****

School of Physics, Engineering and Technology

**BEng Initial Project Report  
  
2022/23**

**Student Name:** Ben Millar

**Project Title:** On-line teaching simulation / visualisation

**Supervisors:** Dave Pearce and Asim Mumtaz

Department of Electronic Engineering  
University of York  
Heslington  
York  
YO10 5DD

Ben Millar

[company name]

# Abstract

This initial report sets out my plan and research for producing a web-based teaching simulation for a Baseband Modulation visualisation. The visualisation will simulate generating a binary signal. Then the signal will be encoded using a line coding scheme defined by the user. The high-frequency content of this signal will then be filtered out. Finally, the visualisation will display this signal’s power spectrum before transmission.

Students will be able to interact with the simulation by altering parameters. Including specifying the entropy of the generated signal, selecting which coding scheme to use, and altering the cut-off frequency of the low-pass filter.

If time permits, I will expand upon these goals by allowing the student to select additional line coding methods such as MLT-3 [1]. Additionally, I would like the student to be able to interact with the generated bit-stream by inverting individual bits and see how this effect the line coding of the signal. Finally, I would like to include an eye diagram [2] to allow the student to visualise the intersymbol interference of the signal after it has been encoded and filtered.

I provide an overview of my background research that led me to develop this project as a web-based visualisation; making use of WebAssembly and Rust to improve the performance of the Discrete Fourier Transform (DFT) across a signal with a large number of data points.

My planned progression for the project is laid out in the form of a Gantt Chart giving the objectives to be completed each week. Additionally, I tabulate the risks that may be encountered during the project as well as state my planned mitigation strategy for each.

This report will be followed up, in May of 2023, by the full report which will expand this report discussing the process of creating this simulation.

# 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 aim is to develop 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. I state the specification of the objectives required to meet these aims in section 3. I expand the must-have and nice-to-have requirements in section 4. Expand upon my approach and give a timeline for my planned progression in section 5. I discuss the risks and mitigation strategies I plan on using in section 6. 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 required for creating a Baseband Communication Visualisation.

## Technology

The Covid-19 pandemic has re-confirmed the importance of developing tools and learning resources that are not just accessible to those with access to University resources. The tools we develop should be accessible to all 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. 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 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 define the standards and best practices used in web development [6]. They define four languages for running code in the browser, these four languages 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.

I also 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.

### WebAssembly

WebAssembly (wasm) is a recent technology adopted in 2019 by the World Wide Web Consortium. It is notable as the second standard for executing code in all modern browsers [17].

WebAssembly allows for code to be written in many languages and then compiled into a low-level binary format. This would theoretically allow for massive computational time savings when delivering computationally intensive programs to the user. This is because you can write the code in an extremely high-performance language, like C, or Rust, and convert it to a binary format for the user to run on their machine far faster than JavaScript could run.

As JavaScript was not designed to be a high-performance language and is compiled by a Just in Time compiler, ahead of execution, it would not be the language of choice for something computationally intensive, such as the Fast Fourier Transform. WebAssembly on the other hand would be complied to binary code far ahead of time and would be downloaded to the user’s machine when they load the web page and be immediately ready for fast execution.

I am planning to use the Fast Fourier Transform in my visualisation to convert a function between the time and frequency domains. This is a very computationally intensive operation, given the large number of sample points needed to capture the higher frequencies of the data generated. These operations could utilise the performance advantages of WebAssembly to allow the function to be programmed in a more appropriate language. And then, compiled to byte code before being run by the user's browser, theoretically showing the user the output much quicker than with JavaScript.

A popular design tool called Figma [18] is currently doing this [19] to deliver an extremely computationally intensive design tool to users via a web browser. Previously something this powerful would only be able to run as a native, compiled desktop application, such as Photoshop.

When deciding upon which language to write my code in for WebAssembly I had to consider three main factors. As my goal was to compare the performance of WebAssembly to JavaScript I had to choose a programming language that has good performance. This means that it should be directly translated to machine code with little alteration of the code by a compiler or virtual machine. The second factor was memory management, I wanted to have full control of the memory being allocated so that I could ensure a fair trial with every trial having the same number of bytes of memory being allocated. Additionally, I did not want to have a garbage collector adding a variable to the trial that I could not control. The garbage collector would also increase the download size of the WebAssembly code being downloaded to the user’s machine, this would dramatically increase the time taken to load the page.

Due to these factors, I narrowed down my choice of language to either C, C++, or Rust. The final factor I considered was support from the language community for WebAssembly. If I wanted to ensure that the project could be maintained by other developers in the future, the language needed to have strong tools for interacting with the webpage’s Document Object Model as well as having tools for translating rust code into WebAssembly code. Rust is the clear choice in this regard being the most desired and frequently used language for WebAssembly, according to the 2022 State of WebAssembly Survey [20]. It has a large number of Open-Source tools for building WebAssembly Code [21], as well as tools for Interacting with JavaScript and the DOM [22]. Due to this, I decided to use Rust as the language when developing the WebAssembly part of this project.

## Baseband Communication

When researching the techniques, I would need for a Baseband Communication prototype I focussed my attention on two areas, the Discrete Fourier Transform and Line Coding Techniques I wish to include in the visualisation.

### The Discrete Fourier Transform

For background reading of the Discrete Fourier Transform (DFT), I read “Digital Signal Processing Concepts and Applications” [23]. This introduced me to some of the concepts that would be key for me to understand how the DFT would represent my data including spectral leakage and the Nyquist Frequency. When looking at ways of best implementing the Discrete Fourier Transform I was helpfully pointed towards “Numerical Recipes: The Art of Scientific Computing” [24]

### 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, introducing new concepts such as return to zero. 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. Due to this, I decided to initially add the following, five, line coding schemes:

* Non-return-to-zero level
* 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 intend on implementing the line coding technique defined by IEEE 802.3[25], which is implemented in their wired Ethernet standards. This technique is commonly used as regardless of which symbol is generated there is always a bit transition, this means the signal is self-clocking. This introduces a useful coding scheme in the real world and can also teach the importance of a signal being self-clocking.
* Bipolar, Duobinary signal [26]. This concept can introduce the advantages of a line coding signal having little or no DC component to the students.

# Specification

To achieve the above aim, the following requirements must be met.

## Requirements needed to implement Baseband Communication

The JavaScript included with the web page must be able to implement the following functionality:

* A binary signal generator, this signal must be able to be displayed to the user.
* Line coder, which must be able to take in the generated signal and encode it using the technique selected by the user. The output of this must be displayed to the user
* A Discrete Fourier Transform, to transform the signal from the time to frequency domain
* A low-pass filter, which takes the output from the Discrete Fourier Transform and removes the high-frequency content.
* Inverse Discrete Fourier Transform, to transform the signal back from the frequency domain to the time domain. This must be displayed to the user to demonstrate the effect of the low-pass filter.
* A graph displaying the power spectral density of the filtered signal.

# Description of requirements

To achieve the aim of the project I have some must-have requirements, these are essential to have to be able to thoroughly test my hypothesis. Additionally, I have listed some nice-to-have requirements that I would like to add to the project if I have time, these would extend the functionality of the visualisation as well as introduce additional concepts to expand the teaching goals of the project.

## Must have requirements

These requirements must be met to meet the aim of creating a Baseband Communication teaching visualisation. These descriptions expand upon the specifications defined in section 3.

### Binary signal generator

The first of which is a binary signal generator able to generate a large, random, sample of data to be coded and transformed. This data must be clearly displayed to the user so they can follow the process of the data as it passes through the baseband visualiser.

### Line coding

The second requirement is a line coder which must be able to encode the data generated by the binary signal generator using several distinct line coding techniques, the student needs to be able to select the line coding scheme they wish to use so they can see how the encoding scheme changes the output frequencies.

### Discrete Fourier Transformation

The third requirement is to be able to pass this encoded signal through a Discrete Fourier Transform (DFT) algorithm. This is to represent the signal as a function of its frequency content for the higher frequencies to be filtered out.

### Low-pass filter

A low-pass filter must be applied to the Frequency domain signal returned from the DFT algorithm. This is essential to cut off the extremely high frequencies that would be generated when trying to load a cable with a signal that changes instantaneously from zero volts to a higher voltage. These extremely high frequencies could cause undesirable coupling and crosstalk with other nearby cables so these frequencies must be filtered out before transmission.

## Inverse Discrete Fourier transform

The final must-have requirement is to use the Inverse Discrete Fourier Transform algorithm to transform the filtered DFT signal back into 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.

## Nice to have requirements

Additionally, I will follow up on these must-have requirements with objectives that would be beneficial to include if time permits, to extend the scope and learning objectives for the project.

### Eye Diagram

Firstly, if time permits, I would like to present the user with an eye diagram overlaying the filtered signal for all possible sequences of binary ones and zeros. This would allow the student to be able to visually compare different line coding schemes by visually understanding where the receiver would need to sample the signal to receive the correct interpretation of the value of the signal.

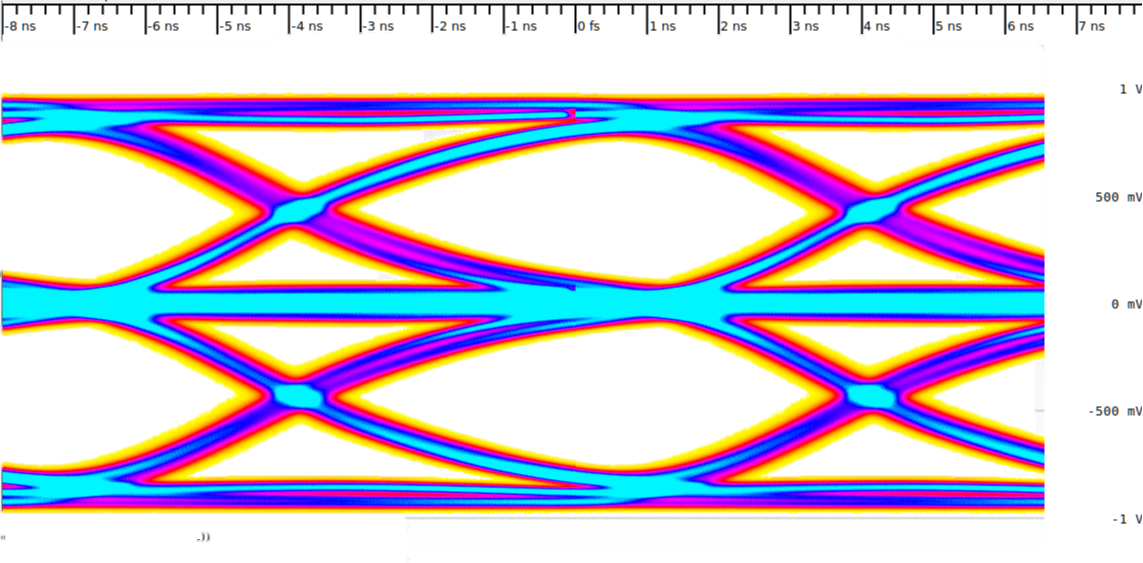
For the receiver to be able to correctly sample a signal there needs to be a distinct “eye” between the high and low signal values. If there is too much intersymbol interference or noise present on the signal this central “eye” will not be present. This means the signal will be unable to be sampled without error.

Figure 1 - Example of an MLT-3 Eye Diagram  
(Taken from: https://commons.wikimedia.org/wiki/File:Eye\_pattern\_MLT3.png)

### Noise

I would like to be able to allow the user to add variable amounts of gaussian noise to the original signal. This would allow the student to better be able to visualise how the signal would look in a real-world situation. Additionally, it would help the student understand how different line coding techniques can be used to mitigate the effects of Gaussian noise.

### Entropy

I would like to add the ability for the student to select the entropy of the original signal. They would do this by altering the probability that a binary one is generated. This would allow them to more easily understand how different line coding schemes affect the output signal. Additionally, by allowing the user to select the probability with a slider, the visualisation can provide feedback as to how their input affects the entropy of the signal by displaying an automatically updating entropy equation. Such as the one shown below.

Where is the probability of the symbol being generated. And, is the number of symbols used in the alphabet for the communication protocol; for the case of a binary signal, this would be two.

### Additional Line Coding Techniques

If time permits I would wish to include additional line coding techniques for the student to be able to select from. In particular, I would wish to include techniques such as 4B5B line coding. This technique maps four input bits onto five output bits for transmission [27]. This technique ensures that there will always be enough bit transitions to produce a self-clocking system regardless of the input bits. Due to this 4B5B encoding is used in the USB Power Delivery specification [28].

Additionally, I would like to include MLT-3 encoding [29] which uses three voltage levels and cycles through the voltage levels -1, 0, 1, 0 to transmit a binary one, but remains at the current voltage level to transmit a binary zero. 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.

### Inverting Bits

The user should be able to click on the signal to invert an individual bit. Doing this will allow the user to understand how each bit affects the final signal and how one-bit alteration could have large effects on the final output signal.

### Cable simulator

Finally, I will like to include a cable simulation where the encoded and filtered signal is simulated to be passed down a cable where the student defines the cable attenuation in Np/m. This would allow the student to view how the signal would be received and simulate the eye diagram at the receiver of the cable to verify if their chosen line coding scheme and filter pole locations would be suitable for a real-world communication protocol with the parameters given.

Additionally, the energy per bit to noise power spectral density (Eb/No) could be calculated for the parameters given. The student could then define the gain of an amplifier at the transmission end of the cable to achieve a desired Eb/No for their simulated communication protocol.

# Timetable

The following Gantt Chart shows my planned progression towards meeting the project specifications. It includes three prototype simulations which I build to test segments of the final simulation independently. This ensured that each segment could be verified to work and allowed for receiving feedback regularly so that it could be implemented without causing the project to fail to hit the deadline.

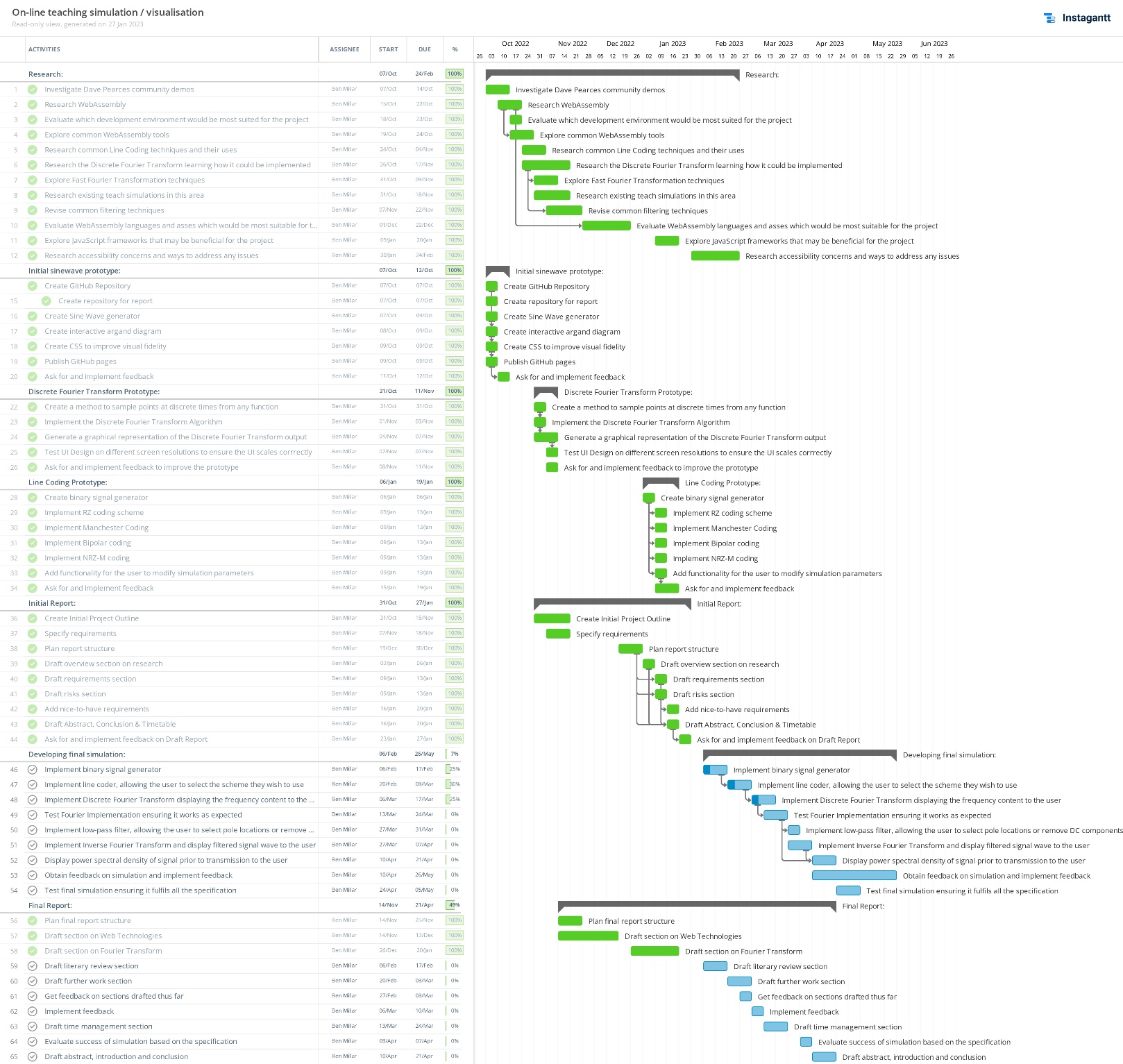
The below Gantt Chart shows the objectives which should be achieved each week.

Figure 2 - Gantt Chart showing planned weekly project progression

# Risks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk | Severity | Probability | Seriousness | Mitigation |
| Losing project code due to laptop crashing. | 10 | 3 | 30 | Back-up code on GitHub, keep an offline backup. |
| Report documents may be lost or become corrupted | 10 | 3 | 30 | Create regular backups of all report documents, and store these backups in multiple places including a cloud backup. |
| Scope creep, leading to being unable to finish the core aims of the project by the deadline. | 10 | 4 | 40 | Stick to the must-have requirements until the core aims are met. Create regular working prototypes to ensure aims are always being addressed. Only implement nice-to-have requirements after the core aim is met |
| The project is not progressing according to the schedule | 8 | 3 | 24 | Follow the timetable closely. Schedule regular supervision sessions and ask for help when it is needed. |

# Ethics statement

After consideration of the University’s code of practice and principles for good ethical governance, no ethical issues were identified in this project.

# Conclusion

In this initial report, I laid out my research for selecting the tools and techniques I did for creating a web-based Baseband Communication simulation. I discussed the must-have and nice-to-have requirements for this simulation. I followed this up with a timetabled plan for the project along with any risks and ethical considerations I may encounter when developing this project.

This report will be followed up by the full- report in May of 2023. The full report will expand this report discussing the process of creating this simulation.

# References

[1] Godred Fairhurst *MLT-3 Line Encoding* erg.abdn.co.uk [Online]. Available: <https://erg.abdn.ac.uk/users/gorry/course/phy-pages/mlt3.html> [Accessed 29 January 2023]

[2] David Herres *The Eye Diagram: What is it and why is it used* testandmeasurementtips.com [Online]. Available: <https://www.testandmeasurementtips.com/basics-eye-diagrams/> [Accessed 29 January 2023]

[3] Microsoft *Silverlight End of Support* support.microsoft.com [Online]. Available: <https://support.microsoft.com/en-us/windows/silverlight-end-of-support-0a3be3c7-bead-e203-2dfd-74f0a64f1788> [Accessed 26 January 2023]

[4] StatCounter *Operating System Market Share Worldwide* sstatcounter.com [Online]. Available: <https://gs.statcounter.com/os-market-share> [Accessed: 22 January 2023]

[5] The University of York *“Minimum PC specification for taught students” York.ac.uk* [Online]. Available: <https://www.york.ac.uk/it-services/purchasing/private/minimum-spec-students/> [Accessed: 20 January 2023]

[6] W3C *“About W3C” W3.org* [Online]. Available: <https://www.w3.org/Consortium/> [Accessed: 19: January 2023]

[7] W3C *“World Wide Web Consortium (W3C) brings a new language to the Web as WebAssembly becomes a W3C Recommendation” W3.org* [Online]. Available: <https://www.w3.org/2019/12/pressrelease-wasm-rec.html.en> [Accessed: 19 January 2023]

[8] Mozilla *“WebAssembly”* developer.mozilla.org [Online]. Available: <https://developer.mozilla.org/en-US/docs/WebAssembly> [Accessed: 18 January 2023]

[9] Trio *“JavaScript Frameworks: What Are They and How Do They Work?”* trio.dev [Online]. Available: <https://www.trio.dev/blog/javascript-framework> [Accessed 25 January 2023]

[10] React *“React” reactjs.org* [Online]. Available: <https://reactjs.org/> [Accessed: 19 January 2023]

[11] Angular *“Angular”* angular.io [Online]. Available: <https://angular.io/> [Accessed: 19 January 2023]

[12] Vue *“Vue”* *vuejs.org* [Online]. Available: <https://vuejs.org/about/faq.html> [Accessed: 19 January 2023]

[13] State of JS *“State of JS About”* 2022.stateofjs.com[Online]. Available: <https://2022.stateofjs.com/en-US/about/> [Accessed: 20 January 2023]

[14] State of JS *“Ratios over time”* 2022.stateofjs.com[Online]. Available: <https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/> [Accessed: 20 January 2023]

[15] TypeScript *“TypeScript”* typescriptlang.org[Online]. Available: <https://www.typescriptlang.org/> [Accessed: 20 January 2023]

[16] Stack Overflow *“Stack Overflow Developer Survey”* insights.stackoverflow.com[Online]. Available: <https://insights.stackoverflow.com/survey/2021#technology> [Accessed 26 January 2023]

[17] Mozilla *WebAssembly* developer.mozilla.org[Online]. Available: <https://developer.mozilla.org/en-US/docs/WebAssembly> [Accessed: 24 November 2022].

[18] Figma *Figma* figma.com [Online] Available: <https://www.figma.com/> [Accessed: 20 January 2023]

[19] Evan Wallace CTO Figma *WebAssembly cut Figma’s load time by 3x* [Online]. Available: <https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/> [Accessed: 24 November 2022].

[20] Scott Logic *The State of WebAssemly 2022* scottlogic.com [Online]. Available: <https://blog.scottlogic.com/2022/06/20/state-of-wasm-2022.html> [Accessed: 22 January 2023]

[21] The Rust and WebAssembly Working Group *wasm-pack* github.com [Online]. Available: <https://github.com/rustwasm/wasm-pack> [Accessed: 22 January 2023]

[22] *Rust and WebAssembly* rustwam.github.io [Online]. Available: <https://rustwasm.github.io/docs/book/reference/tools.html> [Accessed: 22 January 2023]

[23] B. Mulgrew, P. Grant and J. Thompson *Digital Signal Processing: Concepts and Applications.* Bloomsbury Publishing, 2002

[24] W. H.Press, S. A. Teukoisky *Numerical Recipes The Art of Scientific Computing* Cambridge University Press, 2007

[25] *IEEE Ethernet,* IEEE Standard 802.3-2018

[26] Wikipedia. *“Line Coding”,* Wikipedia.org[Online]. Available: <https://en.wikipedia.org/wiki/Line_code> [Accessed: 19 January 2023]

[27] Wikipedia. *“4B5B”, Wikipedia.org.* [Online]. Available: https://en.wikipedia.org/wiki/4B5B[Accessed: 19 January 2023]

[28] USB Implementers Forum *USB Power Delivery* usb.org [Online]. Available: <https://www.usb.org/document-library/usb-power-delivery> [Accessed: 20 January 2023]

[29] Wikipedia *“MLT-3 Encoding”* Wikipedia.org [Online]. Available: <https://en.wikipedia.org/wiki/MLT-3_encoding> [Accessed: 30 January 2023]