# Abstract

This report details my research and work done when producing a web-based baseband communication teaching simulation. The visualisation simulates both the transmitter and receiver of a baseband signal; each stage of this simulation is displayed graphically to the student, showing how the data is transformed by each stage in the process.

The transmitter generates a binary signal. This signal is then encoded using a line coding scheme. The high-frequency content of this signal is filtered out using a low-pass filter. Finally, the transmitter displays this signal ready for transmission in the time domain to the students.

The simulation then adds gaussian noise to the signal, this simulates the thermal noise that would be introduced when a signal travels down a transmission line. The receiver then demodulates the signal, by sampling it at a set interval and then decodes the signal.

Students are able to interact with the transmitter simulation by altering its parameters. They are able to specify the bit period of the generated signal, alter its entropy, or specify a specific bit pattern. They are able to select which encoding scheme to use, as well as what cut-off frequency the signal should be subject to. They are also able to modify how much Gaussian noise to introduce to the signal prior to it reaching the receiver. These factors will affect the bit error rate of the signal as well as the signals energy per bit to noise power spectral density ratio .

The project also introduces students to an eye diagram [2]; this allows students to visualise the intersmbol interference of the signal after it has been encoded and filtered. This can help students to visually identify that if there is not a distinct ‘eye’ present between the high and low signals that the receiver will be unable to demodulate the signal without error.

I provide an overview of my background research that led me to develop this project as a web-based visualisation. I introduce the modern web technology standards that allowed me to produce a visualisation that is able to run on all modern browsers.

My project management procedures that I followed, along with a Gantt Chart which states the objectives to be completed each week. Additionally, I tabulate the risks that may have encountered during the project and state my mitigation strategy for each.

Table of Contents

[Abstract 1](#_Toc134185914)

[1 Introduction 2](#_Toc134185915)

[2 Implementation of Baseband Communication 3](#_Toc134185916)

[2.1 Binary Signal Generator 4](#_Toc134185917)

[2.2 Binary Line Encoder 5](#_Toc134185918)

[2.3 Discrete Fourier Transform 6](#_Toc134185919)

[2.4 Low-Pass Filter 7](#_Toc134185920)

[2.5 Performing an IDFT 7](#_Toc134185921)

[2.6 Demodulating and decoding the signal 8](#_Toc134185922)

[2.7 Additional Features 9](#_Toc134185923)

[2.7.1 Gaussian Noise Generator 9](#_Toc134185924)

[2.7.2 Eye Diagram 10](#_Toc134185925)

[3 Software Testing 11](#_Toc134185926)

[3.1 Automated Testing 11](#_Toc134185927)

[3.1.1 Unit Testing 11](#_Toc134185928)

[3.1.2 End-to-end Testing 12](#_Toc134185929)

[3.1.3 Performance Testing 13](#_Toc134185930)

[References 15](#_Toc134185931)

[Testing References 16](#_Toc134185932)

# Introduction

Teaching visualisations are a useful tool for helping students to understand complex systems. They provide a way for students to not just see how concepts they have been taught work but to interact with these concepts and in doing so develop a greater understanding. By altering the parameters of the simulation, they can learn how these parameters affect the system and understand why these decisions may have been made when implementing real-world systems.

Dave Pearce has published over thirty Silverlight-based visualisations helping students understand concepts including Basic Electronics, Communication Physical Layers, and Communication Protocols and in doing so was awarded the Higher Education Academy Engineering Subject Centre Teaching Award in 2008. This demonstrates the importance of these visualisations to students' learning. However, as of 2019 the technology used to run these demos is no longer supported by any modern web browser [3].

Because of this, it is useful that these visualisations are updated to be able to be run on modern hardware that today’s students can access, not just on university resources but also on their own devices.

The project developed a baseband communication visualisation, based on Dave Pearce’s original demo built with modern web technologies.

I will give an overview of the background reading which led me to decide upon the tools and technologies I chose for this project in section 2. In section 3, I state the specification of the objectives required to meet these aims. I expand the must-have and nice-to-have requirements in section 4. Section 5 expands upon my approach and gives a timeline for my planned progression. Section 6 discusses the risks and mitigation strategies I plan on using. I state any ethical considerations for the project in section 7. Finally, I conclude the report in section 8.

# Overview of background reading

When researching for this project I divided my research into two main sections. The first of these is the technology and software engineering skills that would be required to best realise the project. To create a deliverable that could be accessed by as many students as possible, for as long a time as possible. The second section focuses more on the theory behind the key stages of implementing a baseband communication protocol, such as the Discrete Fourier Transform and the Line Coding techniques that I plan on including.

## Technology

The Covid-19 pandemic has re-confirmed the importance of developing tools and learning resources that are accessible to all students, not just those with access to University resources. The tools we develop should be accessible to students regardless of their location or the technical capability of their computing hardware. Due to this, the most logical platform to develop these tools is a web-based environment; this is because all of the top seven operating systems by market share [4] can access web-based resources.

Developing these tools to be accessed from a webpage allows these tools to be developed once and run by students who are using a range of operating systems. Additionally, web-based resources can be updated without having to ask students to download and install software updates themselves; this allows for bug fixes and patches to quickly be deployed to ensure students are always accessing accurate and relevant information.

The University of York publishes a ‘Minimum PC specification for taught students’ web page [5]. Therefore, it is a requirement that the visualisation can run fluidly on a laptop with those specifications, to ensure that all students following that guidance can engage fully with the content.

### Languages

The World Wide Web Consortium defines the standards and best practices used in web development [6]. They define four languages for running code in the browser, these are HTML, CSS, JavaScript and WebAssembly [7]. These four languages can run in all modern browsers [8]. However, only JavaScript and WebAssembly can be used to implement the interactivity required for this project.

Web Development often uses a JavaScript framework [9]; these are collections of code libraries and components which can be used to help provide a foundation for Web Developers to build their websites. There are various frameworks of JavaScript used for web development, such as React, which is maintained by Meta [10], Angular, which was developed by Google [11], and Vue, which is an independent community-driven project [12]. However, each of these frameworks adds complexity and overhead to developing web apps. Additionally, if future developers wish to maintain/ update the code-base for future cohorts of students or modify the visualisation to introduce new concepts then they would need to be well versed in these frameworks as well as JavaScript.

According to the 2022 State of JavaScript Survey, which was created to identify upcoming trends in the web development ecosystem [13]. There is clear segmentation between these front-end frameworks. Although React is used by 81.8% of respondents, [14] when we consider the interest of JavaScript developers React drops to 47.2%. This may suggest that many developers would be less interested in maintaining and updating the visualisation if I were to use this framework.

TypeScript is a strongly typed programming language that builds on JavaScript [15], it is popular with developers as it allows for type syntax to be added to variables and structures. However, it is translated back into JavaScript before run time. I could develop the project with TypeScript, however, like with the above frameworks it may discourage developers who are unfamiliar with TypeScript’s syntax from maintaining and updating the visualisation.

Because of this, I chose to develop the visualisation with vanilla JavaScript, to ensure that the code can be understood and maintained by as many future developers as possible.

### Development Environment

When deciding which development environment to use for the project I had to ensure that it would be suitable for web development. This meant having native syntax highlighting for JavaScript, CSS and HTML. Additionally, I knew I would be writing in an additional language for WebAssembly so the development environment would have to support syntax highlighting for languages such as C, Rust, or Java.

There are many development environments which fit the first requirement. However, it makes sense to select a development environment which is widely used and popular throughout the industry. The most popular, according to the latest Stack Overflow annual developer survey [16], are Visual Studio Code, Notepad ++, Vim, Sublime Text, and Eclipse. Of those listed, the top four can provide Syntax highlighting for C and Rust.

Additionally, I would like the ability to run a live web server which automatically updates the displayed web page when I make changes to the code, this is an extremely useful tool for fast prototyping.

Of the four development environments remaining, only Visual Studio Code supports this feature. Due to this, I decided to develop the project with Visual Studio Code.

## Baseband Communication

When researching the techniques I would need for a baseband communication simulation I focussed my attention on two areas, the Discrete Fourier Transform (DFT) and Line Coding Techniques I wish to include in the visualisation.

### The Discrete Fourier Transform

When researching the DFT, I read “Digital Signal Processing Concepts and Applications” [23]. This introduced me to some concepts that would be key for me to understand how the DFT works including spectral leakage and the Nyquist frequency. When looking at ways of best implementing the DFT I was helpfully pointed towards “Numerical Recipes: The Art of Scientific Computing” [24]. This included code snippets and useful advice for how best to implement the DFT in software.

# Implementation of Baseband Communication

To help achieve the project goal of developing a baseband communication visualisation, I split the baseband communication process into six key stages. Those stages are:

* Binary signal generation
* Encoding the binary signal
* Sampling and performing a DFT on the encoded signal
* Subjecting the frequency content to a low-pass filter
* Performing an IDFT on the filtered frequency content
* Demodulating and decoding the signal

Of these six stages the first four occur before the transmission of the signal and the last two occur at the receiver. Each of these stages required displaying a graph to the student; these graphs are vertically aligned and are ordered to indicate the progression of a real-world baseband communication system.

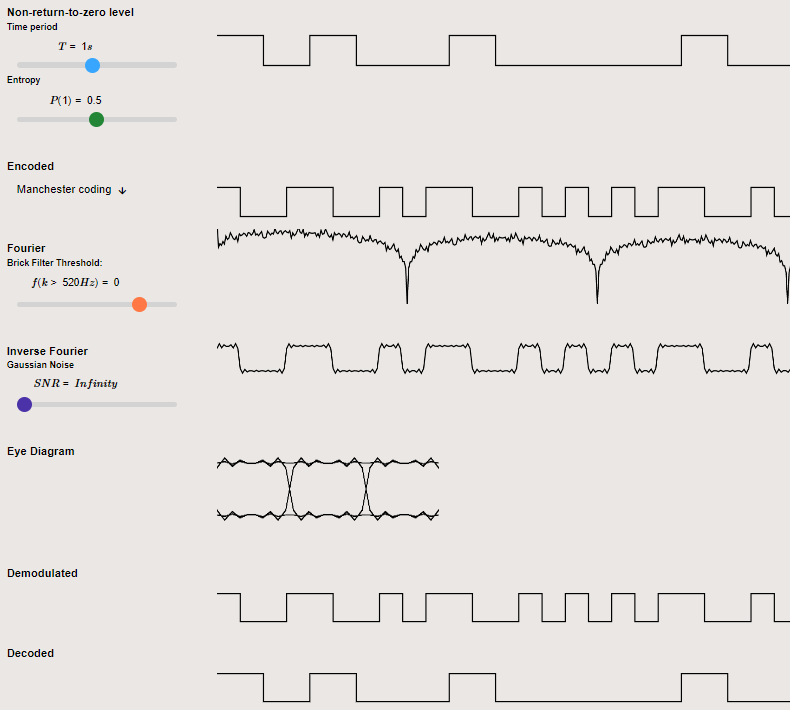


Figure 1 – Screenshot of the visualisation displaying the key stages of Baseband Communication.

## Binary Signal Generator

The first of the key stages was the binary signal generator. It can generate a large, random, sample of data to be coded and transformed. This data is then clearly displayed to the user using non-return-to-zero level encoding.

This is the initial value of the data that will be transformed as it progresses through the baseband communication protocol. The data can be modified by the user in three ways:

* The user can modify the time period of the bits. This alters the bit rate and thus the energy per bit to noise power spectral density ratio of the signal. This is a useful metric for comparing the bit error rate of different encoding schemes.
* The user can manually set the probability of a 1-bit being generated by altering the ‘entropy’ slider between to .  
  This alters the entropy of the bit steam between 0 and 1 according to Shannon’s entropy equation [X].
* This can demonstrate to students how different encoding schemes can handle edge cases, such as a large sequence of binary zeros being transmitted. If the encoding scheme does not vary the signal then the receiver may be unable to maintain a consistent clock frequency to sample the signal correctly.
* The user can invert an individual bit by clicking on the bit position on the graph. This allows the user to manually set a bit pattern so they can see how any specific sequence will be encoded.

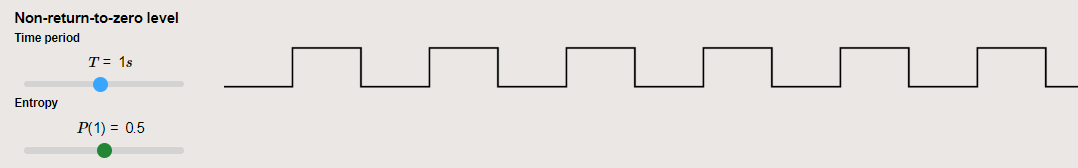


Figure 2 – Screenshot of the binary signal generator.   
Generating a signal with T=1s and P (1) = 0.5.

## Binary Line Encoder

The second stage is a binary line coder, which encodes the data generated by the binary signal generator using several distinct line coding techniques. When deciding which line coding techniques to include I wanted to ensure I was selecting those most useful to the students who were going to use the visualiser. This meant they had to be distinct, each introducing new concepts. They should be used in the real world whilst being easy to understand for students who had previously never been introduced to the concept of line coding. Because of this, I implemented the following six line coding techniques:

* Non-return-to-zero level. For basic display of the generated signal.
* Non-return-to-zero mark. This introduces the concept that data may not just be represented by a single voltage level but may be represented with a bit transition.
* Return to zero. This introduces the concept of return to zero coding to the students, showing that the data does not need to remain at a single level for the entire bit period.
* Biphase-L. Commonly referred to as Manchester Coding. I implemented the line coding technique defined by IEEE 802.3[1], which is implemented in some wired Ethernet standards. This technique can be self-clocked due to each bit pattern generating a bit transition.
* Bipolar, Duobinary signal [2]. This concept can introduce the advantages of a line coding signal having little or no DC component to the students. This is useful to help avoid electro-static discharge.
* Multi-Level Transmit 3 [3]. This introduces the concept of encoding a signal with three voltage levels. Due to the three voltage levels, it requires less bandwidth and emits less electromagnetic interference. Due to its symmetrical nature, it can be connected to a twisted pair of cables and regardless of which way the cables are connected the same signal can be received. This makes it a technique which is used in real-world applications, such as in the FDDI TP-PMD standard that is used in 100BASE-TX as a fast ethernet standard [4].

The student can select the line coding scheme they wish to use so they can see how the encoding scheme changes the frequency content of the wave. These encoding schemes may also be impacted less by noise, or have a greater bit error rate when subject to a low-pass filter. The student will be able to observe these effects further down the visualisation.

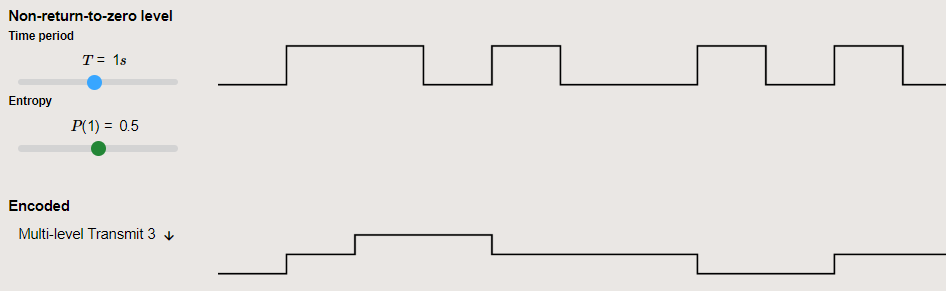


Figure 3 – Screenshot of the visualisation showing a binary signal being  
encoded with MLT-3 Line Coding.

## Discrete Fourier Transform

This key stage passes the encoded signal through a DFT algorithm and displays the frequency content of the student's selected encoding scheme as a spectrogram.

The visualisation first performs a DFT on the student's encoded data. The data from this DFT is then passed down to the subsequent stages. However, to display a more useful depiction of the frequency content of the selected encoding scheme to the student the DFT will continue to process more sets of data. The program then averages the output of all iterations to display a more generalised depiction of the frequency content to the student.

The code for both the DFT and the above implementation can be found in the DFT section of the appendix.



Figure 4 – Screenshot of the visualisation displaying a Spectrogram of a binary signal.

## Low-Pass Filter

When trying to load a cable with a signal that changes instantaneously from zero volts to a higher voltage, extremely high frequencies will be produced. These frequencies can cause undesirable capacitive coupling and crosstalk with other nearby cables, so these frequencies must be filtered out before transmission.

This can be achieved by utilising a low-pass filter. In the visualisation, I utilise a brick-wall filter with a frequency cut-off that can be defined by the student. This is applied to the frequency domain signal returned from the DFT algorithm. In a real-world system, this implementation would not be feasible. Rather, in a real-world analogue system, an LC T-Type filtering circuit could be used to introduce poles to the frequency response, lessening the frequency content after the break frequency [5]. Alternatively, in a digital real-world system, a FIR Filter [6] could be used.

This is an area of the project that, with time, I would like to improve, by implementing more realistic digital filtering techniques.



Figure 5 - Screenshot of the visualisation showing a Spectrogram of a binary signal subject to a brick wall, low-pass, filter with f (k>100Hz) = 0

## Performing an IDFT

Now that the signal has been subjected to a low-pass filter the frequency content is processed using an IDFT. This will then display the filtered version of the signal to the student in the time domain. This is important as it allows the student to see the impact that the low-pass filter has had on the original signal and allows them to understand how the signal will be loaded onto the cable for transmission.

Below are two examples of applying a brick wall low-pass filter to a set of data then displaying the filtered result in the time domain. The first example has a frequency cut off at 300Hz and the second has a frequency cut off of 100Hz.

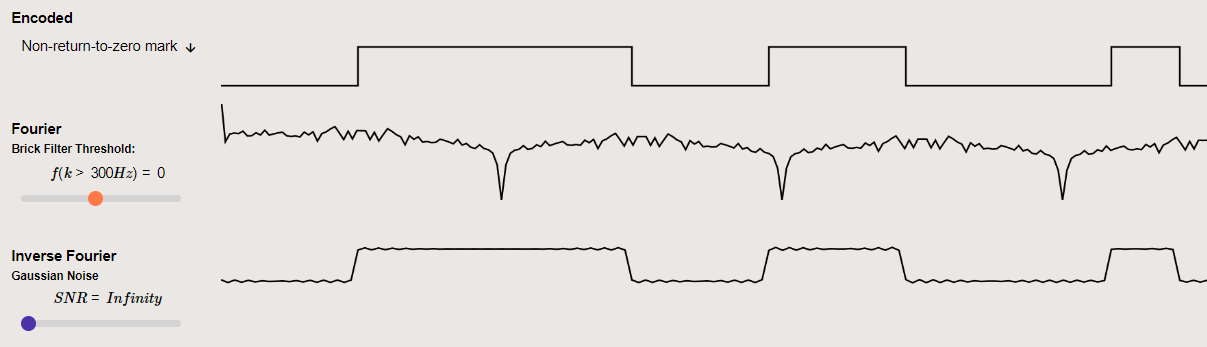


Figure 6 – Screenshot of the visualisation showing the time response of   
a filtered signal with f(k>300Hz) = 0.

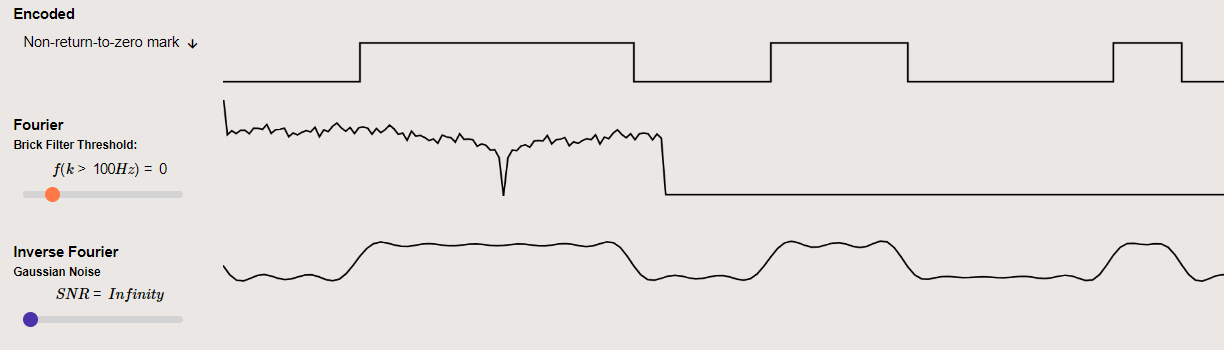


Figure 7 – Screenshot of the visualisation showing the time response of   
the same filtered signal with f(k>=100Hz) = 0.

## Demodulating and decoding the signal

At this stage in a real-world baseband communication system, the filtered signal will be transmitted to the receiver.

It is now the job of the receiver to interpret the incoming signal and attempt to convert it back to the original data stream, as generated by the binary signal generator.

I have implemented the demodulator by sampling the signal at regular intervals determined by the bit period. In a real-world system, the bit period may be standardised or have to be interpreted by the receiver. The current version of the simulation knows the bit period rather than attempting to calculate it. However, this is something that could be implemented with future versions of the visualisation.

Once the signal has been sampled it is then decoded. The demodulated signal may be unable to be decoded correctly due to errors introduced by filtering, or by noise. If this occurs the graph displays this as an error to the user.

If the decoded signal does not match that of the original signal it will be displayed with a red line. This indicates to the student that there was an error introduced when transmitting the data. This is then used to calculate the bit error rate of the system.

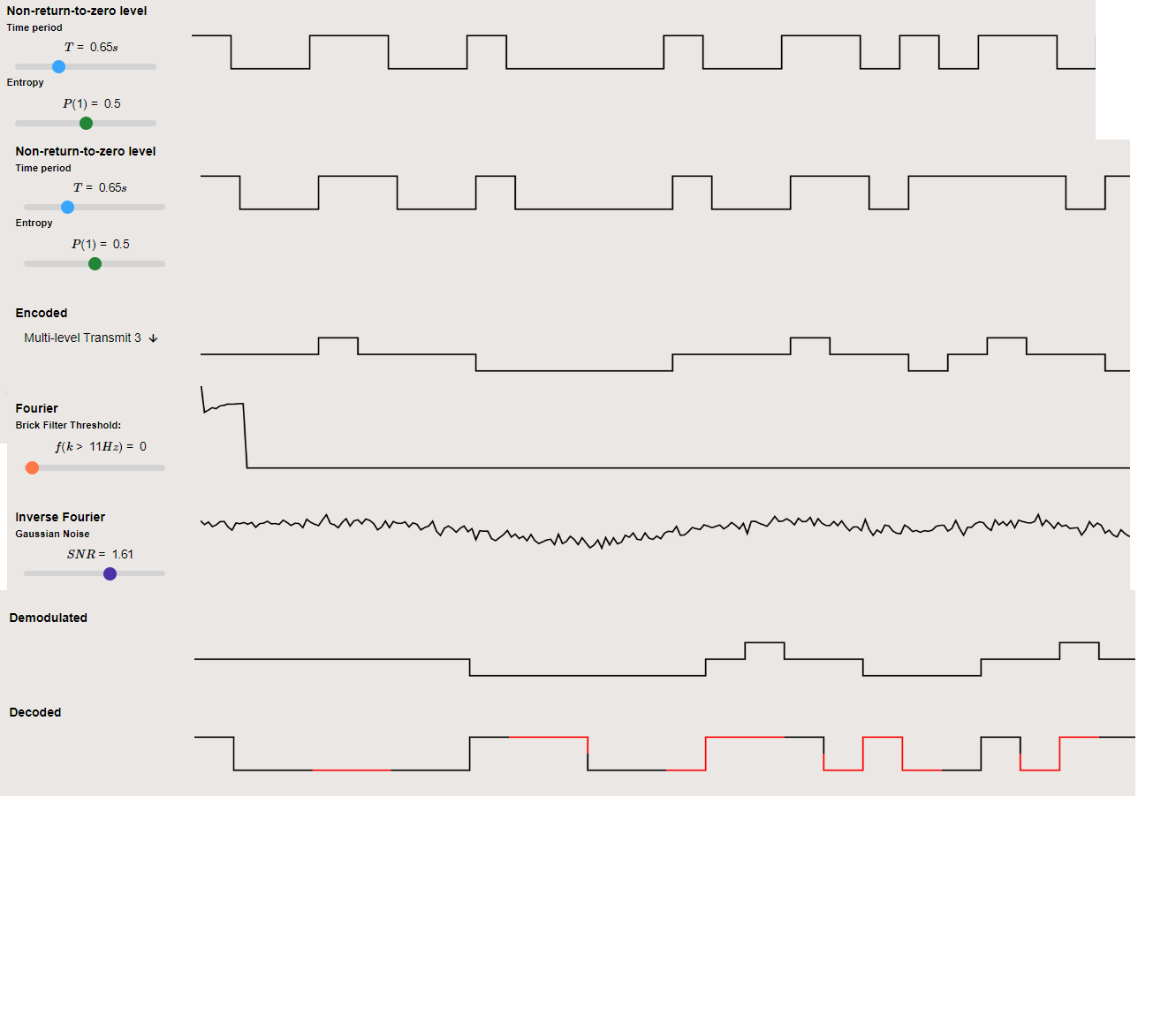


Figure 8 – Screenshot of the visualisation showing a signal being encoded with MLT-3 line coding, filtered, then demodulated and decoded.

## Additional Features

To help demonstrate additional concepts related to baseband communication I have added two further features to the baseband visualisation. A Gaussian noise generator and an eye diagram.

### Gaussian Noise Generator

During transmission in a real-world system, signals will be subjected to noise and interference. This can introduce errors when attempting to demodulate the received signal. To help students understand how different line coding techniques are more or less susceptible to Gaussian noise, when allocated equal power, I added a slider that introduces a variable amount of Gaussian noise to the signal before the signal is demodulated.

### Eye Diagram

To further demonstrate both the effects of noise and the filtering of a signal I implemented an eye diagram. This diagram overlays the filtered signal of all possible sequences of binary ones and zeroes. This allows students to visually compare different line coding schemes by seeing if the receiver would be able to sample the signal at a regular period and receive the correct interpretation of the value of the signal.

If there is not a distinct “eye”, or gap, between the high and low signal values then there is too much intersymbol interference, or noise, present on the signal for a receiver to demodulate the signal correctly at a regular sampling interval; this would result in errors being introduced when demodulating the signal.

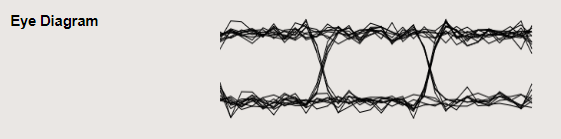


Figure 9 – Eye diagram displaying a signal which is able to be demodulated at regular time intervals without error.

The above eye diagram indicates that it is possible for a receiver to sample and demodulate the signal at regular intervals without introducing errors.

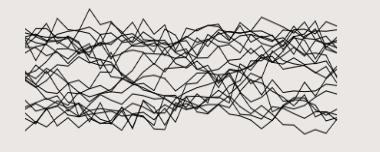


Figure 10 – Eye diagram displaying a signal which is unable to be demodulated at regular time intervals without error.

This eye diagram does not have a clear “eye” between the high and low signal values. This indicates that there is too much noise and intersymbol interference present on the signal thus the receiver will be unable to demodulate the signal without error.

# Software Testing

To ensure the visualisation works as expected it must be adequately tested. The project utilises both automated and manual testing.

Automated testing is performed by software and runs when a change is made to the code. It is extremely useful as it is able to test the underlying code of the software, ensuring that it works as expected. Even if the failure condition is hard to replicate when interacting with the software manually.

On the other hand, manual testing is performed by a user who interacts with the front end of the software and verifies it operates as per the requirements. This type of testing can take a long time and cannot test the underlying code, only what the user can interact with [[1]](#footnote-1).

## Automated Testing

Automated testing is extremely useful during software development to ensure that future changes do not cause issues with previously developed features. The project utilises three, main, types of automated testing: unit testing, end-to-end testing and performance testing. The implementation of these techniques is described below.

### Unit Testing

Unit tests are performed directly on a ‘unit’ of an application, typically a function or class [1]. Their purpose is to input data to the unit and compare the output to a known correct value. This allows them to assert that the unit works as expected. Unit tests should be run independently of other parts of the application as they only evaluate if a single unit works correctly in isolation.

One example from my project where I am utilising unit tests is to test if my complex number functions work as expected. I have set up unit tests to execute the function with set inputs and compare the output of my complex function with known correct values from when I executed the same calculation via MATLAB.

Below is an example of tests which are used to test my complex multiplication and division function. The third parameter of each TestVector object is the known correct value from MATLAB. Importantly, in JavaScript all numbers are stored as 64-bit floating point values. This means that values that are cannot be expressed in base 2 are unable to be stored precisely.

For example, the value is unable to be expressed in base 2, and is instead stored as . This presents major issues when attempting to test the equality of values, as JavaScript can state that . This was an issue when comparing my results from MATLAB to those generated in JavaScript. I resolved this issue by allowing a tolerance of correct values, checking if the final value was within of the expected value.

The unit testing module I built for this project can be found in the testing section of the Appendix.

Text

Description automatically generated

Figure 1 – Examples of complex number unit testing vectors

### End-to-end Testing

End-to-end tests test that the application can meet the requirements by testing the entire workflow. The application has a testing mode which can be run to simulate various input configurations that the user may choose and ensures that the same, correct results are output on the final graph. This ensures that the entire application workflow works as intended and ensures that future changes do not alter known previously correct configurations.

For example, the testing vector below sets the user inputs to the specified values and compares the result to the expected output.

Graphical user interface, text

Description automatically generated

Figure 2 – An example of an end-to-end testing vector

When the test is executed the application updates to display the test inputs and automatically checks the output against the expected output defined in the test vector.

A screenshot of a computer

Description automatically generated with low confidence

Figure 3 – The above test vector being executed

The end-to-end testing module and additional test vectors can be found in the testing section of the appendix.

### Performance Testing

It is important that when the user interacts with the visualisation that it quickly responds and updates the appropriate graphs. An earlier version of my application had severe performance issues where the user would interact with the visualisation, and it would take around 7 seconds to perform the necessary calculations and update the graphs to display the information to the user.

In order to help diagnose this issue I performed performance testing on my application to evaluate what functions of the program were causing the largest impact on performance. To do this, I used profiling tools to record exactly when each function was called and for how long it was executed.

I decided to use the Chromium performance profiler. Which is built into the DevTools on Chromium browsers. I did this as Chromium browsers have the highest desktop market share in 2023[3]. This means that it is likely that it is the most common browser students will be using to access the visualisation. So, it makes sense to use this tool to optimise the visualisation for Chromium browsers as it is likely to be the browser most students use to access the visualisation.

A picture containing graphical user interface

Description automatically generated

Figure 4 – Profiling chart taken pre-optimisation

As you can see from the profiler around 6 seconds of processing time was spent on the ‘plotDataWave’ function, this is far too long and indicates there is a performance problem with this function. We can further break this down and see that the issue resulted from how frequently the ‘getPositionAtTime’ function was being called.

Before performing performance testing I expected the performance bottleneck to be caused by the DFT function as this seemed to be the most computationally intensive function operating with time complexity. However, performance testing revealed this was not the case. This highlights the importance of performance testing as if I had not performed it I could have wasted time optimising part of the code with little impact on performance.



Figure 5 – Breakdown of function execution time

Having found that the bottleneck lay with plotting the graphs rather than with the computationally intensive DFT function, I focused my attention here and re-wrote how the graphs were to be updated. Instead of plotting each point on the graph individually I decided to instead just plot the data points and draw lines between the points.

I repeated the same interaction with the visualisation and again recorded the profiling, with the results shown below.

Timeline

Description automatically generated

Figure 6 –– Profiling chart, taken post-optimisation

This time only 33ms was spent on the task, which is a substantial improvement from 7 seconds. Now the functions which caused the largest impact on performance were the DFT and IDFT functions, which is what I would expect.

Due to performance testing, the user experience when using the visualisation was substantially improved and allows students to interact with the visualisation and receive feedback from their inputs almost immediately.

Doing performance testing was imperative for me to correctly identify which points of the code should be optimised and which operations would not affect performance. If I did not perform performance testing, I would have incorrectly assumed that the DFT was the part of the software which had the biggest impact on performance and wasted development time wrongly optimising that part of the software.

# References

[1] *IEEE Standard for Ethernet,* IEEE Standard 802.3-2018

[2] Wikipedia contributors. *"Bipolar encoding"*.  Wikipedia, The Free Encyclopaedia. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=Bipolar_encoding&oldid=1113660078> [Accessed: 29 April 2023].

[3] Wikipedia contributors. "MLT-3 encoding". Wikipedia, The Free Encyclopaedia. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Line\_code&oldid=1131753198 [Accessed: 29 April 2023].

[4] *IEEE Standards for Local and Metropolitan Area Networks: Supplement - Media Access Control (MAC) Parameters, Physical Layer, Medium Attachment Units, and Repeater for 100Mb/s Operation, Type 100BASE-T*,IEEE Standard 802.3, 1995.

[5] D. Pearce. “A Short Introduction to First-Order Responses”. University of York. [Online]. Available: <https://wiki.york.ac.uk/display/EE/Short+Introductions?preview=/206308218/251169152/24_Short_Intro_to_First_Order_Responses.pdf> [Accessed: 30 April 2023].

[6] D. Halliday. “FIR filter design”. University of York [Online]. Available: <https://wiki.york.ac.uk/download/attachments/198216152/DSP_OHP_Lect10.pdf?version=1&modificationDate=1676897975000&api=v2> [Accessed: 28 April 2023].

# Testing References

[1] S. Pittet. *The different types of software testing.* Atlassian.com. [Online]. Available: <https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing> [Accessed 10 April 2023]

[2] A. Meixner. *The Test Attributes of Controllability and Observability.* accendoreliability.com. [Online]. Available: <https://accendoreliability.com/test-attributes-controllability-observability/> [Accessed 12 April 2023]

[3] Kinsta. *Global Desktop Browser Market Share for 2023.* kinsta.com. [Online]. Available: [Global Desktop Browser Market Share for 2023 - Kinsta®](https://kinsta.com/browser-market-share/) [Accessed 25 April 2023].

1. This is related to concepts known as observability and controllability [2]. Since manual tests typically cannot observe data when an application is running, they cannot test that the individual components are working as intended. This is less of an issue with automated testing as they typically can interact with the underlying code. [↑](#footnote-ref-1)