

**<YEAR 4>**

**Name of Module (Digital Communications 2, COMM3602)**

**Name of Lecturer (Kelly, Anthony T)**

Labs Formal Report

**Student Name** \_Sajjad Ullah, C17344483\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_\_\_\_\_**

**Title \_**Digital Communications 2 Lab Formal Report-2022\_\_\_\_

**DECLARATION**

I hereby certify that the material, which is submitted in this assignment, is entirely my own work and has not been submitted for any academic assessment other than as part fulfilment of the assessment procedures.

Signature of student: ……Sajjad ullah……………….

Date: ……18/12/21…………………………

Contents

[Introduction 5](#_Toc90563690)

[Lab1 Speech Scrambling 6](#_Toc90563691)

[Background theory 6](#_Toc90563692)

[Procedure 6](#_Toc90563693)

[Comments 9](#_Toc90563694)

[Lab2 Baseband Data receiver 10](#_Toc90563695)

[Background theory 10](#_Toc90563696)

[Procedure 10](#_Toc90563697)

[Comments 15](#_Toc90563698)

[Lab3 M-ary coding 16](#_Toc90563699)

[Background theory 16](#_Toc90563700)

[Procedure 16](#_Toc90563701)

[Comments 21](#_Toc90563702)

[Lab4 BER analysis 22](#_Toc90563703)

[Background theory 22](#_Toc90563704)

[Procedure 22](#_Toc90563705)

[Comments 24](#_Toc90563706)

[Lab5 Entropy encoding 25](#_Toc90563707)

[Background theory 25](#_Toc90563708)

[Procedure 25](#_Toc90563709)

[Task 28](#_Toc90563710)

[Comments 28](#_Toc90563711)

[References 29](#_Toc90563712)

**Table of figures**

[Figure 1 Python code for plotting a spectrum of multiple tones 6](#_Toc90563713)

[Figure 2 Spectrum of multiple tones plot 7](#_Toc90563714)

[Figure 3 Frequency inversion scrambler block view 7](#_Toc90563715)

[Figure 4 A signal x(t) with 6 samples 7](#_Toc90563716)

[Figure 5 Invert every second sample to creating new signal 8](#_Toc90563717)

[Figure 6 frequency of new ‘chopped signal’ 8](#_Toc90563718)

[Figure 7 Frequency inversion scrambler filter method 8](#_Toc90563719)

[Figure 8 Python Code to invert signal 9](#_Toc90563720)

[Figure 9 Inverted spectrum of three tones plotted 9](#_Toc90563721)

[Figure 10 8-bit PRBS with channel filtering and noise corruption components 10](#_Toc90563722)

[Figure 11 NRZ, CH & nCH plots 11](#_Toc90563723)

[Figure 12 comparator connected to nCH 11](#_Toc90563724)

[Figure 13 nCH connected to comparator 12](#_Toc90563725)

[Figure 14 hysteresis comparator with switching levels of 3V & 0.5v 14](#_Toc90563726)

[Figure 15 Hysteresis voltage characteristics plot 14](#_Toc90563727)

[Figure 16 Comparing hysteresis circuit to comparator 15](#_Toc90563728)

[Figure 17 PRBS generator & a clock divide by 2 circuit. 16](#_Toc90563729)

[Figure 18 Bit Splitter 17](#_Toc90563730)

[Figure 19 D-A Converter 17](#_Toc90563731)

[Figure 20 Sweep generator 18](#_Toc90563732)

[Figure 21 Mary and filtered Mary plots 18](#_Toc90563733)

[Figure 22 Binary M-ary plot 19](#_Toc90563734)

[Figure 23 4 level M-ary eye Diagram 20](#_Toc90563735)

[Figure 24 Noise on the M-ary signal 20](#_Toc90563736)

[Figure 25 Baseband Data System 22](#_Toc90563737)

[Figure 26 AWGN input 23](#_Toc90563738)

[Figure 27 AWGN output 23](#_Toc90563739)

[Figure 28 error rate calculation results 24](#_Toc90563740)

[Figure 29 Mathlab code for Probability Error Calculation 24](#_Toc90563741)

[Figure 30 Python code to display random N symbols with NS levels 25](#_Toc90563742)

[Figure 31 Symbol distribution of 128 with 8 possible levels 26](#_Toc90563743)

[Figure 32 Code for probability of each symbol occurrence plot 26](#_Toc90563744)

[Figure 33 Symbol probability 0-7 27](#_Toc90563745)

[Figure 34 code and result for Average Level, Entropy & Average Power 27](#_Toc90563746)

# Introduction

This is a report based on multiple laboratory work conducted throughout the semester. It will cover 5 labs.

**Lab 1: speech scrambling**

Here we will provide spectral plots of the inputs and outputs from the frequecny inversion scrambler and discuss the code used in a high level.

Software used here include – Spyder IDE

**Lab 2: Baseband Data Receiver**

Here we provide all output signals. Illustrate if the use of the hysteresis circuit improves the receiver’s performance and show the transfer characteristic of the hysteresis circuit and provide any design equations used.

Software used here include – Pspice

**Lab 3: M-ary Coding**

For this lab, plots will be provided of the filtered M-ary signal and the eye diagram also the effect of noise on signals. Lastly, we will compare the binary against 4-ary cases and discuss why M-ary coding is used in practice.

Software used here include – Pspice

**Lab 4: BER Analysis**

Schematic diagrams will be included for this lab and outputs of the AWGN channel. After we will show how the BER value obtained from simulation relates to the calculation of the error probability.

Software used here include – Simulink & mathlab

**Lab 5: Entropy Coding**

Plots of symbol probabilities for all cases and we will solve the following example question:

what the entropy of a data source with 5 symbols with probabilities of is symbols 1 to 4 defined respectively as:

P1=0.12, P2=0.22, P3=0.36, P4=0.05 and P(1/1)= P(4/1)=0.5, P(1/3)=0.6, P(5/3)=0.4.

Software used here include – Spyder IDE

# Lab1 Speech Scrambling

The task of this lab was to implement a simple frequency inversion scrambler based on the DSP technique in the python programming language.

## Background theory

Speech inversion scrambling is an analogue method of concealing the content of a transmission, the technique renders the speech nearly unintelligible in ordinary radio receivers without a descrambler which reapplies the inversion thus undoing the original scrambler. It works by inverting the frequency spectrum of the signal, making the lowest frequencies the highest and vice versa[1].

In the lab we are implementing a digital inversion scrambler using python.

## Procedure

To begin we first created 3 tones, using 256 samples and a sampling rate of 8 kHz.

Table Signal characteristics of three tones

|  |  |  |  |
| --- | --- | --- | --- |
|  | Tone 1 | Tone 2 | Tone 3 |
| Amplitude | 3v | 2v | 1v |
| Frequency | 500 Hz | 1500 Hz | 3000 Hz |
| Phase | 0 | 0 | 0 |

Using the library *numpy* which has a large collection of mathematical functions and the *scipy fftpack* to compute fast Fourier transforms we can create and plot the spectrum of the three tones.

Text

Description automatically generated

Figure Python code for plotting a spectrum of multiple tones

The resulting plot in figure 2 verified that the signal has indeed been created with the intended amplitudes and frequency.

Chart, histogram

Description automatically generated

Figure Spectrum of multiple tones plot

A frequency inversion scrambler works by passing the input signal into a sampler which will sample at the sampling frequency typically determined by the nyquist rate and then inverts every second sample.

Diagram, schematic

Description automatically generated

Figure Frequency inversion scrambler block view

For example, take a signal x(t) with 6 samples as shown in figure 4.

Chart, line chart

Description automatically generated

Figure A signal x(t) with 6 samples

If we take every second signal and invert with respect to the time axis, then use the new sample positions to create the signal shown in green calling it the ‘chopped signal’

Chart

Description automatically generated

Figure Invert every second sample to creating new signal

To get the frequency of this new signal ( carrier signal) , plot the new samples between amplitude values of +1 and -1. When measured this would have a frequency that is half the sampling rate.

Chart, line chart

Description automatically generated

Figure frequency of new ‘chopped signal’

It is equivalent to multiplying the input signal by the carrier frequency. This produces a Double Side Band (DSB) spectrum cantered on the carrier frequency. To get the scrambled signal we pass this through a low pass filter which outputs the scrambled signal.

Diagram, schematic

Description automatically generated

Figure Frequency inversion scrambler filter method

Now that we know how it works, we can program it as shown below

Text

Description automatically generated

Figure Python Code to invert signal

Chart, histogram

Description automatically generated

Figure Inverted spectrum of three tones plotted

We can verify that the plot is correct, we used 8,000 as the sampling frequency which means that the carrier frequency would be 4,000 Hz. Due to passing the signal through a low pass filter, the double side band it produces will result in any frequency components higher than 4,000 Hz would be removed leaving the lower frequencies which are :

4,000 Hz – 500 Hz = 3,500Hz

4,000 Hz – 1,500 Hz = 2,500Hz

4,000 Hz – 3,000 Hz = 1,000Hz

## Comments

The digital frequency inversion scrambler worked as intended, as verified by the spectral plots the frequencies were indeed inverted. Future work in this lab could involve a speech file and running the program on it, then listening to the scrambled output.

# Lab2 Baseband Data receiver

The objective of this lab was to examine the characteristics of a baseband data receiver and to investigate the use of hysteresis to improve the receivers noise immunity.

## Background theory

A Radio Frequency (RF) receivers’ job is to listen for the appropriate signals from a transmitter, but this is made difficult due to background noise that get added onto the signal. Hysteresis is used to provide resilience to noise or weak signals by defining signal levels in the design; if the signal falls below this level it treated as noise and the output will stay at the last valid level. This is to prevent the receiver amplifying noise and producing invalid signal levels on the output.

## Procedure

The first step was to build an 8-bit shift register configured to give a pseudo-random binary sequence (PRBS) using the 74164 components in Pspice.

The channel filtering is done by the R2C2 components. The corrupted noise signal will be represented by the data signal ‘eight.dat’.



Figure 8-bit PRBS with channel filtering and noise corruption components

To ensure if this the circuit works as intended, we have placed markers at points **NRZ** (non-return to zero) signal, **CH** (channel filtering) and **nCH** ( the output of the NRZ with the noise amplified by a gain block of 3 )



Figure NRZ, CH & nCH plots

From figure 11 we observe that the NCH output has indeed worked with the noise added onto the NRZ signal, this signal would be difficult to distinguish the binary ones and zeros of the original nrz signal, but we shall attempt this with a comparator connected to the output, like the one shown in figure 12 with switching level of 1.75 volts.



Figure comparator connected to nCH

We can observe in figure 13 that the comparator does a good job most of the time at recovering the intended signal but taking a closer look at the 6ms range in the COMP plot, it has produced a spurious pulse due to noise interference which dropped the voltage level at that point in time and triggered the comparator, corrupting our signal.



Figure nCH connected to comparator

This spurious behaviour of an open-loop comparator can be overcome by adding positive feedback between the output and input of the comparator. With positive feedback, the circuit has hysteresis with the output switching occurring between two different switching points

Now we will design a hysteresis comparator with voltage switching levels of 0.5 and 3v to improve the performance of the system.

Table Hysteresis calculations

|  |
| --- |
| **hysteresis equations:**  Comparator +VCC = 5v, let’s call this ‘A’  Comparator -VCC = 0v  let the 0.5v switching level be called ‘VB’  let the 3v switching level be called ‘VA’  The hysteresis voltage (VH) can be found by VA – VB = 2.5v  use the following equations to get the voltage reference value to use in the hysteresis circuit. |
| **finding resistor values:**  The hysteresis voltage (VH) can be found by VA – VB = 2.5v:  VH = VA - VB  =  *,* where A = 5v & VH = 2.5v  2.5 v = 5*,*  0.5 *= , meaning R1 value needs to be x2 the size of R2*  There pick R1 = 100K & R2 = 200K, because they are common resistor values. |
| **Calculation of Vref using VA equation** |

The calculations in table 2 tell use to use a voltage reference of 2v using the component values R1 = 100K ohms & R2 = 200k ohms, to get the desired switching levels.

Connecting the hysteresis circuit shown in figure 14 to the nCH signal and checking the plot produced.



Figure hysteresis comparator with switching levels of 3V & 0.5v

To check if it has been designed as intended at the correct switching levels, we can plot the hysteresis voltage characteristics as shown in figure 15 and examine that the switching voltages are indeed at 0.5v & 3v.



Figure Hysteresis voltage characteristics plot

In figure 16 we have plotted the signals NRZ, COMP from the original comparator and COMP\_HYS\_5v from the hysteresis circuit to compare the improvement. The hysteresis plot does not show the spurious pulse seen in the COMP plot at around the 6ms range and is therefore more resilient to noise with switch levels at 0.5 and 3v.



Figure Comparing hysteresis circuit to comparator

## Comments

The hysteresis comparator was a clear improvement over the standard comparator shown in figure 12, the data picked up at the receiver end will now have a certain degree of resilience from noise/interference picked up during transmission.

# Lab3 M-ary coding

The objective of the M-ary lab was to examine the characteristics of a Mary encoder & bit splitter and to produce the ‘eye’ diagram. Lastly to investigate the eye diagram of a real filtered data signal

## Background theory

M-ary modulation uses multi-level modulation to transmit N bits at a time, its main advantage is the reduced channel bandwidth because of M-ary encoding. The M in M-ary corresponds to the number of levels, conditions, or combinations possible for N-bits[2].

M-ary modulation techniques, typically modulate one parameter of the carrier signal such as such as amplitude, phase, and frequency[2]:

M-ary Amplitude Shift Keying (ASK)

M-ary Frequency Shift Keying (FSK)

M-ary Phase Shift Keying (PSK)

## Procedure

To start off we set up a circuit which consisted of a pseudo random binary sequence (PRBS) and a clock divide by 2 circuit in the configuration shown in figure 17 and labelled the wires for the type of signal passing through.



Figure PRBS generator & a clock divide by 2 circuit.

Simulation was done over 100ms with a step ceiling of 20 us.

Up next was the bit splitter which separates the I and Q signals.



Figure Bit Splitter

Last to be built was the D-A converter with components shown in figure 19.



Figure D-A Converter

To set up the channel filtering and sweep generator, the ‘lopass & Vpulse’ block were used as shown in figure 20.



Figure Sweep generator

We can attach voltage markers on the labels MARY\_BIPOLAR and CH\_OP to view the effect of the lowpass block on the m-ary signal.



Figure Mary and filtered Mary plots

From figure 21 we can tell this is a 4 level M-ary system from the from counting the unique number of ‘steps’ therefore we can encode 2 bits.

To encode binary one bit, we can modify the D-A circuit and ground either ‘I’ or ‘Q’ which will result in only one binary going into the system.



Figure Binary M-ary plot

With binary we can encode only two levels, we see the M-ary Bipolar signal switching between two levels now in figure 22.

**Why is M-ary coding used:** M-ary coding is a modulation technique used for data transmission in which instead of sending N bits at a time, a single signal is used for multi bit transmission. This has the effect of reducing the channel bandwidth. For example, in 4 M-ary each signal represents 2 bits at a time[6].

If we want a digital signal to have **M levels** which can represent voltage levels, frequencies, phases etc then the mathematical expression to calculate the number of bits necessary to produce these levels is given by[6]:

To obtain the ‘eye’ diagram we produced an X-Y plot of the channel output against the sweep time-base signal, generating the plot in figure 23.

The less overlap there is between the levels, the more we can easily detect the individual levels meaning there is enough spacing between the levels that no interference between levels can occur.



Figure 4 level M-ary eye Diagram

In figure 24 we have added noise from the ‘eight.dat’ file and amplified it with a gain of 2, we can observe that the resulting signal ‘NCH\_OP\_GAIN\_2’ can still be seen to have distinct levels even with the noise corruption.



Figure Noise on the M-ary signal

## Comments

The M-ary system we designed in the lab shows how the system functions in block sections, from the bit splitter to the D-A converter it may be seem like a more complex modulation system when compared to easier methods such as Amplitude Modulation (AM) but the bandwidth efficiency derived from the use of m-ary modulation makes it an attractive modulation scheme.

From the work done in the lab we can envision that higher M-ary levels of modulation would be ideal for higher bit transmission, but this would become difficult and require a costly system to discern the levels that are very close together especially when noise effects the end signal like seen in figure 24.

# Lab4 BER analysis

The objective of the Bit Error Rate (BER) analysis lab was to simulate the operation of a baseband data system in the presence of white gaussian noise and to determine the bit error rate for a given signal-to-noise ratio.

## Background theory

BER analysis is a key performance metric of digital transmission systems. It is the number of received bits of a data stream over a communication channel that have been altered due to noise, interference, or distortion[3].

The bit error ratio is the number of bit errors divided by the total number of transferred bits during a studied time interval. Bit error ratio is a unitless performance measure, often expressed as a percentage[3].

## Procedure

To start off we built and simulated the baseband data system shown in figure 25.

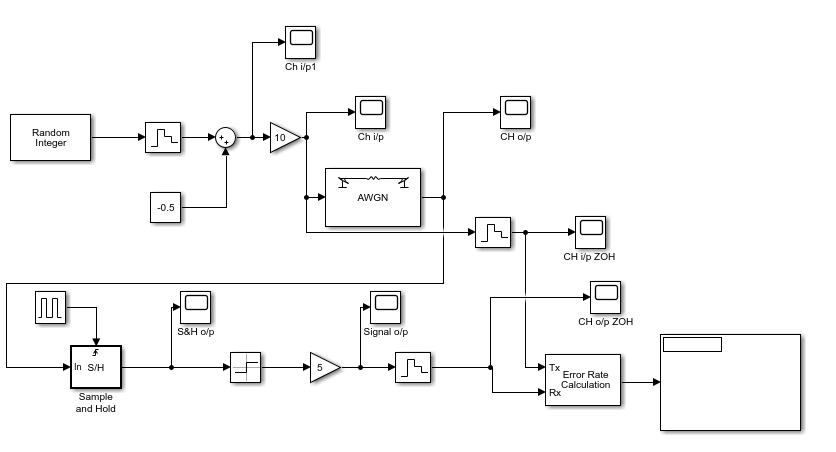


Figure Baseband Data System

The simulation parameters: stop time to 20e-3 sec.

In table 3 are the parameters for the various blocks we have set.

Table Block name and parameters set

|  |  |
| --- | --- |
| **Block name** | **Parameters set** |
| Random Integer | M-ary number: 2  Initial seed: 8  Sample Time: 1e-3 |
| Zero Hold Order 1 | Sample Time: 1e-5 |
| AWGN channel | initial seed: 100  variance: 32 |
| Zero Hold Order 2 | Sample Time: 1e-3 |
| Pulse Generator | Pulse type: time based  Time: simulation time  Amplitude: 1 | Period: 1e-3  Pulse Width: 10 | Phase Delay: 0.5e-3  Click Box: interpret vector as 1-D |
| Error Rate Calculation | Receive delay: 1  Computation delay: 0  Computation mode: Entire frame |

Chart, histogram

Description automatically generated

Figure AWGN input

The AWGN block adds white gaussian noise, the input to the block is the signal in figure 26 while the output is figure 27.

Chart, line chart

Description automatically generated

Figure AWGN output

After running the simulation, the error rate calculation displays the following values where if we read from the top, the first line gives the error rate percentage, the second line represents the number of errors while the last line is the quantity the error was calculated from.

Table

Description automatically generated

Figure error rate calculation results

The equation for probability error is:

Text

Description automatically generated

Using this equation, we can compute the value of PE and compare it against the simulation results.

From figure 26 the peak signal level measured is 5, therefore the peak-to-peak value is 10. Using this value in mathlab code to compute the PE value.

Graphical user interface, text, application

Description automatically generated

Figure Mathlab code for Probability Error Calculation

The calculated PE value of 0.1884 is almost an exact match to the simulation value of 0.1852 with only a 0.4338% difference between observed and calculated values.

Due to a variance of 32, there is a lot of noise on the signal which results in 18.52% of the symbols are incorrect.

## Comments

In this lab insight was gained into the operation of a baseband data system in the presence of white gaussian noise.

We were able to determine the bit error rate for a given signal-to-noise ratio.

The simulated values and calculated values from mathlab were almost an exact match, this match can be seen as verifying the work done in Simulink was correct.

# Lab5 Entropy encoding

The objective of the entropy encoding lab using python was to use the python simulation environment to obtain the average level and the entropy of a set of data symbols.

## Background theory

Entropy encoding is a lossless data compression scheme used to compress digital data by representing frequently occurring patterns with few bits and rarely occurring patterns with many bits[4].

It involves assigning codes to symbols to match code lengths with the probabilities of the symbols. Two of the most common entropy encoding techniques are Huffman encoding and arithmetic encoding[5].

## Procedure

To start we used the following python code to generate 8 symbols ( level 1 – 7) and display 128 random symbols in a symbol distribution plot.

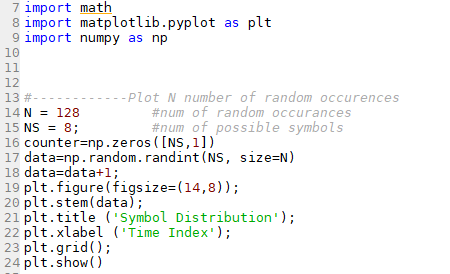


Figure Python code to display random N symbols with NS levels

This will generate the plot shown in figure 31 of 128 random occurrences.

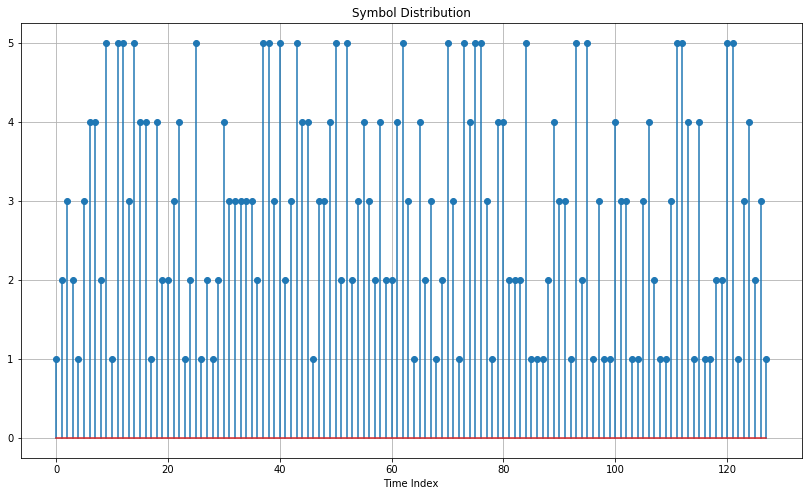


Figure Symbol distribution of 128 with 8 possible levels

To code in figure 32 counts the number of times a symbol/voltage level occurs. The counter has been initialised using counter = np.zeros([NS,1]). Thus counter(X) contains the number of times that symbol X occurs.

Text

Description automatically generated

Figure Code for probability of each symbol occurrence plot

Chart, line chart

Description automatically generated

Figure Symbol probability 0-7

To calculate the average level, entropy and the average power we use the following code shown in figure 34.

Text

Description automatically generatedText

Description automatically generated

Figure code and result for Average Level, Entropy & Average Power

### Task

What is the entropy of a data source with 5 symbols with probabilities of is symbols 1 to 4 defined:

P1=0.12, P2=0.22, P3=0.36, P4=0.05 and P(1/1)= P(4/1)=0.5, P(1/3)=0.6, P(5/3)=0.4

To normally encode 5 symbols, we would need at least 3 bits.

The symbol P5 is = [ 1- (P1+P2+P3+P4) ] = 0.25

We can calculate the entropy H as

Check : Entropy 2.09 < 3 bits, true

## Comments

In this lab we have created python code to plot the number of possible symbols with the number of occurrences into a symbol distribution plot. This was then used to obtain the average level and the entropy of a set of data symbols.

# References

[1]O. Räisänen, "Descrambling the voice inversion scrambler", *absorptions*, 2013. [Online]. Available: https://www.windytan.com/2013/05/descrambling-voice-inversion.html. [Accessed: 12- Dec- 2021].

[2]"Digital Communication - M-ary Encoding", *Tutorialspoint.com*. [Online]. Available: https://www.tutorialspoint.com/digital\_communication/digital\_communication\_m\_ary\_encoding.htm. [Accessed: 12- Dec- 2021].

[3]J. LIM, "Is BER the bit error ratio or the bit error rate?", *EDN*, 2010. [Online]. Available: https://www.edn.com/is-ber-the-bit-error-ratio-or-the-bit-error-rate/. [Accessed: 12- Dec- 2021].

[4]"What is Entropy Coding? | Webopedia", *Webopedia*, 2007. [Online]. Available: https://www.webopedia.com/definitions/entropy-coding/. [Accessed: 12- Dec- 2021].

[5]"entropy encoding", *Planetmath.org*, 2013. [Online]. Available: https://planetmath.org/entropyencoding. [Accessed: 12- Dec- 2021].

[6]"Digital Communication - M-ary Encoding", *Tutorialspoint.com*. [Online]. Available: https://www.tutorialspoint.com/digital\_communication/digital\_communication\_m\_ary\_encoding.htm. [Accessed: 16- Dec- 2021].