University of Southern Queensland

Faculty of Health, Engineering & Sciences

Remote Test and Measurement Access via Acoustic Coupling in Non-WiFi Accessible Environments

A dissertation submitted by

C. Van den Ham

in fulfilment of the requirements of

ENP4111 Research Methodology

towards the degree of

Bachelor of Electrical & Electronic Engineering

Submitted: November, 2024

Abstract

This research project investigates the feasibility of acoustic data transfer in areas where wireless and wired connections are restricted or non-existent. The project is a proof of concept for a mobile device, operated by a technician, to send and retrieve data from an embedded field device, examining the speed and robustness of data transferred between devices under different environmental situations.

This concept has potential applications in scenarios where an embedded device is located in a remote area, and data needs to be retrieved for troubleshooting or monitoring purposes. Additionally, it could be used for semi-remote firmware updates, where the device is accessible to a technician on-site, but remote from the developers.

The data transfer process follows this sequence: The mobile device accesses a text (or JSON) file, converts the text to binary data, modulates the binary using frequency shift keying (FSK) into an audio file, and plays the audio to the embedded device. Once initiated, the embedded device listens for the audio file, records the audio to a buffer, demodulates the FSK signal into binary data using Goertzel's Algorithm, Fast Fourier Transform (FFT) or a bandpass filter, and converts the binary data back to text. It then reads the message from the mobile device and responds by sending an audio signal back (either an acknowledgement of received update, or with diagnostic data) following the same process.

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

C. Van den Ham

[redacted]

Acknowledgments

I would like to thank everyone who supported me throughout this project. Their willingness to listen and allow me to work through problems, even when the topics were unfamiliar, has been greatly appreciated.

A special thanks to my partner, Priscila, for her patience and thoughtful input during many discussions around the design and implementation of DSP. I am also grateful to my family for their ongoing encouragement and support.

I would like to acknowledge my supervisor, Professor John Leis, for his valuable guidance and groundwork around shaping the direction of this project, as well as Dr. Bo Song for providing key insights and suggestions that helped me throughout this project.

C. Van den Ham

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction and Background

Technology has evolved at an incredible pace in recent history. What was thought impossible only a decade ago, is now proving not only possible, but better, and faster than was imagined by many. The phenomenal amounts of data that is sent between palm sized devices eclipses what super computers could achieve only a short while ago (Gou & Wu 2023). Much of this data transmission occurs through cabled connections, though end users are increasingly relying on the use of radio frequencies, such as WiFi and Bluetooth, to transmit the data from the cable (or modem) to the end user device.

While this is advantageous in many ways, transmitting data over radio signals can introduce security risks, which continue to evolve (Ramezanpour, Jagannath & Jagannath 2023). With the rise of Internet of Things (IoT), an increasing number of devices are now connected to servers via WiFi and Bluetooth connections (Lonzetta, Cope, Campbell, Mohd & Hayajneh 2018). This connectivity enhances data retrieval and automation but also introduces security vulnerabilities, especially in environments where sensitive information must be protected. Such environments include hospitals, defence facilities, petrochemical plants, and other critical infrastructure where data security is paramount.

Generally, any equipment housed in these radio frequency (RF) sensitive areas that need to transfer data, will have reliable ways in which to do so. On rare occasions some equipment may need to transfer data that does not often do so. This could be in a time of fault, when there is a system fault, or it could be a piece of monitoring equipment that rarely needs to dispatch its contents. When these anomalies occur, it often means a skilled technician needs to be sent to site to perform the data interrogation.

These RF restricted areas are often located in central locations, so when data needs to be sent to, or retrieved from equipment, technicians are easily able to physically retrieve the data. This would often mean plugging a secure laptop into the device by way of some description of cable, be it proprietary or not. Other times the technician may need to physically inspect the system to assess for faulty equipment.

When these sites are in remote areas, having a technician come to site can take an excessive amount of time. Technicians sometimes have to travel thousands of kilometres to access equipment, often because it is in restricted areas or cabinets. Some of these faults can be as simple as a faulty sensor, resulting in a time consuming, very costly process of fault finding.

This project will explore acoustic coupling to transfer data to and from embedded devices in remote areas. This will offer a simple solution for system interrogation, small firmware updates and general trouble shooting that can be performed by personnel following a standard of procedure.

Technology of this kind is widely used in underwater acoustic modems (Sendra, Lloret, Jimenez & Parra 2016), where radio frequencies experience significant attenuation, leading to poor data transmission. In contrast, acoustic waves propagate efficiently through water, enabling reliable communication. This project will explore various modulation schemes to determine the most effective method for data transmission in such environments.

Modulation types may include Frequency Shift Keying (FSK), Amplitude Shift Keying (ASK), Phase Shift Keying (PSK), Direct Sequence Spread Spectrum (DSSS), Orthogonal Frequency Division Multiplexing (OFDM). These types of modulation are all quite common and range in levels of complexity and bit rate (Indriyanto & Edward 2018*a*). The modulation types used will depend on time and resources available, the purpose of the project is for the transmission to be simple enough that it is accessible to a wide range of equipment, which may restrict complexity.

1.2 Objectives and Aims

The aim of this project is to develop a proof of concept prototype to investigate secure, efficient and robust data transmission through acoustic coupling. The prototype will test the impact of different transmission speeds, microphones, and environmental factors on the accuracy of data transmission.

Future development of this prototype will enable data to be transmitted from a portable device, using an Application Programming Interface (API) for processing, to an embedded device, simulated by an Arduino Uno. The use of these two different devices will facilitate bi-directional communication and data retrieval from the embedded device.

1.2.1 Primary Objectives

Modulation Developement – Analyse different modulation techniques as mentioned in 1.1 with the aim of producing the most robust modulation for aerial acoustic transmission.

Signal Generation – Investigate different aspects of signal generation to optimise for use between a portable device and a low computer powered embedded device.

Demodulation Development – Analyse different demodulation techniques with the aim of producing the most consistent, low error demodulation.

Hardware Testing – Test various hardware components, particularly different microphones, to evaluate their performance in terms of sensitivity, frequency response, and cost-effectiveness for consistent performance.

Speed Testing – Investigate the maximum transmission speed at which data can still be reliably demodulated, focusing on the impact of increased speed on error rates and latency.

Preamble Testing – Implement and test the use of a preamble and postamble to ensure the system reliably detects the start and end of messages. This will establish the foundation for implementing a secure handshake mechanism, serving as a precursor to future authentication protocols or security key exchanges to safeguard bi-directional communication between systems.

1.2.2 Secondary Objectives

Prototype Developement – Develop a functioning prototype of the acoustic access system. This will involve hardware design and construction, taking into account the need for integrating the speaker and microphone components on the target device. The prototype will also need to have the ability to run software for the modulation and demodulation of the data to be transmitted.

Interface Design – Develop an API with an intuitive and user friendly user interface (UI) for field personnel to interact with.

Field Testing and Validation – Conduct testing to evaluate the system's performance and usability.

Cost Benefit Analysis – Evaluate the cost of the system including new equipment but also retrofitting equipment that is already in use.

1.2.3 Expected Outcomes

Validation – Identify the validity of sending and retrieving data remotely using acoustic coupling.

Acoustic Coupling Optimisation – Identify what methods of modulation and demodulation are the most effective for acoustic coupled data transfer speeds and accuracy. This will allow for recommendations to be made for any future work undertaken in this field.

Technical Performance Data – Determine technical abilities of the system, including maximum operating range, achievable data rates, latency constraints and usability.

1.3 Contributions to Research

This project leverages established technology in acoustic data transfer, integrating it with modern advancements to enable remote data retrieval and communication across long distances, where it was previously constrained to very close proximity. The research contributes to the field by assessing the feasibility and performance of acoustic data transfer methods in scenarios where bandwidth is constrained and the relative positioning of speakers and microphones is variable.

The project explores and evaluates different modulation and demodulation techniques for their robustness and reliability in dynamic environments where noise, orientation, and timing present challenges, this study identifies effective solutions for acoustic data transfer under real-world conditions. The results aim to provide valuable insights into optimising acoustic data systems where wireless and wired connections are restricted or non-existent.

Chapter 2

Literature Review

2.1 Chapter Overview

The evolution of acoustic modulation techniques from their inception has been marked by significant innovations aimed at enhancing data transfer speeds, robustness, security, and error reduction (Baggeroer 1984, Kishore & Rallapalli 2019, Zhang, Zhan, Chen, Li, Ren, Wang & Ma 2014). Notably, research has ventured into embedding data within musical compositions, an endeavour to harmonise acoustic data transfer with human experiences more comfortably (Zhou, Wang, Ren, Koutsonikolas, Su & Chen 2019, Cho, Choi & Kim 2015). These advancements underscore the potential of acoustic coupling as an effective medium for data exchange across various domains.

Much of the later research and development has unfolded within the marine environment, concentrating on underwater acoustic modulation. This focus is partly due to the limitations of radio frequency (RF) modulation underwater, where acoustic signals offer a viable alternative for communication. However, in the broader context of data transmission, especially in terrestrial and air environments, RF modulation currently leads in meeting the demands of our increasingly data-hungry world due to its superior data transfer speeds. This project will concentrate on acoustic coupling, not for it's speed, but for it's added inherent security in close-range data transfer, compared to that of RF.

A significant consideration for this project is that some target devices may possess limited processing power. The efficiency of data transfer is intrinsically linked to the computational capacity available, highlighting the need for optimisation strategies that minimise processing demands while maximising data transfer efficacy.

Accordingly, this project concentrates on exploring and developing acoustic data transfer methodologies optimised for minimal processing overhead on devices with constrained computational resources. By prioritising techniques that capitalise on the inherent properties of acoustic coupling for efficient data exchange, this research aims to make modulation methods more accessible to a broader spectrum of devices, particularly those with limited processing capabilities.

This endeavour aims to not only contribute to the body of knowledge in acoustic data transmission but also address a pivotal challenge in making these technologies accessible to a wider array of devices in RF restricted environments.

2.2 Digital Modulation Techniques

There are a number of parameters that should be taken into consideration when selecting an appropriate modulation technique. The right modulation technique will minimise bit error rates, be robust while considering bit rate and have a balance of power / cost effectiveness and meeting the demand for bit transfer speeds (Kishore & Rallapalli 2019).

2.2.1 Bit Error Rate

The data that is transmitted will always be exposed to some sort of external noise along the transmission. This noise can lead to some bits in the transmission being compromised, this is called the bit error rate (BER) (Leis 2018). The BER can be found dividing the number of received bits that have been compromised, by the total number of bits transmitted,

$$BER = \frac{Number\text{ of error bits}}{\text{Total Number of bits}} \tag{2.1}$$

(Kishore & Rallapalli 2019)

Bit errors in digital systems occur when noise and other interferences affect the transmitted signal, potentially so much so that the demodulated signal reads the wrong bit. A simple illustration of a binary data transmission affected by noise is shown in Figure 2.1, where a square wave representing a binary sequence encounters additive noise, resulting in a received signal that combines both the original data and noise.

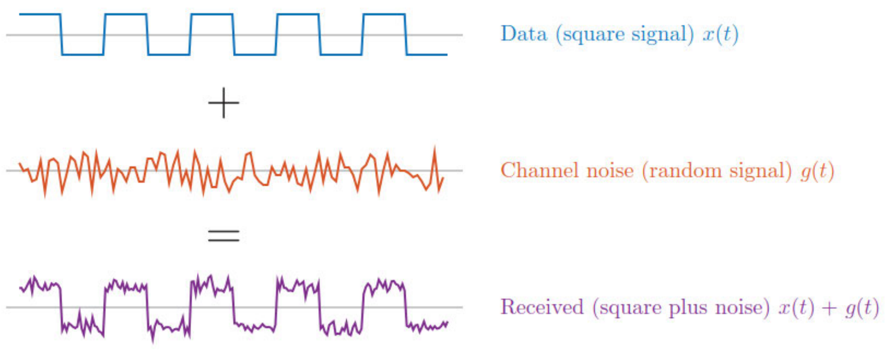

Data (square signal) $x(t)$
$+$
Channel noise (random signal) $g(t)$
$=$
Received (square plus noise) $x(t) + g(t)$

Figure 2.1: A binary signal with added AWGN (Leis 2018).

As modulation techniques become more complex and utilise less bandwidth, the susceptibility of a signal to misinterpretation due to BER increases. The BER is one parameter that can be measured to evaluate the performance of a modulation technique on a channel (Xiong 2006).

2.2.2 Amplitude Shift Keying (ASK)

Amplitude Shift Keying (ASK) is a modulation technique that encodes data onto a sinusoidal carrier signal by varying the amplitude. A basic form of ASK is On-Off Keying (OOK), where the presence or absence of a signal indicates binary 1s and 0s, respectively (Leis 2018), This can also be referred to as Binary ASK or B-ASK. ASK can be extended to M-ary ASK (M-ASK), where M amplitude levels allow for encoding $\log_2(M)$ bits per symbol, improving bandwidth efficiency. Figure 2.2 illustrates an example of 4-ASK modulation (Kishore & Rallapalli 2019, Lopes & Aguiar 2003).

Despite its simplicity and effectiveness in radio frequency (RF) communication, ASK presents several challenges in acoustic environments. The modulation is particularly vulnerable to signal degradation from environmental noise, reflections, and rapid signal power attenuation. These issues significantly limit the practicality of ASK for acoustic data transmission, leading to it being rarely utilised in such contexts (Lopes & Aguiar 2001).

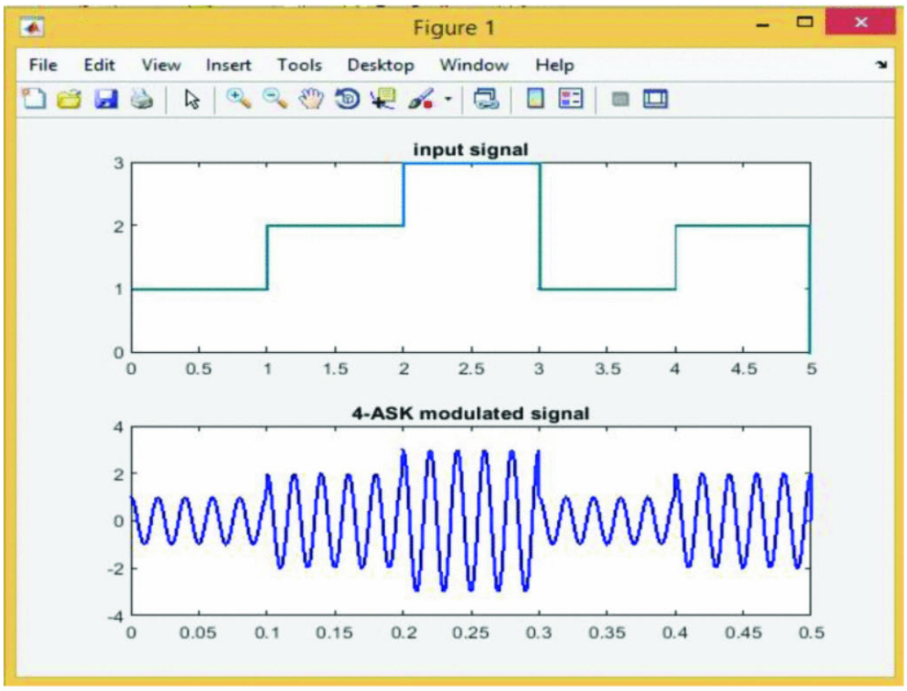


Figure 2.2: 4-ASK input signal and modulated signal (Kishore & Rallapalli 2019).

2.2.3 Frequency Shift Keying (FSK)

Frequency Shift Keying (FSK) is a modulation technique that transmits digital information through discrete frequency changes of a carrier wave. In contrast to amplitude-based modulation schemes, FSK's consistent amplitude renders it inherently resistant to amplitude-related noise, making it advantageous for applications such as telemetry and paging systems (Kishore & Rallapalli 2019). The utilisation of FSK in digital radio communication has been extensive, primarily due to its straightforward implementation and compatibility with non-linear amplifiers that boost power efficiency (Kishore & Rallapalli 2019, Leis 2018).

Binary FSK (BFSK) represents the simplest form of FSK, employing just two frequencies to encode binary '1s' and '0s'. To increase the transmission's bandwidth efficiency, M-ary FSK (M-FSK) can be implemented, where 'M' distinct frequencies allow for the encoding of $\log_2(M)$ bits per symbol. Although this method complicates the receiver design, it enables the accommodation of a higher data rate within the same bandwidth (Leis 2018).

Figure 2.3 illustrates an example of 4-FSK modulation, where four separate frequencies correlate to binary pairs, effectively doubling the data rate compared to BFSK. An essential metric for evaluating the performance of an FSK system is the Bit Error Rate

(BER), which generally decreases as the modulation level 'M' increases, denoting improved performance. However, this improvement in BER often comes at the cost of increased bandwidth requirements (Kishore & Rallapalli 2019).

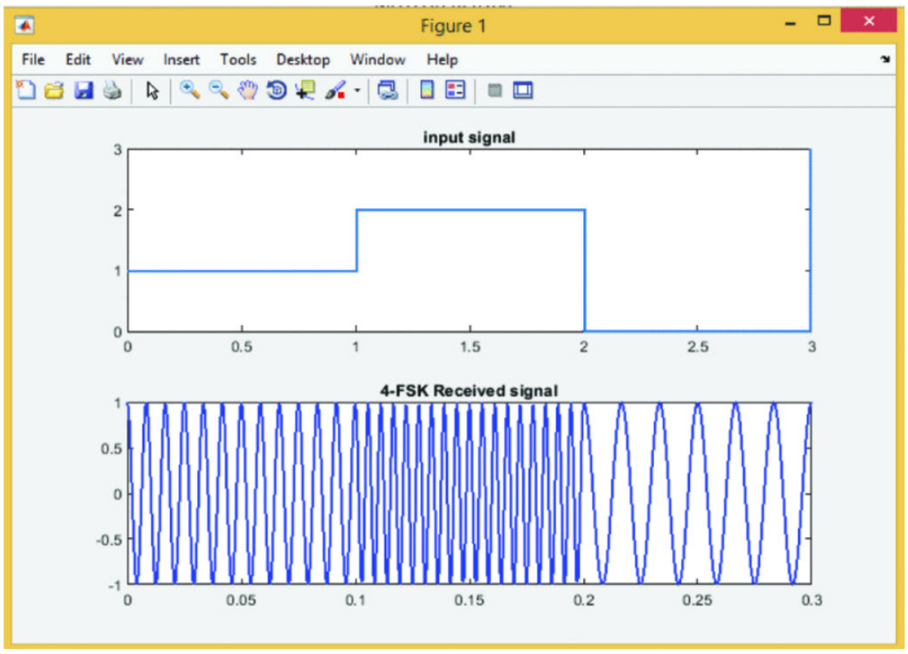


Figure 2.3: 4-FSK modulation illustrating three different levels (Kishore & Rallapalli 2019).

While FSK is inherently robust, making it a suitable choice for a range of communication scenarios, its spectral efficiency is less than optimal when compared to other digital modulation methods. This has led to FSK being overlooked in high-performance digital communication systems requiring efficient spectrum usage (Leis 2018, Lee, Kim, Choi & Choi 2015). Nonetheless, FSK's simplicity and reliability maintain its relevance in the design of contemporary communication systems (Indriyanto & Edward 2018*b*).

2.2.4 Phase Shift Keying (PSK)

Phase Shift Keying (PSK) is a modulation technique that encodes digital information by varying the phase of a carrier wave. Unlike ASK, which manipulates amplitude, or FSK, which alters frequency, PSK modifies the carrier phase to represent data bits, making it inherently more resistant to signal amplitude variations that can occur due to channel impairments (Leis 2018). A fundamental form of PSK is Binary Phase Shift Keying (BPSK), where two phases are utilised to represent binary '0' and '1'. This technique can be extended to M-ary PSK (M-PSK), allowing the encoding of $\log_2(M)$ bits per symbol by employing M distinct phase states, thus enhancing bandwidth efficiency.

The primary challenge in PSK is the exact recovery of the carrier phase at the receiver, which is crucial for correctly demodulating the received signal. This recovery process is complicated by the fact that many communication channels, especially those in mobile and aerial acoustic environments, exhibit time-varying characteristics that can induce phase shifts in the transmitted signal (Lee et al. 2015). For instance, in aerial acoustic communication, the phase of a transmitted signal can drift over time due to factors like sampling frequency offset in the microphone, making PSK less suitable for long-range communication without complex phase compensation algorithms (Lee et al. 2015).

As depicted in Figure 2.4, the phase of the carrier wave form is manipulated to indicate different bits, while the amplitude and frequency remain unchanged. The figure on the left shows a test signal of 0, 1 repeating, while the figure on the right shows a pseudo random binary sequence (PRBS). It can be noted where the bit doesn't change, the carrier system continues at the same phase.

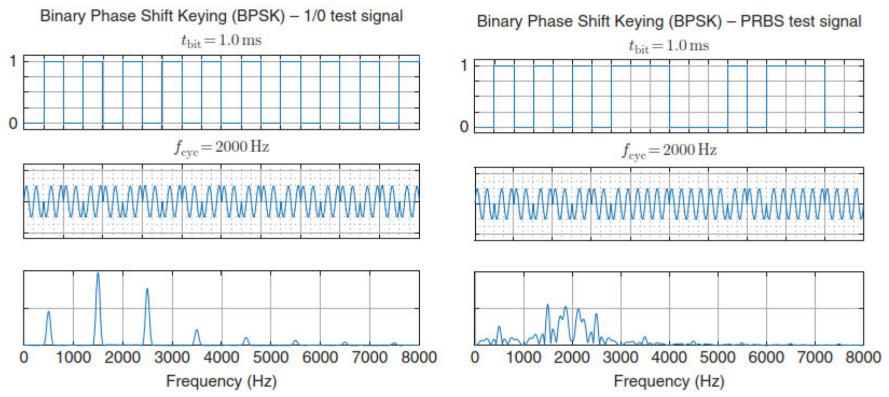


Figure 2.4: Ideal BPSK with test I/O signal and PRBS signal (Leis 2018).

Despite these challenges, PSK remains a popular choice for digital communication systems due to its efficient bandwidth utilisation and high data rate capabilities. Advanced forms of PSK, such as Quadrature Phase Shift Keying (QPSK) and higher-order M-PSK, are widely used in various applications, including satellite communication, WiFi, and cellular telephony, due to their ability to combine high spectral efficiency with relatively good resistance to noise and interference (Leis 2018).

In an ideal environment, PSK offers better power efficiency than ASK and a similar bit rate and BER to FSK proving a powerful modulating technique. One restriction of PSK is that the signal detection is much more complex than both FSK and ASK, meaning more computing power is necessary to demodulate the signal (Kishore & Rallapalli 2019). In relation to the project, the extra processing power required will be a significant disadvantage.

2.2.5 Other Modulation Techniques

There are a multitude of advanced digital modulation techniques, that use multiple techniques, or even multiple components from the same technique to increase bit transfer rates and efficiency.

Many of these are not suited to acoustic modulation as this project requires and therefore will not be investigated further.

Two advanced methods that could be used in acoustic coupling are:

Quadrature Amplitude Modulation (QAM) combines both Amplitude Shift Keying (ASK) and Phase Shift Keying (PSK) to transmit more bits per symbol by varying both the amplitude and the phase of the carrier signal. Given the vulnerabilities of ASK to signal degradation in noisy environments and PSK's sensitivity to phase disturbances as discussed in Sections 2.2.2 and 2.2.4, the use of QAM in this project may be impractical due to similar susceptibilities in an acoustic medium.

Orthogonal Frequency-Division Multiplexing (OFDM) excels in handling channel imperfections like multipath fading and interference through the use of closely spaced orthogonal sub-carriers. This characteristic renders OFDM an attractive option for wireless communications. Nonetheless, the implementation of OFDM in scenarios with constrained processing resources, such as devices aimed for acoustic data transmission, is hampered by its high computational demands and the necessity for accurate synchronisation to preserve sub-carrier orthogonality (Proakis & Salehi 2008). The complexity and power requirements associated with OFDM processing could be prohibitive for low-powered acoustic devices.

2.3 A Deeper Look into FSK for Acoustic Data Transmission

Frequency Shift Keying (FSK) stands out as a robust digital modulation technique, particularly beneficial for acoustic data transmission in environments where conventional communication methods may be unsuitable. This section explores the nuances of FSK modulation, its application in underwater and aerial acoustic data transmission, and the research that has showcased its adoption in challenging environments.

In FSK, digital data is transmitted through changes in the frequency of a carrier wave. A typical carrier signal for FSK modulation can be represented by the equation:

$$c(t) = A_c \sin(2\pi f_c t + \phi), \tag{2.2}$$

where A_c is the amplitude of the carrier wave, f_c is the carrier frequency, t is time, and ϕ is the phase of the signal at $t = 0$. The binary data that needs to be transmitted is encoded into a bit pattern, with each bit corresponding to a specific frequency of the carrier wave.

In binary FSK (BFSK), two frequencies represent binary '1' and '0', respectively. For example, a '1' may be represented by increasing the carrier frequency to f_{c1}, and a '0' by changing it to f_{c0} (Leis 2018). The frequency of the carrier signal varies according to the bit pattern of the message, resulting in a modulated signal that can be expressed as:

$$s(t) = \begin{cases} A_c \sin(2\pi f_{c1} t + \phi), & \text{for binary '1'}, \\ A_c \sin(2\pi f_{c0} t + \phi), & \text{for binary '0'}. \end{cases} \tag{2.3}$$

This modulation technique allows the transmission of digital information over a carrier wave by varying its frequency in accordance to the binary data being sent. The receiver demodulates the signal by detecting these frequency changes, reconstructing the original digital message (Xiong 2006).

2.3.1 Underwater Acoustic Modem Utilising FSK Modulation

The research by Indriyanto and Edward (Indriyanto & Edward 2018*b*) introduces an underwater acoustic modem designed for ultrasonic frequencies using FSK modulation. Their design aims to overcome the inherent obstacles of underwater communication, such as signal attenuation and multipath reflection, by leveraging the simplicity and reliability of FSK. The modem incorporates low-cost, waterproof ultrasonic sensors as transducers, highlighting a cost-effective approach to underwater communication. Performance evaluations demonstrate the modem's ability to transmit text data effectively through water, achieving a 1200bps rate with a minimal bit error rate (BER), showcasing the potential of FSK in underwater environments.

2.3.2 FSK in Aerial Acoustic Communication

Drawing insights from Lopes and Aguiar (Lopes & Aguiar 2003), the application of FSK in aerial acoustic communication provides an alternative to conventional wireless communication technologies. Their study explores the modulation of sound in air, employing FSK to encode data into multiple frequency tones, resulting in a method that is not only technically effective but also creates sounds that are pleasant or familiar to the human ear. This approach illustrates the versatility of FSK in adapting to different transmission media while maintaining the integrity of data transmission.

2.3.3 Performance Evaluation in AWGN Channels

Kishore and Rallapalli's (Kishore & Rallapalli 2019) comprehensive evaluation of digital modulation techniques, including FSK, in Additive White Gaussian Noise (AWGN) channels provides valuable insights into the performance of FSK relative to other modulation schemes. Their analysis indicates that while FSK's constant amplitude confers an inherent resistance to amplitude-related noise, making it advantageous in noisy environments, its spectral efficiency is lower compared to techniques like QAM. This trade-off underscores the importance of selecting the appropriate modulation technique based on the specific requirements and constraints of the communication environment.

2.3.4 Advancements and Practical Implementations

The practical implementation of FSK for acoustic data transmission has seen notable advancements, with studies like those by Ochi et al. (Ochi, Shimura, Sawa, Amitani & Nakamura 2000) demonstrating successful underwater transmission using FSK at ultrasonic frequencies. Their work, achieving reliable long-range communication in oceanic conditions, together with the exploration of FSK in aerial and underwater acoustic communication by (Lopes & Aguiar 2003) and (Indriyanto & Edward 2018*b*), underscores the significant potential of FSK in diverse acoustic environments.

2.3.5 Conclusion

FSK modulation emerges as a compelling choice for acoustic data transmission, offering a balance between simplicity, reliability, and performance in challenging environments. Whether applied in the depths of the ocean or the expanse of the air, FSK's adaptability and resilience to environmental factors make it a valuable tool in the advancement of acoustic communication technologies.

2.4 FSK Demodulation

2.4.1 Coherent vs Noncoherent Detection of BFSK

In frequency-shift keying (FSK) modulation, there are two primary methods of signal detection: coherent and noncoherent detection.

Coherent detection requires precise synchronisation of the receiver with both the frequency and phase of the transmitted signal. This synchronisation allows for accurate demodulation, providing better performance in terms of bit-error rate (BER) at lower signal-to-noise ratios (SNR) compared to noncoherent methods. However, coherent detection is sensitive to frequency and phase variations, making it more complex to implement in environments where signal distortion, multi-path fading, and noise are significant factors (Sklar 2021, Proakis & Salehi 2008).

Noncoherent detection, on the other hand, does not require this level of synchronisation, which simplifies the receiver design. Instead of relying on phase information, noncoherent detection only uses the frequency information of the received signal (Proakis & Salehi 2008). This makes it more robust in environments with frequency drift or phase noise, such as acoustic transmission, where echo and noise are common issues. However, the trade-off is a moderate degradation in performance, as noncoherent detection generally requires a higher SNR to achieve the same BER as coherent detection (Middlestead 2017).

The decision between using coherent or noncoherent detection in a system depends on the specific application requirements, such as complexity constraints, power availability, and the nature of the transmission environment. In this project, noncoherent detection is chosen due to its simplicity and tolerance to environmental variations in acoustic communication, despite the increased SNR requirement (Glenn 1966).

2.4.2 Envelope Detection

The demodulation methodology employs envelope detection, specifically linear and square-law detection, to identify the mark and space tones in the BFSK signal. Linear envelope detection, though not optimal for BFSK, offers a simple implementation that is tolerant of small frequency shifts (Sklar 2021). The bit-error performance of this method can be described by the following equation:

$$P_{be} = \frac{1}{2}\left[1 - \operatorname{erf}\left(\frac{m_z}{2\sigma_z}\right)\right] \tag{2.4}$$

where m_z and σ_z are the mean and standard deviation of the Gaussian-distributed output from the low-pass filter (Middlestead 2017). In cases where the signal-to-noise ratio (SNR) is high, square-law detection is employed, which offers better noise robustness and improved bit-error performance:

$$P_{be} = \frac{1}{2}\left[1 - \operatorname{erf}\left(\frac{E_b}{2N_0WT_b}\right)\right] \tag{2.5}$$

2.4.3 Frequency Discriminator

For demodulation, a frequency discriminator is used to distinguish between the mark and space tones. This approach provides a straightforward means of detecting frequency shifts without the need for precise phase information. The output from the discriminator is smoothed using a low-pass filter to accurately determine bit transitions. This method is particularly effective in environments with small frequency shifts (Middlestead 2017).

2.4.4 Bit-Error Rate Analysis

The bit-error rate (BER) performance of the demodulation system is analysed using the theoretical models described by Middlestead (Middlestead 2017). The following equation is used to compute the BER for noncoherent detection under varying SNR conditions:

$$P_{be} = \frac{1}{2}\left[1 - \mathrm{erf}\left(\frac{\pi}{8}\frac{E_b}{N_0 W T_b}\right)\right] \tag{2.6}$$

These theoretical results could serve as a baseline for comparing the performance of a model under different conditions, including variations in baud rate, microphone sensitivity, and environmental noise.

2.4.5 Fast Fourier Transform (FFT)

The Fast Fourier Transform (FFT) is an efficient algorithm to compute the Discrete Fourier Transform (DFT) of a signal (Cooley & Tukey 1965). While the DFT converts a time-domain signal into its frequency-domain representation, the FFT dramatically improves the computational efficiency of this process. The DFT is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j(2\pi kn/N)} \tag{2.7}$$

where $x(n)$ is the time-domain signal, $X(k)$ is the frequency-domain representation, N is the number of samples, and k represents the frequency bin. The FFT achieves its efficiency through a divide-and-conquer approach. Instead of computing each output separately (as in the DFT, which has a complexity of $O(N^2)$), the FFT recursively divides the computation into smaller parts, reducing the overall complexity to $O(N \log N)$ (Proakis & Salehi 2008). This significant reduction in computational complexity makes the FFT much faster than the direct DFT calculation, especially for large N.

The FFT is widely used in signal processing applications such as FSK demodulation to detect frequency components. Since FSK uses shifts between specific frequencies to encode data, the FFT can efficiently extract these frequencies from the signal by analysing its spectral content (Leis 2011, Lyons 2010).

Advantages:

- Simultaneous Frequency Analysis: The FFT can analyse multiple frequency components simultaneously, making it particularly useful when dealing with signals composed of multiple frequencies or in the presence of noise.
- Efficiency: The computational complexity of the FFT is $O(N \log N)$, which is significantly lower than directly computing the DFT (Cooley & Tukey 1965).

Disadvantages:

- Computational Complexity for Real-Time Applications: While efficient, the FFT may still be too computationally expensive for real-time applications, particularly when only a few frequencies need to be detected.
- Resolution vs. Time Trade-off: The frequency resolution is inversely proportional to the time window size, which may limit its performance in real-time scenarios.

2.4.6 Goertzel Algorithm

The Goertzel algorithm is a computationally efficient method designed to detect specific frequency components in a signal. Unlike the FFT, which computes the entire frequency spectrum, the Goertzel algorithm focuses on a single frequency bin (Lyons 2010). It is particularly useful in FSK demodulation, where only a few known frequencies need to be detected.

The Goertzel algorithm can be expressed as:

$$y[n] = x[n] + 2\cos(2\pi k/N)y[n-1] - y[n-2] \tag{2.8}$$

where $x[n]$ is the input signal, $y[n]$ is the output, k represents the target frequency bin, and N is the total number of samples.

Advantages:

Table 2.1: Single-Bin DFT Computational Comparisons (Lyons 2010)

Method	**Real multiplies**	**Real additions**
Single-bin DFT	$4N$	$2N$
FFT	$2N \log_2 N$	$N \log_2 N$
Goertzel	$N+2$	$2N+1$

- Optimised for Single Frequency Detection: It is computationally efficient when detecting specific frequencies, as in FSK demodulation.
- Low Memory Usage: The Goertzel algorithm operates in small chunks, using significantly less memory than the FFT.

Disadvantages:

- Limited Spectrum Analysis: The algorithm needs to be executed separately for each frequency, which limits its ability to analyse a wide range of frequencies simultaneously.

2.4.7 Bandpass Filters

A bandpass filter allows signals within a specific frequency range to pass while attenuating signals outside that range. In FSK demodulation, bandpass filters are used to isolate the frequency components corresponding to the binary states of the transmitted signal.

The transfer function of a digital bandpass filter (e.g., FIR or IIR) can be expressed as:

$$H(z) = \sum_{n=0}^{N-1} b[n] z^{-n} \tag{2.9}$$

where $b[n]$ represents the filter coefficients and z^{-n} represents the delay.

Types of Bandpass Filters:

- *Analog Filters*: Traditional analog filters, constructed using resistors, capacitors, and inductors, are fast but less flexible.

- *Digital Filters (FIR/IIR)*: Digital FIR filters provide linear phase response, while IIR filters are more computationally efficient but may introduce phase distortion.

Matched filters can also be used to maximise the signal-to-noise ratio (SNR) by matching the filter's response to the expected frequency of the incoming signal.

Advantages:

- Effective for isolating specific frequencies.
- Low computational load in analog systems.

Disadvantages:

- In digital systems, high-order filters may require more computational power (Middlestead 2017).

2.4.8 Zero-Crossing Detection

Zero-crossing detection is a simple method for detecting frequency changes by measuring the time between consecutive zero crossings of the signal (Sklar 2021). In FSK, this method can detect frequency shifts by calculating the zero-crossing rate, which directly corresponds to the frequency of the signal.

Let T_z represent the time between zero-crossings. The frequency f can be estimated as:

$$f = \frac{1}{2T_z} \tag{2.10}$$

Advantages:

- Simplicity: It is easy to implement in both hardware and software.
- Low Computational Load: This method is computationally efficient, making it ideal for real-time applications.

Disadvantages:

- Noise Sensitivity: The method is sensitive to noise, which can lead to erroneous zero-crossings and, consequently, incorrect frequency estimates.
- Limited Accuracy: It is less accurate for closely spaced frequencies or in the presence of significant signal distortion.

2.4.9 Phase-Locked Loop (PLL)

A Phase-Locked Loop (PLL) is a feedback control system that locks the phase of a local oscillator to the phase of an input signal. In FSK demodulation, the PLL tracks the frequency shifts in the incoming signal and adjusts its local oscillator accordingly to demodulate the signal.

The basic mathematical model for a PLL involves a phase detector, loop filter, and a voltage-controlled oscillator (VCO). The VCO generates an output signal with a phase $\theta_{out}(t)$, which is adjusted to minimise the phase difference $\theta_{in}(t) - \theta_{out}(t)$ (Proakis & Salehi 2008).

Advantages:

- Robust Against Frequency Drift: The PLL can adapt to slight variations in the input frequency, making it robust against frequency drift.
- Noise Handling: The feedback loop allows the system to reject high-frequency noise and maintain lock on the signal.

Disadvantages:

- Complexity: More complex than zero-crossing detection and may introduce delay due to the lock time required by the PLL.

2.4.10 Conclusion

The literature around demodulation techniques provides evidence that there is a clear compromise between robustness, resolution, and computational power.

It is important to take into account the requirements of the project's ultimate implementation goals when deciding on which demodulation technique will be used. The ideal technique will require:

- **Low computational power** - the embedded device may only have a small amount of memory with which to process the data.
- **Moderate to high speed** - the actual speed of acoustically transferring the data will be a constraint in this project, for this reason, it is important that the processing doesn't take an excessively long time.
- **High resilience to noise** - the environment that the data transfer will take place will possibly have machinery and people operating in reasonable proximity so the system should be robust and the frequency resolution be able to distinguish the target frequencies from other acoustics in the area.
- **Low implementation complexity** - the model will need to be implemented on different platforms, using different coding languages, therefore should be easy to implement on different systems.

Table 2.2: Comparison of FSK Demodulation Techniques

Technique	**Speed**	**Robustness**	**Computational Power**	**Frequency Resolution**	**Implementation Complexity**
Envelope Detection	High	Moderate	Low	Low	Low
FFT	Moderate	High	High	High	Moderate
Goertzel Algorithm	High	Moderate	Low	Moderate	Low
Bandpass Filters	High	High	Moderate	Moderate	Moderate
Zero-Crossing Detection	Very High	Low	Very Low	Low	Very Low
PLL	Moderate	High	Moderate	High	High

From Table 2.2, the technique that stands out, covering the most necessary parameters the best, is the Goertzel Algorithm. It offers a good balance of high speed, moderate robustness, low computational power requirements, moderate frequency resolution, and low implementation complexity. Envelope Detection and Zero-Crossing Detection also merit consideration for their simplicity and low computational requirements, though lower frequency resolution may limit their ability to distinguish target frequencies from ambient noise in the operational environment.

Based on this analysis, the Goertzel Algorithm will be the primary focus for implementation and testing in this project, with Envelope Detection as a potential alternative if higher noise resilience is required.

2.5 Hardware

The development of an embedded device prototype for this project will require consideration of several factors, including processing power, power consumption, connectivity options, and compatibility with various sensors and devices. The prototype aims to closely resemble a real-world application scenario, ensuring both feasibility and practicality in deployment. Among the many choices available for the embedded device's processor, two likely options have been evaluated: the Raspberry Pi Microprocessor and the Arduino Microcontroller (Arduino Uno R3 and R4 have been included). This section analyses these options, evaluating their strengths and weaknesses in the context of the project requirements to ascertain the most suitable platform for the prototype development.

Table 2.3: Comparison of specifications between Arduino Uno R4, Raspberry Pi Zero & Arduino Uno R3 (Foundation 2024, Arduino 2024*b*, Arduino 2024*c*).

Parameter	**Raspberry Pi Zero**	**Arduino Uno R4**	**Arduino Uno R3**
Price	\$29.50	\$49.95	\$34.95
Microcontroller	Broadcom BCM2835, Single-core Cortex-A7 (ARMv6) 32-bit SoC	RA4M1 32-Bit	ATmega328P 8-Bit
Clock Speed	1 GHz	48 MHz (R4 WiFi ESP32 @ 240 MHz)	16 MHz
External Interrupts	N/A	14	2
USB Connector	Mini HDMI, Micro USB OTG	USB-C (also HID)	USB-B

Continued on next page

Table 2.3 – *Continued from previous page*

Parameter	**Raspberry Pi Zero**	**Arduino Uno R4**	**Arduino Uno R3**
Real Time Clock (RTC)	NO	YES	NO
I/O Pin Source/Sink Current	N/A	8mA	20mA
Digital to Analog Converter (DAC)	N/A	12-Bit	NONE
Analog to Digital Converter (ADC)	N/A	14-Bit	10-Bit
CAN Bus Interface	N/A	YES	NO
WiFi & Bluetooth	YES	YES (R4 WiFi Only)	NO
Flash Memory	Micro-SD card slot (for OS and storage)	256kB	32kB
SRAM	512MB LPDDR2 SDRAM	32kB	2kB

2.6 Raspberry Pi vs. Arduino: A Comparative Overview

2.6.1 Raspberry Pi

- **Overview:** The Raspberry Pi is a series of small single-board computers developed for promoting basic computer science education and various applications.
- **Processing Power:** Equipped with ARM processors capable of executing complex computations and supporting software requiring substantial processing power.
- **Power Consumption:** While recent models have improved, Raspberry Pis are not the most power-efficient devices. Power optimisation techniques may be necessary for battery-powered projects.
- **Connectivity:** Offers extensive connectivity options including Ethernet, Wi-Fi,

Bluetooth, and multiple USB ports, making it suitable for projects requiring network connectivity or peripheral integration.

- **Compatibility with Sensors:** Supports a wide range of sensors and devices through GPIO pins, USB ports, and various interfaces like I2C, SPI, and UART, allowing for versatile sensor integration.

2.6.2 Arduino

- **Overview:** Arduino is an open-source electronics platform based on easy-to-use hardware and software, designed for creating interactive objects or environments.
- **Processing Power:** Equipped with simpler microcontrollers compared to Raspberry Pi, suitable for projects focused on sensor interaction and basic device control.
- **Power Consumption:** Highly power-efficient, ideal for battery-powered applications, offering advantages for mobility and extended operational periods.
- **Connectivity:** While some models offer USB, Wi-Fi, and Bluetooth, connectivity options are generally more limited compared to Raspberry Pi, requiring additional modules for extensive networking.
- **Compatibility with Sensors:** Analog and digital input/output pins enable direct interaction with various sensors, making it well-suited for physical computing and sensor-based data collection projects.

2.6.3 Conclusion

The choice between Raspberry Pi and Arduino depends on specific project requirements and limitations. Raspberry Pi excels in complex computations, extensive data processing, and network connectivity, while Arduino excels in power efficiency, sensor integration, and physical computing applications. The choice should ensure the embedded device remains as close to the capabilities that a real embedded device may have. One very large advantage the Arduino Uno R4 possesses over its counterparts is the fact it has both an Analog to Digital Converter and a Digital to Analog Converter, this will be an advantageous addition for digital signal processing.

Chapter 3

Methodology

3.1 Reason for Research

As discussed in Chapter 1, the proliferation of IoT devices has facilitated remote data acquisition, often relying on WiFi and Bluetooth connections. However, these wireless technologies can present security vulnerabilities that may be unacceptable in certain environments. In situations where access to traditional wireless communications is restricted or denied, a secure alternative for remote data acquisition becomes crucial.

This research explores the potential of acoustic data transfer as a viable solution in environments where conventional wireless methods are unsuitable due to security concerns, technical limitations, or system malfunctions. While acoustic data transfer is not a new concept, its application in modern contexts, particularly as a secure alternative to WiFi and Bluetooth, warrants investigation.

The study will focus on:

1. Developing a MATLAB model to simulate acoustic data exchange between a portable device and an embedded system.
2. Evaluating the theoretical data transfer speeds achievable through acoustic methods using the MATLAB simulation.
3. Assessing the robustness of simulated acoustic data transfer under various conditions, considering different modulation techniques and simulated equipment quality.

4. Analysing the feasibility and potential applications of acoustic data transfer as a secure alternative to traditional wireless methods based on simulation results.

5. Laying the groundwork for future development of physical prototypes, including both the portable device and the embedded system.

By examining these aspects through simulation, this research aims to determine whether acoustic data transfer merits further development into a functional prototype for use in security-sensitive or wireless-restricted environments. The results will inform future work on physical implementation and real-world testing.

3.2 How This Project Will Be Undertaken

This project will undertake several different aspects:

- **MATLAB Model**: The proof of concept for this project will be based on the MATLAB simulation model. Whilst the ultimate goal will be to have a working prototype of a portable device and an embedded device, the testing of different parameters and scenarios will be undertaken within the MATLAB model.
- **Portable Device**: Where the embedded device will be an emulation, the portable device will be akin to what would actually be used in practice.

  This will consist of a mobile phone with access to a speaker and a microphone. The portable device should have sufficient processing power, a high-quality microphone and speaker, and the ability to run custom software. The mobile phone will access an API, written in JavaScript, which will establish a communication link with the embedded device and allow data to be bidirectionally transferred. The phone will emulate the portable device used by field personnel, ensuring it meets practical usability requirements.
- **API**: The API will facilitate the communication between the portable and embedded devices. It must support initialisation, data encoding and decoding, data transmission, error checking, and termination protocols. Parameters to be defined include baud rate, modulation scheme, security measures, and data format. The API should be intuitive and user-friendly, allowing personnel with minimal training to operate the system efficiently.
- **Embedded Device**: This will consist of an Arduino R4, utilising an audio amplifier and speaker module, and a microphone sound sensor module to emulate the embedded device. The Arduino R4 is chosen for its advanced features including the Digital to Analog Converter (DAC) (Arduino 2024*a*) which is crucial for processing audio signals. The microphone module must be suitable for capturing acoustic signals with sufficient fidelity, ensuring accurate and reliable data transmission.. The speaker module will be used to transmit data acoustically. Initially, we will use Binary Frequency Shift Keying (BFSK) with two frequencies, and may expand to 4-FSK to increase data rates. The embedded device may also be equipped with a memory card to store information for later retrieval, similar to how some instru-

ments log data.

- **Arduino Program**: The Arduino program is responsible for executing the low-level operations that enable communication with the portable device. Much like the API does for the portable device, the Arduino program handles the handshake operation, data encoding and decoding, data transmission, error checking, and termination protocols for the embedded device.

3.3 Data Encoding Method

The data encoding method selected for this project is Frequency Shift Keying (FSK). FSK was chosen due to its robustness in noisy environments and its simplicity of implementation on both the hardware and software levels. As stated in Chapter 2, FSK has a consistent amplitude, improving its robustness to amplitude related noise, as the receiver is only seeking differences in frequency of the signal. This gives it an advantage for applications such as telemetry and paging systems (Kishore & Rallapalli 2019). This attribute gives FSK a clear advantage over techniques such as ASK for this project, increasing the potential to maintain data integrity in potentially noisy environments.

BFSK will be implemented initially, which uses two distinct frequencies to represent binary '0' and '1'. If time and resources permit, the project may expand to 4-FSK to investigate increased data transmission rates. The benefits of M-ary FSK (MFSK), where multiple frequencies allow for encoding more bits per symbol, have been highlighted in chapter 2 for increasing bandwidth efficiency, despite the added complexity in receiver design necessary (Leis 2018). Figure 2.3 from the literature review and Figure 3.2 illustrate examples of 4-FSK modulation, where four separate frequencies correlate to binary pairs, effectively doubling the rate compare to BFSK. An example of BFSK is shown in Figure 3.1, demonstrating the simplicity of only using two frequencies.

Comparing the two modulation techniques, paying close attention to the spectrum plot, it is clear there is a much more obvious distinction between the frequencies in BFSK than 4-FSK, the side lobes of the FSK plot are easy to distinguish, where in 4-FSK, the lower frequency resolution coupled with effect of the side lobes creates a much less distinct bin magnitude. The example in Figure 3.1 uses the higher frequency twice that of the low frequency. The distinction between the two frequencies can be increased further, by

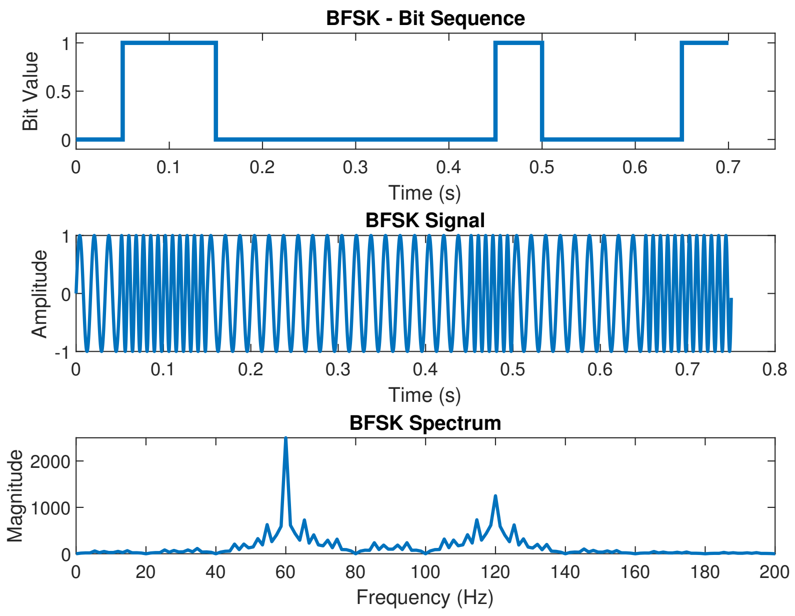


Figure 3.1: BFSK modulation illustrating the simplicity of two frequencies.

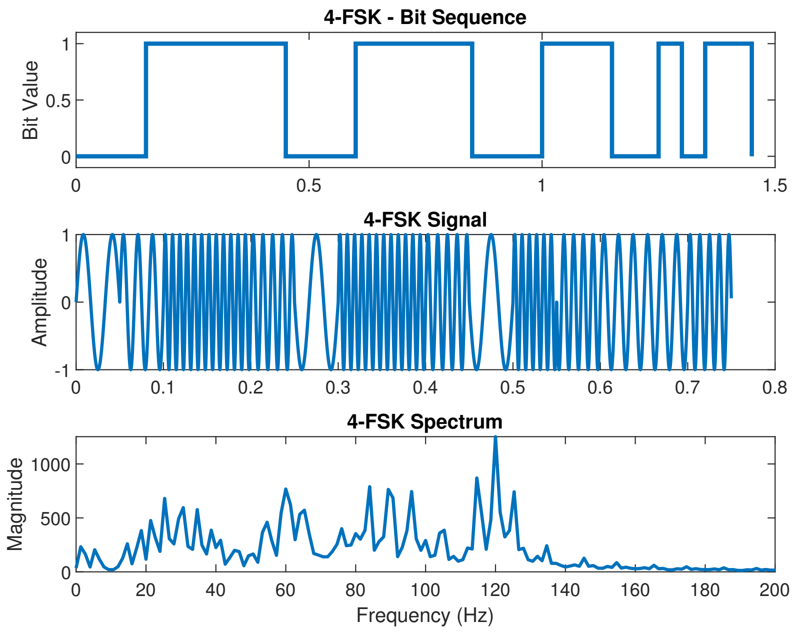


Figure 3.2: 4-FSK modulation illustrating the relatively unclear distinction of frequencies in the spectrum plot.

widening the gap between a high and a low frequency. This will result in more accurate message transmission, though, as stated, will mean a slower bit rate than using a higher order FSK. Considering the environments that the embedded device could be located, and the limited performance of the microphones and speakers that could practically be used on the equipment, accuracy is likely to be of high importance.

MATLAB simulations can be used to compare the performance of BFSK with other modulation schemes such as ASK and PSK, focusing on bit error rates (BER) with varying rates of signal-to-noise ratios (SNR). According to Kishore and Rallapalli (Kishore & Rallapalli 2019), while FSK's constant amplitude provides intrinsic resistance to amplitude-related noise, its spectral efficiency is lower compared to techniques like QAM.

To improve efficiency while using equipment with limited processing power, like the Arduino, and to improve the processing speeds of the equipment, a lookup table (LUT) will be used to generate the sine wave frequencies required for FSK modulation. This will avoid using excess processing power and allow encoding and decoding at a speed closer to real time (Leis 2018).

By using a LUT, high data transmission rates can be maintained using low power equipment which will increase the viability of the project being put into practice in industry. The use of FSK will provide reliable communication in noisy environments to ensure accurate results from the different trials of data transmission.

3.4 Data Demodulation

The demodulation technique chosen for this project is the Goertzel algorithm, which aligns well with the FSK modulation scheme discussed in Section 3.3. As explored in Chapter 2, the Goertzel algorithm offers an optimal balance of robust demodulation, low computational requirements, and good frequency resolution, making it particularly suitable for this project's objectives.

3.4.1 Goertzel Algorithm Implementation

The Goertzel algorithm is especially efficient for detecting specific frequencies, which is ideal for FSK demodulation where predetermined frequencies representing binary '0' and '1' (or multiple frequencies in the case of M-ary FSK). Its computational efficiency makes it well-suited for implementation on embedded devices with limited processing power, such as the Arduino platform used in this project. The algorithm can be expressed mathematically as:

$$y[n] = x[n] + 2\cos(2\pi k/N)y[n-1] - y[n-2] \tag{3.1}$$

where $x[n]$ is the input signal, $y[n]$ is the output, k represents the target frequency bin, and N is the total number of samples.

3.4.2 Advantages for FSK Demodulation

The Goertzel algorithm offers several advantages for FSK demodulation in this project:

- **Computational Efficiency**: As shown in Table 3.1, the Goertzel algorithm requires fewer computations compared to FFT for single frequency detection, making it ideal for real-time processing on embedded systems.
- **Frequency Selectivity**: It allows precise detection of the specific frequencies used in our FSK scheme, improving noise resistance.
- **Low Memory Usage**: The algorithm operates on small chunks of data, reducing memory requirements compared to FFT-based methods.
- **Flexibility**: It can be easily adapted for both BFSK and 4-FSK implementations, aligning with our potential expansion plans.

Table 3.1: Computational Comparison: Goertzel vs. Other Methods

Method	**Real Multiplications**	**Real Additions**
Single-bin DFT	$4N$	$2N$
FFT	$2N \log_2 N$	$N \log_2 N$
Goertzel	$N + 2$	$2N + 1$

3.5 MATLAB Model Overview

The MATLAB model serves as a tool for simulating and analysing the FSK modulation and Goertzel algorithm demodulation process, the MATLAB scripts can be found in Appendix D. This model allows for testing of various parameters and scenarios before physical implementation. The key components of the MATLAB model include:

3.5.1 Signal Generation

The model begins by accessing a .txt file containing the message to be transmitted. This text is then converted from ASCII to binary data, creating a stream of bits for modulation. It should be noted, the message is expected to be in JSON format for field communication, the .txt file will be used as part of the simulation for easy human assessment

The message used for testing purposes will be a 125 character, 1000 bit message, This will give an error accuracy of 0.1%. The message is as follows: "Hello, World rose in 1976, by B. Kernighan, before that, Alexander G Bell in 1876: Mr. Watson, come here, I want to see you.!" The "!" will be used as an end of message character, this can be replaced with a postamble in future works.

A preamble is implemented to aid in signal detection and synchronisation. The preamble consists of a specific bit pattern that precedes the actual message data. This preamble is crucial for establishing bit boundaries and synchronising the receiver with the incoming signal (Sklar 2021).

3.5.2 FSK Modulation

The model implements Binary Frequency Shift Keying (BFSK) modulation. Two distinct frequencies ($f1$ and $f2$) are defined to represent binary '0' and '1' respectively. The selection of these frequencies is crucial and should consider the Nyquist theorem to avoid aliasing.

The Nyquist-Shannon theorem states that the minimum sampling frequency must be at least twice that of the highest frequency component present in the original signal to ensure accurate signal reproduction (Leis 2011).

A sampling rate (fs) is defined, set at 8 kHz to balance between signal quality and computational load. While the Arduino R4 can handle faster sampling rates, higher rates increase computational demands. Having a lower sampling rate does limit the frequencies able to be used; if necessary, this could be increased to further test different frequencies' resilience to noise. The chosen sampling rate must satisfy the Nyquist criterion ($fs > 2 * fmax$, where $fmax$ is the highest frequency used in the FSK modulation) to prevent aliasing.

The baud rate, which determines the symbol transmission speed, is also defined. This parameter directly affects the bit duration (T) through the relationship $T = 1/baudrate$. A higher baud rate results in a shorter bit duration, potentially increasing transmission speed but also increasing the risk of inter-symbol interference and reducing the system's tolerance to noise and timing errors.

3.5.3 Signal Processing

To simulate real-world conditions and test signal robustness, the model incorporates Additive White Gaussian Noise (AWGN). AWGN is a basic noise model used in information theory to mimic the effect of many random processes that occur in nature (Proakis & Salehi 2008).

3.5.4 Signal Normalisation and Output

Before output, the signal is normalised to prevent clipping. The processed signal is then written to a .wav file for further analysis or playback.

3.5.5 Demodulation Process

The demodulation process begins with recording or importing the modulated audio signal. The model then implements the Goertzel algorithm for frequency detection, which is particularly efficient for identifying specific frequencies in FSK signals (Lyons 2010).

Sliding Window Approach

A sliding window approach is used for preamble detection and subsequent bit decoding. This method involves moving a window of samples across the received signal, analysing each window for the presence of the preamble or data bits. The window size is typically set to match the expected length of the preamble or a single bit, depending on the stage of demodulation. The step size of the window can be adjusted to balance between accuracy and computational efficiency. A smaller step size (e.g., one sample) provides more precise detection but requires more computational power, while a larger step size (e.g., half a bit duration) is faster but may miss the optimal detection point.

Preamble Detection and Symbol Training

The preamble serves a dual purpose: it allows the receiver to detect the start of the message and acts as a "training sequence" for symbol detection. By analysing the known preamble pattern, the receiver can calibrate its detection thresholds for the subsequent data bits.

During preamble detection, the sliding window compares the received signal against the expected preamble pattern. A correlation or matching score is computed for each window position. When this score exceeds a predefined threshold, the preamble is considered detected, and the receiver can synchronise its timing for subsequent bit detection.

Goertzel Algorithm Implementation

For each window of samples corresponding to a potential bit, the Goertzel algorithm is applied to detect the presence of the two FSK frequencies ($f1$ and $f2$). The algorithm computes the energy at these specific frequencies, allowing for efficient detection without the need for a full FFT. Key parameters in the demodulation process include:

- Symbol length calculation based on the sampling rate and baud rate
- Definition of frequency bins for the Goertzel algorithm
- Implementation of the sliding window approach for preamble and bit detection

- Bit decision making based on energy levels at the target frequencies

Message Display and Bit Pattern Analysis

After demodulation, the recovered message is displayed for visual inspection. This allows for a quick assessment of the demodulation quality and can help identify any obvious errors or patterns. The bit pattern is then further analysed to assess for accuracy of the received message. During this analysis, various metrics can be computed:

- Bit Error Rate (BER): Comparing the received bit pattern with the known transmitted pattern to calculate the proportion of incorrectly received bits.
- Signal-to-Noise Ratio (SNR): Estimating the SNR based on the strength of the detected signals versus the noise floor.
- Timing Offset: Analysing any drift in bit timing over the course of the message, which could indicate synchronisation issues.

Error Detection and Correction

Depending on the implementation, error detection and correction mechanisms can be incorporated. This might include:

- Parity bits: Simple error detection by including an extra bit to make the number of 1s even or odd.
- Cyclic Redundancy Check (CRC): More robust error detection by appending a checksum to the message.
- Forward Error Correction (FEC): Techniques like Reed-Solomon codes that allow for error correction without re-transmission.

This MATLAB model provides a flexible platform for optimising the acoustic data transmission system before hardware implementation. It allows for iteration of different modulation parameters, demodulation techniques, and noise conditions, providing insights for the physical prototype design.

3.6 Analysis and Testing

3.6.1 Microphone Testing

Microphone testing forms a crucial component in assessing the feasibility of this prototype. The ideal microphone should offer reasonable quality without being prohibitively expensive. While the microphones under test are relatively basic, performance variations are anticipated.

Test Signal

The test will utilise a section of the preamble: [1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1]. Preliminary testing has shown this sequence to produce intriguing results, making it an excellent basis for comparison.

Microphones Under Test

HP Victus 16-d1xxx Laptop Built-in Microphone This dual microphone array is designed to capture room acoustics. Although the recording environment will be quiet, it is not entirely soundproof, which may influence the recording quality.

Sennheiser PXC 550 Headphones Microphone This integrated microphone is also designed for ambient sound capture.

External Microphone (via PXC 550 headphones) This dedicated close-talking microphone is expected to be the best performer due to its design for capturing nearby voice input while minimising surrounding noise.

Testing Considerations

It is important to note that Windows has built-in ”Audio Enhancements” that can filter out the signals being captured. Unawareness of this feature could lead to significant time spent troubleshooting the reason the target frequencies are not being captured.

Testing Methodology

1. **Signal Generation:** The test signal will be played through a calibrated speaker at a consistent volume and distance from each microphone.
2. **Recording Environment:** Tests will be conducted in a quiet, though not completely soundproof, room to simulate realistic conditions.
3. **Software Configuration:** Windows audio enhancements will be disabled to prevent filtering of the target signals.
4. **Performance Metrics:** Each microphone will be evaluated based on:
   - Signal-to-Noise Ratio (SNR)
   - Frequency response accuracy
   - Bit Error Rate (BER) in the decoded signal
5. **Waveform Analysis:** Time-domain and frequency-domain plots will be generated for visual comparison.

Expected Outcomes

The external microphone is anticipated to outperform the integrated options due to its specialised design for close-range voice capture. However, the actual results may reveal unexpected strengths or weaknesses in each microphone's performance.

This testing methodology will provide a comprehensive comparison of the microphones' performance in capturing the FSK signals, helping to determine the most suitable option for the future prototype.

3.6.2 Distance from Source Testing

Testing the performance of the system at different distances between the source and the microphone has real world application for the demodulation. In the field, obstructions may occur, meaning the distance between devices can not always be controlled. Assessing the performance at different distances will assist in outlining necessary conditions for data transfer, for standards of procedure documentation.

Testing Methodology

1. **Signal Generation:** The test signal will be played through a calibrated speaker at consistent volume at distances around <0.1 m, 0.5 m, 1.0 m, 2 m, and 5 m from the microphone. These distances may be updated after preliminary results. Where the microphone testing saw only a small portion of the preamble used for testing, the distance testing and subsequent tests will use the full, 1000 bit message.
2. **Recording Environment:** Tests will be conducted in a large, open indoor space to minimise reflections while simulating realistic conditions.
3. **Microphone Setup:** The best-performing microphone from previous tests will be used for all distance measurements.
4. **Performance Metrics:** At each distance, the following will be evaluated:
    - Signal-to-Noise Ratio (SNR)
    - Frequency response accuracy
    - Bit Error Rate (BER) in the decoded signal
5. **Waveform Analysis:** Time-domain and frequency-domain plots will be generated for visual comparison across distances.

Expected Outcomes

Signal amplitude is expected to degrade as distance increases, potentially affecting SNR and frequency response. This will assess the systems ability to demodulate signals that are degraded, it is expected the BER will remain very low, due to the robustness of the demodulation. This testing methodology will provide a comprehensive comparison of the system's performance at various distances, helping to determine the effective range for reliable FSK signal transmission and reception.

3.6.3 Data Transmission Speed Testing

As noted previously, acoustic data transmission is not a high speed form of transmission and this prototype would be used as an alternate method when there were few other options. It is still important that the speeds are able to reach a reasonable rate to ensure the feasibility and further investigation of the prototype.

Testing Methodology

1. **Signal Generation:** Test signals will be generated at varying speeds: 25bps, 50bps, 100bps, 200bps, and 400bps. Additional speeds may be tested as necessary to sufficiently stress test the demodulation process.
2. **Recording Environment:** Tests will be conducted in a controlled, quiet environment to minimise external interference.
3. **System Setup:** The optimal microphone and distance configuration from previous tests will be used for all speed measurements.
4. **Performance Metrics:** For each transmission speed, the following will be evaluated:
    - Signal-to-Noise Ratio (SNR)
    - Frequency response accuracy
    - Bit Error Rate (BER) in the decoded signal
5. **Waveform Analysis:** Time-domain and frequency-domain plots will be generated for visual comparison across different transmission speeds.

Expected Outcomes

As transmission speed increases, signal quality may degrade, potentially affecting SNR, frequency response accuracy, and BER. Higher speeds are expected to result in reduced spectral resolution, which could impact the system's ability to distinguish between frequency states.

This testing methodology will provide a comprehensive comparison of the system's performance at various data transmission speeds, helping to determine the optimal balance between speed and reliability for efficient FSK signal transmission and reception.

3.6.4 Noise Resilience Testing

Understanding the system's performance under various noise conditions is crucial for assessing its viability in real-world environments. This test aims to evaluate the robustness of the FSK system against different levels of Additive White Gaussian Noise (AWGN).

Testing Methodology

1. **Signal Generation:** A consistent FSK signal will be generated and combined with AWGN at different Signal-to-Noise Ratio (SNR) levels: 10dB, 20dB, 30dB, and 40dB.
2. **Noise Introduction:** AWGN will be digitally added to the clean FSK signal to ensure precise control over the SNR levels.
3. **Recording Environment:** Tests will be conducted in a controlled, quiet environment to minimise external interference.
4. **System Setup:** The optimal microphone, distance, and speed configuration from previous tests will be used for all noise measurements.
5. **Performance Metrics:** For each SNR level, the following will be evaluated:
    - Bit Error Rate (BER) in the decoded signal
    - Frequency response accuracy
    - Signal detection reliability
6. **Waveform Analysis:** Time-domain and frequency-domain plots will be generated for visual comparison across different noise levels.

Expected Outcomes

As the SNR decreases (i.e., noise levels increase), the system's performance is expected to degrade. Higher noise levels are likely to result in increased BER and reduced frequency response accuracy. However, FSK modulation is known for its robustness against noise, so the system is expected to maintain some level of reliability even at lower SNR levels.

This testing methodology will provide a comprehensive assessment of the FSK system's noise resilience, helping to determine its effectiveness in various noise environments and identify potential limitations or areas for improvement in the signal processing algorithms.

3.6.5 Preamble Detection

Objective

This test evaluates the accuracy of the preamble detection algorithm in identifying the start of a message. Accurate preamble detection is crucial for ensuring reliable bi-directional communication and optimising parameters such as transmission speed.

Testing Methodology

1. **Signal Generation:** A clean FSK signal will be generated at different bit rates, based on the results from speed testing in section 3.6.3. The tested speeds will include 25 bps, 50 bps, and 100 bps.
2. **Recording Environment:** Tests will be conducted in a controlled, quiet environment to minimise external interference.
3. **System Setup:** The optimal microphone, distance, and speed configuration from previous tests will be used for all preamble detection tests.
4. **Performance Metrics:** For each bit rate, multiple tests will be conducted. The algorithm will output the detected index for the start of the message, which will be compared to the actual start of the message determined visually from the signal plot. The accuracy of detection will be calculated for each test, and an average accuracy across all tests at each speed will be recorded.

Expected Outcomes

It is expected that preamble detection will be highly accurate at lower bit rates (e.g., 25 bps), with accuracy decreasing as the bit rate increases. This metric is critical for determining the maximum speed at which reliable bi-directional communication can occur.

3.7 Additional Performance Tests

While the previous tests provide a comprehensive assessment of the system under various conditions, additional tests are necessary to evaluate and optimise the autonomous performance of the demodulation process. These tests focus on fine-tuning specific aspects of the signal processing algorithms.

3.7.1 Sliding Window Step Size Optimisation

Objective

To determine the optimal step size for the sliding window approach in preamble detection, balancing accuracy and computational efficiency.

Testing Methodology

1. **Variable Step Sizes:** Test the preamble detection algorithm with various step sizes, ranging from 1 sample to a significant fraction of the preamble length.
2. **Performance Metrics:** For each step size, evaluate:
    - Accuracy of index start position
    - Computational time
    - Overall bit recognition performance
3. **Analysis:** Compare the trade-offs between accuracy and efficiency for different step sizes.

Expected Outcomes

Smaller step sizes are expected to provide more accurate index start positions but require more computational resources. The goal is to identify a step size that offers an optimal balance between accuracy and efficiency.

3.7.2 Sample Skipping Optimisation

Objective

To determine the optimal number of samples to skip at the beginning and end of each symbol to reduce edge detection errors while maintaining signal strength.

Testing Methodology

1. **Variable Skip Sizes:** Test the demodulation algorithm with different numbers of skipped samples at symbol edges.
2. **Performance Metrics:** For each skip size, evaluate:
   - Symbol detection accuracy
   - Frequency response strength
   - Overall bit error rate
3. **Analysis:** Assess the relationship between skip size and performance across different transmission speeds.

Expected Outcomes

Increasing the number of skipped samples is expected to improve symbol detection accuracy up to a point, after which the reduced signal strength may negatively impact performance. The optimal skip size may vary depending on the transmission speed.

3.8 Hardware Selection

3.8.1 Embedded Device

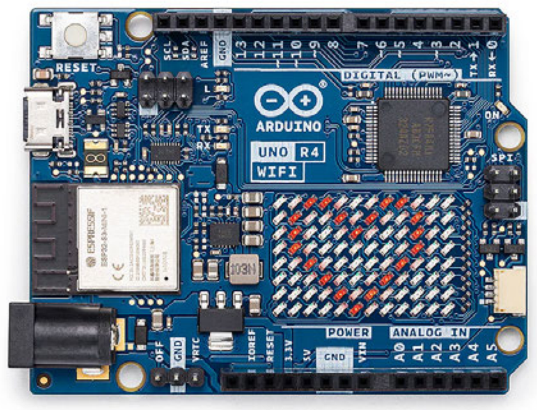


Figure 3.3: Arduino Uno R4 to be Used to Emulate Embedded Device (Arduino 2024*b*).

- **Arduino Uno R4:** Selected for its DAC, which is essential for processing analogue audio signals. The Arduino R4 also offers sufficient processing power and memory for the required signal processing tasks. The Arduino Uno R4 has significant improvements on its predecessor in the Arduino R3 as outlined in Table 2.3 in Section 3.8. Some of the most significant differences are:
  - 32-bit micro-controller in the R4 compared to an 8-bit micro-controller in the R3
  - 48 MHz clock speed in the R4 compared to 16 MHz in the R3.
  - 12-bit DAC in the R4 compared to none in the R3.
  - 14-bit ADC in the R4 compared to a 10-bit ADC in the R3.
  - 256 kB flash memory in the R4 compared to 32 kB in the R3.

  Table 2.3 also compares the Arduino Uno to the Raspberry Pi Zero, while the Raspberry Pi Zero has some very impressive attributes and is clearly a powerful piece of equipment, the convenience of the Arduino Uno R4 having a built in DAC and ADC as well as being a microcontroller with simple integration features gives it a real advantage in this scenario.

- **Microphone Sound Sensor Module:** Chosen for its ability to capture acoustic signals with the necessary fidelity. The selected module should have a wide frequency response and low noise levels.

Figure 3.4: Arduino Compatible Microphone Module (jaycar.com.au).

- **Audio Amplifier and Speaker Module:** The speaker will transmit acoustic signals, and the amplifier ensures the signals are transmitted at sufficient volume. The chosen speaker should have a flat frequency response in the range used for data transmission.

Figure 3.5: Arduino Compatible Amplifier and Speaker Module (jaycar.com.au).

- **Memory Card Module:** To store data locally on the embedded device for later retrieval. This is particularly useful for logging data in remote locations.

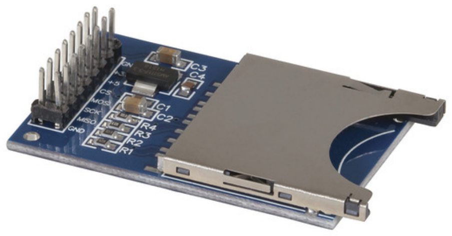


Figure 3.6: Arduino Compatible SD Card Module (jaycar.com.au).

3.8.2 Portable Device

The device that will be used to emulate the portable device used to communicate with the embedded device is a Samsung Galaxy S21 5G (S21) mobile phone. This is a great example of a phone that personnel might have in the field, it is not top of the range phone nor is it a cheap, poor quality phone, it sits somewhere in the middle.

Like all modern day mobile phones, the microphone and speaker generally record and produce reasonably clear audio. The phone has ample processing power and even a built in DAC to ensure it will be able to handle data encoding and decoding through the API.

Parameter	**Samsung Galaxy S21 5G**
CPU	Octa-Core
Clock Speed	2.9, 2.4, 1.8 GHz
DAC	32-bit
ADC	None
RAM	8 GB
Operating System	Android 14

Table 3.2: Specifications of Samsung Galaxy S21 5G Mobile Phone (Samsung 2021).

3.9 Data Transfer Procedure

The data transfer procedure involves the following steps:

1. **Initialisation:** The portable device sends an initialisation signal to the embedded device to establish a communication link.
2. **Data Encoding:** Data is encoded using BFSK (or 4-FSK if expanded) before transmission.
3. **Data Transmission:** The portable device transmits the encoded data acoustically to the embedded device. A preamble is sent to calibrate the high and low bit frequencies.
4. **Data Reception:** The embedded device receives the acoustic signals through the microphone and decodes the data.
5. **Error Checking:** Implement error checking mechanisms such as checksums to ensure data integrity. If errors are detected, request re-transmission.
6. **Data Processing:** The received data is processed and stored in the embedded device or used to update firmware/configurations.

3.10 API Architecture

The purpose of the API is to facilitate secure and efficient bidirectional data transfer between a portable device and an embedded device. Using an API for the portable device will ensure cross platform compatibility for a range of personnel, making the functionality more accessible to a wider audience. The API will ensure field personnel can easily initiate data requests, send data, and manage data through a user friendly interface, a draft of which can be found in Figure 3.8. This will also allow robust security protocols to protect any sensitive information.

3.10.1 Structure

The API is developed using JavaScript and adheres to RESTful principles to ensure scalable and maintainable communication between the portable and embedded devices. It will utilise HTTP protocol for data transfer, ensuring compatibility with a wide range of devices and platforms.

Data will be exchanged in JavaScript Object Notation (JSON) format, keeping it lightweight, easy to read, and simple to parse. The API is built using the Node.js framework, providing efficiency and robustness for server-side JavaScript applications. For testing purposes, text files will be used to enable simple adjustments.

The API consists of several endpoints that handle specific tasks related to data transfer and communication management. Each endpoint is designed to perform a discrete function, allowing for modularity and easy debugging. A flowchart outlining the endpoints operation and flow can be found in Figure 3.7.

3.10.2 Endpoints Overview

1. **Initialisation Endpoint**('/initiateHandshake'): Initiates the handshake process between the portable and embedded device. This uses a pre-defined key that both devices will have stored.
2. **Data Request Endpoint**('/requestData'): Requests data from the embedded device. This can be a request for specific data such as pressure or temperature, or an

entire diagnostic data request which would generally be used for fault finding.

3. **Resend Request Endpoint**('/requestResend'): This endpoint will request data is sent again if the API identifies the BER is unacceptable. The user should change position or environment to attempt to provide a connection more suitable for positive communication.

4. **Data Sending Endpoint**('/sendData'): Sends data from the portable device to the embedded device.

5. **Disengage Endpoint**('/disengage'): Terminates the communication session.

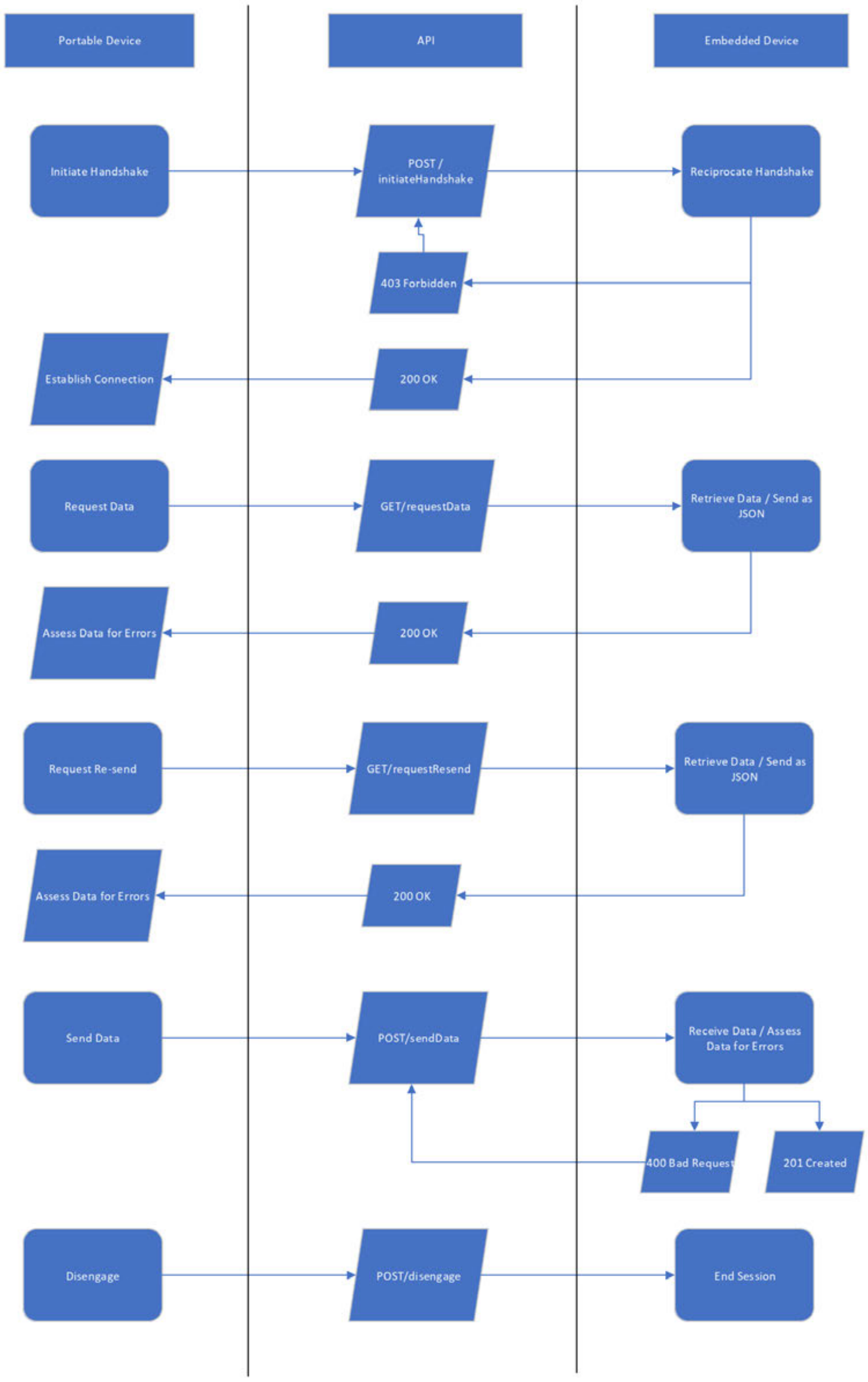


Figure 3.7: API Flowchart

3.10.3 Communication Flow

1. **Initiation**

    - The user on the portable device triggers the ”Initiate Handshake” button and presses the corresponding button on the embedded device.
    - A POST request is sent to the ’/initiateHandshake’ endpoint.
    - The embedded device processes the request and responds with the corresponding handshake.

2. **Data Request**

    - The user requests data from the embedded device by pressing the ”Request Data” button on the portable device.
    - A GET request is sent to the ’/requestData’ endpoint.
    - The embedded device retrieves and sends back the requested data in JSON format.

3. **Data Re-send**

    - If the BER is determined to be unacceptable, the user will be notified.
    - Once the user has attempted to improve the connection environment conditions, they can request the data to be re-sent by pressing the ”Request Re-send” button on the portable device.
    - A GET request is sent to the ’/requestResend’ endpoint.
    - The embedded device processes the request and re-sends the last message.

4. **Data Sending**

    - To send data from the portable device to the embedded device, the user will press the ”Send Data” button.
    - A POST request with the data payload is sent to the ’/sendData’ endpoint.
    - The embedded device receives and processes the data.
    - The embedded device will have the same chance to request the data be re-sent if the BER is found to be unacceptable.

5. **Disengagement**

    - To end the session, the user will press the "Disengage" button on the portable device.
    - A POST request is sent to the '/disengage' endpoint.
    - The embedded device processes the request and terminates the session.

Error Handling and Security

- **Error Handling**: Each endpoint includes error handling to manage and respond to issues such as invalid requests and data corruption. Standard HTTP Status codes (e.g., 200 for success, 400 for bad request, 500 for server error etc.) are used.
- **Security**: The API uses HTTPS to encrypt data in transit, to attempt to protect sensitive information. Authentication mechanisms such as API keys will be implemented to control access to the API.

Scalability and Maintenance

- The API's modular structure allows for easy updates, scalability and debugging. Each endpoint can be developed, tested and deployed independently.

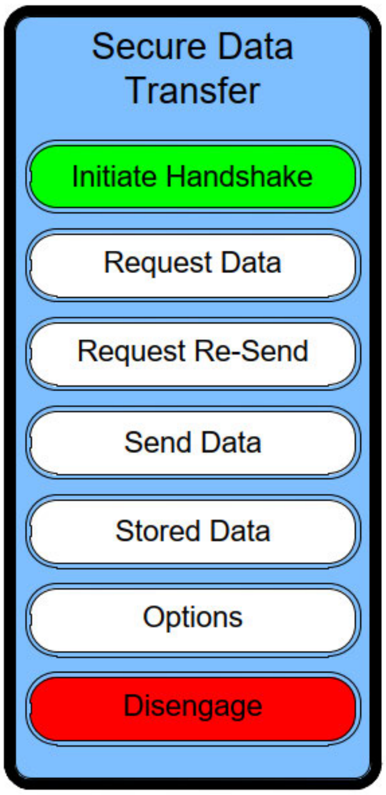


Figure 3.8: Draft of the API GUI

3.11 Arduino Program Architecture

The purpose of the Arduino program is to facilitate secure and efficient bidirectional data transfer between an embedded device and a portable device. The Arduino program will ensure that the embedded device can effectively communicate with the portable device, handle data requests, send data, and manage communication protocols. This will also include implementing security measures to protect sensitive data.

3.11.1 Structure

The Arduino program is developed using C++, allowing it to be easily accessible for most people that would wish to use the code in the future. It is designed to handle low-level operations required for communication between embedded and portable devices. It will manage serial communication protocols to ensure reliable data transfer.

Data will be encoded and decoded using the selected modulation scheme, investigating a balance between the most reliable and the best performing transmission. The Arduino program includes functions for for data processing, error detection and handling communication protocols.

The program consists of several functions that handle specific tasks related to data transfer and communication management. Like the API, each function is designed to perform a discrete task, allowing for modularity and ease of debugging.

3.11.2 Tasks Performed

1. **Initialisation and Handshake**: The Arduino program receives and instruction to listen for a handshake from the portable device. This ensures that both devices are ready for data transmission, sets the parameters for the communication session and ensures only messages from devices with the correct key are communicated.

2. **Data Encoding and Transmission**: Data is encoded using FSK and transmitted to the portable device. The Arduino program includes routines for converting digital data to analog signals (DAC) and vice versa (ADC) to facilitate this process.

3. **Error Checking and Data Integrity**: To ensure data integrity, the Arduino program implements error detection mechanisms, such as checksums. If an error is detected, the program can request a re-transmission. Error correction algorithms will also be explored.

4. **Termination Protocols**: The communication session is terminated by either a timeout function or by a direct request from the portable device, ensuring the embedded device is only open to receiving or sending data when authorised to do so.

5. **Security Measures**: The Arduino program includes basic security measures to verify the authenticity of the portable device and protect data during transmission. This includes authentication checks and will explore data encryption.

3.11.3 Device Compatibility Testing

- **Objective**: Ensure the system's compatibility with a range of different devices.

- **Method**: Test the communication system with various makes and models of smartphone or tablets.

- **Expected Results**: It is expected this test will have positive outcomes though it is an important test to do to prove the viability of the system.

Chapter 4

Results of the Project Experiments

4.1 MATLAB Model Results

4.1.1 Microphone Testing

Microphone testing methodology can be found in Section 3.6.1. It should be noted, as there is a 0.5 second buffer at the start of the modulated message, this will be removed from many of the plots featured, to increase the resolution of the captured audio. This will result in the recorded audio starting at different samples between recordings.

Modulated Signal

The following Figures 4.1 and 4.2 show the signal that has been modulated using FSK, this is the signal that will be played by the portable device and recorded, for demodulation in the MATLAB model. The signal is modulated at a rate of 25 bps, with frequencies $f1$ = 1 kHz and $f2$ = 2 kHz.

Key properties to note on the modulated plots, as seen in Figures 4.1 and 4.2, are the amplitude varying from 1.0 to -1.0 and the cleanliness of the signal. A microphone sufficient for the needs of this project should be able to capture the signal at as close to an ideal, clean signal as possible, there should be as little as possible variation from the modulated signal, to the captured signal from the microphone.

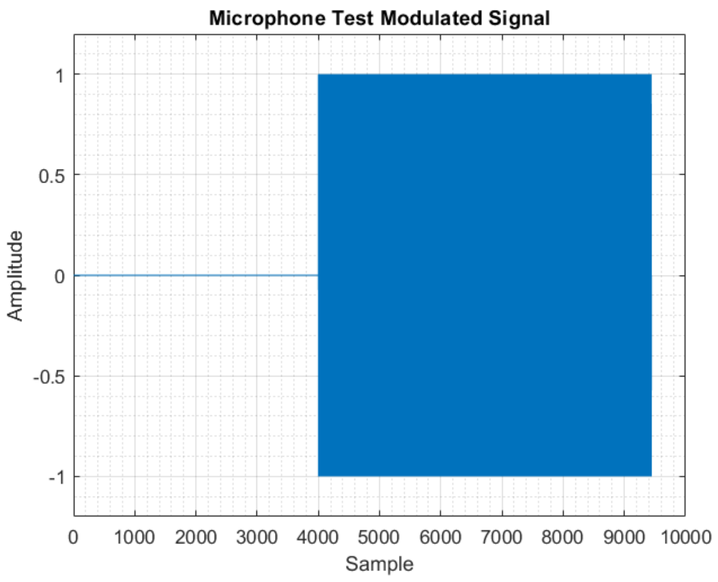


Figure 4.1: Modulated Microphone Whole Signal

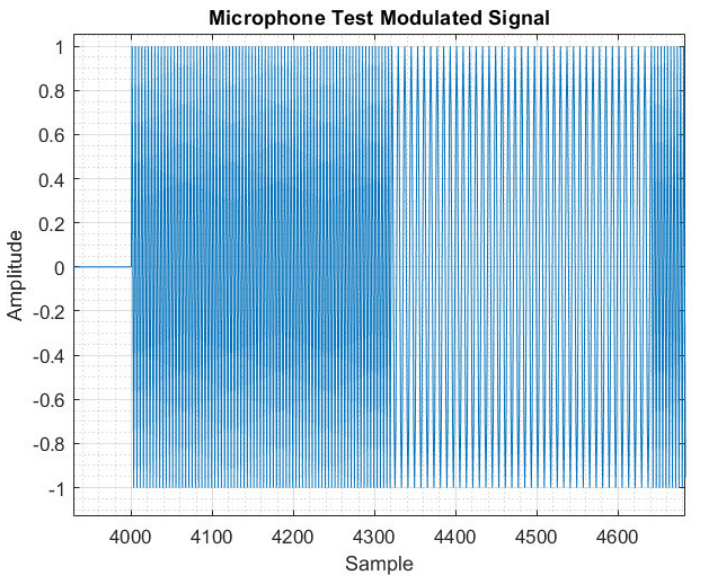


Figure 4.2: Modulated Microphone Signal Showing Two Bits [0 1]

Table 4.1: Microphone Performance Comparison

Microphone	**SNR (dB)**	**Frequency Response**	**BER**
HP Victus 16-d1xxx	34.2	Excellent	0.00
Sennheiser PXC 550	3.52	Poor	0.375
External Microphone	15.44	Fair	0.00

HP Victus 16-d1xxx Microphone

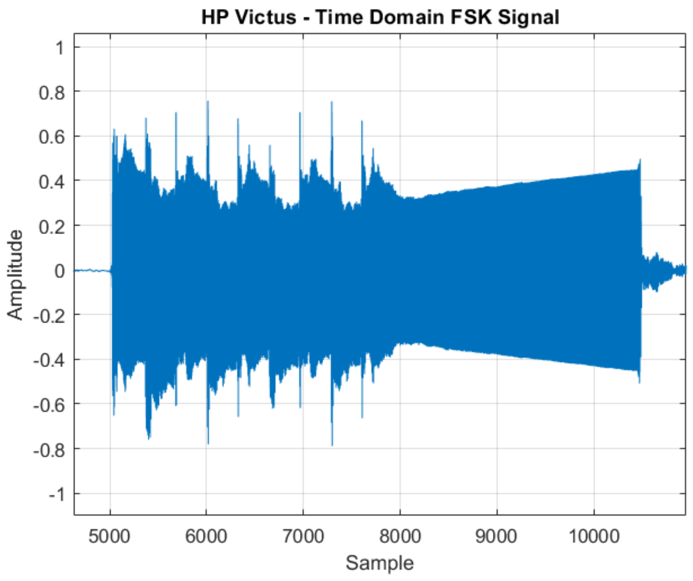


Figure 4.3: Victus Microphone Array Recording Time Domain - Full Signal

The HP Victus microphone array showed curious energy spikes at the change of frequency resulting in a noisy captured signal. The eight continuous bits (from sample $\tilde{7}600$ to sample $\tilde{1}0500$) allows extended analysis of this frequency change anomaly. It appears the signal somewhat recovers from the noisy capture after around one symbol length (320 samples in this case), once it has recovered, the signal seems to be quite clean, but also steadily grows in amplitude. This could be due the array of microphones (2) capturing sounds from around the room, which, while quiet and sound dampened, isn't of a recording studio quality and sound reflections are entirely possible. It may also also be due to the energy output from the speaker, though the signal recording doesn't show this energy growth in Figure 4.12.

Regardless of the curious energy growth in the continuous 8 bit pattern, the received signal is reasonably clean and the single bit recording is impressive, as seen in Figure 4.4, and the bit recovery looks very clean as seen in Figure 4.6.

Sennheiser PXC 550 Microphone

As mentioned in Section 3.6.1, microphone software often has noise suppression software in their driver, this proved true with the built in microphone on the HP Victus laptop, and is believed to have been a factor when using the built in microphone on the PXC 550

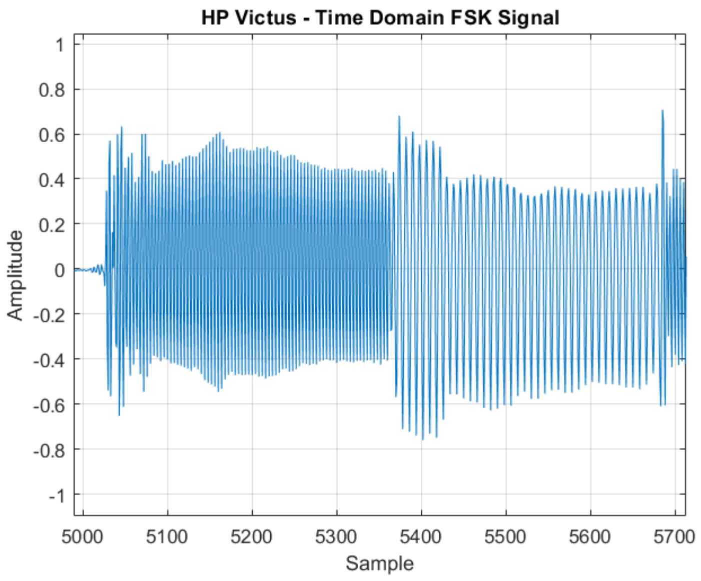


Figure 4.4: Victus Microphone Array Recording Time Domain Showing Two Bits [0 1]

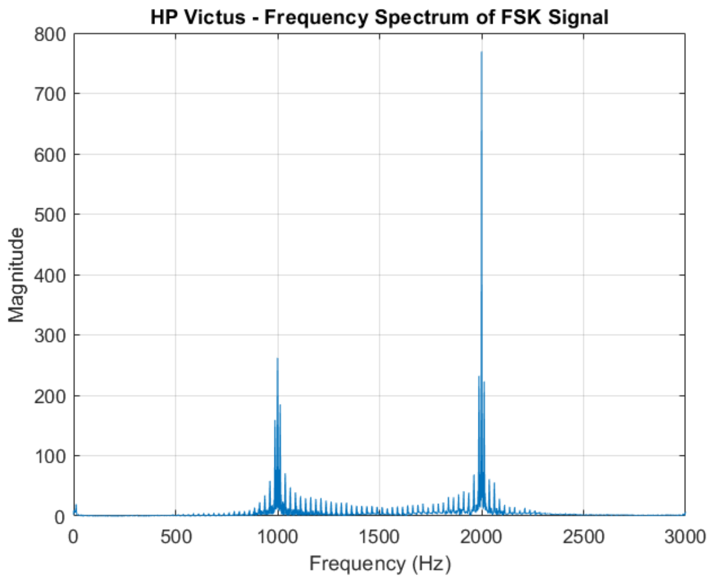


Figure 4.5: Victus Microphone Array Recording Frequency Response

headphones.

As seen in Figure 4.11, the recorded signal is not discernible when viewed to at an amplitude scale of 1. Figure 4.12 shows a zoomed in amplitude scale of $1 * 10^{-4}$, here the signal is visible, though the resolution is not of a standard to be used for demodulation.

Figures 4.9 and 4.10 reinforce the sentiment that this microphone should not be used in this project.

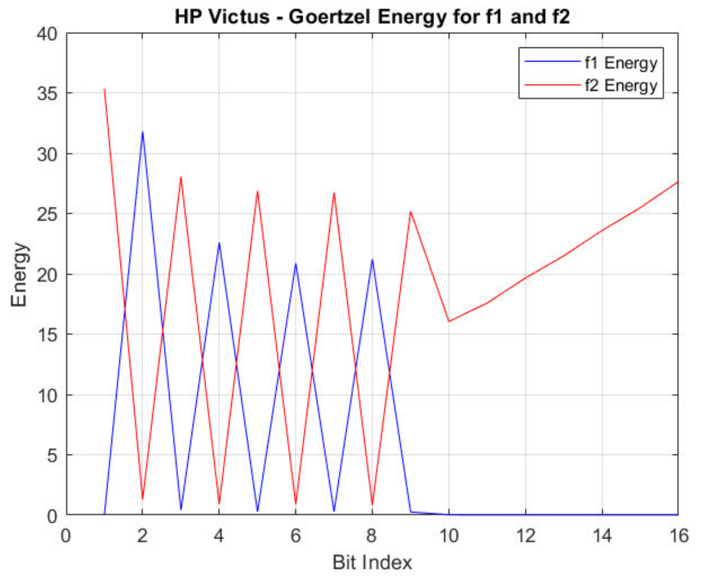


Figure 4.6: Victus Microphone Array Recording Goertzel Algorithm Energy for $f1$ and $f2$

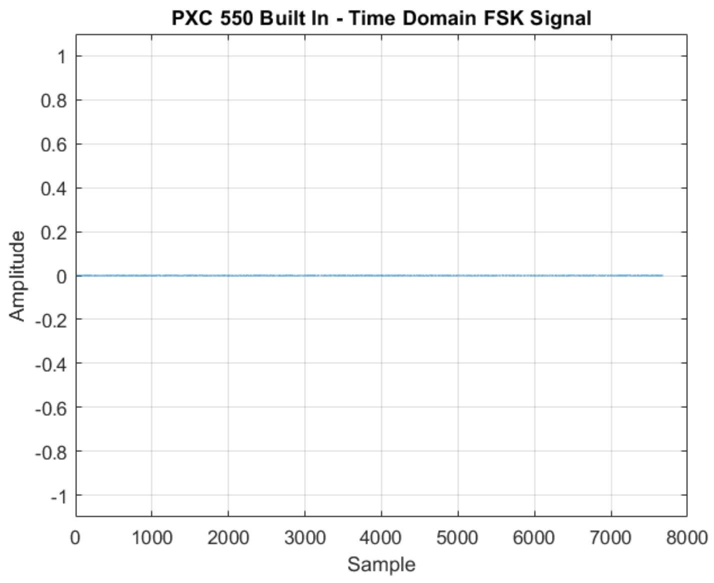


Figure 4.7: Sennheiser PXC 550 Recording Time Domain - Full Recording

It should be noted that every effort shy of editing the driver of the microphone was taken in an attempt to record a usable signal.

Sennheiser PXC 550 External Microphone

The external microphone for the PXC 550 is assumed to use the same driver software as the built in microphone, in spite of this, the external microphone produced much better results, though the effects of the noise suppression was evident.

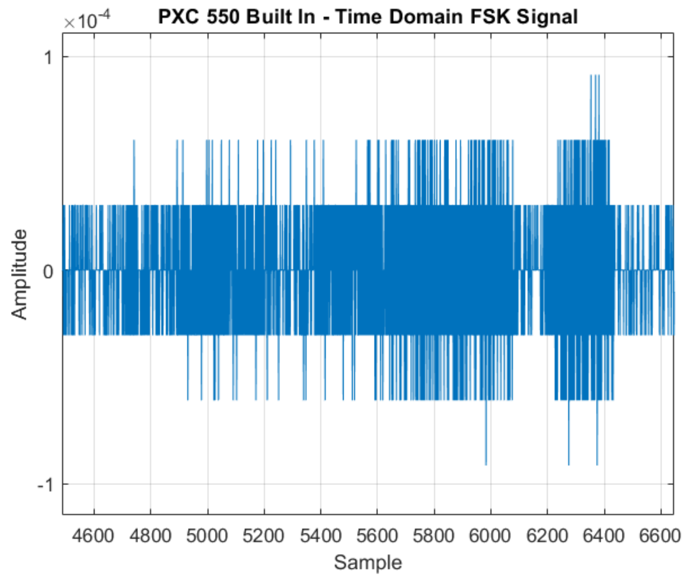


Figure 4.8: Sennheiser PXC 550 Recording Time Domain Showing Whole Signal

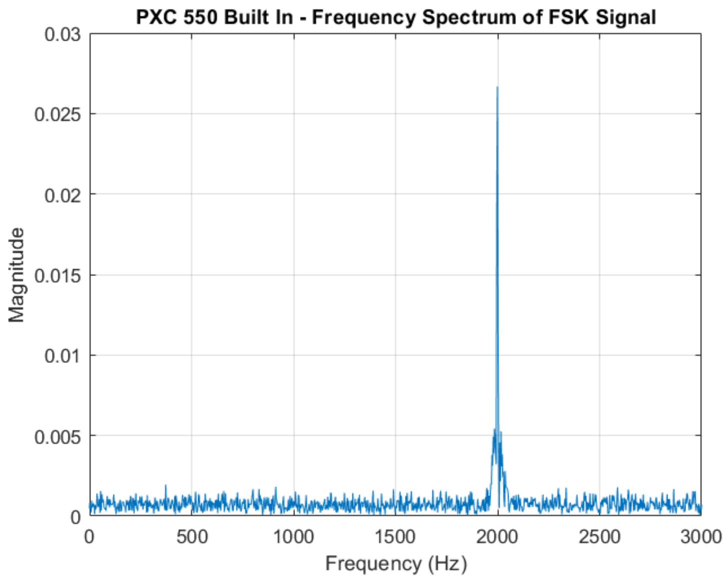


Figure 4.9: Sennheiser PXC 550 Recording Frequency Response

Figures 4.12 and 4.13 show the full recording of the signal and two bits of that signal respectively. A cleaner signal is seen with this microphone, compared with that of the HP Victus Microphone Array, the amplitude spikes are less prominent and the signals show a more uniform amplitude. Unfortunately Figure 4.12 shows the full extent of the recording, the signal amplitude is so low when on an amplitude scale of, that it is unable to be seen, it isn't until the amplitude is scaled to $1 * 10^{-4}$ that the captured signal becomes useful.

The demodulation using this microphone was very clear, and could be manipulated to

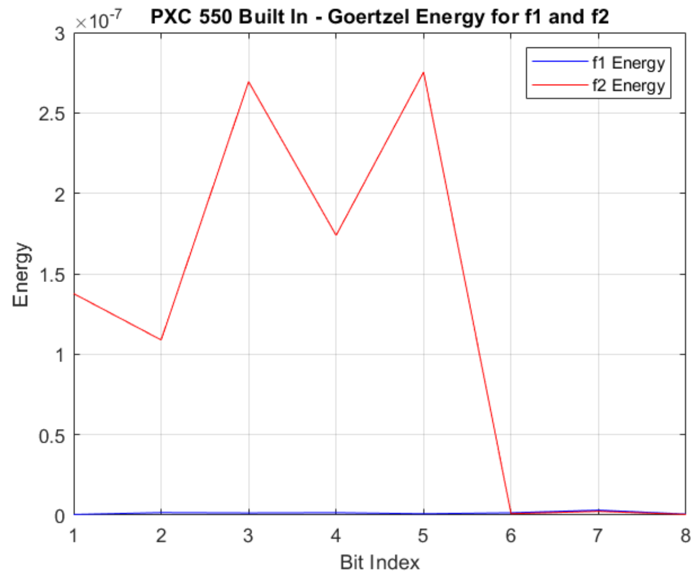


Figure 4.10: Sennheiser PXC 550 Recording Goertzel Algorithm Energy for $f1$ and $f2$

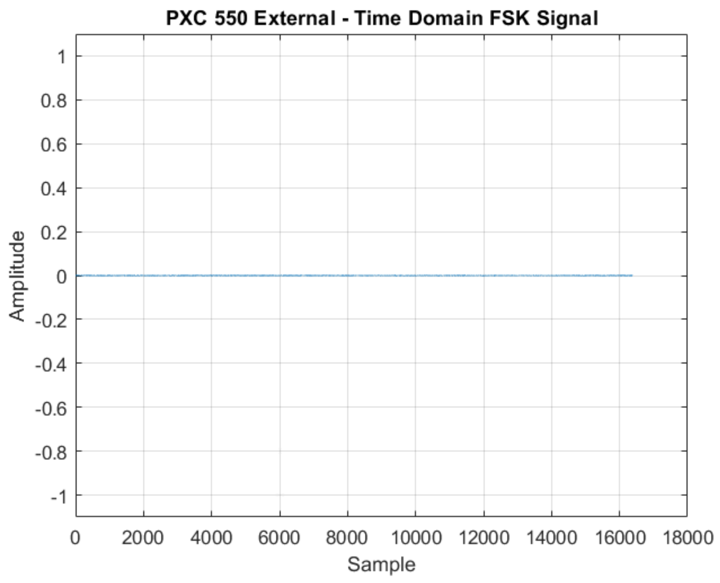


Figure 4.11: Sennheiser PXC 550 External Microphone Recording Time Domain - Full Recording

give strong results. The energies captured, shown in Figure 4.15, show a very clean demodulated signal. Unfortunately, the energy captured in $f1$ (1 kHz) is far too low to give reliable results.

As a result of the poor energy level captured, this microphone will not be used for further testing.

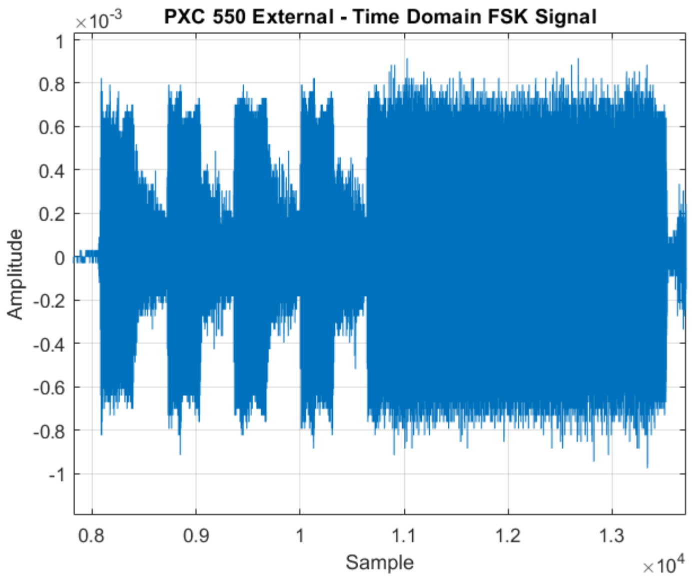


Figure 4.12: Sennheiser PXC 550 External Microphone Recording Time Domain Showing Whole Signal

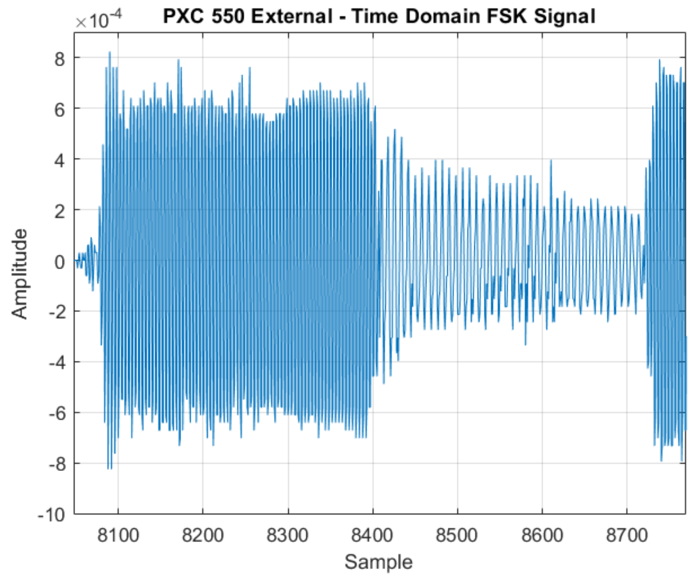


Figure 4.13: Sennheiser PXC 550 External Microphone Recording Time Domain Showing Two Bits [0 1]

4.1.2 Distance From Source Testing

Distance from source testing methodology can be found in Section 3.6.2. The demodulation will utilise a manually chosen starting index. This will result in the recorded audio starting at different samples between recordings. The baud rate for this test was 40 bps at frequencies of $f1 = 1$ kHz and $f2 = 2$ kHz. The microphone used was the HP Victus 16-d1xxx Microphone Array.

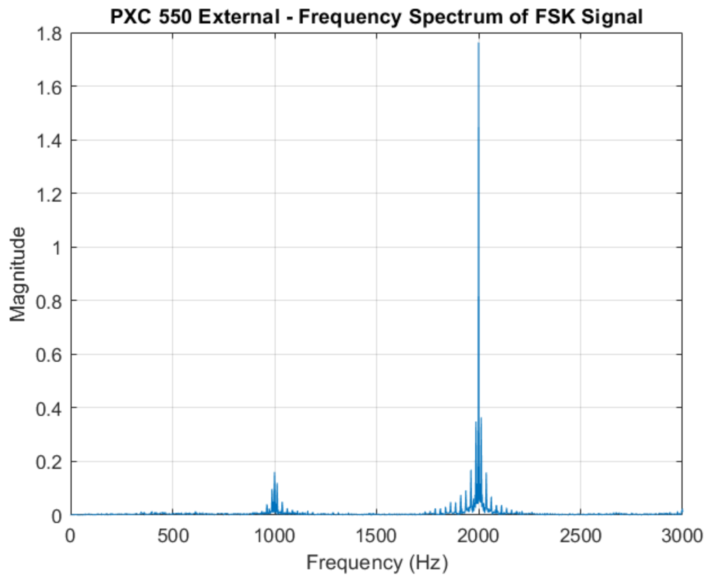


Figure 4.14: Sennheiser PXC 550 External Microphone Recording Frequency Response

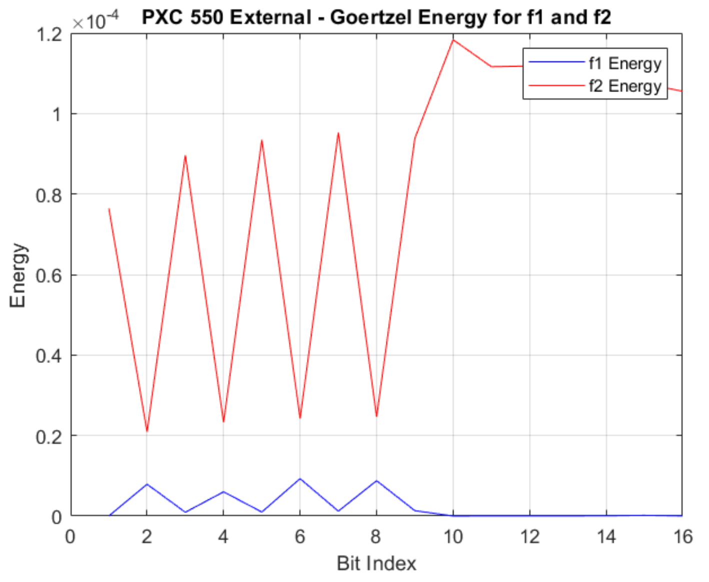


Figure 4.15: Sennheiser PXC 550 External Microphone Recording Goertzel Algorithm Energy for $f1$ and $f2$

As mentioned in Section 3.6.2, the message tested for this and proceeding tests, is the full 1000 bit message, this means full message plots will lose some resolution, this will be addressed in subsequent plots, zoomed in on two bits, to see the recorded audio in finer detail.

Table 4.2: Distance Test Results

Distance (m)	SNR (dB)	BER (%)	Amplitude
<0.1	31.05	0.00	0.2754
0.5	33.67	0.00	0.1644
1	22.44	0.00	0.0699
2	25.76	0.3333	0.0669
5	16.02	0.0833	0.0262

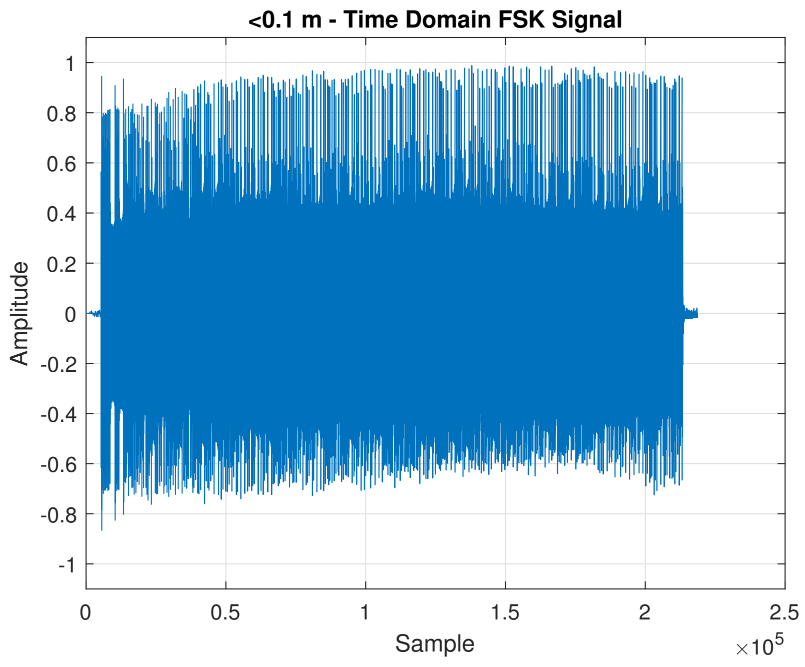


Figure 4.16: Time Domain Plot Recorded from a Distance of 0.1 m Showing Full Message

<0.1 m

Figure 4.16 shows the full time domain recorded signal, with Figure 4.17 zoomed in on two bits. Whilst the signal here is close to what we expect, there is a clear DC offset to the signal, with the upper limit close to 1 but the lower limit sitting around -0.7. This was not as apparent in further tests but was included to demonstrate the robustness of the demodulation technique used.

As seen in Figure 4.19, the apparent DC offset has not appeared to affect the Goertzel energy plot, showing very clear bit transitions.

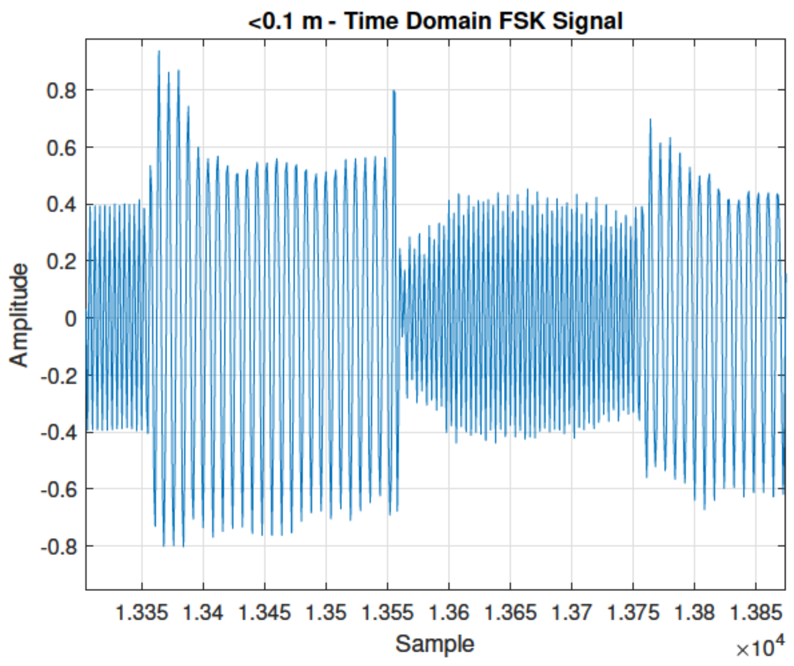


Figure 4.17: Time Domain Plot Recorded from a Distance of 0.1 m Showing Two Bits [0 1]

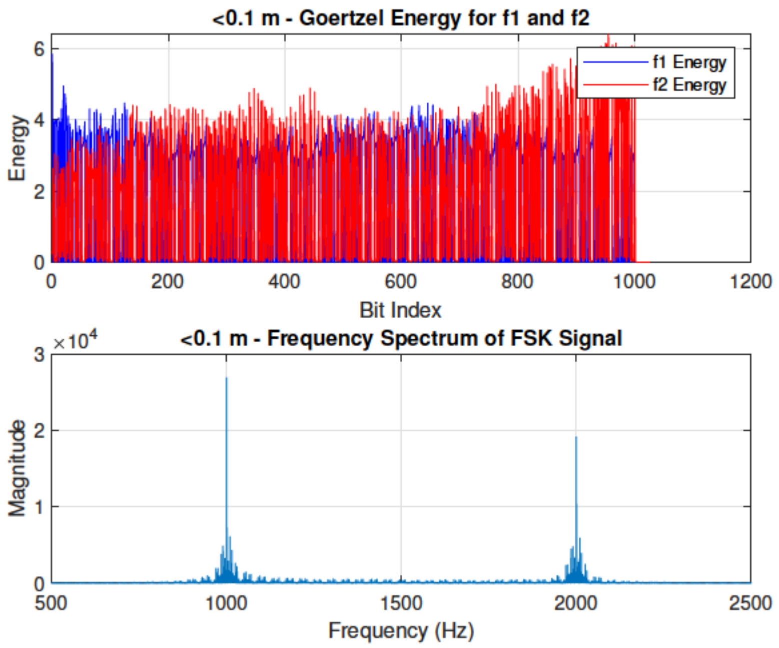


Figure 4.18: Full Message Goertzel Energy and Frequency Spectrum Plot Recorded from a Distance of 0.1 m

0.5 m

Figure 4.20 shows the time domain plot of the full message, recorded from 0.5 m. The biggest contrast to the message recorded from <0.1 m is the amplitude, where the former

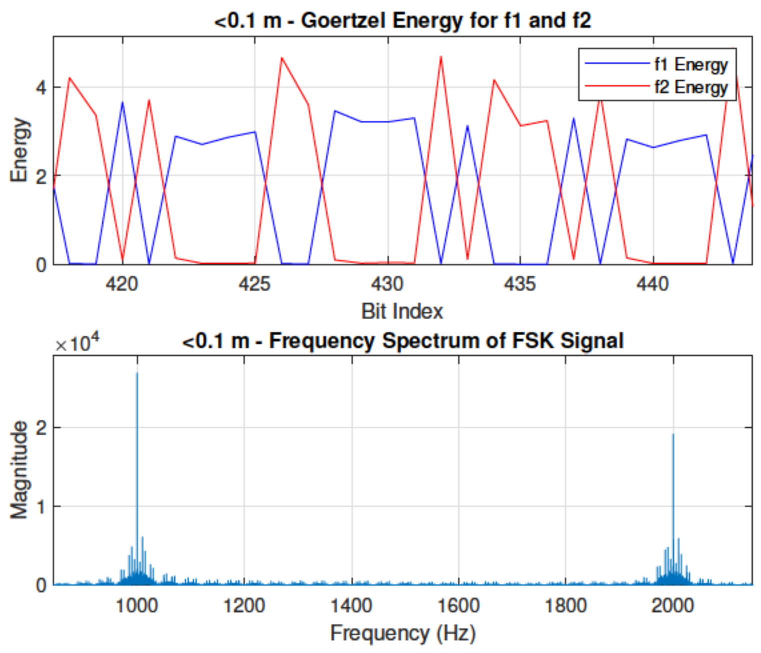


Figure 4.19: Zoomed in Goertzel Energy and Frequency Spectrum Plot Recorded from a Distance of 0.5 m

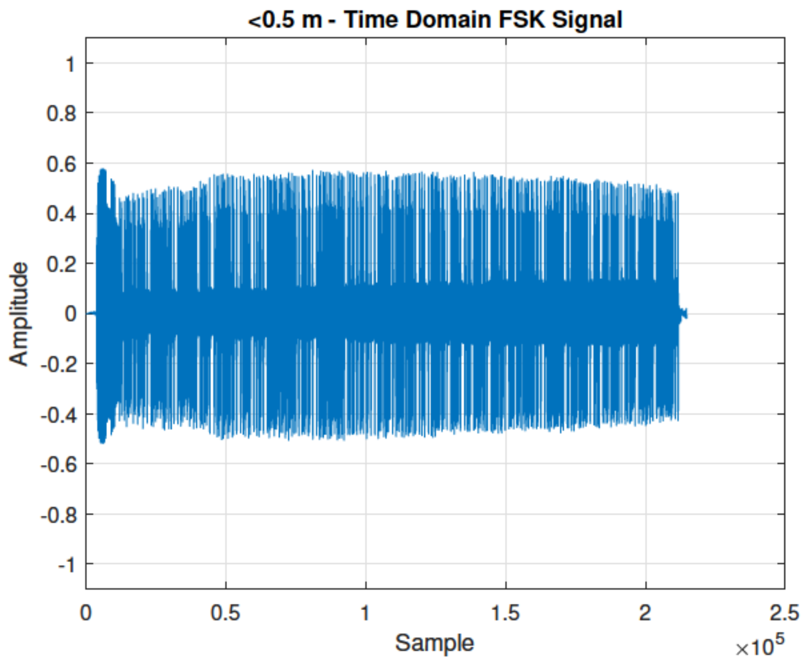


Figure 4.20: Time Domain Plot Recorded from a Distance of 0.5 m Showing Full Message

plot amplitude had a total amplitude of around 1.8, the amplitude recorded from 0.5 m amplitude is around 1 (from 0.5 to -0.5). This smaller amount of energy capture may lead to an insufficient amount of energy for demodulation.

It should be noted in Figure 4.21, the low amplitude of bit 0 ($f156$), this may lead to

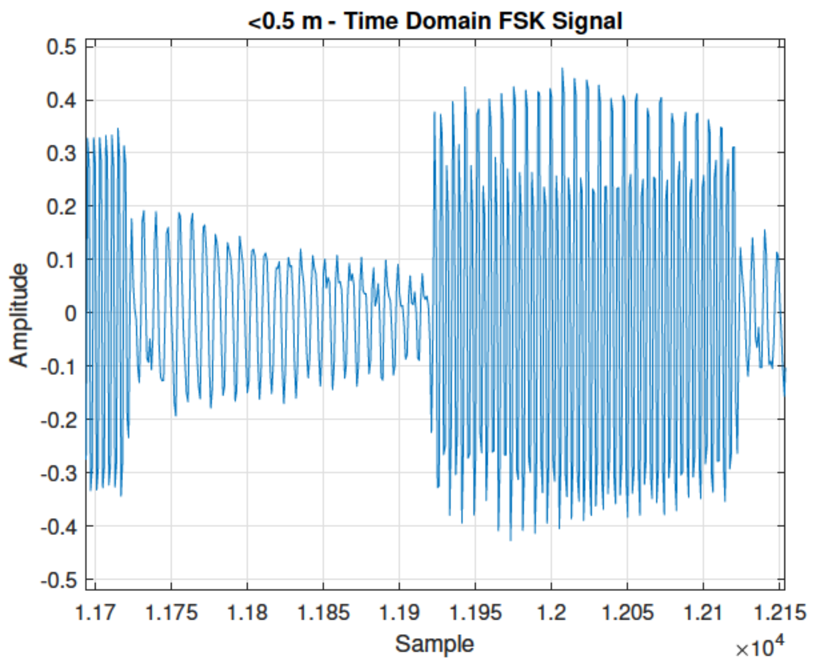


Figure 4.21: Time Domain Plot Recorded from a Distance of 0.5 m Showing Two Bits [0 1]

poor results in demodulation.

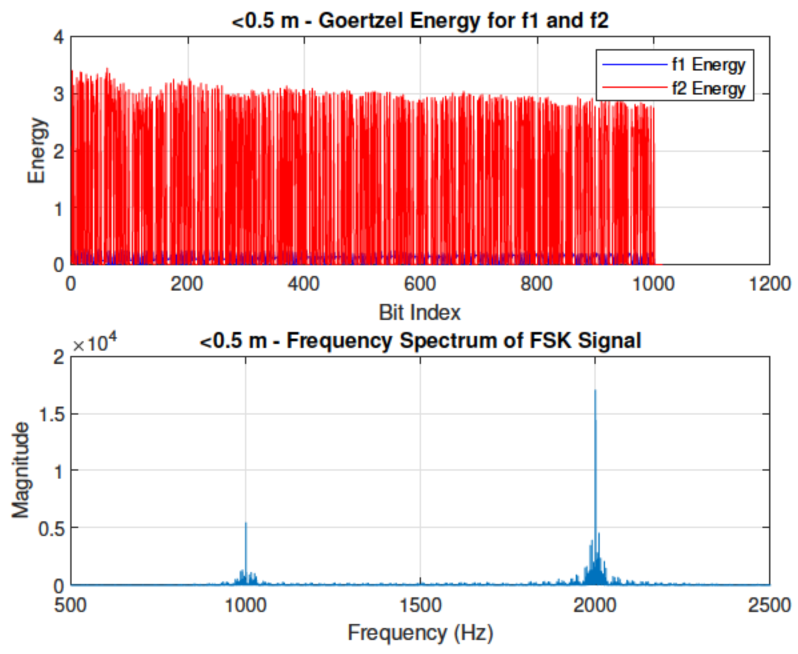


Figure 4.22: Full Message Goertzel Energy and Frequency Spectrum Plot Recorded from a Distance of 0.5 m

The low power of $f1$ is accentuated in Figure 4.23 and 4.22, whilst the demodulation has remained unaffected, it seems it is only the relatively high energy of $f2$ that has allowed the demodulation to take place accurately. Whilst this is arguably a poor recording, it

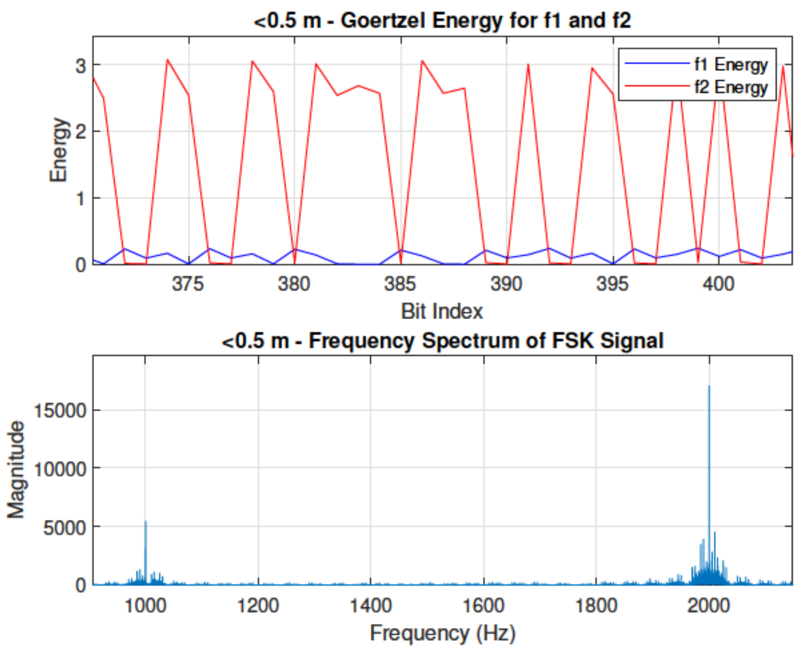


Figure 4.23: Zoomed in Goertzel Energy and Frequency Spectrum Plot Recorded from a Distance of 0.5 m

reinforces the robustness of the demodulation technique.

1 m

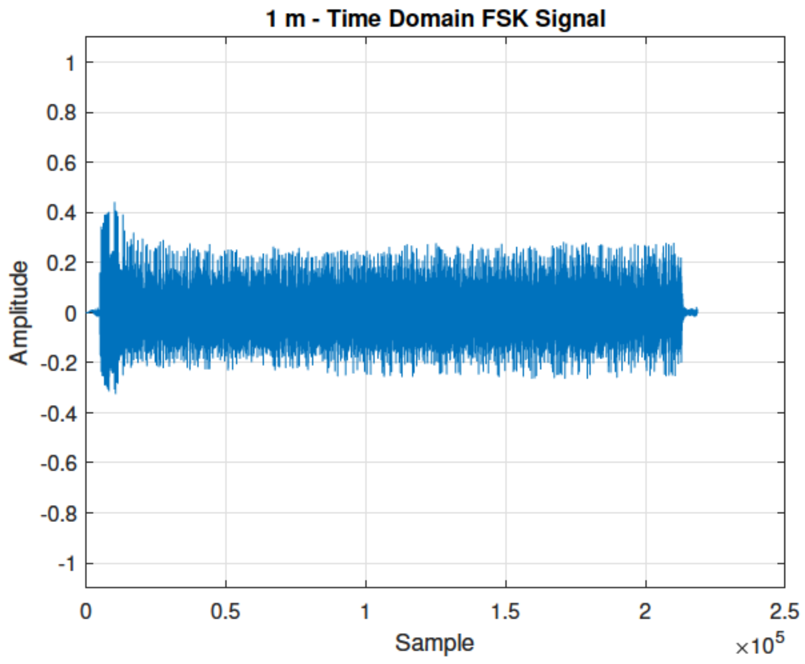


Figure 4.24: Time Domain Plot Recorded from a Distance of 1 m Showing Full Message

Figure 4.24 shows the attenuation of the acoustic signal in an aerial medium, within 1 metre from the microphone, the total amplitude has attenuated from 1.8 to 0.4. This highlights the need for close proximity between the portable device and the embedded device.

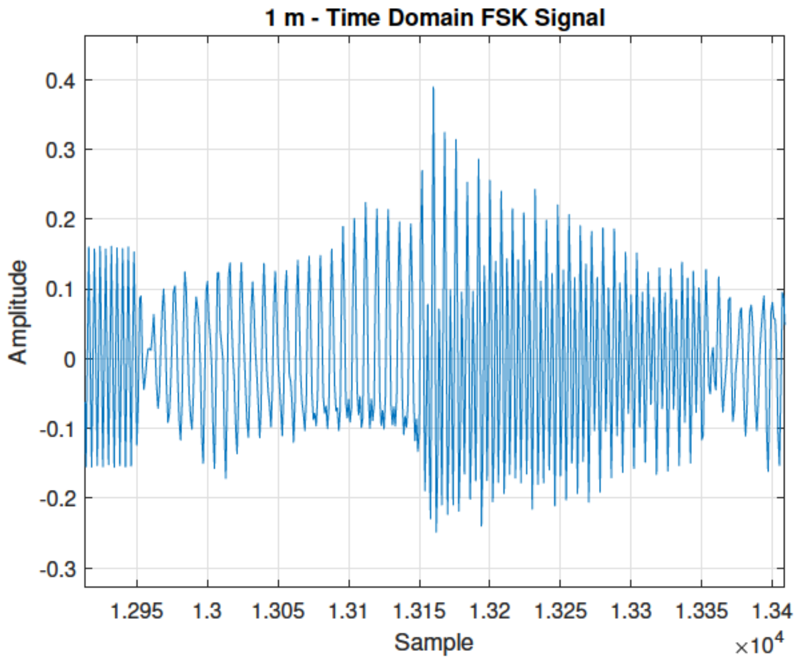


Figure 4.25: Time Domain Plot Recorded from a Distance of 1 m Showing Two Bits [0 1]

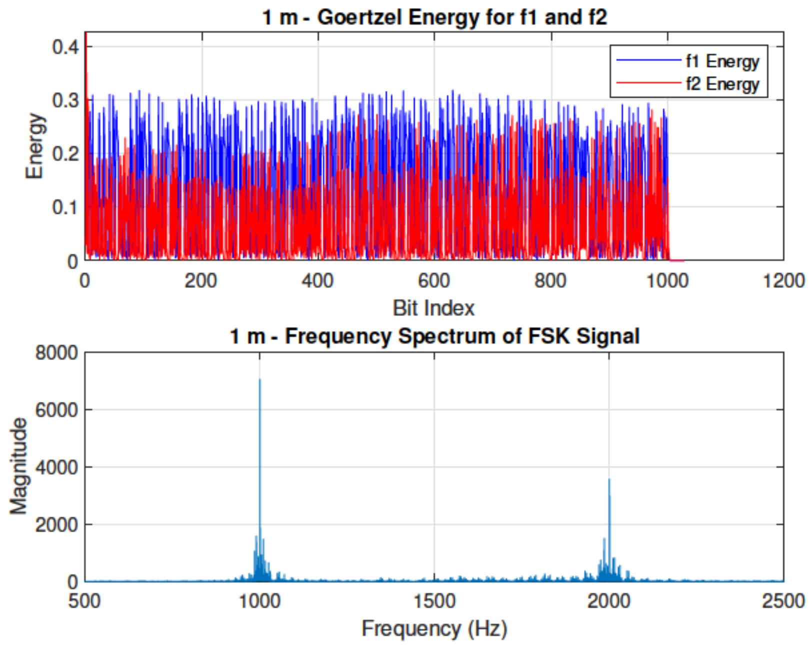


Figure 4.26: Full Message Goertzel Energy and Frequency Spectrum Plot Recorded from a Distance of 1 m

The zoomed in Goertzel Energy plot in Figure 4.27 shows the crossover of energies is

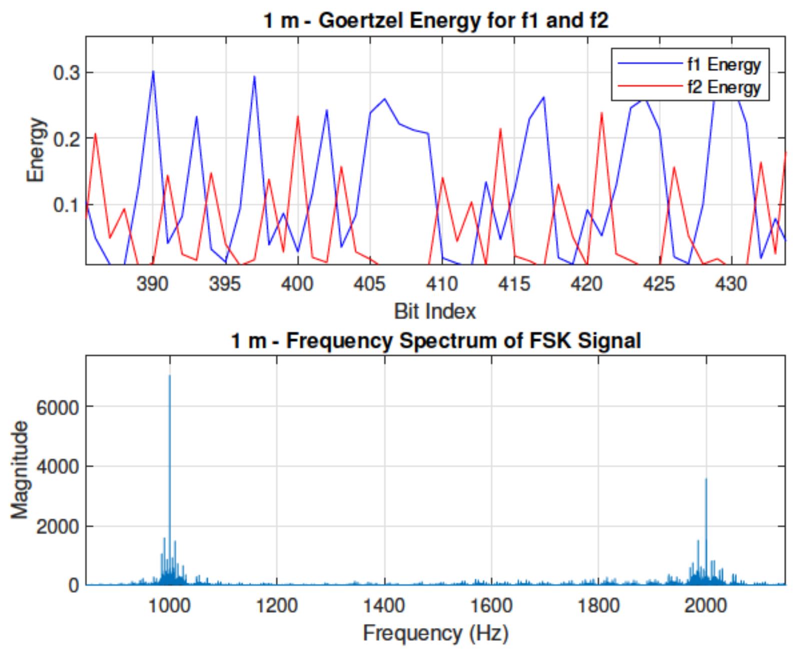


Figure 4.27: Zoomed in Goertzel Energy and Frequency Spectrum Plot Recorded from a Distance of 1 m

still quite pronounces, though pales in comparison to that of Figure 4.19. This further highlights the importance of close proximity between devices.

2 m

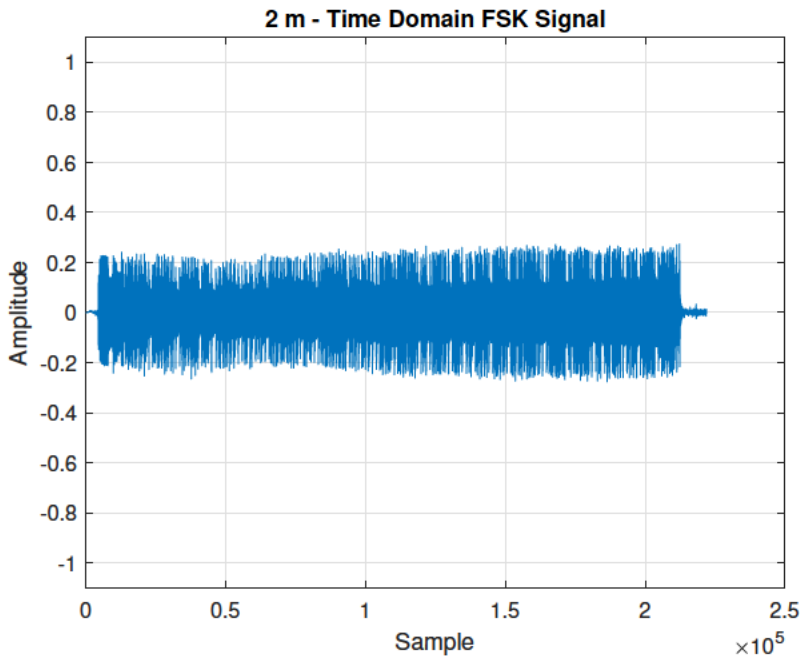


Figure 4.28: Time Domain Plot Recorded from a Distance of 2 m Showing Full Message

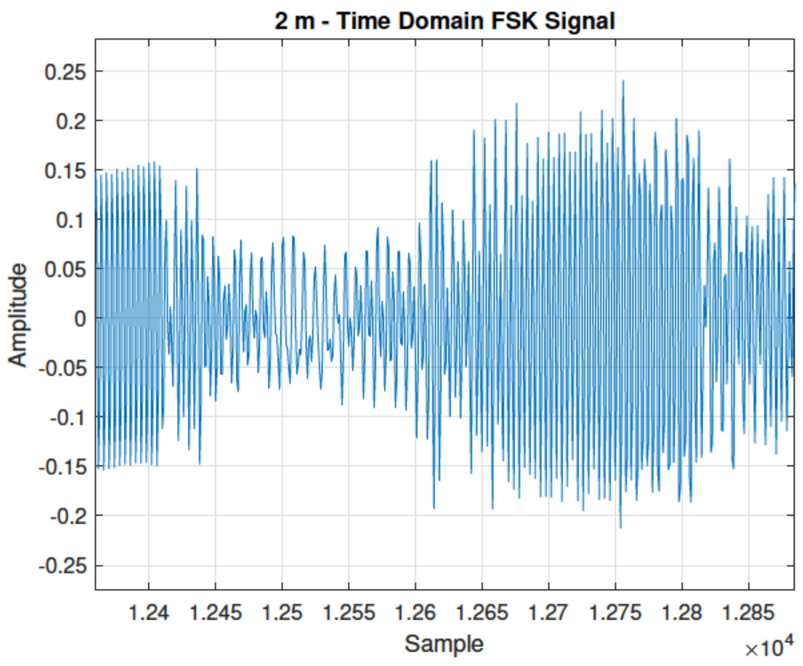


Figure 4.29: Time Domain Plot Recorded from a Distance of 2 m Showing Two Bits [0 1]

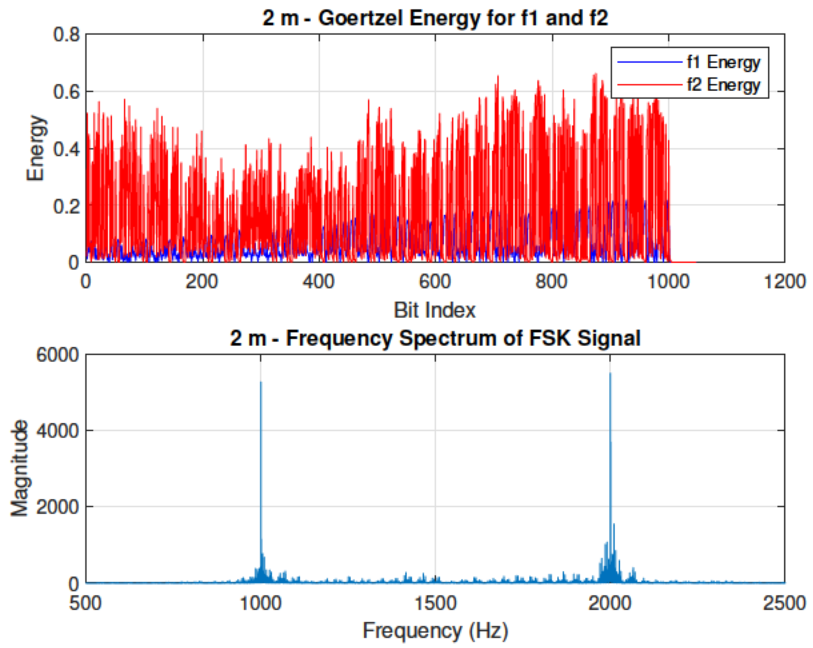


Figure 4.30: Full Message Goertzel Energy and Frequency Spectrum Plot Recorded from a Distance of 2 m

Figures 4.28, 4.29, 4.30 and 4.31, recorded at a distance of two metres, show a similar capture to the signal recorded at one metre. The point of difference here is that the energies captured reduce even further due to the attenuation of the signal through air.

It is this signal recorded at two metres that starts to show errors in the demodulation and

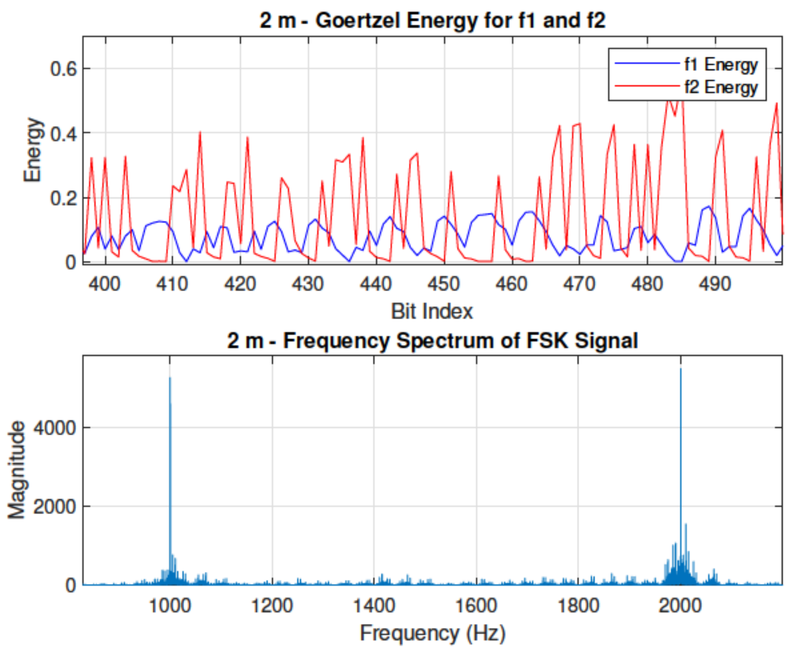


Figure 4.31: Zoomed in Goertzel Energy and Frequency Spectrum Plot Recorded from a Distance of 2 m

suggests the maximum distance that the signal should be recorded is one metre or less. The zoomed in energy capture in Figure 4.31 shows a rounded edge of the bit recognition in $f1$ energy, as opposed to the sharp response of that in Figure 4.19.

5 m

The signal captured from five metres shows significant attenuation of the signal, as expected. The image seen in Figure 4.32 shows a hardly discernible difference between the time domain capture of two different bits. This is contrasted in table 4.2, where a bit error rate of 0.0833 was calculated. The recovered message was as such:

H-l,/- -orl- 2ose)n -976, "y . -ernigjin-(j-n-r- that, Alexander G Bell in 1876: Mr. Watson, come here, I want to see you.!

This is not a reliable signal capture, but has potential for the demodulation to be able to capture signals with low signal power, possibly with further filtering.

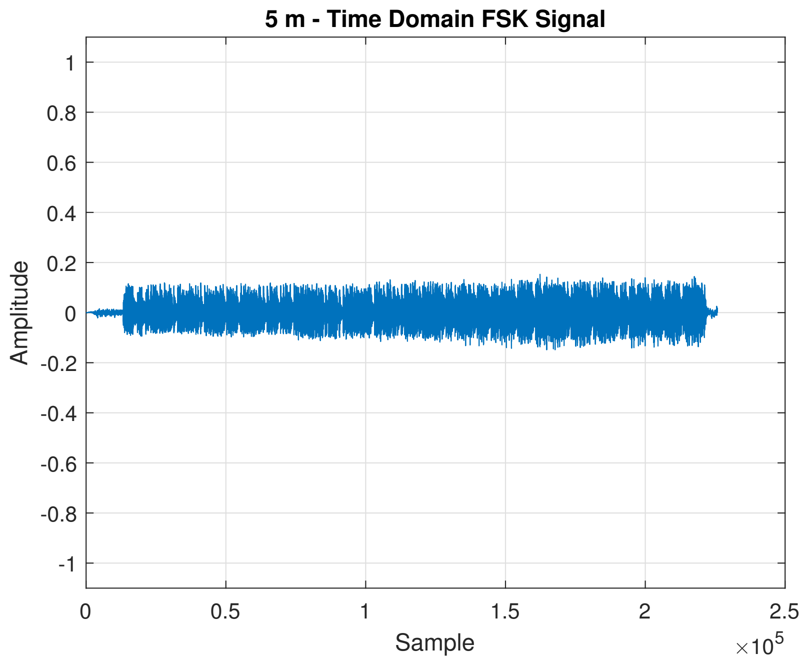


Figure 4.32: Time Domain Plot Recorded from a Distance of 5 m Showing Full Message

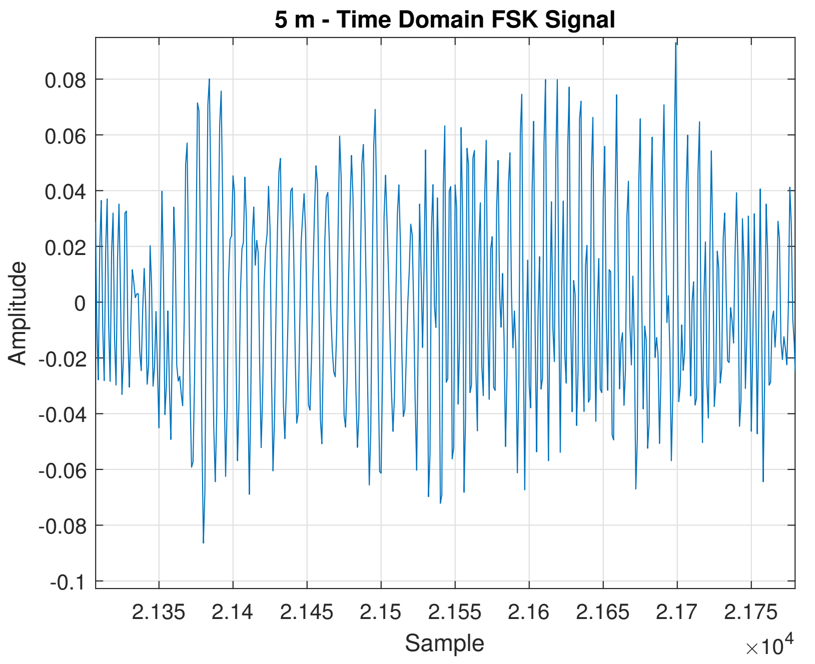


Figure 4.33: Time Domain Plot Recorded from a Distance of 5 m Showing Two Bits [0 1]

4.1.3 Data Transmission Speed Testing

As mentioned in section 3.6.3, a high data transfer speed is not essential in this application, however, it is necessary to have a data speed that makes the system feasible. This is an arbitrary speed as the data to be manipulated to suit the speed - if an accurate speed is

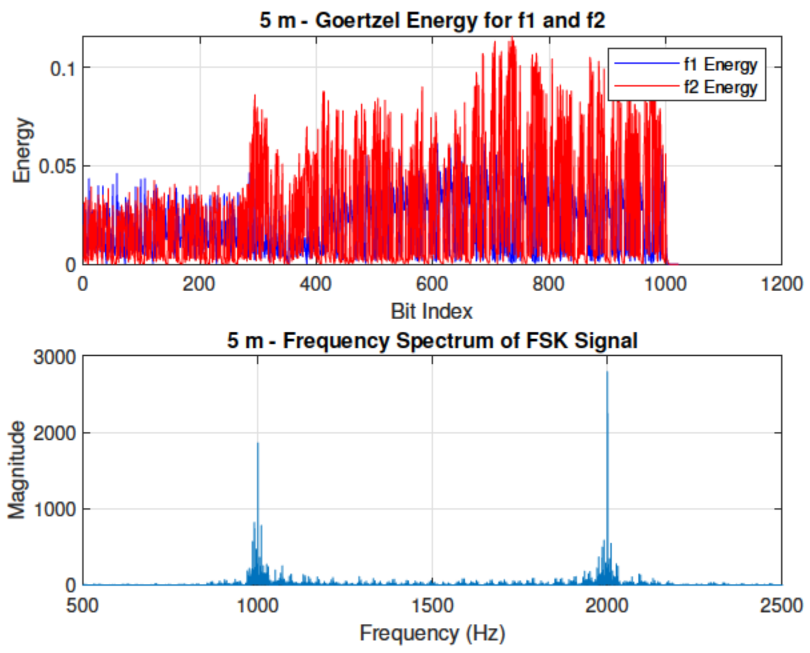


Figure 4.34: Full Message Goertzel Energy and Frequency Spectrum Plot Recorded from a Distance of 5 m

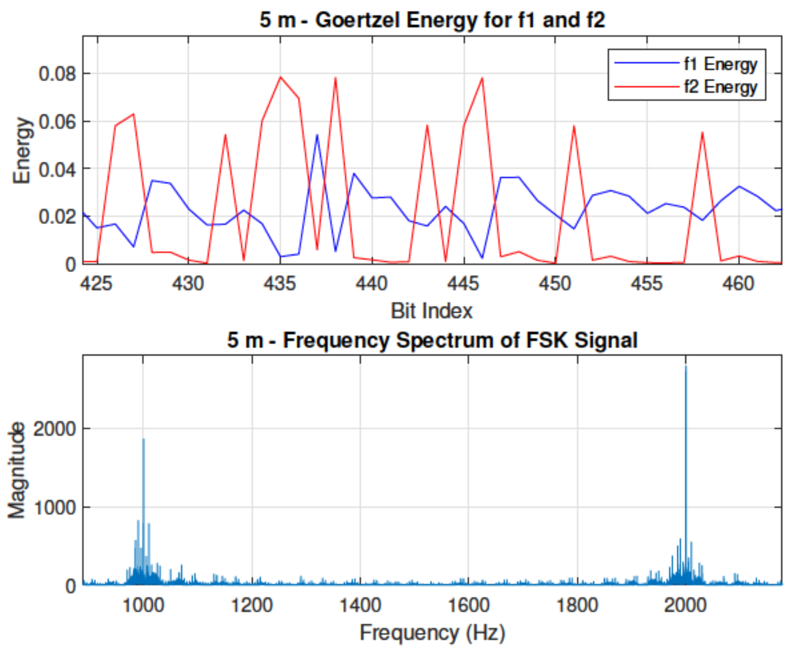


Figure 4.35: Zoomed in Goertzel Energy and Frequency Spectrum Plot Recorded from a Distance of 5 m

still low, it may mean the use of the system is limited to short bursts of data acquisitions, rather than things like firmware updates.

Table 4.3: Speed Test Results

Speed (bps)	**BER (%)**
25	0.00
50	0.00
100	0.00
200	0.1060
400	0.4880

25 bps

The signal at 25 bps was the original test signal speed and as such has shown a good response. This test, in particular, was included in the report to showcase the robust demodulation.

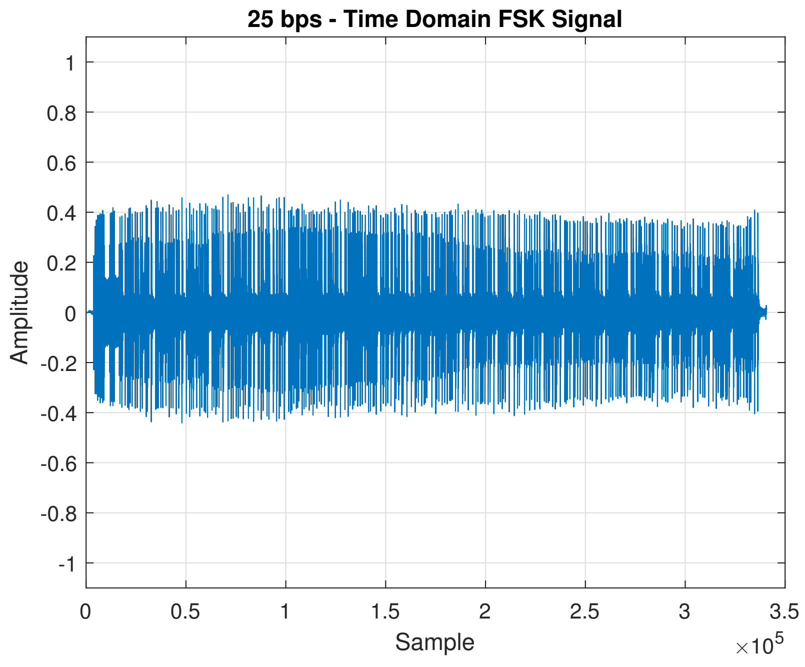


Figure 4.36: 25 bps Baud Rate Time Domain Plot Showing Full Message

The time domain plots, Figures 4.36 and 4.37 show the low amplitude recording, particularly $f1$. The energy capture, seen in 4.39 shows the result of this low energy recording. A similar pattern is observed in Figure 4.23, with an emerging trend of inconsistent energy capture for $f1$. This may be due to environmental factors like room resonance or interference, which could amplify or attenuate specific frequencies, or limitations in microphone sensitivity at these frequencies.

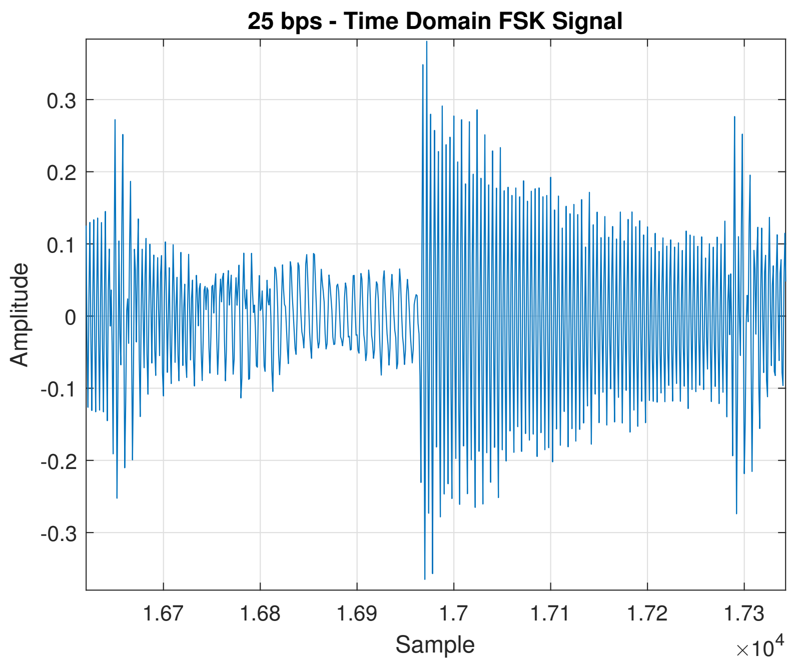


Figure 4.37: 25 bps Baud Rate Time Domain Plot Showing Two Bits [0 1]

It is hypothesised that resonance in the environment or equipment may be influencing the energy disparity between $f1$ and $f2$. Additionally, the timing windows used for sampling might play a role in capturing more energy at one frequency over the other. If the issue were to persist in future tests, investigations could occur into chirp signals to try to identify problem frequencies.

For the purposes of this project, what is apparent is the robust demodulation technique.

50 bps

A speed of 50 bps fetched excellent results. The time domain signal was captured as a clean signal and the energy levels between the two different frequencies was much more consistent.

100 bps

The time domain plots of the signal recorded at 100 bps, particularly Figure 4.44, again show low energy levels of $f1$.

Somewhat surprisingly, when taking into account Figure 4.46, the BER for this recording

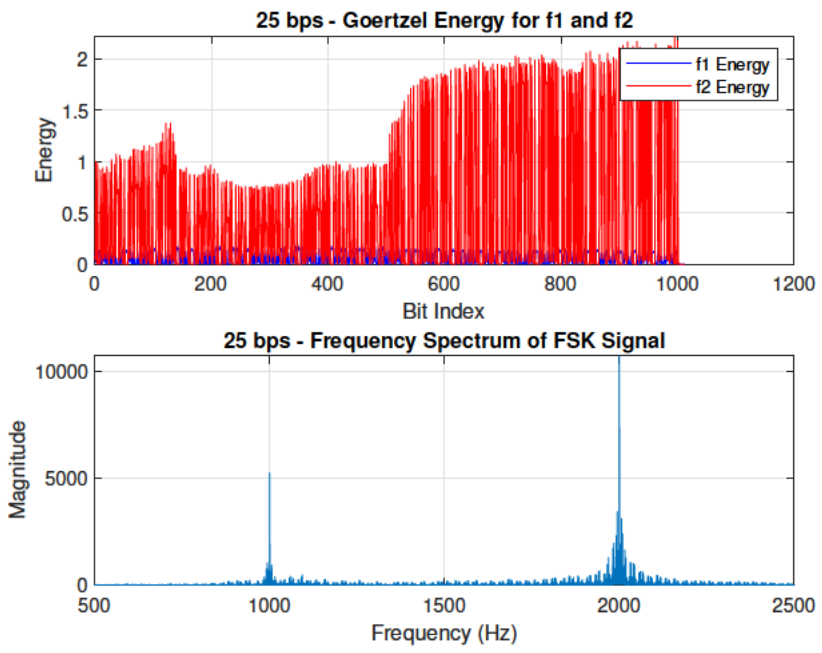


Figure 4.38: Full Message Goertzel Energy and Frequency Spectrum Plot Recorded at a Speed of 25 bps

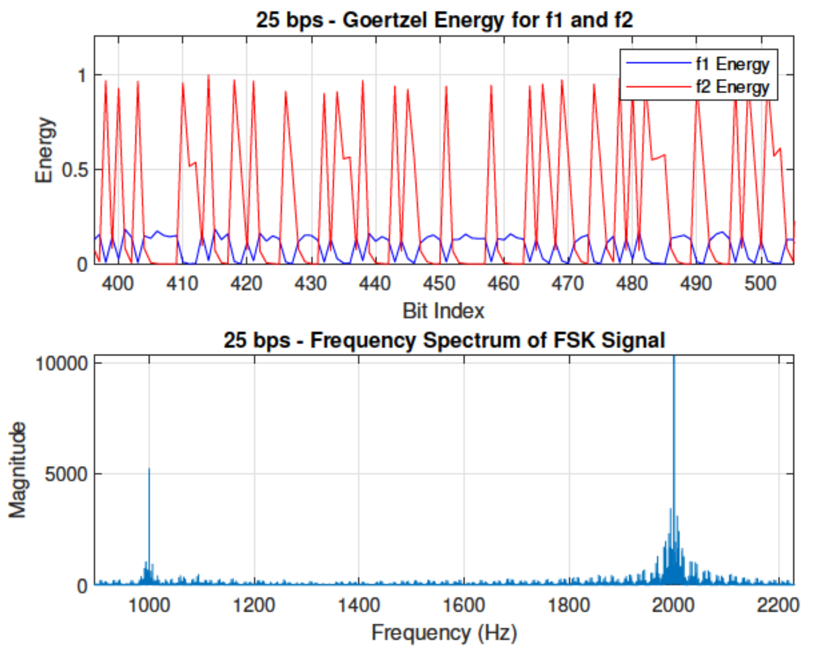


Figure 4.39: Zoomed in Goertzel Energy and Frequency Spectrum Plot Recorded at a Speed of 25 bps

was 0.00, the full message was demodulated with no errors. This is surprising when looking at the plot, with such low energies recorded from $f1$.

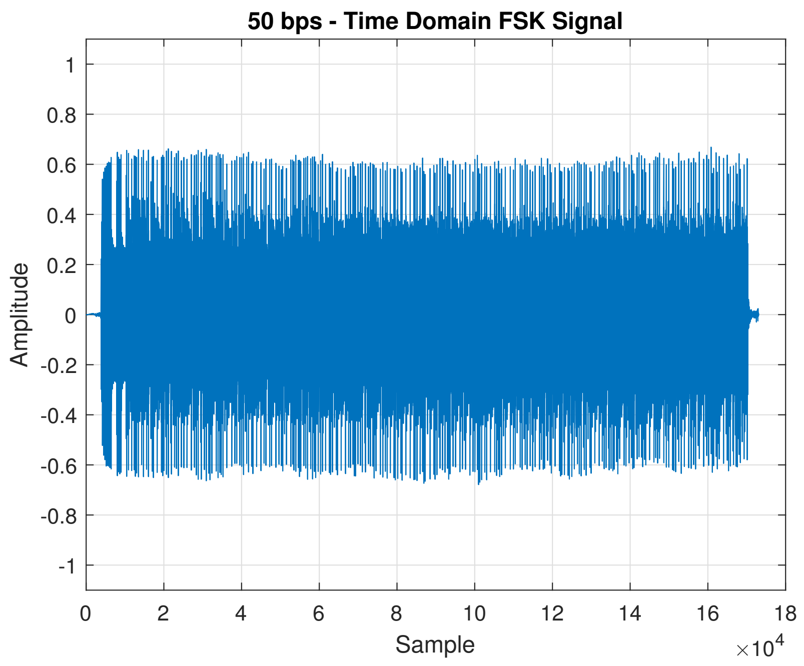


Figure 4.40: 50 bps Baud Rate Time Domain Plot Showing Full Message

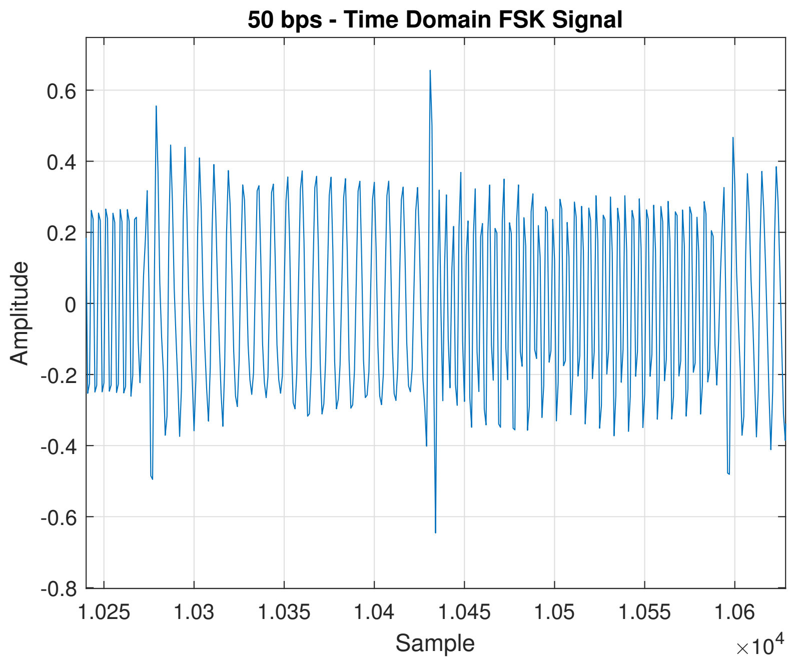


Figure 4.41: 50 bps Baud Rate Time Domain Plot Showing Two Bits [0 1]

200 bps

The signal captured at 200 bps showed errors creeping in. With a BER of 0.160, the message demodulated was close to being decipherable but showed too many errors for a quality signal to be received.

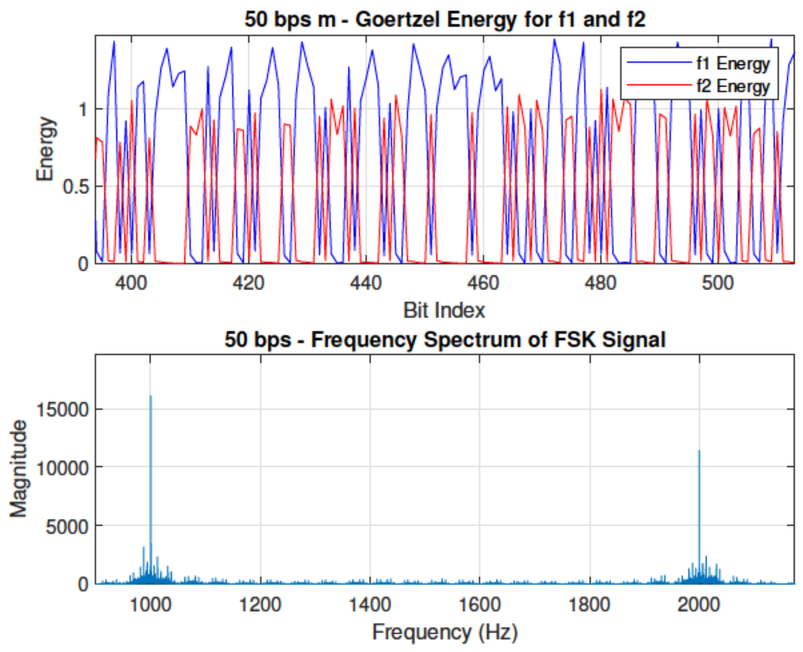


Figure 4.42: Zoomed in Goertzel Energy and Frequency Spectrum Plot Recorded at a Speed of 50 bps

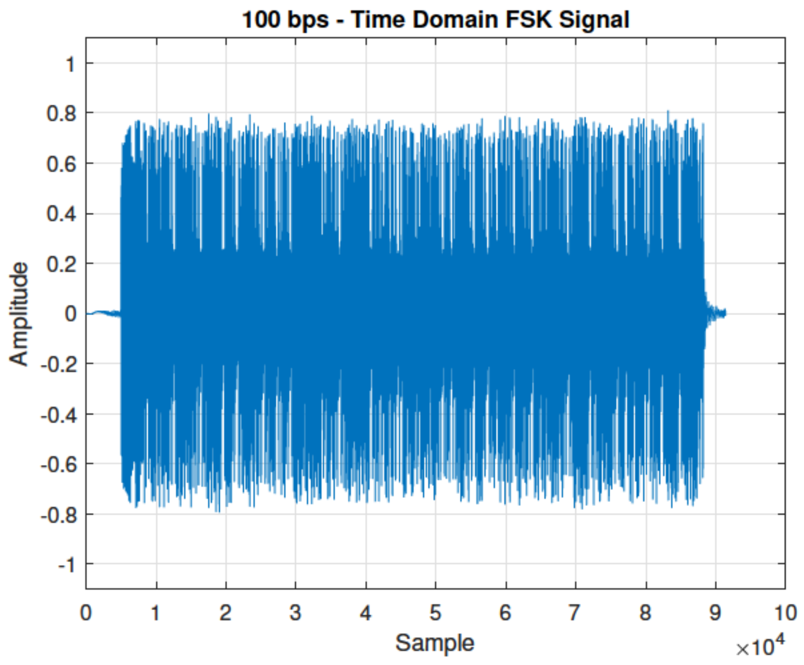


Figure 4.43: 100 bps Baud Rate Time Domain Plot Showing Full Message

Again, as in the case of the 100 bps signal and others, $f1$ energy was too low, though in this case, there appears to also be a small timing error. The peaks of $f1$ don't coincide with the troughs of $f2$.

For the current system, it appears as though the bit rate of 200 bps requires more work

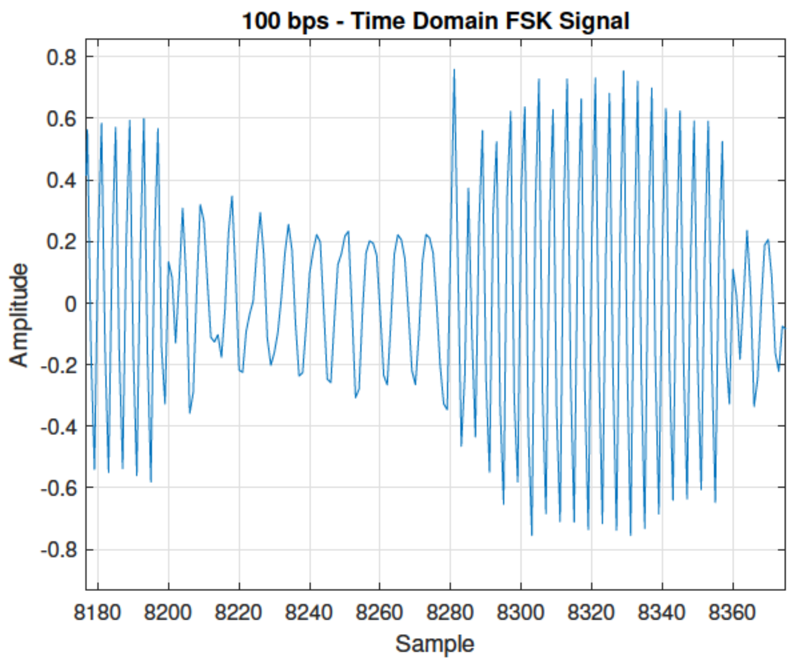


Figure 4.44: 100 bps Baud Rate Time Domain Plot Showing Two Bits [0 1]

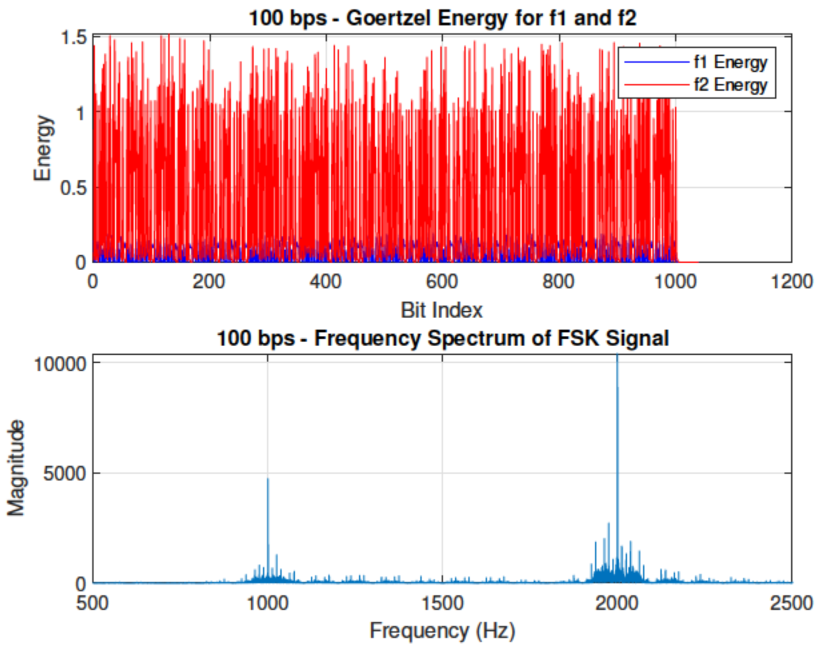


Figure 4.45: Full Message Goertzel Energy and Frequency Spectrum Plot Recorded at a Speed of 100 bps

for a reliable system.

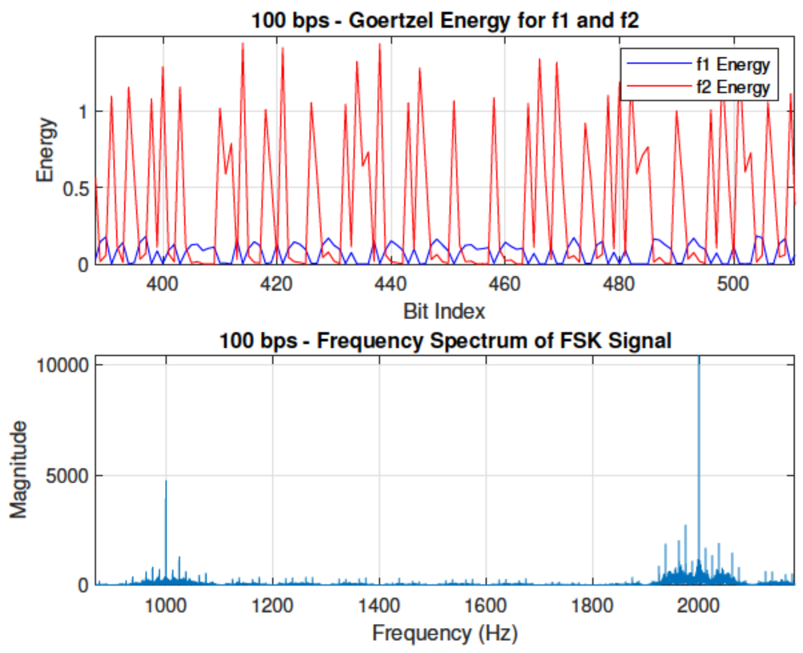


Figure 4.46: Zoomed in Goertzel Energy and Frequency Spectrum Plot Recorded at a Speed of 100 bps

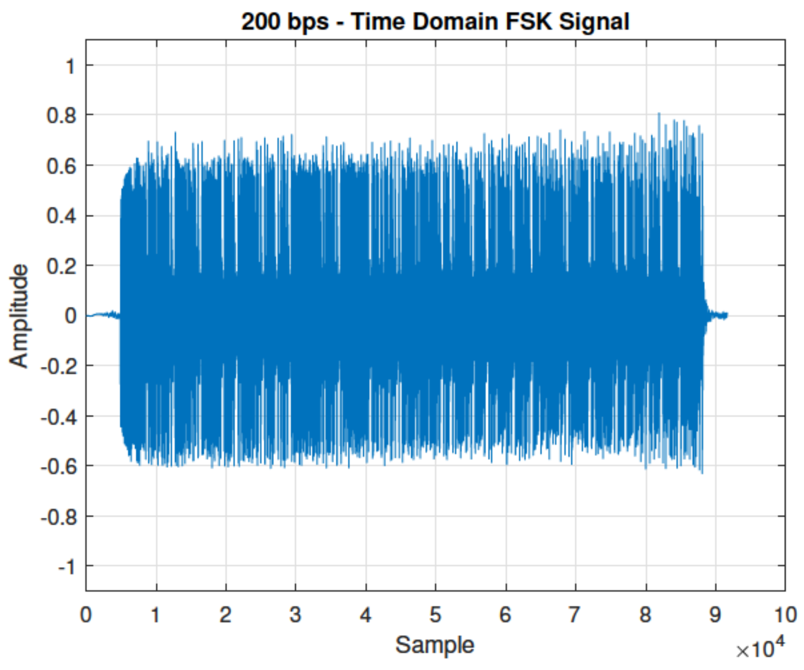


Figure 4.47: 200 bps Baud Rate Time Domain Plot Showing Full Message

400 bps

The 400 bps audio signal sounded the way it performed and the way it looks in Figures 4.51 to 4.54. Audibly, it sounded like the speaker was unable to keep up with the change

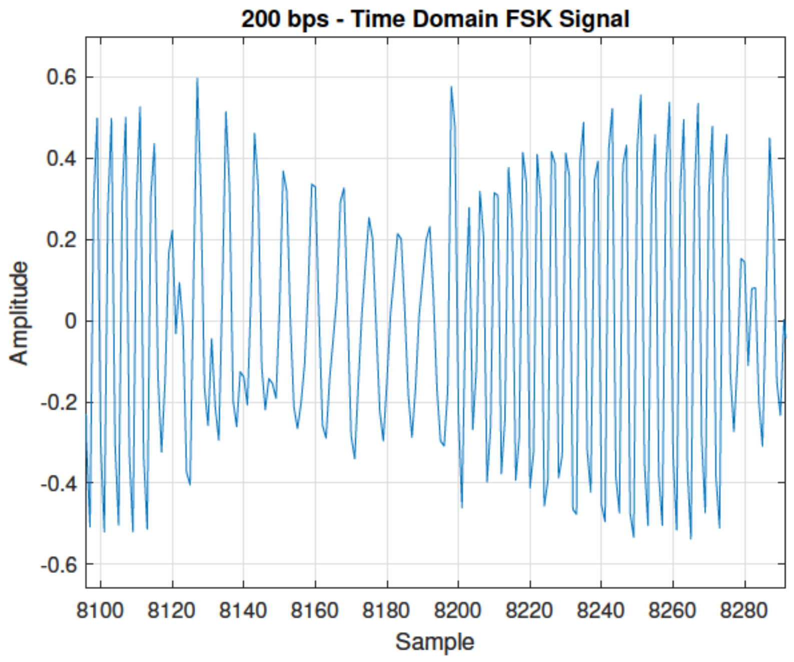


Figure 4.48: 200 bps Baud Rate Time Domain Plot Showing Two Bits [0 1]

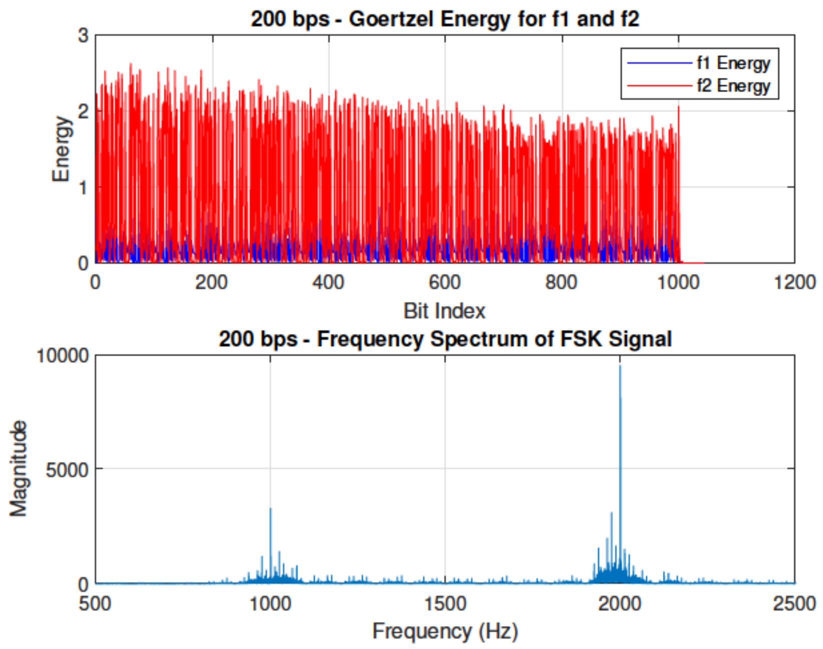


Figure 4.49: Full Message Goertzel Energy and Frequency Spectrum Plot Recorded at a Speed of 200 bps

rate of the frequencies and for these parameters it appeared to have reached a functional limit.

The disparity between the two energies captured has lead to a very poor result. Whilst the BER was 0.4880, it's important to remember the old adage that "a broken clock is

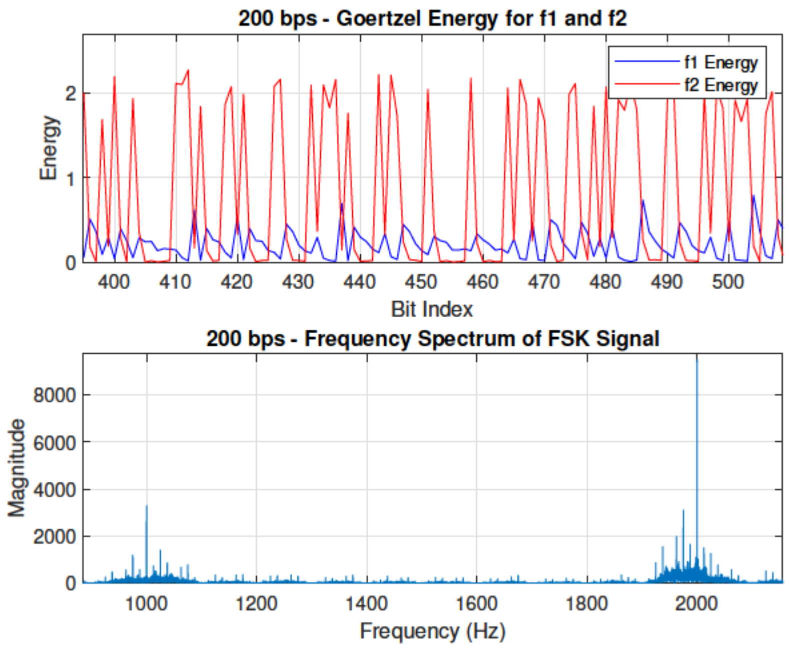


Figure 4.50: Zoomed in Goertzel Energy and Frequency Spectrum Plot Recorded at a Speed of 200 bps

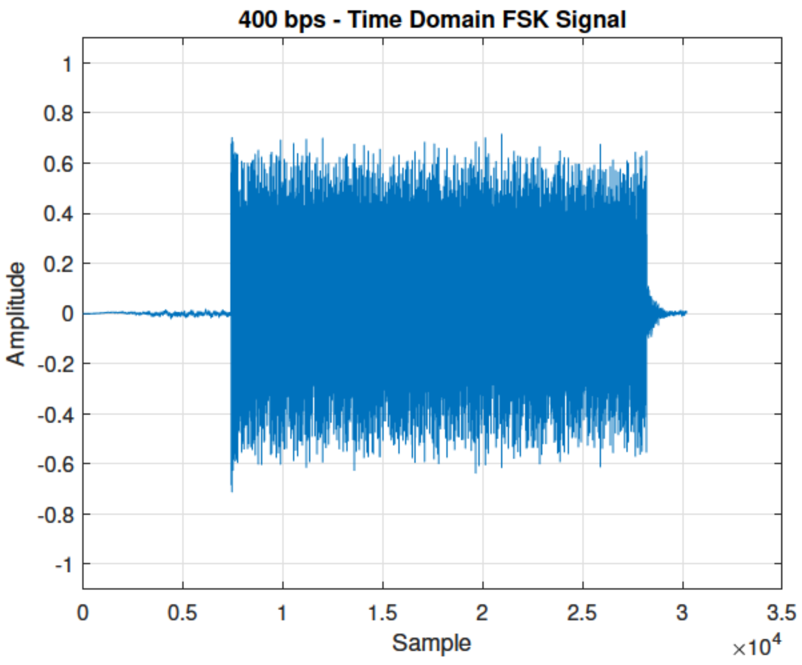


Figure 4.51: 400 bps Baud Rate Time Domain Plot Showing Full Message

right twice a day.” In a system where there are only two options (0 or 1) a BER of close to 50% is a very poor result. This test showed no successful demodulation.

Further testing could be done with frequencies that are closer together, while the small difference in frequency may make demodulation harder, it could illicit a better response

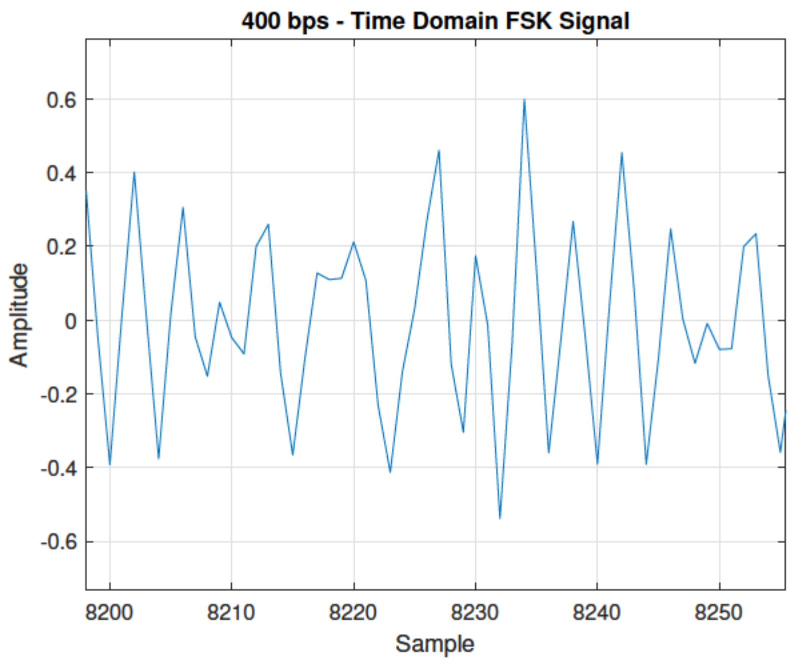


Figure 4.52: 400 bps Baud Rate Time Domain Plot Showing Two Bits [0 1]

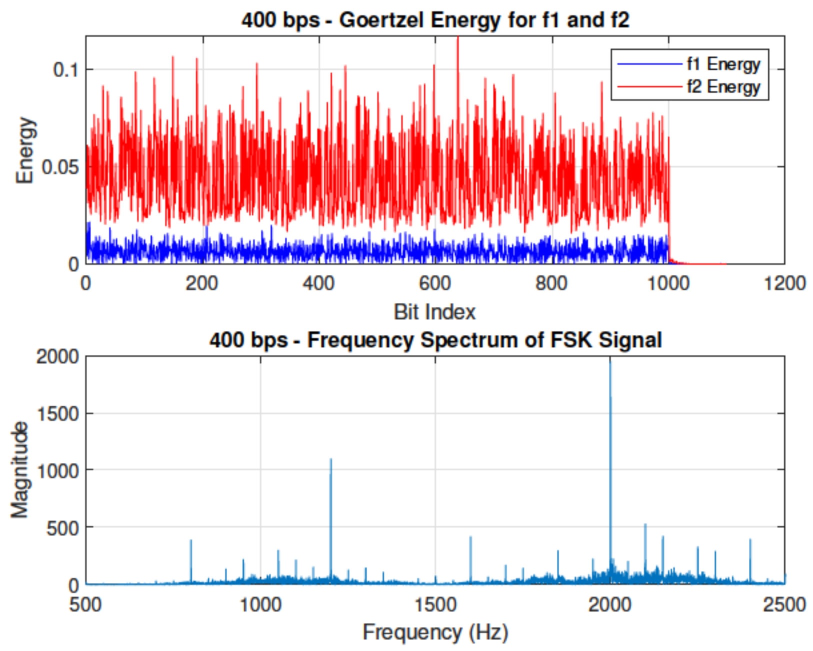


Figure 4.53: Full Message Goertzel Energy and Frequency Spectrum Plot Recorded at a Speed of 400 bps

from the speaker and a more even energy capture between different frequencies.

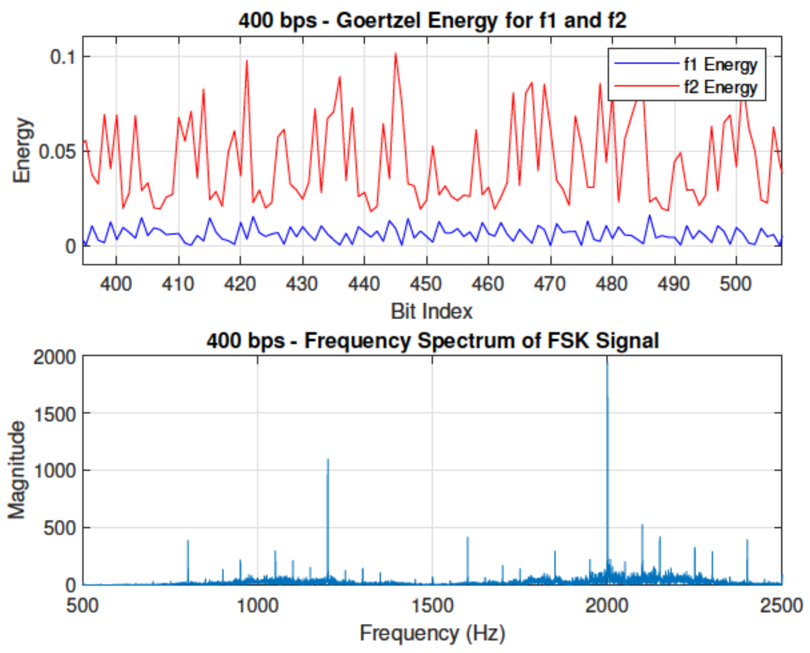


Figure 4.54: Zoomed in Goertzel Energy and Frequency Spectrum Plot Recorded at a Speed of 400 bps

4.1.4 Noise Resilience Testing

For the environment this project is investigating, resilience to noise if arguably the most important parameter. For this project to produce feasible results, it must perform in areas that are noise polluted by machinery, fans and other equipment.

As seen in 4.4, the system performed extremely well with a noise effected signal.

Table 4.4: Noise Test Results

SNR (dB)	**BER (%)**	**Rec. SNR**
0	0.00	6.38
10	0.00	11.25
20	0.00	16.95
30	0.00	26.14
40	0.00	29.56
Clean	0.00	32.83

Clean Signal (No Noise)

The first test was a benchmark test with no noise added to the signal, the results of which can be seen in Figures 4.55 to 4.58.

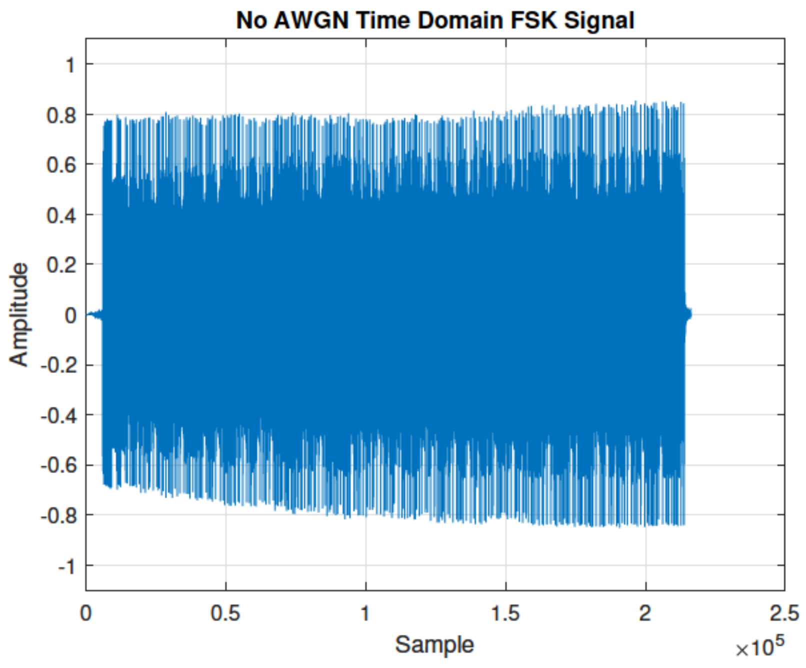


Figure 4.55: Clean Signal Time Domain Plot Showing Full Message

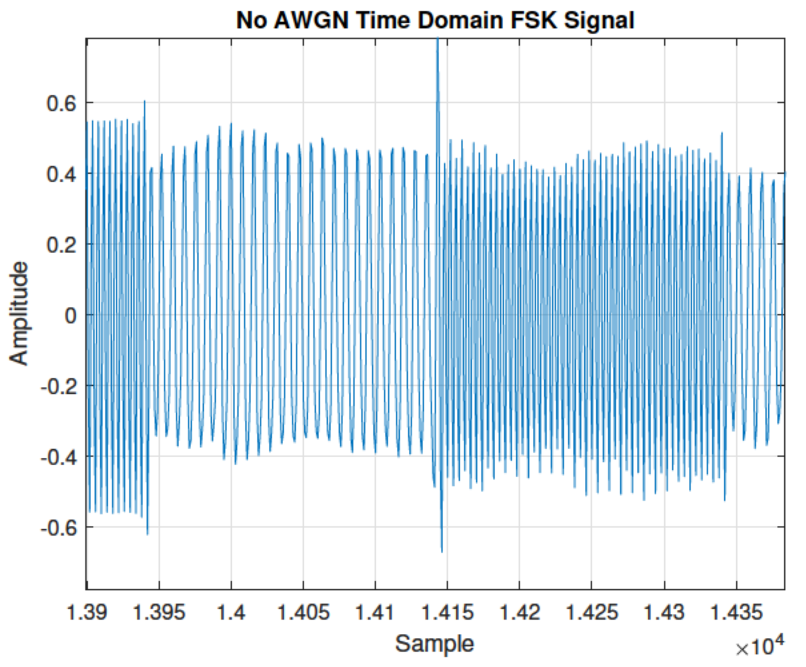


Figure 4.56: Clean Signal Time Domain Plot Showing Two Bits [0 1]

The clean signal performed well, as expected and creates a good benchmark.

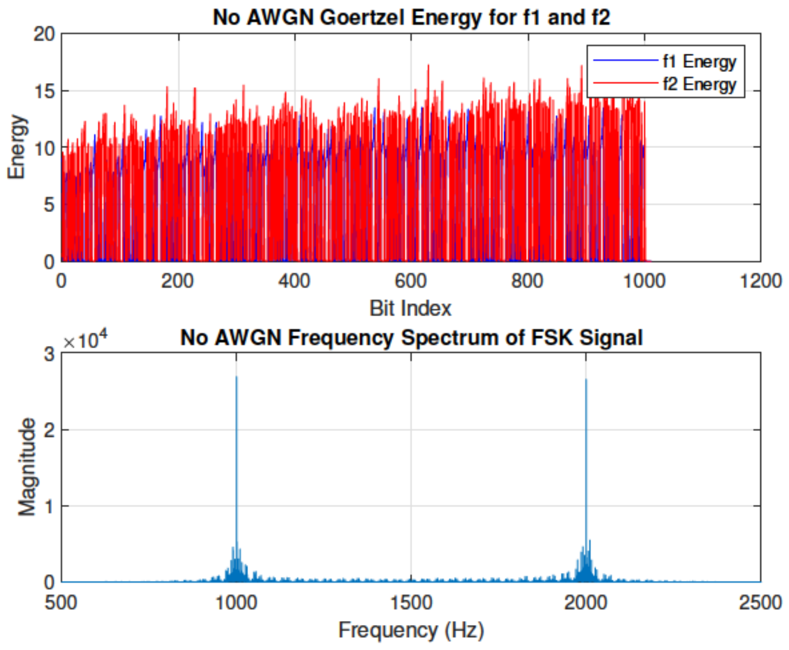


Figure 4.57: Full Message Goertzel Energy and Frequency Spectrum Plot for Clean Signal

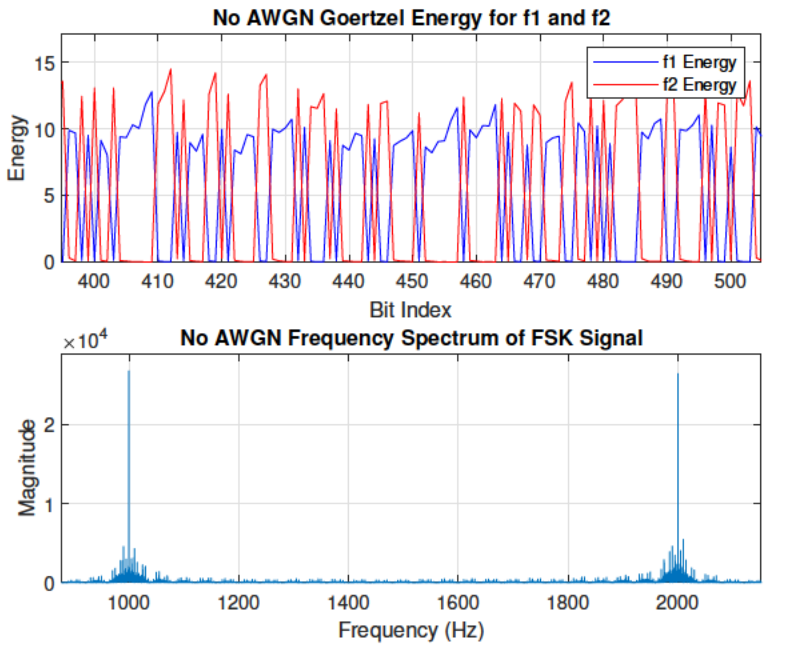


Figure 4.58: Zoomed in Goertzel Energy and Frequency Spectrum Plot for Clean Signal

Following on from section 4.1.4, it is interesting to note the disparity between $f1$ and $f2$ is not nearly as pronounced in this test. This coincided with a necessary change of test environment, whilst this can't be used as conclusive evidence, it is a good indication that the environment may play a role in the effectiveness of the recorded signals.

These results demonstrate that the system can accurately demodulate signals even in environments with significant noise interference, reinforcing the applicability of this model in real-world settings where background noise is prevalent.

40 dB SNR

At an SNR of 40 dB, there is little effect of noise expected to play a part in the received signal.

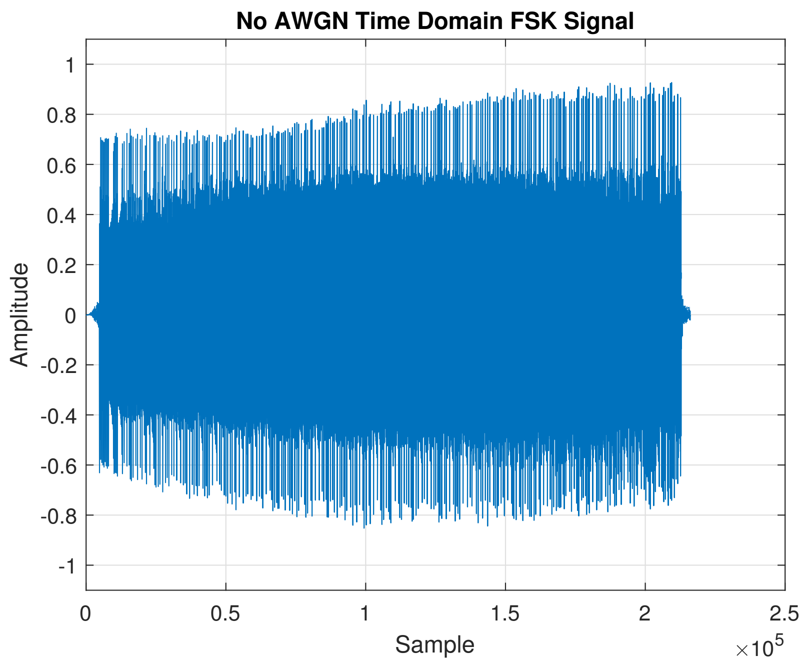


Figure 4.59: 40 dB SNR Time Domain Plot Showing Full Message

In practice, while the noise was audible, it was a minimal difference from the clean signal. Figure 4.62 shows just how clean the received signal is.

30 dB SNR

The signal modulated with an SNR of 30 dB again showed little evidence of the added noise in the captured signal. The signal was able to be demodulated successfully.

Figure 4.66 does show confirm a curious pattern in the accrual of the energy signal for each frequency emerging for this set of tests. The energy captured for each frequency appears to continue to increase before abruptly stopping, giving a sharp end to the bit.

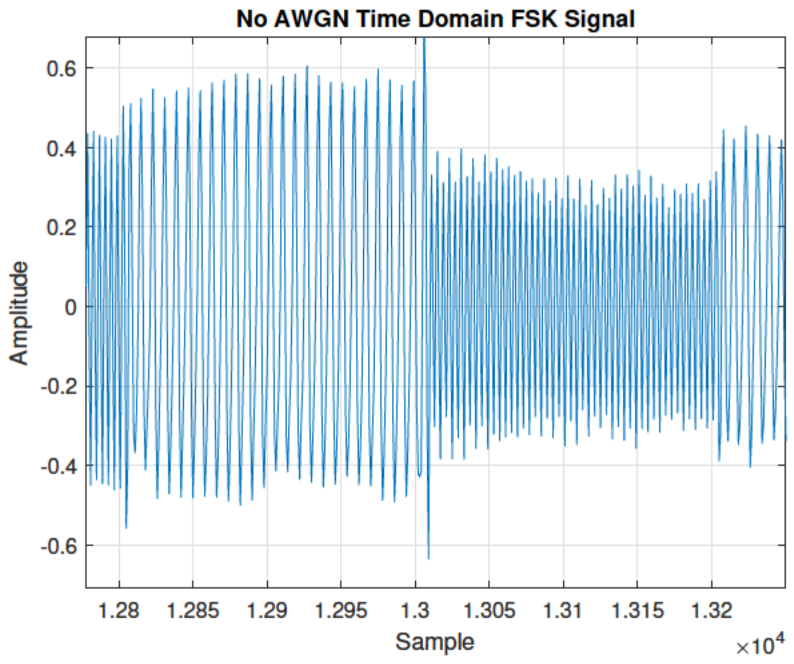


Figure 4.60: 40 dB SNR Time Domain Plot Showing Two Bits [0 1]

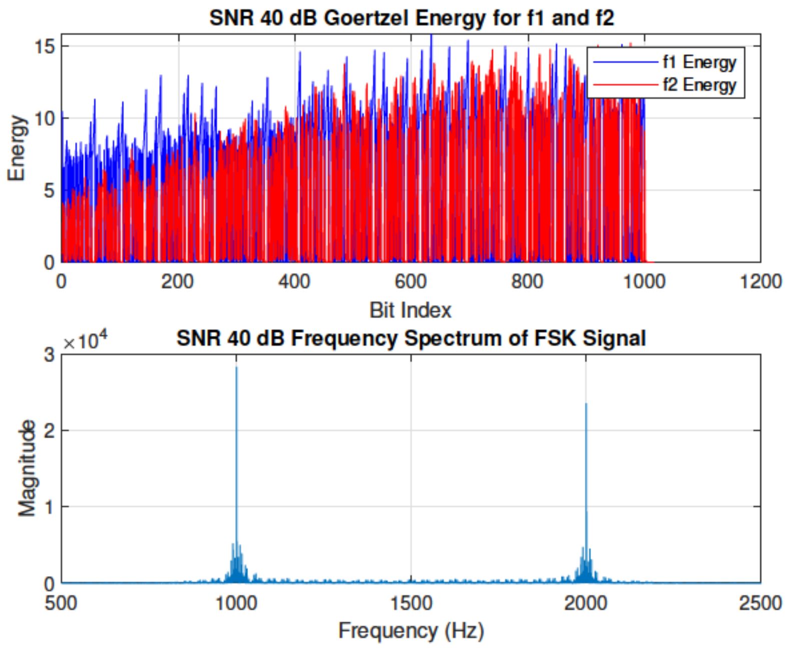


Figure 4.61: Full Message Goertzel Energy and Frequency Spectrum Plot Modulated at 40 dB SNR

This becomes more pronounced with the addition of more noise to the signal.

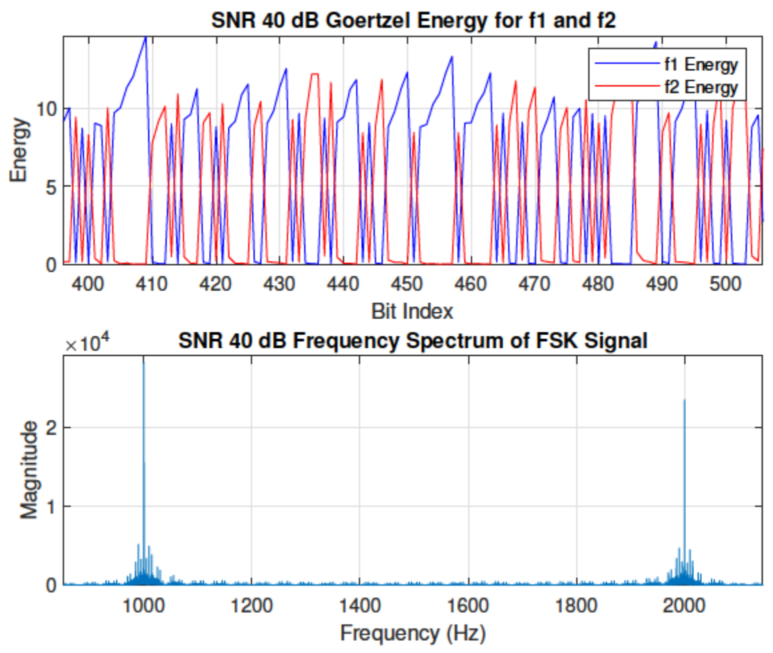


Figure 4.62: Zoomed in Goertzel Energy and Frequency Spectrum Plot Modulated at 40 dB SNR

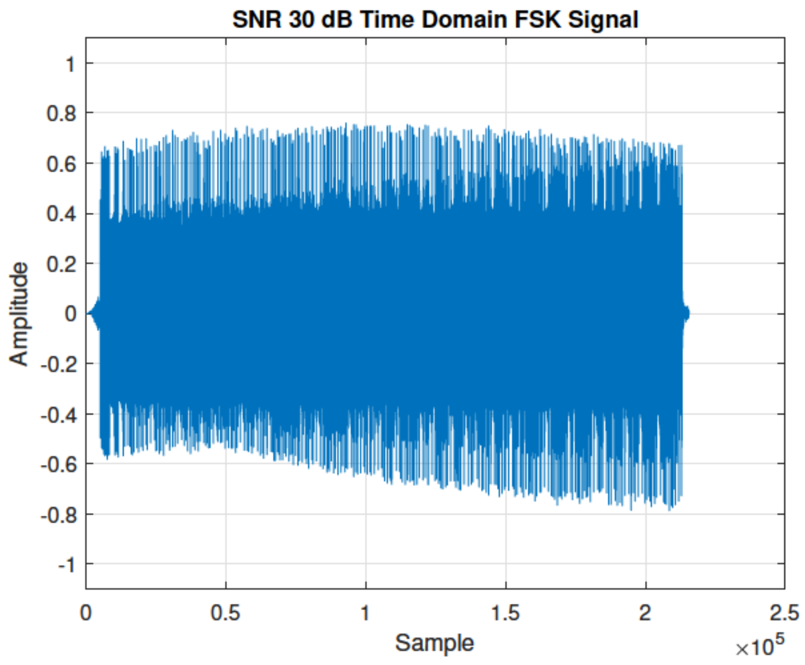


Figure 4.63: 30 dB SNR Time Domain Plot Showing Full Message

20 dB SNR

The 20 dB noisy signal again shows great demodulation performance in the presence of noise. At this noise level, the signal is clearly muffled the a lot of white noise able to be

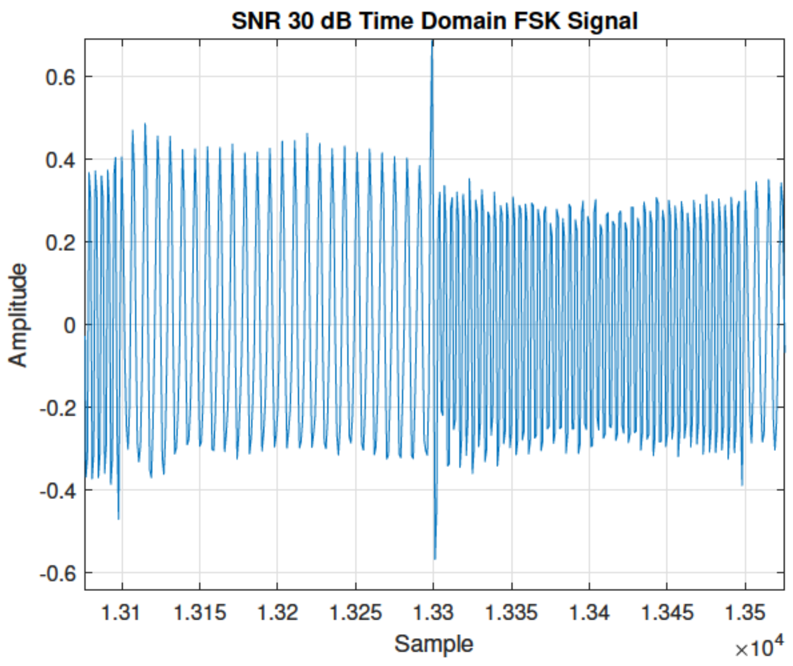


Figure 4.64: 30 dB SNR Time Domain Plot Showing Two Bits [0 1]

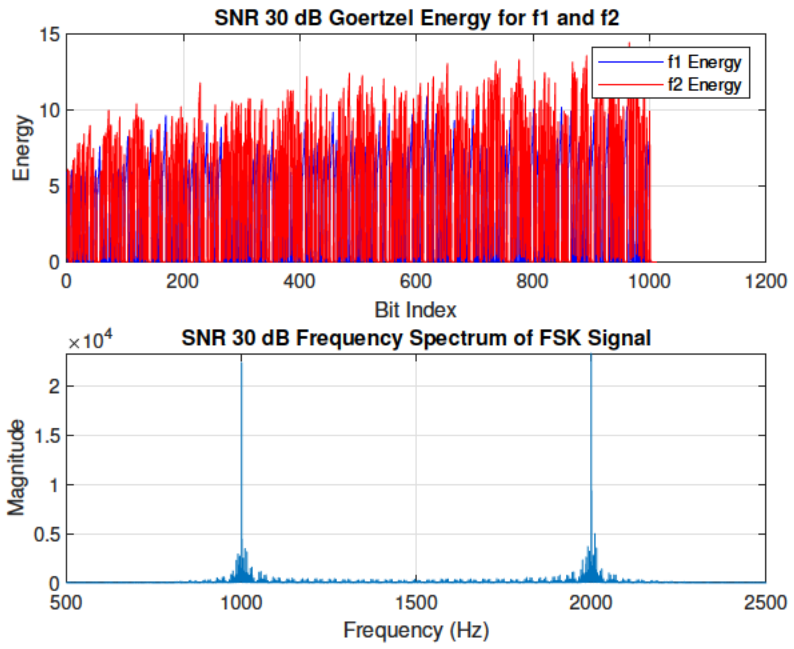


Figure 4.65: Full Message Goertzel Energy and Frequency Spectrum Plot Modulated at 30 dB SNR

heard.

Despite the noise, the time domain signal captured remains very clean and the frequencies are easily discernible.

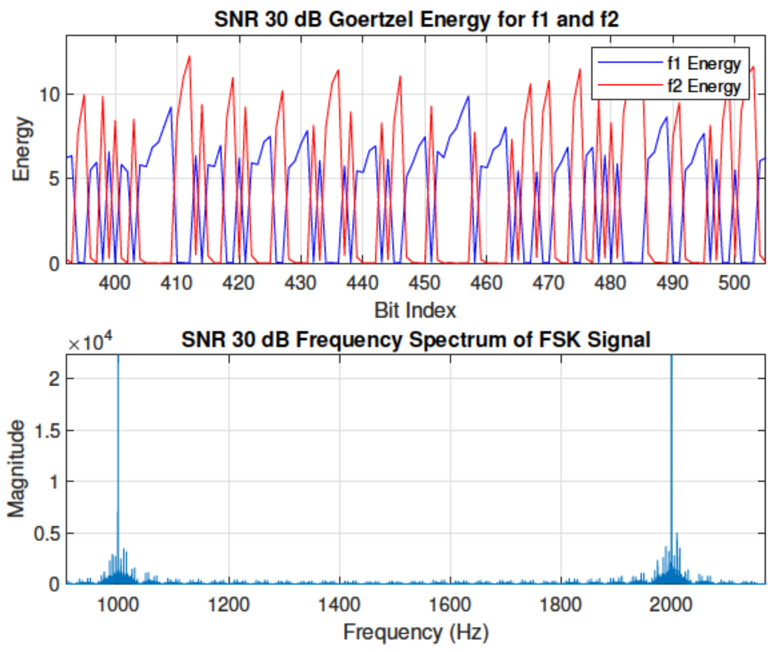


Figure 4.66: Zoomed in Goertzel Energy and Frequency Spectrum Plot Modulated at 30 dB SNR

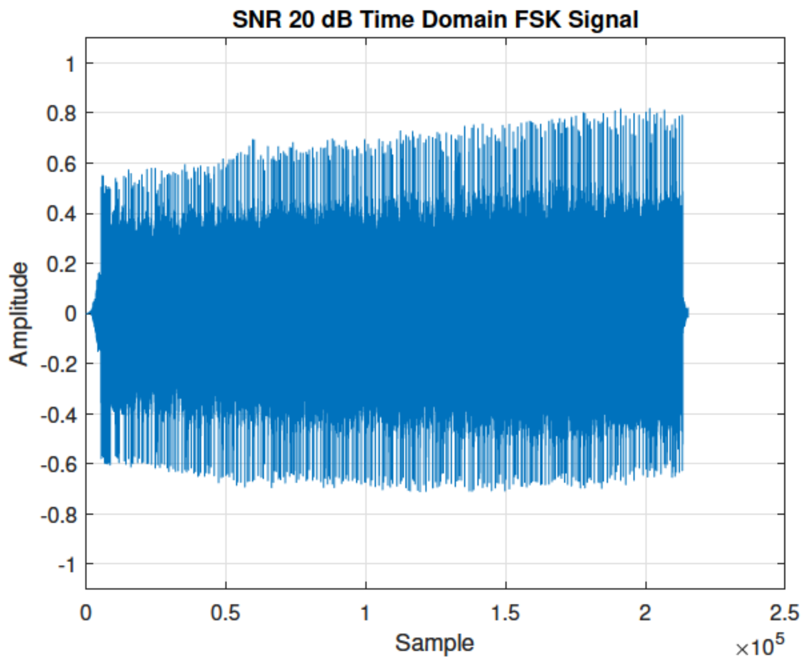


Figure 4.67: 20 dB SNR Time Domain Plot Showing Full Message

Figure 4.69 shows and interesting result where the energies of both frequencies appear to increase in size with the length of the signal.

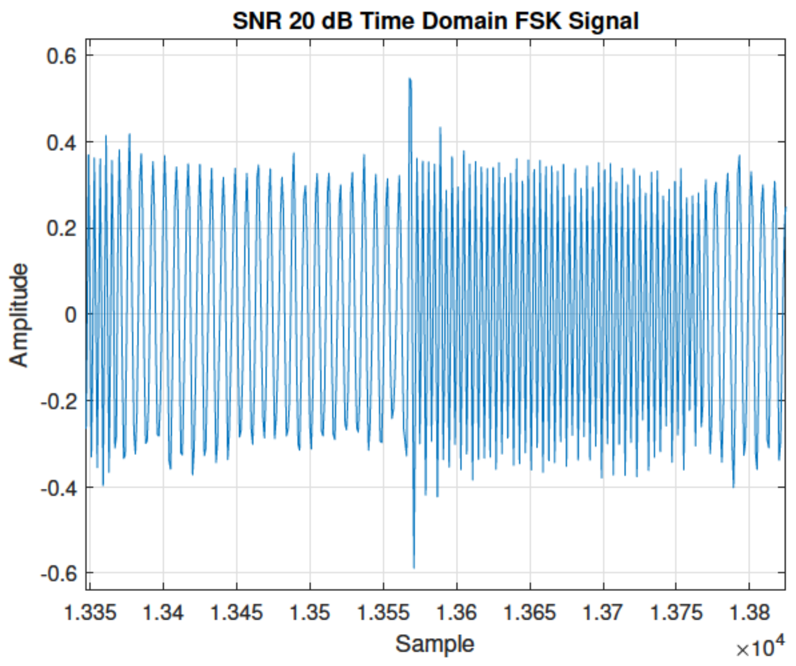


Figure 4.68: 20 dB SNR Time Domain Plot Showing Two Bits [0 1]

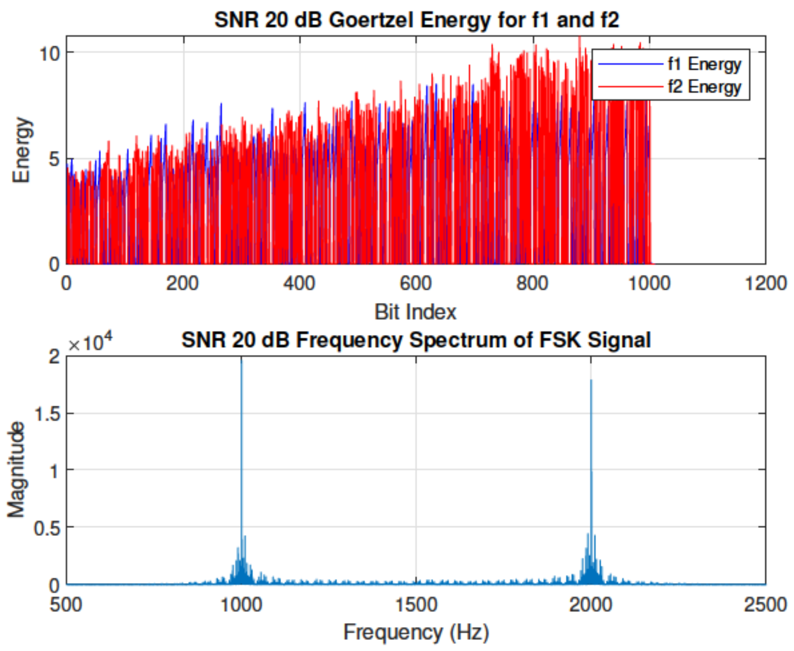


Figure 4.69: Full Message Goertzel Energy and Frequency Spectrum Plot Modulated at 20 dB SNR

10 dB SNR

The signal modulated with an SNR of 10 dB shows a visibly noisy signal in Figure 4.71. Audibly, this signal was very noise and the message was difficult to ascertain to the ear.

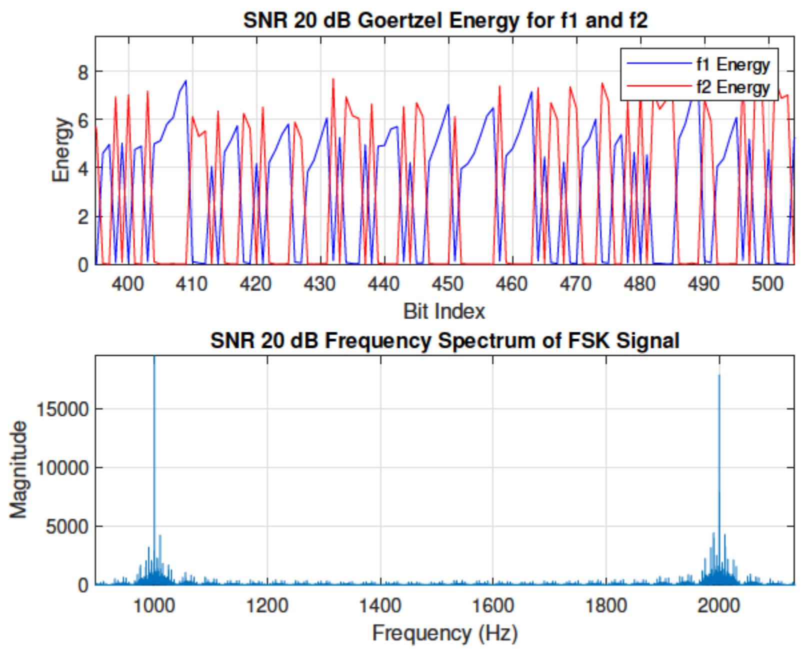


Figure 4.70: Zoomed in Goertzel Energy and Frequency Spectrum Plot Modulated at 20 dB SNR

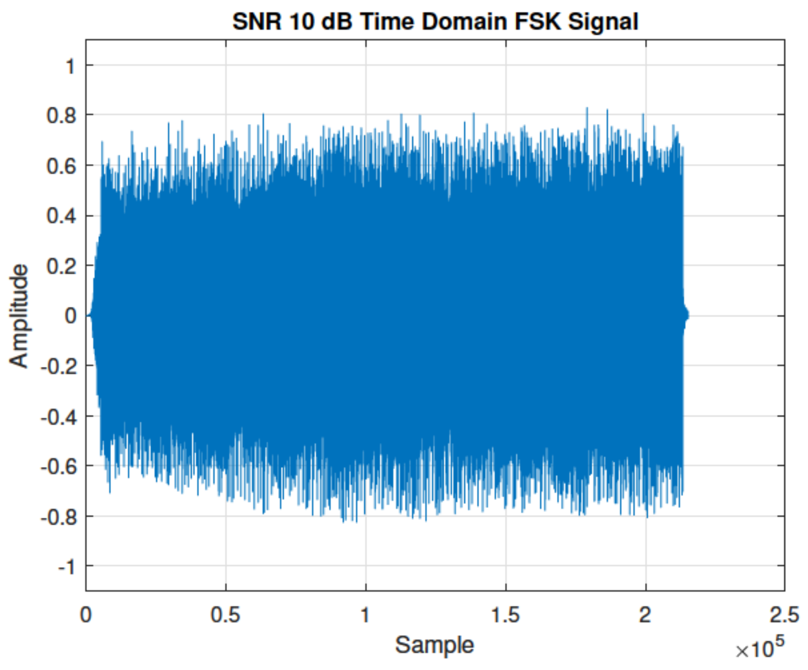


Figure 4.71: 10 dB SNR Time Domain Plot Showing Full Message

As seen in Figure 4.74, the system had no problem demodulating the signal, with a 100% success rate.

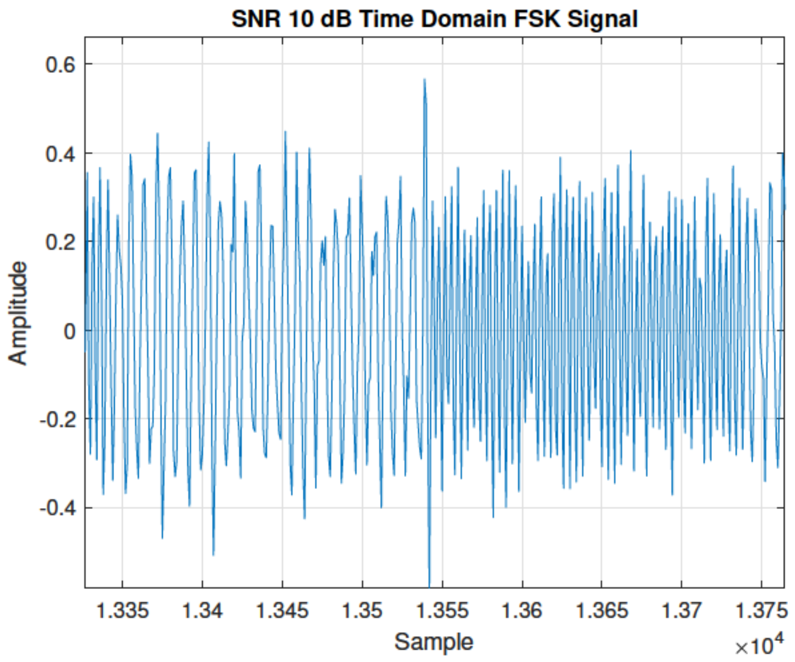


Figure 4.72: 10 dB SNR Time Domain Plot Showing Two Bits [0 1]

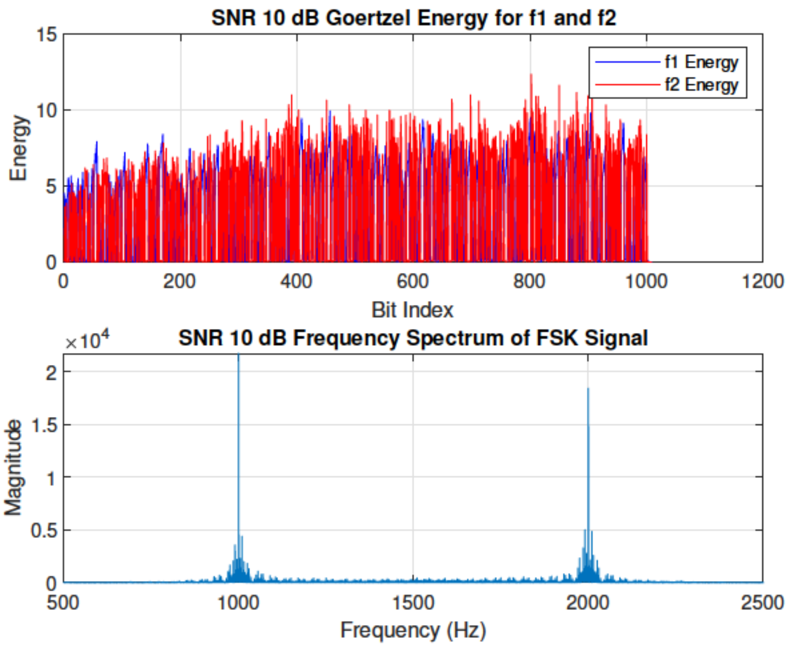


Figure 4.73: Full Message Goertzel Energy and Frequency Spectrum Plot Modulated at 10 dB SNR

0 dB SNR

This audio was very difficult to hear the message within. After hearing the audio, it was assumed there was little chance for an accurately demodulated message.

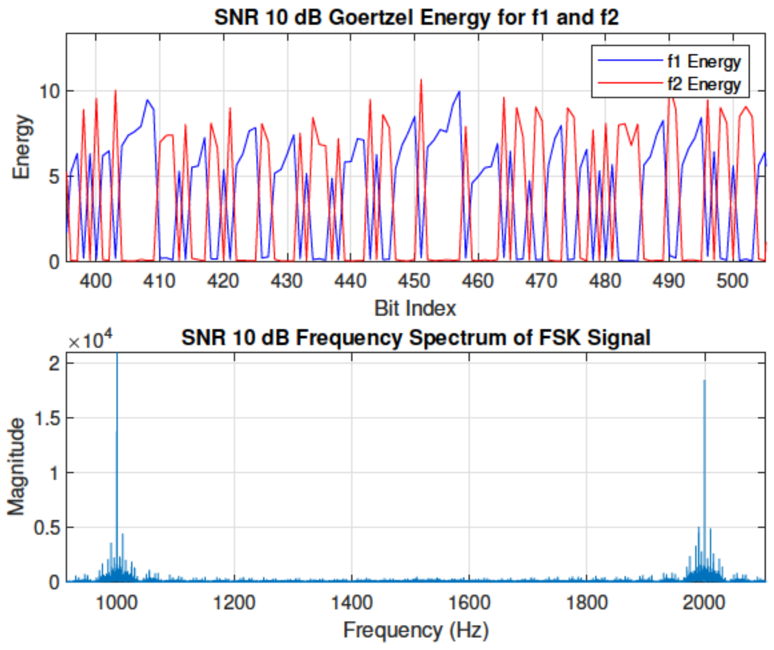


Figure 4.74: Zoomed in Goertzel Energy and Frequency Spectrum Plot Modulated at 10 dB SNR

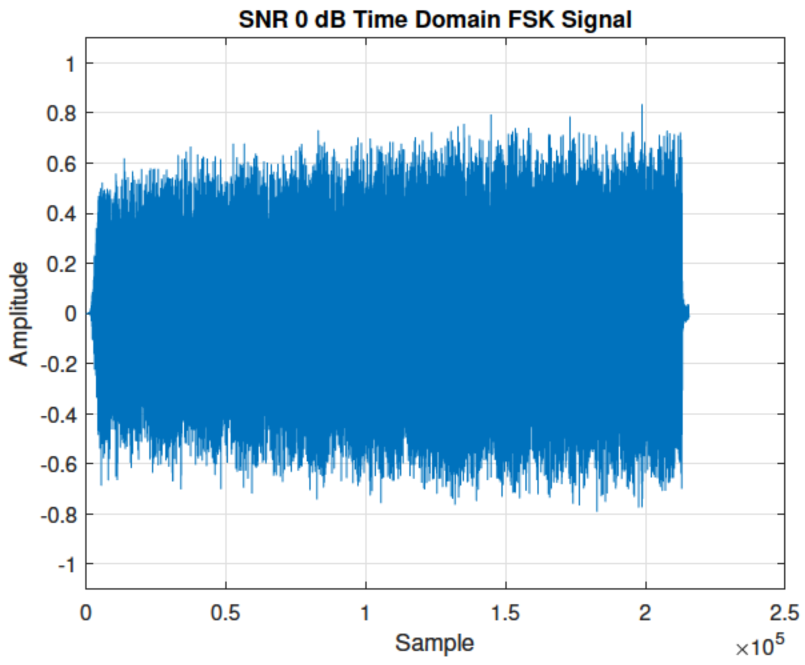


Figure 4.75: 0 dB SNR Time Domain Plot Showing Full Message

Figure 4.76 shows just how noisy the signal was. remembering the message start index is being chosen manually for this test, the starting point was very difficult to determine.

Despite the white noise being at parity with the modulated signal at 0 dB, the system had no trouble demodulating the message, again, with a 100% success rate.

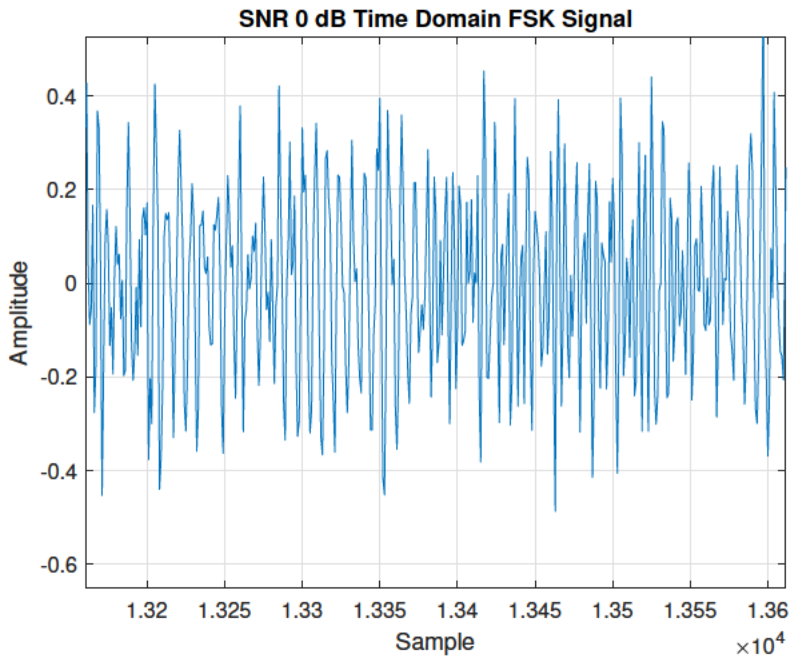


Figure 4.76: 0 dB SNR Time Domain Plot Showing Two Bits [0 1]

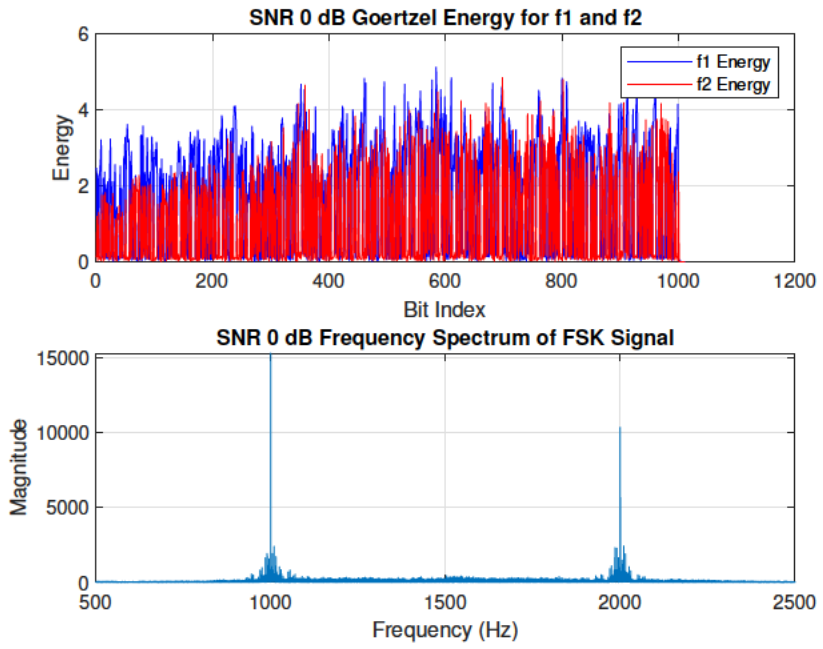


Figure 4.77: Full Message Goertzel Energy and Frequency Spectrum Plot Modulated at 0 dB SNR

Figure 4.78 shows just how strong a response was performed by the demodulation system in this instance, suggesting that the SNR could be much lower and the demodulation still be accurate.

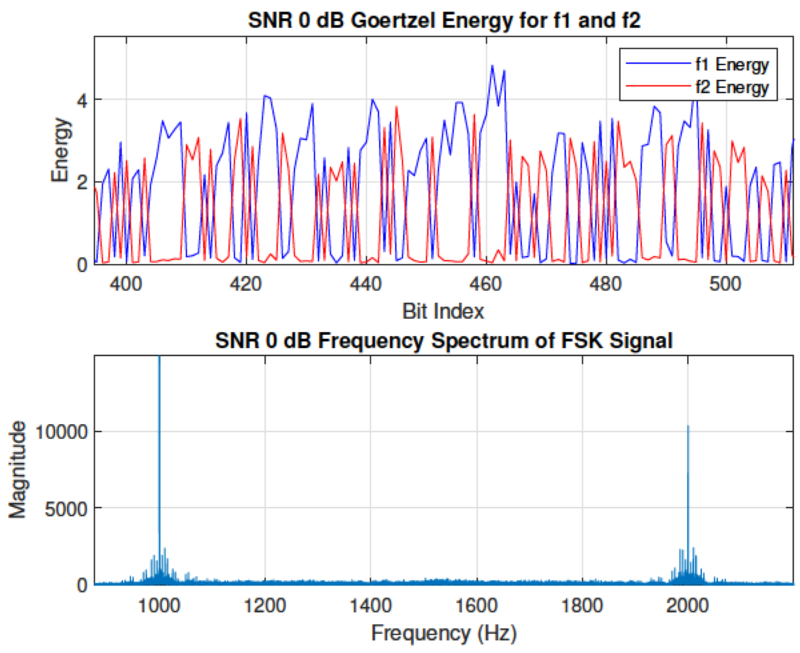


Figure 4.78: Zoomed in Goertzel Energy and Frequency Spectrum Plot Modulated at 0 dB SNR

4.1.5 Preamble Detection

For bidirectional communication to occur between devices, the programs must be able to detect the start and end of messages. For this to occur, the preamble (that can work as both message detection and as a "handshake") and postamble detection must work effectively.

As seen in 4.5, there was a small correlation between the speed of transmission and the accuracy of the start of message detection, though this can be attributed to the reducing symbol length at increasing speeds, determining accuracy percentage. Conversely, there can be no correlation made between speed and accuracy as from a sample index perspective.

It should be noted that the manual detection of the message was only done by eye, from the signal plot (as can be seen in Figure 4.80) and may not be perfectly accurate itself.

Table 4.5: Preamble Detection Accuracy at Different Bit Rates

Bit Rate (bps)	Accuracy (bits)	Accuracy (%)
25	3.667	98.85
50	9.333	94.17
100	7.333	90.83

25 bps

At a bit rate of 25 bps, the symbol length is 320 samples. This allows for some margin of error, and the accuracy does not significantly impact the ability to decode the message.

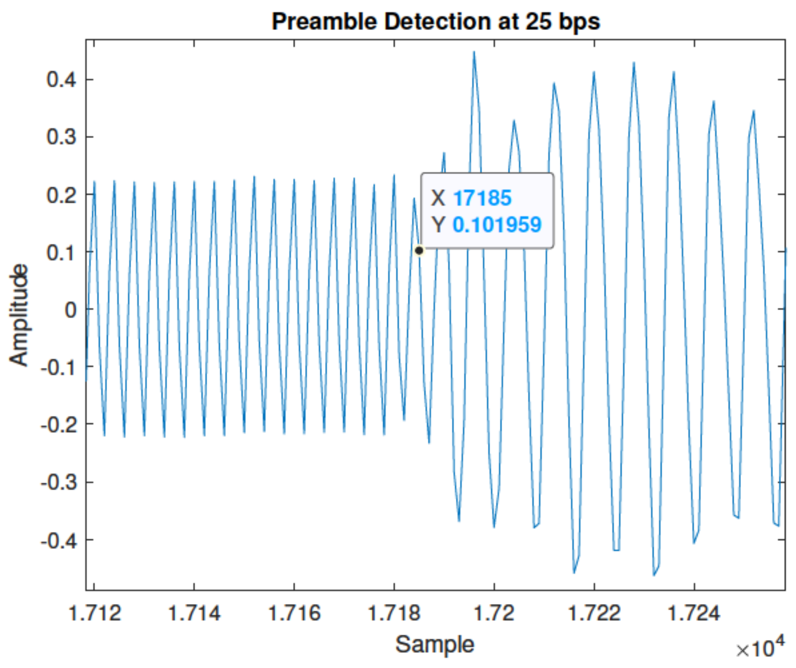
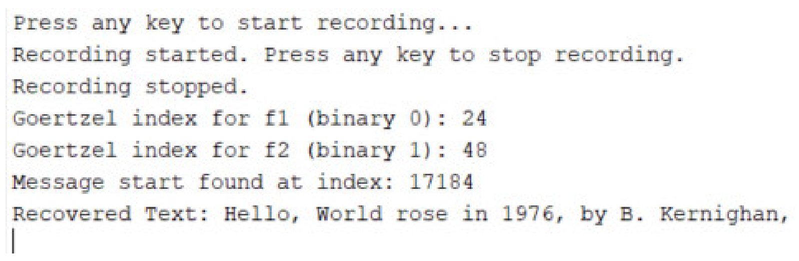


Figure 4.79: 25 bps recorded signal showing manually found start of message

Figure 4.79 shows the process of manually determining the sample index of the start of the message. The signal plot is zoomed in to the end of the preamble and the start of the message, where the sample representing the transition from preamble to message is selected.

Figure 4.80 shows the output in the command window from the MATLAB program `ENP4111_Working_Preamble_demod.m`. The output displays the Goertzel bin indices for $f1$ and $f2$, the detected sample index at the start of the message and the recovered message.



```
Press any key to start recording...
Recording started. Press any key to stop recording.
Recording stopped.
Goertzel index for f1 (binary 0): 24
Goertzel index for f2 (binary 1): 48
Message start found at index: 17184
Recovered Text: Hello, World rose in 1976, by B. Kernighan,
|
```

Figure 4.80: Excerpt from Command Window showing index of start of message detected

Table 4.6: Preamble Detection Accuracy at 25 bps

	Detected start of message (sample)	**Actual start of message (sample)**	**Accuracy (bits)**	**Accuracy (%)**
Test 1	17184	17185	1	99.69
Test 2	18356	18364	8	97.50
Test 3	19081	19083	2	99.37

50 bps

At a bit rate of 50 bps, the symbol length is 160 samples. The shorter symbol length results in a slightly reduced accuracy compared to 25 bps, but the preamble detection remains within acceptable limits for message decoding.

Table 4.7: Preamble Detection Accuracy at 50 bps

	Detected start of message (sample)	**Actual start of message (sample)**	**Accuracy (bits)**	**Accuracy (%)**
Test 1	13511	13527	16	90.00
Test 2	13566	13571	5	96.88
Test 3	11265	11258	7	95.63

100 bps

At a bit rate of 100 bps, the symbol length is further reduced to 80 samples. This reduction in symbol length leads to a noticeable decrease in accuracy, though it remains within acceptable bounds for message detection.

Table 4.8: Preamble Detection Accuracy at 100 bps

	Detected start of message (sample)	Actual start of message (sample)	Accuracy (bits)	Accuracy (%)
Test 1	10390	10396	6	92.50
Test 2	9857	9865	8	90.00
Test 3	9099	9091	8	90.00

Preamble Testing Summary

Overall the preamble detection was successful across all tests, with only minor inaccuracies observed. These inaccuracies become more pronounced at higher bit rates, where the symbol length is shorter, resulting in a smaller window for skipping samples and adjusting for errors.

Despite these challenges, the overall performance of the preamble detection algorithm was satisfactory, demonstrating reliable detection of the start of the message in all cases. The accuracy slightly decreased as the bit rate increased, but it remained within acceptable limits for effective communication.

4.2 JavaScript API Development

The development of the API was a significant part of this project, taking up a significant portion of the time spent. The goal was to create a web-based interface that could modulate and demodulate acoustic signals using Frequency Shift Keying (FSK) through access to a server. However, several challenges were encountered along the way, particularly with hosting the API and integrating all the necessary functionality. The full code can be found in Appendix E.

4.2.1 Progress and Challenges

The API started as a locally hosted HTML page, which was incrementally improved. Initially, the focus was on creating a simple user interface (UI) for uploading text files, selecting baud rates, and transmitting modulated audio signals. While a lot of the core functionality was implemented — such as file upload, binary conversion, modulation, audio output, basic recording, and an attempt at demodulation, the API never progressed beyond local hosting.

One of the major challenges was transitioning from a locally hosted environment to a fully hosted API. Despite learning about hosting techniques and attempting to deploy the API on a server, this step was not successfully completed. Additionally, as the project evolved, the demodulation technique transitioned from using Fast Fourier Transform (FFT), to Goertzel's algorithm in MATLAB. Unfortunately, this update was not reflected in the API, which still uses FFT for demodulation.

4.2.2 Functionality Implemented

Despite these challenges, several key features were successfully implemented in the locally hosted HTML page:

- **Graphical User Interface (GUI):** A simple interface uses buttons, inputs, file uploads and file saving to allow users to access all the functions of the API as seen in Figure 4.81.

  Modulation and Transmission

  - **File Upload:** This allows the user to upload a text file with the message to be sent to the embedded device.
  - **Baud Rate:** The Baud Rate input (initialised at 25 bps) allows the user to adjust the baud rate as necessary.
  - **Transmit:** After uploading the message file and clicking the transmit button, the process of conversion, modulation, audio playback, and .WAV file save is initiated, some of the functions are included below:
    - **Text to Binary Conversion:** The uploaded text is converted into binary data before modulation.

      ```
      function textToBinary(text) {
      return text.split('').map(char => {
      return char.charCodeAt(0).toString(2).padStart(8, '0');
      ```

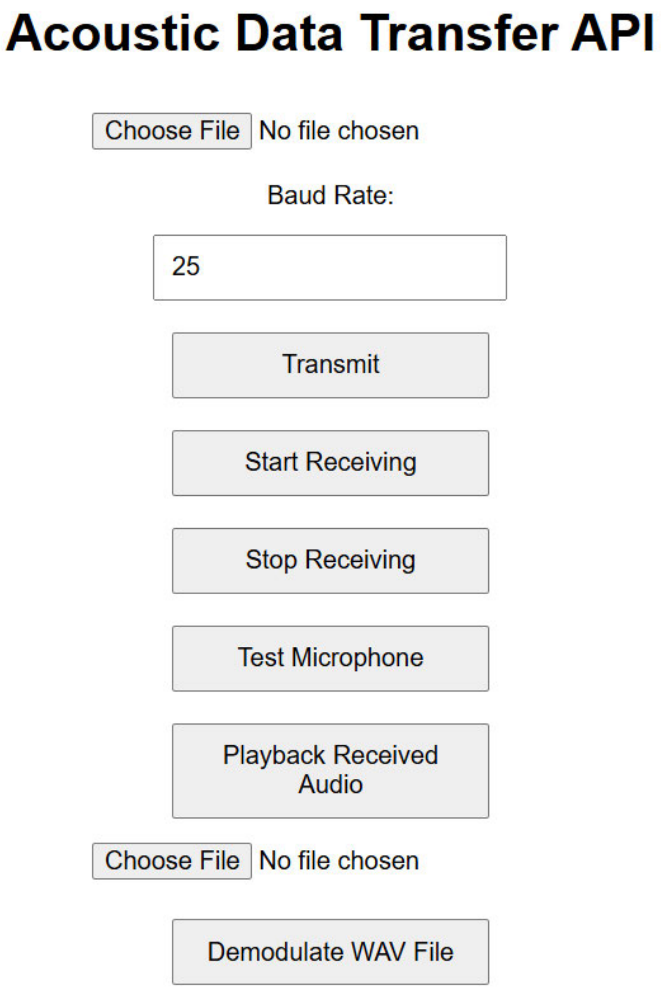


Figure 4.81: Screenshot of the Acoustic Data Transfer API

```
}).join('');
}
```

* **Modulation of Binary Data:** The binary data is modulated into an audio signal using FSK modulation.

```
function modulate(binaryData, baudRate) {
let audioBuffer = [];
let duration = SAMPLE_RATE / baudRate;
for (let bit of binaryData) {
let frequency = (bit === '0') ? FREQ_LOW : FREQ_HIGH;
for (let i = 0; i < duration; i++) {
let sample = Math.sin(2 * Math.PI * frequency * i / SAMPLE_RATE);
audioBuffer.push(sample);
}
}
return audioBuffer;
}
```

∗ **Audio Output:** The modulated signal is played back through speakers.

```
function playAudio(audioBuffer) {
let buffer = audioContext.createBuffer(1, audioBuffer.length, SAMPLE_RATE);
buffer.copyToChannel(new Float32Array(audioBuffer), 0);
let source = audioContext.createBufferSource();
source.buffer = buffer;
source.connect(audioContext.destination);
source.start();
}
```

∗ **Save file as .WAV:** The audio file is saved, this is essentially for testing purposes.

Recording and Demodulation

Basic recording functionality was implemented using the Web Audio API, though demodulation remains incomplete.

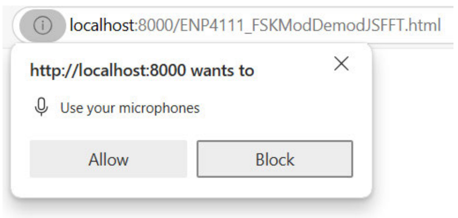
- **Start Receiving:** Requests permission to record audio through the web browser and starts recording to a buffer and begins demodulation.

```
function startReceiving() {
// Audio stream setup code...
processor.port.onmessage = (event) => {
if (!stopReceivingFlag) {
processAudioBuffer(event.data); // Process received audio data
audioChunks.push(...event.data); // Store received audio chunks
}
};
}
```

- **Stop Receiving:** Stops recording, saves audio as a .WAV file (mainly for testing purposes) and finishes the demodulation process. As discussed, the demodulation remains unsuccessful and will need to be updated to using Goertzel's Algorithm in line with the MATLAB simulation.

```
function stopReceiving() {
console.log("Stop Receiving button clicked");
if (audioContext && haveAudioAccess) {
stopReceivingFlag = true; // Set flag to stop receiving
source.disconnect(processor); // Disconnect the audio source
processor.disconnect(audioContext.destination); // Disconnect the processor
audioContext.close(); // Close the audio context
haveAudioAccess = false; // Reset audio access flag
document.getElementById('statusMessage').innerText = 'Receiving stopped.';
}
}
```

- **Additional Testing Functionality:** The second file upload and demodulate functions have been used only for testing purposes, allowing a .WAV file to be uploaded and demodulation implemented manually without having to record a new transmission.

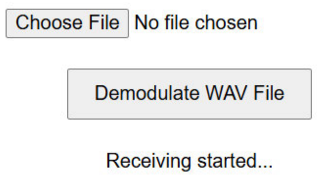


localhost:8000/ENP4111_FSKModDemodJSFFT.html

http://localhost:8000 wants to

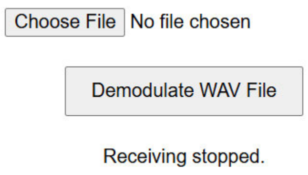
Use your microphones

(a) Audio Permission Request



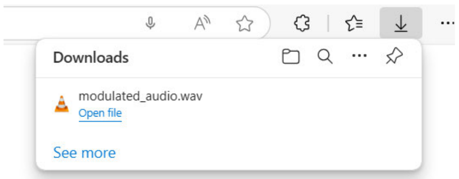
Receiving started...

(b) Audio Receiving



Receiving stopped.

(c) Audio Stopped



modulated_audio.wav

(d) Saved Audio

Figure 4.82: Screenshots of API functionality: Audio Permission Request, Audio Receiving, Audio Stopped, and Saved Audio

4.2.3 Future Work

While significant progress was made in developing the core functionality of the API, several areas remain incomplete:

- Transitioning from FFT to Goertzel's algorithm for more efficient modulation/demodulation.
- Hosting the API on a remote server for broader accessibility.
- Completing the demodulation process to fully decode received messages.

4.3 Embedded Device Development

Due to the time taken developing the API, the coding for the Arduino didn't get developed past the initial hardware set up and the code will not be included in this paper. See Figure 4.83 for a photo of the device to be used in future work.

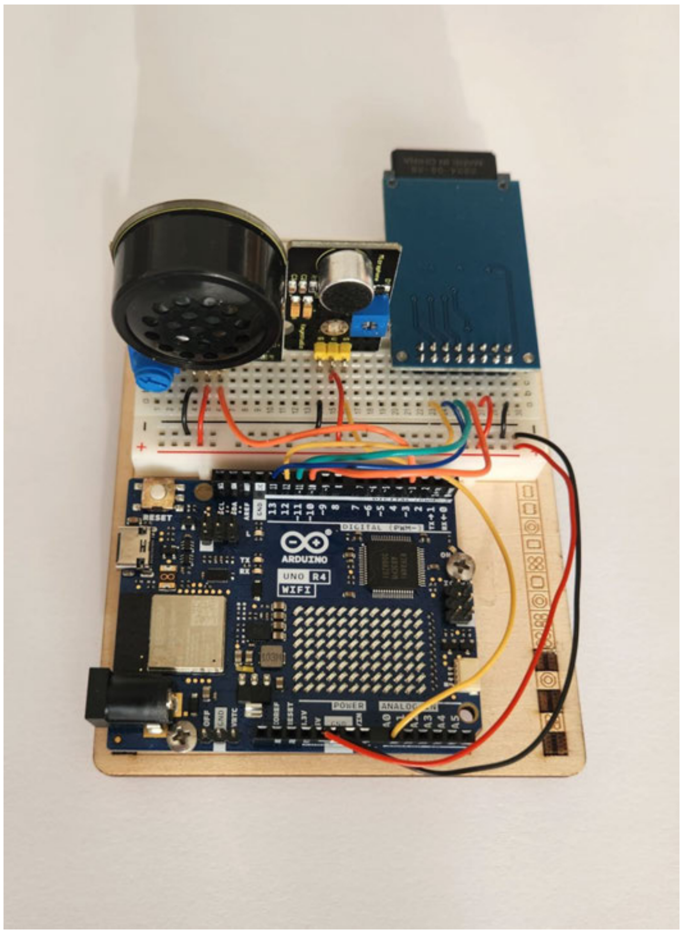


Figure 4.83: Hardware Set Up for Embedded Device Development

Chapter 5

Conclusions and Further Work

5.1 Conclusions

5.2 Conclusions

The experiments conducted demonstrate the viability of an acoustic-coupled data transmission system for secure and robust communication for remote test and measurement access. The results show that the system successfully modulates and demodulates signals over significant distances and under various environmental conditions, including the presence of noise.

5.2.1 Key Findings

- **Frequency Selection**: The optimal operating frequencies for this system were identified as $f_1 = 1$ kHz and $f_2 = 2$ kHz. While higher frequencies (above 3 kHz) were tested initially, they were found to be unpleasant for human operators due to their high pitch, making them impractical for field use. The selected frequencies balanced performance and user comfort, providing efficient transmission without causing discomfort.
- **Distance Performance**: The system exhibited strong performance at a range of distances, proving that the acoustic signal can be transmitted over considerable distances while maintaining data integrity. This demonstrates the potential for use

in practical applications where devices might not be in immediate proximity.

- **Data Transfer Speed**: While the current system functions as a proof of concept, there is room for improvement in terms of data transfer speeds. Enhancing the system's filtering and frequency optimisation could potentially increase the transmission rate, allowing for faster and more efficient communication.
- **Noise Resilience**: One of the most promising findings was the system's excellent resilience to noise. Even in the presence of significant background noise, the system maintained a high level of accuracy in demodulation, with Bit Error Rates (BER) remaining consistently low across various noise levels. This robustness makes the system highly applicable for environments with high levels of ambient noise, such as industrial settings.

5.2.2 Overall Viability

The testing demonstrates that acoustic-coupled data transfer is a feasible method for communication between devices. The system's ability to maintain signal integrity across different distances and noise conditions highlights its potential for real-world applications, particularly in environments where other forms of wireless communication may be impractical or unreliable.

5.2.3 Achievements of Project Objectives

The following objectives have been addressed:

Primary Objectives

Modulation Development: The MATLAB model was developed to test various modulation techniques, focusing on Frequency Shift Keying (FSK) for robustness in aerial acoustic transmission. Through simulations, the performance of FSK was evaluated under different conditions, helping to identify strengths and limitations.

Signal Generation: The MATLAB simulations included an analysis of signal generation tailored to work effectively between a portable device and a low-power embedded device.

Demodulation Development: Demodulation techniques were examined in MATLAB, including the use of a sliding window approach for enhanced detection accuracy. This contributed to a better understanding of how to minimise errors in the demodulation process, although further refinement would be needed in a practical implementation.

Hardware Testing: Various hardware components were selected and tested, particularly microphones. Sensitivity, frequency response, and cost-effectiveness were evaluated, influencing decisions on hardware choices. While testing provided valuable insights, some advanced testing of hardware in real environments remains as future work.

Speed Testing: Initial speed tests were conducted using MATLAB to assess the maximum data rate at which reliable demodulation could be achieved. These tests provided a foundation for understanding the relationship between transmission speed, error rates, and latency, though further validation on the physical prototype is required.

Preamble Testing: A preamble was designed and tested in the MATLAB model to assess its effectiveness in signalling the start and end of messages. This work sets the foundation for implementing a secure handshake mechanism and establishes a baseline for future security features, such as authentication protocols.

Secondary Objectives

Prototype Development: A functioning prototype of the acoustic access system was partially completed. The hardware, including an Arduino-based embedded device, was assembled with microphone and speaker components integrated. However, due to time constraints, software development on the Arduino to handle modulation and demodulation was not completed.

Interface Design: An HTML page was developed to act as an interface for the API, with functional buttons for audio recording, saving files, and initiating data transfer. Although the API demonstrated functionality locally, it did not operate fully as a remote API. Future work will focus on expanding its capabilities for cross-platform accessibility.

Field Testing and Validation: Field testing in real-world environments could not be completed due to the incomplete state of the physical prototype and API. However, simulation-based validation was performed in MATLAB, which provided preliminary insights into the system's performance and limitations.

Cost-Benefit Analysis: While a detailed cost analysis was not conducted, initial assessments of hardware components helped gauge the economic feasibility of the project. Further cost evaluation, including retrofitting existing equipment, remains an area for future work.

Summary of Objectives Achieved

In summary, the project successfully met several primary objectives through MATLAB simulations and prototype development, providing valuable insights into modulation, demodulation, and signal generation techniques. Secondary objectives, such as interface design and preliminary hardware setup, were also addressed, though the system's full functionality was not realised due to time constraints. These achievements lay the groundwork for further development and real-world testing of the acoustic data transfer system.

5.3 Further Work

5.3.1 MATLAB Model Refinement

While the current MATLAB model has proven functional, several areas can be further refined to improve overall system performance and robustness:

- **Sliding Window Optimisation**: The sliding window used in signal detection and demodulation could be optimised for faster processing times. By refining this mechanism, the system may become more responsive and adaptable to real-time data transmission needs. This could use an iterative approach with a binary search, where the power received from each frequency is compared in halves, starting with a large window and honing in smaller and smaller.
- **Additional Filtering**: Introducing more advanced filtering techniques could sharpen the bit recognition response, improving the accuracy and reliability of demodula-

tion. Implementing high-pass or band-pass filters around the key frequencies may help to further reduce interference and noise.

- **Exploring Different Frequency Ranges**: There is potential to experiment with additional frequencies to find the optimal trade-off between data rate and transmission reliability. For instance, utilising frequencies closer together could increase data transfer speeds, while still being feasible for average-quality speakers such as those in mobile phones. Further testing would determine if this would introduce challenges to demodulation accuracy.

5.3.2 JavaScript API Development

The JavaScript API, which serves as the real-time interface for the portable device, requires further refinement:

- **Implementing Goertzel's Algorithm**: The API needs to be updated to include the Goertzel's Algorithm, this could be done by installing the Goertzel.js library for efficient development.
- **Extending Functionality**: While the current version provides a foundation, more work is needed to fully implement the functionalities developed in the MATLAB model. The API needs to handle real-time acoustic signal processing more efficiently, including signal modulation and demodulation with error correction.
- **Improving Interface Design**: The API should be optimised for easier integration with hardware systems like Arduino, allowing seamless communication and control between devices.

5.3.3 Arduino Code Enhancements

The Arduino code, which enables the physical communication between devices, requires further improvement:

- **Synchronising with MATLAB Model**: The current Arduino implementation needs to be further developed to reflect the advancements made in the MATLAB model, ensuring accurate modulation and demodulation.

- **Error Handling and Optimisation**: More work is needed to ensure that the system can handle errors robustly, especially in real-world noisy environments. Optimising the Arduino code for performance and power consumption could further enhance the system's efficiency and reliability.

5.3.4 Exploration of Chirp Signals

Another potential area of research is the use of chirp signals to help identify problem frequencies and ensure more accurate frequency detection during modulation and demodulation:

- **Identifying Resonance Issues**: Chirp signals could be used to detect any resonance or frequency interference that may impact the acoustic signal. This would allow for adjustments to the operating frequencies and the introduction of countermeasures in both hardware and software.
- **Potential for Adaptive Modulation**: By exploring chirp signals, the system could potentially adapt to the best frequency ranges for transmission, depending on the environmental conditions.

References

Arduino (2024*a*), 'Arduino uno r4 wi-fi datasheet', https://docs.arduino.cc/resources/datasheets/ABX00087-datasheet.pdf.

Arduino (2024*b*), 'Arduino uno r4 wifi', https://store-usa.arduino.cc/products/uno-r4-wifi?selectedStore=us.

Arduino (2024*c*), 'Arduino uno rev3 smd', https://store-usa.arduino.cc/collections/audio-sound/products/arduino-uno-rev3.

Baggeroer, A. (1984), 'Acoustic telemetry-an overview', *IEEE Journal of oceanic engineering* **9**(4), 229–235.

Cho, K., Choi, J. & Kim, N. S. (2015), 'An acoustic data transmission system based on audio data hiding: method and performance evaluation', *EURASIP Journal on Audio, Speech, and Music Processing* **2015**, 1–14.

Cooley, J. W. & Tukey, J. W. (1965), 'An algorithm for the machine calculation of complex fourier series', *Mathematics of computation* **19**(90), 297–301.

Foundation, R. P. (2024), 'Raspberry pi zero', https://www.raspberrypi.com/products/raspberry-pi-zero/.

Glenn, A. (1966), 'Analysis of noncoherent fsk system with large ratios of frequency uncertainties to information rates, part i—binary systems', *RCA Review* **XXVII**(2), 272–303.

Gou, F. & Wu, J. (2023), 'Novel data transmission technology based on complex iot system in opportunistic social networks', *Peer-to-Peer Networking and Applications* **16**(2), 571–588.

Indriyanto, S. & Edward, I. Y. M. (2018*a*), Ultrasonic underwater acoustic modem using frequency shift keying (fsk) modulation, *in* '2018 4th International Conference on Wireless and Telematics (ICWT)', pp. 1–4.

Indriyanto, S. & Edward, I. Y. M. (2018*b*), Ultrasonic underwater acoustic modem using frequency shift keying (fsk) modulation, *in* '2018 4th International Conference on Wireless and Telematics (ICWT)', IEEE, pp. 1–4.

Kishore, G. & Rallapalli, H. (2019), '" performance assessment of m-ary ask, fsk, psk, qam and fqam in awgn channel", 2019 international conference on communication and signal processing (iccsp), chennai, india'.

Lee, H., Kim, T. H., Choi, J. W. & Choi, S. (2015), Chirp signal-based aerial acoustic communication for smart devices, *in* '2015 IEEE Conference on Computer Communications (INFOCOM)', IEEE, pp. 2407–2415.

Leis, J. W. (2011), *Digital signal processing using MATLAB for students and researchers*, John Wiley & Sons.

Leis, J. W. (2018), *Communication systems principles using MATLAB*, John Wiley & Sons.

Lonzetta, A. M., Cope, P., Campbell, J., Mohd, B. J. & Hayajneh, T. (2018), 'Security vulnerabilities in bluetooth technology as used in iot', *Journal of Sensor and Actuator Networks* **7**(3), 28.

Lopes, C. V. & Aguiar, P. M. (2001), Aerial acoustic communications, *in* 'Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No. 01TH8575)', IEEE, pp. 219–222.

Lopes, C. V. & Aguiar, P. M. (2003), 'Acoustic modems for ubiquitous computing', *IEEE pervasive computing* **2**(3), 62–71.

Lyons, R. G. (2010), *Understanding digital signal processing, 3rd ed.*, Prentice Hall.

Middlestead, R. W. (2017), *Digital communications with emphasis on data modems: Theory, analysis, design, simulation, testing, and applications*, John Wiley & Sons.

Ochi, H., Shimura, T., Sawa, T., Amitani, Y. & Nakamura, T. (2000), 'Experiments for acoustic digital data communication using frequency shift keying modulation', *Japanese Journal of Applied Physics* **39**(5S), 3184.

Proakis, J. G. & Salehi, M. (2008), *Digital communications*, McGraw-hill.

Ramezanpour, K., Jagannath, J. & Jagannath, A. (2023), 'Security and privacy vulnerabilities of 5g/6g and wifi 6: Survey and research directions from a coexistence perspective', *Computer Networks* **221**, 109515.

Samsung (2021), 'Samsung galaxy s21 5g specifications', https://www.samsung.com/africa_en/smartphones/galaxy-s21-5g/specs/.

Sendra, S., Lloret, J., Jimenez, J. M. & Parra, L. (2016), 'Underwater acoustic modems', *IEEE Sensors Journal* **16**(11), 4063–4071.

Sklar, B. (2021), *Digital communications: fundamentals and applications*, Pearson.

Xiong, F. (2006), *Digital modulation techniques*, Artech.

Zhang, B., Zhan, Q., Chen, S., Li, M., Ren, K., Wang, C. & Ma, D. (2014), 'Priwhisper: Enabling keyless secure acoustic communication for smartphones', *IEEE Internet of Things Journal* **1**(1), 33–45.

Zhou, M., Wang, Q., Ren, K., Koutsonikolas, D., Su, L. & Chen, Y. (2019), 'Dolphin: Real-time hidden acoustic signal capture with smartphones', *IEEE Transactions on Mobile Computing* **18**(3), 560–573.

Appendix A

Project Specification

ENP 4111 Research Project

Project Specification and Work Plan

For:	**Christopher Van den Ham**
Student ID:	[redacted]
Topic:	Remote Test and Measurement Access via Acoustic Coupling in Non-WiFi Accessible Environments
Supervisor:	John Leis
Sponsorship:	Faculty of Health, Engineering & Sciences

A.1 Project Specification

A.1.1 Introduction and Background

Technology has evolved at an incredible pace in recent history. What was thought impossible only a decade ago, is now proving not only possible, but better, and faster than was imagined. The phenomenal amounts of data that is sent between palm sized devices eclipses what super computers could achieve only a short while ago (Gou & Wu 2023). Much of this data transmission occurs through cabled connections, though end users are increasingly relying on the use of radio frequencies to transmit the data from the cable (or modem) to the end user device.

Whilst this advantageous in many ways, there are some instances where sending data over radio signals has the potential to pose security risks which are ever evolving (Ramezanpour et al. 2023). These areas include hospitals, department of defence buildings, petrochemical plants, and other instances where sensitive data or important infrastructure must be kept secure.

Generally, any equipment housed in these radio frequency (RF) sensitive areas that need to transfer data, will have reliable ways in which to do so. On rare occasions some equipment may need to transfer data that does not typically do so. This could be in a time of system fault, or a piece of monitoring equipment that rarely needs to dispatch its contents. When these anomalies occur, it typically means a skilled technician needs to be sent to site to perform the data interrogation.

These RF restricted areas are often located in central locations, so when data needs to be sent to, or retrieved from equipment, technicians are easily able to physically retrieve the data. This would typically mean plugging a secure laptop into the device by way of some description of cable, be it proprietary or not. In some instances, the technician may need to physically inspect the system to assess for faulty equipment.

When these sites are located in remote areas, having a technician come to site can take an excessive amount of time. Technicians sometimes have to travel thousands of kilometres to access equipment, often because it is in restricted areas or cabinets. Some of these faults can be as simple as a faulty sensor, resulting in a time consuming, very costly process of fault finding.

This project will explore acoustic coupling to transfer data to and from embedded devices in remote areas. This will offer a simple solution for system interrogation, small firmware updates and general troubleshooting that can be performed by personnel following a standard of procedure.

Technology like this is commonplace in underwater acoustic modems (Sendra et al. 2016), where there is a significant amount of attenuation at certain frequencies, resulting in poor data transmission. The project will look at several types of modulation to test which is the most effective for data transmission.

Modulation types may include Frequency Shift Keying (FSK), Amplitude Shift Keying (ASK), Phase Shift Keying (PSK), Direct Sequence Spread Spectrum (DSSS), Orthogonal Frequency Division Multiplexing (OFDM) and others. These types of modulation are all quite common and range in levels of complexity and bit rate (Indriyanto & Edward 2018*a*). The modulation types used will depend on time and resources available, the idea of the project is for the transmission to be simple enough that it is accessible to a wide range of equipment, which may restrict complexity.

A.1.2 Objectives and Aims

The main aim of the project is to develop and test a prototype, capable of transferring data bidirectionally between a portable device, and the embedded end device.

Specific Objectives

Prototype Developement – Develop a functioning prototype of the acoustic access system. This will involve hardware design and construction, taking into account the need for integrating the speaker and microphone components on the target device. The prototype will also need to be have the ability to run software for the modulation and demodulation of the data to be transmitted.

Interface Design – Develop an Application Programming Interface (API) with an intuitive and user friendly user interface (UI) for field personnel to interact with.

Acoustic Communication Optimistation – Investigate the effectiveness of different modulation techniques, as mentioned in A.1.1, to test for performance indicators such as baud rate and bit rate.

Security – Investigate and implement secure mechanisms to ensure only authorised data can be sent and retrieved.

Field Testing and Validation – Conduct testing to evaluate the system's performance and usability.

Cost Benefit Analysis – Evaluate the cost of the system including new equipment but also retrofitting equipment that is already in use.

Expected Outcomes

Validation – Identify the validity of sending and retrieving data remotely using acoustic coupling.

Acoustic Coupling Optimisation – Identify what methods of modulation and demodulation are the most effective for acoustic coupled data transfer speeds and accuracy. This will allow for recommendations to be made for any future work undertaken in this field.

Technical Performance Data – Determine technical abilities of the system, including maximum operating range, achievable data rates, latency constraints and usability.

A.2 Work Plan

A.2.1 Timeline

January	Develop a project plan and submit the proposal.
February	Specification and work plan, begin literature review, begin methodology.
March	Finalize literature review, continue methodology, begin hardware design, begin software design, begin framework for draft dissertation.
April	Continue methodology, continue hardware design, continue software design, continue draft dissertation.
May	Finalize methodology, continue hardware design, begin construction, continue software design, continue draft dissertation.
June	Finalize hardware design and construction, continue software design, begin software implementation, continue draft dissertation, begin framework for final dissertation.
July	Finalize software design and implementation, begin integration testing, continue draft dissertation, begin presentation preparation, continue final dissertation.
August	Finalise integration testing and complete prototype, continue draft dissertation, continue presentation preparation, continue final dissertation.
September	Finalise draft dissertation, continue presentation preparation, continue final dissertation.
October	Finalise presentation, complete reflection, continue final dissertation.
November	Complete final dissertation.

Refer to Figure A.1 for a visual representation by way of a Gantt chart.

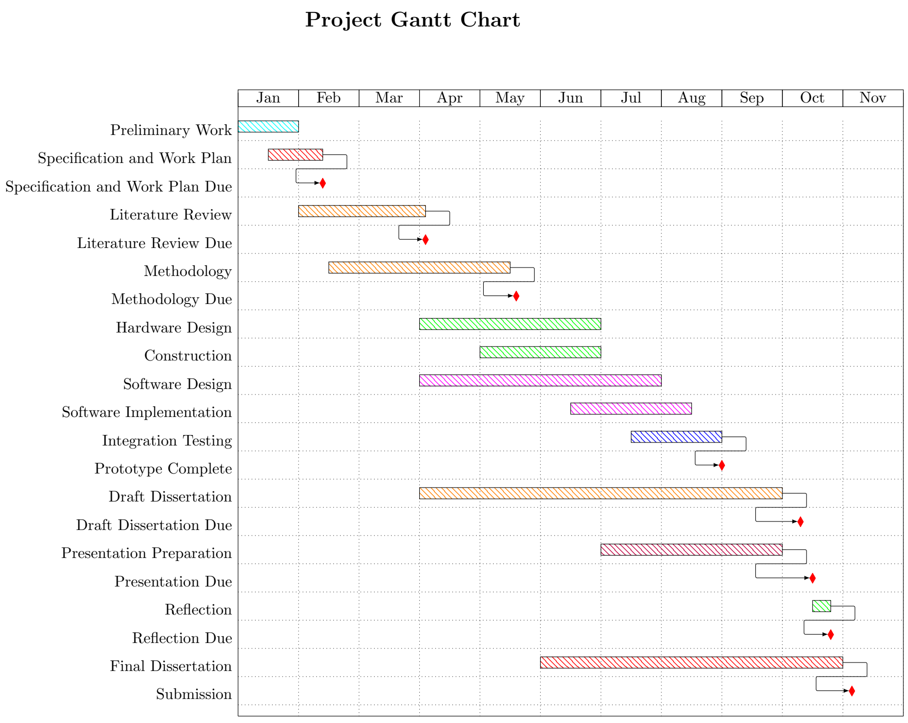


Figure A.1: ENP4111 Research Project Gantt Chart.

A.2.2 Resources Required

- Equipment - Necessary equipment will be finalised in the Hardware Design phase of the project, equipment listed here is subject to change.
  - Embedded Device
    - Arduino Uno R3 (Including bread board and miscellaneous components such as resistors, capacitors etc.).
    - Arduino compatible audio amplifier with speaker module.
    - Arduino compatible microphone sound sensor module.
    - Arduino compatible SD card reader.
  - Portable Device
    - Portable device capable of running JavaScript API, preferably with built in microphone and speaker.
- Software
  - Arduino IDE Software.
  - Source Code Editor like Notepad++ or similar.

  - Browser capable of testing JavaScript API like Chrome or Fire Fox.

- Access
  - Access to areas with different levels of noise isolation to test the device in different scenarios.

References

Arduino (2024*a*), 'Arduino uno r4 wi-fi datasheet', https://docs.arduino.cc/resources/datasheets/ABX00087-datasheet.pdf.

Arduino (2024*b*), 'Arduino uno r4 wifi', https://store-usa.arduino.cc/products/uno-r4-wifi?selectedStore=us.

Arduino (2024*c*), 'Arduino uno rev3 smd', https://store-usa.arduino.cc/collections/audio-sound/products/arduino-uno-rev3.

Baggeroer, A. (1984), 'Acoustic telemetry-an overview', *IEEE Journal of oceanic engineering* **9**(4), 229–235.

Cho, K., Choi, J. & Kim, N. S. (2015), 'An acoustic data transmission system based on audio data hiding: method and performance evaluation', *EURASIP Journal on Audio, Speech, and Music Processing* **2015**, 1–14.

Cooley, J. W. & Tukey, J. W. (1965), 'An algorithm for the machine calculation of complex fourier series', *Mathematics of computation* **19**(90), 297–301.

Foundation, R. P. (2024), 'Raspberry pi zero', https://www.raspberrypi.com/products/raspberry-pi-zero/.

Glenn, A. (1966), 'Analysis of noncoherent fsk system with large ratios of frequency uncertainties to information rates, part i—binary systems', *RCA Review* **XXVII**(2), 272–303.

Gou, F. & Wu, J. (2023), 'Novel data transmission technology based on complex iot system in opportunistic social networks', *Peer-to-Peer Networking and Applications* **16**(2), 571–588.

Indriyanto, S. & Edward, I. Y. M. (2018*a*), Ultrasonic underwater acoustic modem using frequency shift keying (fsk) modulation, *in* '2018 4th International Conference on Wireless and Telematics (ICWT)', pp. 1–4.

Indriyanto, S. & Edward, I. Y. M. (2018*b*), Ultrasonic underwater acoustic modem using frequency shift keying (fsk) modulation, *in* '2018 4th International Conference on Wireless and Telematics (ICWT)', IEEE, pp. 1–4.

Kishore, G. & Rallapalli, H. (2019), '" performance assessment of m-ary ask, fsk, psk, qam and fqam in awgn channel", 2019 international conference on communication and signal processing (iccsp), chennai, india'.

Lee, H., Kim, T. H., Choi, J. W. & Choi, S. (2015), Chirp signal-based aerial acoustic communication for smart devices, *in* '2015 IEEE Conference on Computer Communications (INFOCOM)', IEEE, pp. 2407–2415.

Leis, J. W. (2011), *Digital signal processing using MATLAB for students and researchers*, John Wiley & Sons.

Leis, J. W. (2018), *Communication systems principles using MATLAB*, John Wiley & Sons.

Lonzetta, A. M., Cope, P., Campbell, J., Mohd, B. J. & Hayajneh, T. (2018), 'Security vulnerabilities in bluetooth technology as used in iot', *Journal of Sensor and Actuator Networks* **7**(3), 28.

Lopes, C. V. & Aguiar, P. M. (2001), Aerial acoustic communications, *in* 'Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No. 01TH8575)', IEEE, pp. 219–222.

Lopes, C. V. & Aguiar, P. M. (2003), 'Acoustic modems for ubiquitous computing', *IEEE pervasive computing* **2**(3), 62–71.

Lyons, R. G. (2010), *Understanding digital signal processing, 3rd ed.*, Prentice Hall.

Middlestead, R. W. (2017), *Digital communications with emphasis on data modems: Theory, analysis, design, simulation, testing, and applications*, John Wiley & Sons.

Ochi, H., Shimura, T., Sawa, T., Amitani, Y. & Nakamura, T. (2000), 'Experiments for acoustic digital data communication using frequency shift keying modulation', *Japanese Journal of Applied Physics* **39**(5S), 3184.

Proakis, J. G. & Salehi, M. (2008), *Digital communications*, McGraw-hill.

Ramezanpour, K., Jagannath, J. & Jagannath, A. (2023), 'Security and privacy vulnerabilities of 5g/6g and wifi 6: Survey and research directions from a coexistence perspective', *Computer Networks* **221**, 109515.

Samsung (2021), 'Samsung galaxy s21 5g specifications', https://www.samsung.com/africa_en/smartphones/galaxy-s21-5g/specs/.

Sendra, S., Lloret, J., Jimenez, J. M. & Parra, L. (2016), 'Underwater acoustic modems', *IEEE Sensors Journal* **16**(11), 4063–4071.

Sklar, B. (2021), *Digital communications: fundamentals and applications*, Pearson.

Xiong, F. (2006), *Digital modulation techniques*, Artech.

Zhang, B., Zhan, Q., Chen, S., Li, M., Ren, K., Wang, C. & Ma, D. (2014), 'Priwhisper: Enabling keyless secure acoustic communication for smartphones', *IEEE Internet of Things Journal* **1**(1), 33–45.

Zhou, M., Wang, Q., Ren, K., Koutsonikolas, D., Su, L. & Chen, Y. (2019), 'Dolphin: Real-time hidden acoustic signal capture with smartphones', *IEEE Transactions on Mobile Computing* **18**(3), 560–573.

Appendix B

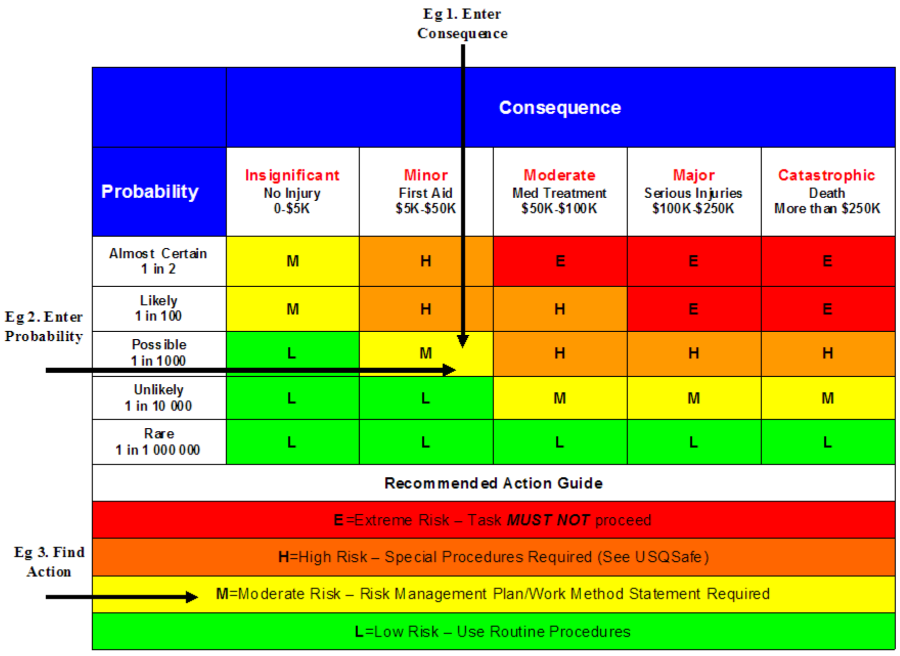
Risk Assessment

University of Southern Queensland Offline Version

USQ Safety Risk Management System

Note: This is the offline version of the Safety Risk Management System (SRMS) Risk Management Plan (RMP) and is only to be used for planning and drafting sessions, and when working in remote areas or on field activities. It must be transferred to the online SRMS at the first opportunity.

Safety Risk Management Plan – Offline Version			
Assessment Title:	Professional Engineer Research Project	Assessment Date:	1/05/2024
Workplace (Division/Faculty/Section):	Faculty of Engineering and Science	Review Date:(5 Years Max)	31/12/2024
Context			
Description:			
What is the task/event/purchase/project/procedure?	Remote Test and Measurement Access via Acoustic Coupling in Non-WiFi Accessible Environments		
Why is it being conducted?	Final year project		
Where is it being conducted?	Student's Home		
Course code (if applicable)	ENP4111	Chemical name (if applicable)	N/A
What other nominal conditions?			
Personnel involved	Christopher Van den Ham, John Leis		
Equipment	Laptop Computer, Arduino R4, Mobile Phone		
Environment	Office		
Other			
Briefly explain the procedure/process	FSK modulation, data transfer and demodulation will be performed.		
Assessment Team - who is conducting the assessment?			
Assessor(s)	John Leis, Belal Yousif		
Others consulted:			


Eg 1. Enter Consequence
Eg 2. Enter Probability
Eg 3. Find Action

	Consequence				
Probability	**Insignificant** No Injury 0-$5K	**Minor** First Aid $5K-$50K	**Moderate** Med Treatment $50K-$100K	**Major** Serious Injuries $100K-$250K	**Catastrophic** Death More than $250K
Almost Certain 1 in 2	M	H	E	E	E
Likely 1 in 100	M	H	H	E	E
Possible 1 in 1000	L	M	H	H	H
Unlikely 1 in 10 000	L	L	M	M	M
Rare 1 in 1 000 000	L	L	L	L	L

Recommended Action Guide
E=Extreme Risk – Task ***MUST NOT*** proceed
H=High Risk – Special Procedures Required (See USQSafe)
M=Moderate Risk – Risk Management Plan/Work Method Statement Required
L=Low Risk – Use Routine Procedures

Step 1 (cont)	Step 2	Step 2a	Step 2b	Step 3			Step 4				
Hazards: From step 1 or more if identified	***The Risk:*** What can happen if exposed to the hazard without existing controls in place?	***Consequence:*** What is the harm that can be caused by the hazard without existing controls in place?	***Existing Controls:*** What are the existing controls that are already in place?	***Risk Assessment:*** Consequence x Probability = Risk Level			***Additional controls:*** Enter additional controls if required to reduce the risk level	***Risk assessment with additional controls:***			
				Probability	**Risk Level**	**ALARP? Yes/no**		**Consequence**	**Probability**	**Risk Level**	**ALARP? Yes/no**
Example											
Working in temperatures over 35⁰ C	Heat stress/heat stroke/exhaustion leading to serious personal injury/death	catastrophic	Regular breaks, chilled water available, loose clothing, fatigue management policy.	possible	high	No	temporary shade shelters, essential tasks only, close supervision, buddy system	catastrophic	unlikely	mod	Yes
Extended hours at computer	RSI, Deep Vein Thrombosis	Moderate	Regular Breaks, Ergonomic position and equipment	Rare	Low	Yes		Select a consequence	Select a probability	Select a Risk Level	Yes or No
Excessive audible volume from acoustic signal	Hearing Damage	Moderate	Keep volume only as loud as necessary on both devices	Rare	Low	Yes		Select a consequence	Select a probability	Select a Risk Level	Yes or No
Hard Drive Malfunction Resulting in lost data	Lost data, Lost work, Lost time	Insignificant	Physical backup hard drive, cloud backup storage	Possible	Low	Yes		Select a consequence	Select a probability	Select a Risk Level	Yes or No
Project Related Stress	Mental health detriment	Moderate	Project planning, ask for assistance if necessar	Unlikely	Moderate	Yes		Select a consequence	Select a probability	Select a Risk Level	Yes or No
Poor Housekeeping	Tripping Risk, Spill drink on computer	Minor	Ensure Housekeeping is kept to a high standard	Unlikely	Low	Yes		Select a consequence	Select a probability	Select a Risk Level	Yes or No

Step 5 - Action Plan (for controls not already in place)			
Additional controls:	*Resources:*	*Persons responsible:*	*Proposed implementation date:*
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.

Step 6 - Approval			
Drafter's name:	Christopher Van den Ham	Draft date:	1/05/2024
Drafter's comments:	Common Hazards associated with Computer work in an office environment		
Approver's name:		Approver's title/position:	
Approver's comments:			
I am satisfied that the risks are as low as reasonably practicable and that the resources required will be provided.			
Approver's signature:		Approval date:	Click here to enter a date.

Appendix C

Ethical Clearance

There are no Ethical Clearances applicable to this project.

Appendix D

MATLAB Scripts

D.1 Introduction to this Appendix

These MATLAB scripts are referred to throughout this paper and facilitate testing of the system. There are three MATLAB scripts:

1. `ENP4111_FSK_Mod.m`: This script is used for modulating a message into an audio signal and saving as a .WAV file.
2. `ENP4111_Manual_Demodulation.m`: This script is used for recording and demodulation, testing parameters for system optimisation and was used for most of the testing, the starting index of the message is input manually in this script.
3. `ENP4111_Working_Preamble_demod.m`: This script is used for recording and automatic demodulation, utilising the preamble detection to find the starting index of the message.

These should all be fairly self explanatory, especially with explanations given in this paper. The main parameters that will need to be matched and adjusted are frequencies, sample rate and baud rate. Some parts have been commented out as part of testing and will need to be reinstated for full functionality.

breaklines=true

Listing D.1: FSK Modulation Script.

```
% ------------------FSK Modulation with Enhanced Preamble and Buffer
    -----ENP4111--------------
% File       Name : ENP4111_FSK_Mod.m
% Authors    Name : Chris Van den Ham U1114389
% Environment     : Matlab R2023b
%
% ---------------------------------------------------------------------
% Message to be transmitted
text = 'Hello, World in 1976, by B. Kernighan, before that,
    Alexander G Bell in 1876: Mr. Watson, come here, I want to
    see you.!';
% text = 'Hello World!';
micTest = [ 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1]; % Mic Test
    modulation

% Convert text to binary
binaryData = reshape(dec2bin(text, 8).' - '0', 1, []);

% Define the enhanced preamble (e.g., '10101010 10101010
    10101010 11111111')
preamble = [1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0
    1 0 1 0 1 0 1 1 1 1 1 1 1 1]; % 24 bits + unique 8 bits

% Concatenate preamble and message
binaryDataWithPreamble = [preamble, binaryData];
```

```matlab
% FSK parameters
f1 = 1000; % Frequency for binary '0'
f2 = 2000; % Frequency for binary '1'
fs = 8000; % Sampling rate
baudRate = 100; % Baud rate in bits per second
T = 1 / baudRate; % Bit duration (in seconds)

% Preallocate the FSK signal
symbolLength = fs * T; % Number of samples per bit
t = 0:1/fs:T-1/fs; % Time vector for one bit duration
fskSignal = zeros(1, length(t) * length(binaryDataWithPreamble))
    ;

% 500ms buffer at the start (0.5 seconds of silence)
bufferLength = fs * 0.5; % 0.5 seconds of silence
buffer = zeros(1, bufferLength); % Create the silence buffer

% Generate FSK signal with preamble and message
for i = 1:length(binaryDataWithPreamble)
    if binaryDataWithPreamble(i) == 0
        fskSignal((i-1)*length(t)+1:i*length(t)) = cos(2*pi*f1*t
            );
    else
        fskSignal((i-1)*length(t)+1:i*length(t)) = cos(2*pi*f2*t
            );
    end
end

% Concatenate the buffer and the FSK signal
fskSignalWithBuffer = [buffer, fskSignal];

% Normalize the noise-free signal to prevent clipping
fskSignalWithBuffer = fskSignalWithBuffer / max(abs(
    fskSignalWithBuffer));

% Desired SNR in dB for the noisy signal
snrDb = 20; % Adjust this value based on your noise level
    requirement

% Add AWGN to the FSK signal to create the noisy version
% noisyFskSignalWithBuffer = ENP4111addAwgnManual(
    fskSignalWithBuffer, snrDb);

% Normalize the noisy signal to prevent clipping
% noisyFskSignalWithBuffer = noisyFskSignalWithBuffer / max(abs(
    noisyFskSignalWithBuffer));

% Save the noise-free signal as a WAV file
filenameNoiseFree = 'ENP4111modulatedSignal_noiseFree_withBuffer
    .wav';
audiowrite(filenameNoiseFree, fskSignalWithBuffer, fs);

% Save the noisy signal as a WAV file
% filenameNoisy = 'ENP4111modulatedSignal_withNoise_withBuffer.
    wav';
% audiowrite(filenameNoisy, noisyFskSignalWithBuffer, fs);

% Plot the noise-free FSK signal
figure;
subplot(2,1,1);
plot(fskSignalWithBuffer);
xlabel('Sample');
ylabel('Amplitude');
title('FSK Modulated Signal (Noise-Free with Buffer)');
```

```
% % Plot the noisy FSK signal
% subplot(2,1,2);
% plot(noisyFskSignalWithBuffer);
% xlabel('Sample');
% ylabel('Amplitude');
% title('FSK Modulated Signal (With Noise and Buffer)');
```

Listing D.2: Recording and Manual Demodulation Script.

```
%—————————————Manual Start Demodulation——ENP4111——————————
% File       Name : ENP4111_Manual_Demodulation.m
% Authors    Name : Chris Van den Ham U1114389
% Environment     : Matlab R2023b
%
%———————————————————————————————————————————————
% Parameters
f1 = 1000;          % Frequency for binary '0'
f2 = 2000;          % Frequency for binary '1'
fs = 8000;          % Sampling frequency
baudRate = 25;      % Baud rate
T = 1/baudRate;     % Bit duration

% Number of samples per bit
symbolLength = fs * T;

% Define number of samples to disregard at start and end of each
     bit
skipSamples = floor(symbolLength / 15);  % Number of samples to
    skip at the start and end of each bit
N_new = symbolLength - 2 * skipSamples;  % New number of samples
     after removing skipSamples

% Compute Goertzel bin indices for f1 and f2 with the new N
k1 = round((f1 * N_new) / fs);  % Index for f1
k2 = round((f2 * N_new) / fs);  % Index for f2

disp(['Adjusted Goertzel index for f1: ', num2str(k1)]);
disp(['Adjusted Goertzel index for f2: ', num2str(k2)]);

% Define the range of bins to search around k1 and k2
binRange = -1:1;  % Range of +-1 bin around the calculated index

% Check energy detection using Goertzel's Algorithm
startIdx = input('Enter start index for message:'); % Input
    starting sample of message
numBits = floor((length(fskSignal) - startIdx + 1) /
    symbolLength);
binaryStream = zeros(1, numBits);

% New Hamming window for the reduced segment size
hammingWindow = hamming(N_new)';

% Initialise Goertzel energy arrays
energyF1Plot = zeros(1, numBits);
energyF2Plot = zeros(1, numBits);

% Loop through bit windows
for i = 1:numBits
    periodStart = startIdx + (i-1)*symbolLength + skipSamples +
        1;
    periodEnd = periodStart + N_new - 1;

    if periodEnd > length(fskSignal)
        break;
    end

    segment = fskSignal(periodStart:periodEnd);
    windowedSegment = segment .* hammingWindow;

    % Apply Goertzel's Algorithm over the range of bins for f1
    energy_f1 = 0;
    for kOffset = binRange
        S_f1 = goertzel(windowedSegment, k1 + kOffset);
        energy_f1 = energy_f1 + abs(S_f1(end))^2;  % Sum energy
           over the range of bins
    end

    % Apply Goertzel over the range of bins for f2
    energy_f2 = 0;
    for kOffset = binRange
        S_f2 = goertzel(windowedSegment, k2 + kOffset);
        energy_f2 = energy_f2 + abs(S_f2(end))^2;  % Sum energy
           over the range of bins
    end

    energyF1Plot(i) = energy_f1;
    energyF2Plot(i) = energy_f2;

    % Bit Decision - This could use filtering for more accurate
       results
    if energy_f1 > energy_f2
        binaryStream(i) = 0;
    else
        binaryStream(i) = 1;
    end
end

% Convert binary stream to ASCII
if mod(length(binaryStream), 8) ~= 0
    binaryStream = [binaryStream zeros(1, 8 - mod(length(
       binaryStream), 8))];
end

binaryMatrix = reshape(binaryStream, [8, length(binaryStream)
   /8])';
asciiValues = bin2dec(num2str(binaryMatrix));

%% End of message detection
endCharAscii = 33;                                    % ASCII
   value for '!'
endIndex = find(asciiValues == endCharAscii, 1);     % Find first
    occurrence of '!'

% If '!' is found, truncate the ASCII values at the end of the
   message
if ~isempty(endIndex)
    asciiValues = asciiValues(1:endIndex);
end

% Convert ASCII values to characters
recoveredText = char(asciiValues)';  % Convert ASCII values to
   characters

% Display the recovered ASCII message
disp(['Recovered Text: ', recoveredText]);

%% Time Domain Plot
figure;
plot(fskSignal);
title('SNR 0 dB Time Domain FSK Signal');
xlabel('Sample');
ylabel('Amplitude');
ylim([-1.1 1.1]);
```

```
grid on;

%% Goertzel Energies
figure;
subplot(2,1,1);
plot(1:numBits, energyF1Plot, 'b', 1:numBits, energyF2Plot, 'r')
    ;
title('SNR 0 dB Goertzel Energy for f1 and f2');
xlabel('Bit Index');
ylabel('Energy');
legend('f1 Energy', 'f2 Energy');
grid on;

%% Create frequency plot
N = length(fskSignal);
fftSignal = fft(fskSignal);
fftShift = fftshift(fftSignal);
freq = (-N/2:N/2-1)*(fs/N);

subplot(2,1,2);
plot(freq, abs(fftShift));
title('SNR 0 dB Frequency Spectrum of FSK Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
xlim([f1 - 500, f2 + 500]);  % Dynamically adjust the x-axis
    limits based on f1 and f2
grid on;

%% Binary Stream Conversion and Display
% Convert binary stream into binary string for display
binaryString = num2str(binaryStream);
binaryString = binaryString(binaryString ~= ' ');  % Remove
    spaces from the string

% Split the binary string into 8-bit chunks (bytes) and display
    each byte
% for debugging
fprintf('Binary Stream (Grouped in Bytes):\n');
for i = 1:8:length(binaryString)
    byte = binaryString(i:i+7);
    fprintf('%s ', byte);
end
fprintf('\n');

%% BER Calculation
originalMessage = 'Hello, World rose in 1976, by B. Kernighan,
    before that, Alexander G Bell in 1876: Mr. Watson, come here,
     I want to see you.!';
originalBinary = logical(dec2bin(originalMessage, 8)' - '0');  %
     Convert to binary

% Truncate binaryStream to the same length as the original
    message
minLength = min(length(originalBinary), length(binaryStream));
truncatedBinaryStream = binaryStream(1:minLength);  % Only take
    the bits up to the message length otherwise there will be
    false BER

% Calculate BER between the original and demodulated bits
ber = sum(originalBinary(1:minLength) ~= truncatedBinaryStream)
    / minLength;

% Display the Bit Error Rate
fprintf('Bit Error Rate (BER): %.4f\n', ber);
```

```
%% SNR Calculation
% Define preamble (as bit sequence for length, could just define
     the length)
preamble = [1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0
    1 0 1 0 1 0 1 1 1 1 1 1 1];

% Calculate how many samples the preamble spans
preambleLengthBits = length(preamble);                          %
    Length of preamble in bits
preambleLengthSamples = preambleLengthBits * symbolLength;  %
    Convert to number of samples

% Define buffer size (0.5 seconds)
bufferSizeSeconds = 0.5;
bufferSizeSamples = round(fs * bufferSizeSeconds);  % Convert
    0.5s to samples

% Calculate the start of the noise buffer (0.5s before the
    preamble)
noiseEndIdx = startIdx - preambleLengthSamples - 1;
noiseStartIdx = max(1, noiseEndIdx - bufferSizeSamples);  % don'
    t go out of bounds

% Extract the noise buffer from the signal
noiseBuffer = fskSignal(noiseStartIdx:noiseEndIdx);

% Calculate the noise power (mean squared amplitude of the noise
    )
noisePower = mean(noiseBuffer.^2);

% Calculate the signal power over the actual message
endIdx = startIdx + numBits * symbolLength - 1;  % Corrected
    endIdx
if endIdx > length(fskSignal)
    warning('endIdx exceeds the signal length. Adjusting endIdx
        to the length of fskSignal.');
    endIdx = length(fskSignal);  % Adjust to fit within bounds
end

signalPower = mean(fskSignal(startIdx:endIdx).^2);

% Calculate the Signal-to-Noise Ratio (SNR) in dB
snrValue = 10 * log10(signalPower / noisePower);

% Display the calculated SNR
fprintf('Signal-to-Noise Ratio (SNR): %.2f dB\n', snrValue);

%% Amplitude Average of Signal
% % This is used for distance testing
% % Compute average signal amplitude over the message portion of
     the FSK signal
% signalSegment = fskSignal(startIdx:endIdx); % Extract the
    message portion of the FSK signal
% avgAmplitude = mean(abs(signalSegment));     % Compute average
    absolute amplitude
%
% % Display the average signal amplitude
% fprintf('Average Signal Amplitude: %.4f\n', avgAmplitude);
```

Listing D.3: Recording and Demodulation Script Utilising Preamble Destection.

```
% ----------------FSK Demodulation Script with Updated Goertzel and
    Preamble Detection----ENP4111-------------
% File      Name : ENP4111_Working_Preamble_demod.m
% Authors   Name : Chris Van den Ham U1114389
% Environment    : Matlab R2023b
```

```
%
% ————————————————————————————————————————
% Parameters
f1 = 1000;          % Frequency for binary '0'
f2 = 2000;          % Frequency for binary '1'
fs = 8000;          % Sampling frequency
baudRate = 100;      % Baud rate
T = 1/baudRate;     % Bit duration

% Initialize the audiorecorder object
recorder = audiorecorder(fs, 16, 1);

% Display instructions to the user
disp('Press any key to start recording...');
pause;  % Wait for the user to press a key

% Start recording
disp('Recording started. Press any key to stop recording.');
record(recorder);

% Wait for the user to press a key to stop recording
pause;
stop(recorder);
disp('Recording stopped.');

% Get the recorded data
fskSignal = getaudiodata(recorder, 'double');

% Play the recorded audio
% sound(fskSignal, fs);

% Plot the recorded signal
figure;
plot(fskSignal);
title('Preamble Detection at 100 bps');
xlabel('Sample');
ylabel('Amplitude');

%%
% Preamble: 10101010 followed by unique 11111111
preamble = [repmat([1 0 1 0], 1, 2), 1 1 1 1 1 1 1 1, repmat([1
    0 1 0], 1, 2), 1 1 1 1 1 1 1 1];
preambleLength = length(preamble) * fs * T;

% Number of samples per bit
symbolLength = fs * T;

% Define the skipSamples and N_new parameters
skipSamples = floor(symbolLength / 5);  % Number of samples to
    skip at the start and end of each bit
N_new = symbolLength - 2 * skipSamples;   % New number of
    samples after removing the skipped samples

% Update the Goertzel coefficients for f1 and f2 based on N_new
k1 = round((f1 * N_new) / fs);  % Index for f1
k2 = round((f2 * N_new) / fs);  % Index for f2

% Define the range of bins to search around k1 and k2
binRange = -1:1;  % Range of +/- 1 bin around the calculated
    index

disp(['Goertzel index for f1 (binary 0): ', num2str(k1)]);
disp(['Goertzel index for f2 (binary 1): ', num2str(k2)]);

% Initialize cross-correlation results
startIdx = -1;
```

```
% Sliding window approach to find the preamble
windowSize = preambleLength;  % Match the segment size to
    preamble length
stepSize = floor(symbolLength / symbolLength);  % Step by one-
    quarter symbol length
matchThreshold = 1.0;  % Threshold for partial match (90% of
    bits must match)

% Initial small-step search to lock onto the preamble
for i = 1:stepSize:(length(fskSignal) - windowSize)
    % Extract the current window
    segment = fskSignal(i:i+windowSize-1);

    % Create binary stream to compare with preamble
    detectedBits = zeros(1, length(preamble));

    % Go through the preamble detection
    for j = 1:length(preamble)
        bitStart = (j-1)*symbolLength + skipSamples + 1;
        bitEnd = bitStart + N_new - 1;

        % Make sure the bit window doesn't exceed the segment
            bounds
        if bitEnd > length(segment)
            break;
        end

        segmentBit = segment(bitStart:bitEnd);

        % Apply Goertzel to the full segment for f1 and f2, over
             the range of bins
        % Initialise energies
        energy_f1 = 0;
        energy_f2 = 0;

        for kOffset = binRange
            if k1 + kOffset <= N_new && k2 + kOffset <= N_new
                S_f1 = goertzel(segmentBit, k1 + kOffset);
                S_f2 = goertzel(segmentBit, k2 + kOffset);

                energy_f1 = energy_f1 + abs(S_f1(end))^2;
                energy_f2 = energy_f2 + abs(S_f2(end))^2;
            end
        end
        energy_f1 = 2 * energy_f1;  % Scale the energy of f1

        % Decide on the bit based on the energy levels
        if energy_f1 > energy_f2
            detectedBits(j) = 0;
        else
            detectedBits(j) = 1;
        end
    end

    % Calculate the proportion of matching bits
    matchPercentage = sum(detectedBits == preamble) / length(
        preamble);

    % If matchPercentage is greater than or equal to the
        threshold, preamble detected
    if matchPercentage >= matchThreshold
        startIdx = i + length(preamble) * symbolLength + (floor(
            symbolLength / 3));  % Start after preamble
        disp(['Message start found at index: ', num2str(startIdx
            )]);
        break;
    end
end

% If the preamble was detected
if startIdx ~= -1
    % Work around for startIdx to start from end of symbol,
       rather than
    % middle
    % startIdx = startIdx + (symbolLength / 2);
    startIdx = startIdx + 100;

    % Initialize binary stream
    numBits = floor((length(fskSignal) - startIdx + 1) /
       symbolLength);
    binaryStream = zeros(1, numBits);

    % Bit Detection using Goertzel
    for i = 1:numBits
        periodStart = startIdx + (i-1) * symbolLength +
           skipSamples + 1;
        periodEnd = periodStart + N_new - 1;

        % Boundary check to prevent exceeding array bounds
        if periodEnd > length(fskSignal)
            break;  % Exit the loop if there's not enough data
               left for another bit
        end

        segment = fskSignal(periodStart:periodEnd);

        % Apply Goertzel to the full segment for f1 and f2, over
            the range of bins
        energy_f1 = 0;
        energy_f2 = 0;

        for kOffset = binRange
            if k1 + kOffset <= N_new && k2 + kOffset <= N_new
                S_f1 = goertzel(segment, k1 + kOffset);
                S_f2 = goertzel(segment, k2 + kOffset);

                energy_f1 = energy_f1 + abs(S_f1(end))^2;
                energy_f2 = energy_f2 + abs(S_f2(end))^2;
            end
        end

        % Decide on the bit based on the energy levels
        if energy_f1 > energy_f2
            binaryStream(i) = 0;
        else
            binaryStream(i) = 1;
        end
    end

    % % Print the detected binary stream
    % disp('Detected binary stream:');
    % disp(binaryStream);
    % % ---------- Binary Stream Conversion and Display ----------
    % % Convert binary stream into binary string for display
    % binaryString = num2str(binaryStream);
    % binaryString = binaryString(binaryString ~= ' ');  %
       Remove spaces from the string
    %
    % % Split the binary string into 8-bit chunks (bytes) and
       display each byte
    % fprintf('Binary Stream (Grouped in Bytes):\n');
    % for i = 1:8:length(binaryString)
    %     byte = binaryString(i:i+7);
    %     fprintf('%s ', byte);
    % end
    % fprintf('\n');  % New line after binary stream

    % Convert binary stream to ASCII
    if mod(length(binaryStream), 8) ~= 0
        binaryStream = [binaryStream zeros(1, 8 - mod(length(
           binaryStream), 8))];
    end

    binaryMatrix = reshape(binaryStream, [8, length(binaryStream
       )/8])';
    asciiValues = bin2dec(num2str(binaryMatrix));
    recoveredText = char(asciiValues)';

    % End of message detection
    endCharAscii = 33;  % ASCII value for '!'
    endIndex = find(asciiValues == endCharAscii, 1);  % Find
       first occurrence of '!'

    % If '!' is found, truncate the ASCII values at the end of
       the message
    if ~isempty(endIndex)
        asciiValues = asciiValues(1:endIndex);
    end

    % Print the recovered text
    disp(['Recovered Text: ', char(asciiValues)']);
else
    disp('Preamble not detected.');
end
```

Appendix E

JavaScript API Code

E.1 JavaScript Code `ENP4111_FSKModDemodJSFFT.html` for Modulation and Demodulation

The following listing shows the progress of the API to facilitate acoustic data transfer.

If the reader wishes to run this code, I have written a short instruction list for running a local host:

1. Open terminal
2. Change file directory to necessary directory

   cd (file path)

   eg. cd C:\Users\Gary\Documents\API
3. Set local host (will host on port 8000) in terminal: python -m http.server
4. Find file in browser

   localhost:8000/filename.html

   eg. localhost:8000/API.html

Find file in browser localhost:8000/filename.html eg. localhost:8000/micTest.html

Listing E.1: Modulation and Demodulation HTML Document

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Acoustic Data Transfer API</title>
    <style>
        /* Basic styling for the body and container */
        body, html {
            margin: 0;
            padding: 0;
            height: 100%;
            display: flex;
            align-items: center;
            justify-content: center;
            font-family: Arial, sans-serif;
        }
        .container {
            text-align: center;
        }
        .button-container, .wav-container {
            display: flex;
            flex-direction: column;
            gap: 10px;
            align-items: center;
        }
        /* Styling for file input to make it wider */
        input[type="file"] {
            width: 300px;
            font-size: 16px;
            padding: 10px;
        }
        /* Styling for buttons and number input */
        button, input[type="number"] {
            width: 200px;
            margin: 5px auto;
            padding: 10px;
            font-size: 16px;
        }
        /* Styling for messages displayed on the page */
        #receivedMessage, #statusMessage {
            margin-top: 20px;
        }
    </style>
</head>
<body onload="onLoad()">
    <div class="container">
        <h1>Acoustic Data Transfer API</h1>
        <div class="button-container">
            <input type="file" id="fileInput" />
            <label for="baudRate">Baud Rate:</label>
            <input type="number" id="baudRate" value="25" />
            <button id="transmitBtn">Transmit</button>
            <button id="startReceiveBtn">Start Receiving</button
                >
            <button id="stopReceiveBtn">Stop Receiving</button>
            <button id="testMicrophoneBtn">Test Microphone</
                button>
            <button id="playbackReceivedBtn">Playback Received
                Audio</button>
        </div>
        <div class="wav-container">
            <input type="file" id="wavInput" />
            <button id="demodulateBtn">Demodulate WAV File</
                button>
        </div>
        <p id="receivedMessage"></p>
        <p id="statusMessage"></p>
    </div>

    <!-- Include the jsfft library ---- to include library, need
        to install node.js, then npm install jsfft-->

    <script>
        //-------------------------- Constants and Configuration
            --------------------------
        const SAMPLE_RATE = 8000; // The sampling rate of the
            audio in Hz
        const FREQ_LOW = 1000; // Frequency representing binary
            '0'
        const FREQ_HIGH = 2000; // Frequency representing binary
            '1'
        let INIT_BAUD_RATE = 25; // Default baud rate,
            adjustable via UI

        //-------------------------- Variables --------------------------
        let currentAudioBuffer = []; // Buffer for storing the
            audio data to be transmitted
        let audioContext, processor, source; // Web Audio API
            components
        let haveAudioAccess = false; // Flag indicating if audio access is granted
        let stopReceivingFlag = false; // Flag to control the receiving process
        let audioChunks = []; // Array to store received audio chunks
        let receivedBinary = ''; // String to store received binary data

        //---------------------- Page Load Function ----------------------
        function onLoad() {
            console.log("Page loaded");
            setupEventListeners(); // Set up event listeners on page load
        }

        //---------------------- Setup Event Listeners ----------------------
        function setupEventListeners() {
            document.getElementById('transmitBtn').addEventListener('click', transmit);
            document.getElementById('startReceiveBtn').addEventListener('click', startReceiving);
            document.getElementById('stopReceiveBtn').addEventListener('click', stopReceiving);
            document.getElementById('testMicrophoneBtn').addEventListener('click', testMicrophone);
            document.getElementById('playbackReceivedBtn').addEventListener('click', playbackReceivedAudio);
            document.getElementById('demodulateBtn').addEventListener('click', demodulateWAVFile);
        }

        //---------------------- Transmit Function ----------------------
        function transmit() {
            console.log("Transmit button clicked");
            let fileInput = document.getElementById('fileInput');
            let baudRate = parseInt(document.getElementById('baudRate').value) || INIT_BAUD_RATE;

            if (fileInput.files.length > 0) {
                let reader = new FileReader();
                reader.onload = function (e) {
                    let text = e.target.result; // Get text from the uploaded file
                    console.log("Text read from file:", text);
                    let binaryData = textToBinary(text); // Convert text to binary
                    console.log("Binary data:", binaryData);
                    let audioBuffer = modulate(binaryData, baudRate); // Modulate binary data to audio
                    currentAudioBuffer = audioBuffer;
                    playAudio(audioBuffer); // Play the modulated audio
                    saveAsWav(audioBuffer); // Save the audio as a WAV file
                };
```

```
                reader.readAsText(fileInput.files[0]); // Read the file as text
            } else {
                alert('Please select a file to transmit');
            }
        }

        //------------------------------ Text to Binary Conversion ------------------------------
        function textToBinary(text) {
            return text.split('').map(char => {
                return char.charCodeAt(0).toString(2).padStart(8, '0');
            }).join(''); // Convert each character to binary and join them
        }

        //------------------------------ Modulation Function ------------------------------
        function modulate(binaryData, baudRate) {
            console.log("Modulating binary data");
            let audioBuffer = [];
            let duration = SAMPLE_RATE / baudRate; // Calculate duration for each bit

            // For each bit, generate corresponding tone
            for (let bit of binaryData) {
                let frequency = (bit === '0') ? FREQ_LOW : FREQ_HIGH;
                for (let i = 0; i < duration; i++) {
                    let sample = Math.sin(2 * Math.PI * frequency * i / SAMPLE_RATE);
                    audioBuffer.push(sample);
                }
            }

            console.log("Modulation complete");
            return audioBuffer; // Return the modulated audio buffer
        }

        //------------------------------ Play Audio Function ------------------------------
        function playAudio(audioBuffer) {
            console.log("Playing Audio");
            audioContext = new (window.AudioContext || window.webkitAudioContext)(); // Create audio context
            let buffer = audioContext.createBuffer(1, audioBuffer.length, SAMPLE_RATE);
            buffer.copyToChannel(new Float32Array(audioBuffer), 0); // Copy audio data to buffer
            let source = audioContext.createBufferSource(); // Create audio source
            source.buffer = buffer; // Set buffer to source
            source.connect(audioContext.destination); // Connect source to audio output
            source.start(); // Start playback
        }

        //------------------------------ Save as WAV Function ------------------------------
        function saveAsWav(audioBuffer) {
            console.log(”Saving as WAV”);
            const silenceDuration = 1; // Duration of silence before and after the audio
            const silenceSamples = silenceDuration * SAMPLE_RATE;
            const silenceBuffer = new Array(silenceSamples).fill(0);
            let extendedAudioBuffer = silenceBuffer.concat(audioBuffer, silenceBuffer); // Add silence around audio

            let wavEncoder = new WavAudioEncoder(SAMPLE_RATE, 1); // Create WAV encoder
            wavEncoder.encode([new Float32Array(extendedAudioBuffer)]); // Encode audio data
            let blob = wavEncoder.finish(); // Get the WAV file blob
            let url = URL.createObjectURL(blob); // Create URL for the blob
            let a = document.createElement('a');
            a.href = url;
            a.download = 'modulated_audio.wav'; // Set download file name
            a.click(); // Trigger download
            URL.revokeObjectURL(url); // Revoke the URL
        }

        //------------------------ Start Receiving Function ------------------------
        async function startReceiving() {
            console.log(”Start Receiving button clicked”);
            if (navigator.mediaDevices) {
                try {
                    const constraints = { audio: { sampleRate: SAMPLE_RATE } }; // Set audio constraints
                    const audioStream = await navigator.mediaDevices.getUserMedia(constraints); // Get audio stream
                    audioContext = new AudioContext({ sampleRate: SAMPLE_RATE }); // Create audio context
                    await audioContext.audioWorklet.addModule('processor.js'); // Load audio worklet processor
                    processor = new AudioWorkletNode(audioContext, 'audio-processor'); // Create processor node
                    source = audioContext.createMediaStreamSource(audioStream); // Create media stream source
                    source.connect(processor); // Connect source to processor
                    processor.connect(audioContext.destination); // Connect processor to output
                    processor.port.onmessage = (event) => {
                        if (!stopReceivingFlag) {
                            processAudioBuffer(event.data); // Process the audio data received
                            audioChunks.push(...event.data); // Store received audio chunks
                        }
                    };
                    haveAudioAccess = true; // Audio access
```

```
                        granted
                    document.getElementById('statusMessage').innerText = 'Receiving started...';
                    alert('Audio access granted');
                } catch (err) {
                    alert('Failed to get audio stream');
                }
            } else {
                alert('navigator.mediaDevices is undefined or not supported in this browser.');
            }
        }

        //------------------------ Stop Receiving Function ------------------------
        function stopReceiving() {
            console.log("Stop Receiving button clicked");
            if (audioContext && haveAudioAccess) {
                stopReceivingFlag = true; // Set flag to stop receiving
                source.disconnect(processor); // Disconnect the audio source
                processor.disconnect(audioContext.destination); // Disconnect the processor
                audioContext.close(); // Close the audio context
                haveAudioAccess = false; // Reset audio access flag
                document.getElementById('statusMessage').innerText = 'Receiving stopped.';
            }
        }

        //------------------------ Process Audio Buffer Function ------------------------
        function processAudioBuffer(data) {
            console.log("Processing audio buffer");

            // Pad the data to the next power of two for FFT
            let paddedData = padToPowerOfTwo(data);
            let real = new Float32Array(paddedData);
            let imag = new Float32Array(real.length); // Imaginary part for FFT

            // Perform FFT using jsfft
            FFT.fft(real, imag); // This modifies 'real' and 'imag' arrays in place

            console.log("FFT Result (Real):", real);
            console.log("FFT Result (Imag):", imag);

            // Calculate magnitude and proceed with demodulation
            let magnitude = calculateMagnitude(real, imag);
            let binaryData = demodulate(magnitude); // Convert magnitude to binary data
            console.log("Demodulated Binary Data:", binaryData);
            let receivedText = binaryToText(binaryData); // Convert binary data to text
            console.log("Received Text:", receivedText);
            document.getElementById('receivedMessage').innerText = receivedText; // Display received text
        }

```

```
    //——————————————— Pad Data to Power of Two ———————————————Would not be necessary for Goertzel's Algorithm
    function padToPowerOfTwo(data) {
        let len = data.length;
        let nextPowerOfTwo = Math.pow(2, Math.ceil(Math.log2(len))); // Calculate next power of two
        let paddedData = new Array(nextPowerOfTwo).fill(0);
        for (let i = 0; i < len; i++) {
            paddedData[i] = data[i];
        }
        return paddedData; // Return padded data array
    }

    //——————————————— Calculate Magnitude ———————————————
    function calculateMagnitude(real, imag) {
        let magnitude = [];
        for (let i = 0; i < real.length; i++) {
            magnitude.push(Math.sqrt(real[i] ** 2 + imag[i] ** 2)); // Calculate magnitude
        }
        return magnitude; // Return magnitude array
    }

    //——————————————— Demodulate Function ———————————————
    function demodulate(magnitude) {
        let binaryData = '';
        let freqLowIndex = Math.round(FREQ_LOW * SAMPLE_RATE / magnitude.length);
        let freqHighIndex = Math.round(FREQ_HIGH * SAMPLE_RATE / magnitude.length);
        let threshold = (magnitude[freqLowIndex] + magnitude[freqHighIndex]) / 2; // Set threshold

        // Iterate over the magnitude data to decode binary information
        for (let i = 0; i < magnitude.length; i += SAMPLE_RATE / INIT_BAUD_RATE) {
            let segment = magnitude.slice(i, i + SAMPLE_RATE / INIT_BAUD_RATE);
            let maxMagnitude = Math.max(...segment);
            binaryData += maxMagnitude > threshold ? '1' : '0'; // Determine binary '1' or '0'
        }
        return binaryData; // Return demodulated binary data
    }

    //——————————————— Binary to Text Conversion ———————————————
    function binaryToText(binaryData) {
        let text = '';
        for (let i = 0; i < binaryData.length; i += 8) {
            let byte = binaryData.slice(i, i + 8);
            text += String.fromCharCode(parseInt(byte, 2)); // Convert binary to text
        }
        return text; // Return converted text
    }

    //——————————————— Test Microphone Function ———————————
```

```
        --------------------
    function testMicrophone() {
        console.log("Testing microphone...");
        startRecording(); // Start recording audio

        setTimeout(() => {
            stopRecording(); // Stop recording after 5 seconds
            playbackReceivedAudio(); // Play back the recorded audio
        }, 5000); // 5 seconds timeout
    }

    //-------------------- Start Recording Function --------------------
    async function startRecording() {
        console.log("Start Recording button clicked");
        try {
            const constraints = { audio: { sampleRate: SAMPLE_RATE } }; // Set audio constraints
            const audioStream = await navigator.mediaDevices.getUserMedia(constraints); // Get audio stream
            if (audioStream) {
                audioContext = new AudioContext({ sampleRate: SAMPLE_RATE }); // Create audio context
                await audioContext.audioWorklet.addModule('processor.js'); // Load audio worklet processor
                processor = new AudioWorkletNode(audioContext, 'audio-processor'); // Create processor node
                source = audioContext.createMediaStreamSource(audioStream); // Create media stream source
                source.connect(processor); // Connect source to processor
                processor.connect(audioContext.destination); // Connect processor to output
                processor.port.onmessage = (event) => {
                    audioChunks.push(...event.data); // Store received audio chunks
                };
                haveAudioAccess = true; // Audio access granted
                document.getElementById('statusMessage').innerText = 'Recording...';
            } else {
                alert('Failed to get audio stream');
            }
        } catch (err) {
            alert('Failed to get audio stream');
            console.error('Error accessing audio:', err);
        }
    }

    //-------------------- Stop Recording Function --------------------
    function stopRecording() {
        console.log("Stop Recording Initiated");
        if (audioContext && haveAudioAccess) {
            source.disconnect(processor); // Disconnect the
```

```
                audio source
            processor.disconnect(audioContext.destination);
                // Disconnect the processor
            if (audioContext.state !== 'closed') {
                audioContext.close(); // Close the audio
                    context if not already closed
            }
            haveAudioAccess = false; // Reset audio access
                flag
            document.getElementById('statusMessage').
                innerText = 'Recording Stopped.';
        }
    }

    //------------------------- Playback Received Audio
        -------------------------
    function playbackReceivedAudio() {
        console.log("Playing back received audio");
        if (audioChunks.length === 0) {
            console.error("No audio data to play back.");
            document.getElementById('statusMessage').
                innerText = 'No audio data to play back.';
            return;
        }
        const audioBuffer = new Float32Array(audioChunks);
            // Create a Float32Array from audio chunks
        const playbackContext = new AudioContext({
            sampleRate: SAMPLE_RATE }); // Create a new audio
             context
        const buffer = playbackContext.createBuffer(1,
            audioBuffer.length, SAMPLE_RATE);
        buffer.copyToChannel(audioBuffer, 0); // Copy audio
            data to the buffer
        const source = playbackContext.createBufferSource();
             // Create an audio source
        source.buffer = buffer; // Assign buffer to the
            source
        source.connect(playbackContext.destination); //
            Connect the source to the audio output
        source.start(); // Start playback
        document.getElementById('statusMessage').innerText =
             'Playing back audio...';
    }

    //------------------------- Demodulate WAV File
        -------------------------
    function demodulateWAVFile() {
        console.log("Demodulating WAV File");
        let wavInput = document.getElementById('wavInput');
        if (wavInput.files.length > 0) {
            let file = wavInput.files[0];
            let reader = new FileReader();
            reader.onload = function (e) {
                let audioContext = new AudioContext();
                audioContext.decodeAudioData(e.target.result
                    , function (buffer) {
                    let channelData = buffer.getChannelData
                        (0); // Get audio data from buffer
                    processAudioBuffer(channelData); //
                        Process the audio buffer
                });
            };
            reader.readAsArrayBuffer(file); // Read the WAV file as an ArrayBuffer
        } else {
            alert('Please select a WAV file');
        }
    }

    //------------------------------ WAV Encoder Class ------------------------------
    class WavAudioEncoder {
        constructor(sampleRate, channelCount) {
            this.sampleRate = sampleRate; // Set the sample rate
            this.channelCount = channelCount; // Set the number of channels
            this.buffer = []; // Initialize the buffer for audio data
            this.length = 0; // Initialize the length of the audio data
        }

        encode(buffers) {
            // Iterate over each sample in the buffers
            for (let i = 0; i < buffers[0].length; i++) {
                for (let j = 0; j < buffers.length; j++) {
                    this.buffer.push(buffers[j][i] * 32767.5 - 0.5); // Scale and push data
                }
                this.length += this.channelCount; // Update length
            }
        }

        finish() {
            let dataSize = this.length * this.channelCount * 2; // Calculate data size
            let buffer = new ArrayBuffer(44 + dataSize); // Create buffer for WAV header and data
            let view = new DataView(buffer);
            this._writeString(view, 0, 'RIFF'); // Write RIFF header
            view.setUint32(4, 36 + dataSize, true); // Write file size minus 8 bytes
            this._writeString(view, 8, 'WAVE'); // Write WAVE format identifier
            this._writeString(view, 12, 'fmt '); // Write fmt chunk
            view.setUint32(16, 16, true); // PCM chunk size
            view.setUint16(20, 1, true); // Audio format (1 = PCM)
            view.setUint16(22, this.channelCount, true); // Number of channels
            view.setUint32(24, this.sampleRate, true); // Sample rate
            view.setUint32(28, this.sampleRate * this.channelCount * 2, true); // Byte rate
            view.setUint16(32, this.channelCount * 2, true); // Block align
            view.setUint16(34, 16, true); // Bits per sample
            this._writeString(view, 36, 'data'); // Write
```

```
data chunk identifier
                view.setUint32(40, dataSize, true); // Write data size
                this._floatTo16BitPCM(view, 44, this.buffer); // Write the PCM data
                return new Blob([view], { type: 'audio/wav' }); // Create the Blob and return it
            }

            _writeString(view, offset, string) {
                for (let i = 0; i < string.length; i++) {
                    view.setUint8(offset + i, string.charCodeAt(i)); // Write characters to DataView
                }
            }

            _floatTo16BitPCM(output, offset, input) {
                for (let i = 0; i < input.length; i++, offset += 2) {
                    let s = Math.max(-1, Math.min(1, input[i])); // Clipping the input to [-1, 1]
                    output.setInt16(offset, s < 0 ? s * 0x8000 : s * 0x7FFF, true); // Convert to 16-bit PCM
                }
            }
        }

        //---------------------- On Page Load ----------------------
        window.onload = onLoad; // Call onLoad function when the page loads
    </script>
</body>
</html>
```

E.2 Processor Code `processor.js` for Audio Processing

The following JavaScript processor code must be used with the above code to process audio.

Listing E.2: Processor JavaScript

```
// processor.js
class AudioProcessor extends AudioWorkletProcessor {
    process(inputs, outputs, parameters) {
        const input = inputs[0];
        if (input.length > 0) {
            const inputData = input[0];
            this.port.postMessage(inputData);
        }
        return true;
    }
}

registerProcessor('audio-processor', AudioProcessor);
```