COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

Radio-Enabled Off-Grid Telemetry

Alamosa

Dawson David

Micah Bird

Liam Kerrigan

Nate Moore

CSCI 370 Fall 2025

Advisor: Dr. Bahar

Table of Contents

I. Introduction

The team directly worked with Scott Jensen of Radio Alamosa Limited to develop a web-enabled data-to-audio encoder and decoder telemetry system. The primary goal was to establish a reliable radio communication methodology to allow users to send and receive data in off-grid environments. As Scott stated: "This is going to be GMRS 2.0, which is basically taking GRMS radio and adding digital capabilities."

Data input sources are arbitrary and can include IoT devices or direct user input. A wide swath of applications of this system exist. For example, a device on a remote weather station could transmit its information to a base station. The system utilized audio encoding and decoding through the WebAudio API, drawing inspiration from older modem standards (such as the Bell 202 standard, a standard Scott instructed us to start with). Significant research and iterating were required, along with utilizing Mines resources available to the team. Regular weekly meetings with Scott took place, with mentorship and necessary tools provided by Radio Alamosa Limited. Data transmission employed URI datatypes, and the system's ultimate goal was for a range of up to 75 nautical miles with a 50-watt transmitter. Due to the lack of radio hardware from the client, the final product was unable to be tested with this kind of hardware. The team operated within the FCC regulatory framework and tested different radio channels managed by the walkie talkie devices that were provided by Scott.

The main stakeholder of this project was Scott Jensen, however, end-users could include individuals participating in outdoor activities (such as Backcountry skiing, or backpacking) who needed reliable location sharing. Maintenance responsibilities would fall to Radio Alamosa Limited upon project completion. The project took advantage of established technologies to ultimately build this encoding/decoding system into a progressive web app (PWA).

II. Functional Requirements

The functional requirements of this project would best be described as user stories:

- As a user, I am able to reliably encode and decode data in a fashion that can be transmitted and received by a radio device.
- As a user with no internet connectivity, I am able to access and utilize the web application (PWA).
- As a user, I am able to connect my radio via a standard 3.5mm headphone jack, allowing communication with other radios using readily available hardware.
- As a user, when receiving a data transmission, the web application successfully decodes the message.
- As a developer, the system implements error detection and correction mechanisms (such as checksums and parity bits) to ensure data integrity during transmission.
- As a user, I am able to modify the underlying configuration of how data is transmitted/received.
- As a developer, I ensure that the system is tolerant to noise and has error correction capabilities.
- As a developer, I ensure the system adheres to all relevant FCC rules and regulations regarding radio transmissions, maintaining legal compliance and preventing interference with other devices.

The following is a table of requirements for the project.

Requirement	Rationale	Relationships
FCC Compliance	Legal operation and avoidance of interference.	All requirements must operate within the constraints of FCC regulations.
Data Decoding	The receiving and understanding of the data being sent.	Reliant on accurate data encoding and reception.
Data Transmission Accuracy	Ensures reliability of data. Unreliable data renders the system useless.	Directly impacts user trust and the core functionality of the application. Error detection/correction mechanisms are essential.
PWA Functionality (Offline)	The core use case requires offline functionality. The entire system is predicated on operating in areas without internet.	Reliant on a fully functional PWA.

All of the requirements of the table are of utmost importance. As for the implementation of these user stories and requirements, a pipeline will be in place for how the input of data is handled, which is discussed in further detail in the system architecture section.

III. Non-Functional Requirements

The team prioritized the reliable and accurate transmission of data over all other considerations, however the following non-functional requirements were accomplished in the project timeline:

- Is efficient enough to work on low end devices (phone, IOT devices, etc.).
- Has fast data transmission speeds (1200+ baud).
- Uses libraries with permissible licenses (the MIT license).
- Operates within 11.25 khz of bandwidth.

IV. Risks

Risk	Likelihood	Impact	Mitigation Plan
Accidental Violation of FCC Regulations during Testing	Likely	Major	Thoroughly research all relevant FCC regulations [1] prior to testing. Be aware of testing restrictions, such as power level limitations and frequency band allotments. Conduct all testing in a controlled environment.
Technical Complexity of Audio Processing Exceeds Team Skillset	Likely	Major	Dedicate initial project time to foundational research in audio processing techniques (modulation, demodulation, codecs). Seek mentorship from faculty with relevant expertise.
Resource Constraints on Mobile Devices Impeding PWA Functionality	Low	Medium	Profile PWA performance on typical mobile devices and optimize resource usage (memory, CPU).

V. Definition of Done

The project was considered complete when all functional user stories were successfully implemented and verified. This included:

- Reliable data encoding and decoding for transmission and reception via audio.
- Offline functionality of the PWA, ensuring usability without internet connectivity.
- Standard 3.5mm headphone jack connectivity for radio integration using readily available hardware.
- Successful decoding of received data transmissions within the web application.
- Configurable transmission/reception settings for user customization.
- Noise tolerance and error correction capabilities to maintain reliability in suboptimal conditions.
- Implementation of error detection and correction mechanisms (checksums, FEC) to ensure data integrity.
- Compliance with FCC regulations for legal and interference-free radio transmissions.

Additionally, the following non-functional requirements were met:

- Efficiency on low-end devices (phones, IoT devices).
- High-speed data transmission (1200+ baud).
- Use of permissively licensed libraries (MIT license).
- Operation within an 11.25 kHz bandwidth.

Before the conclusion of the semester, the team had a formal code review with the client, and he expressed his satisfaction with our final product. The final deliverables (including all source code and documentation) were shared electronically via a transfer of ownership of a GitHub organization to Scott Jensen.

VI. Proof of Concept Prototype

In our initial meeting, our client voiced concern that his envisioned system might not even be feasible, particularly due to his requirement that the modem be implemented as a web application and use the Web Audio API. To verify that the project was feasible, group member Liam spent the first two weeks focused on making a

proof of concept prototype using the Bell 202 standard. The Bell 202 standard was recommended by our client because of its simplicity. It is less efficient than more sophisticated modulation schemes, but it would not have been wise to jump in the deep end by starting with trying to implement those more sophisticated schemes.

The Bell 202 standard describes transmitting 1s and 0s with two special tones: 1200 Hz for a 1 and 2200 Hz for a 0. Figure 1 is an example of a Bell 202 signal. It can be seen how the waveform alternates between a high frequency 2200 Hz wave and a low frequency 1200 Hz wave to describe a sequence of 1s and 0s.

Figure 1. Bell 202 prototype export waveform (made with Audacity)

Core to the function of this initial prototype was the Goertzel algorithm. Our initial thought was to use a Fourier Transform instead, but the Fourier transform wastes effort analyzing every possible frequency. Using the Goertzel algorithm was more efficient because it directly targeted just the frequencies we needed. The Bell 202 prototype uses two Goertzel filters, one tuned to 1200 Hz and the other to 2200 Hz, to detect which tone is present. With a rudimentary timing recovery system to align the Goertzel filter windows to the symbol rate of the transmitter, i.e. synchronizing the transmitter and receiver, this system was able to successfully demodulate Bell 202 signals.

The Bell 202 prototype was implemented with just JavaScript and the WebAudio library, just as our client originally recommended. The prototype fulfilled its purpose of demonstrating feasibility; at this point we knew that we could create the product our client envisioned, so we started investigating ways to implement the more efficient but also more complex modulation schemes such as GMSK.

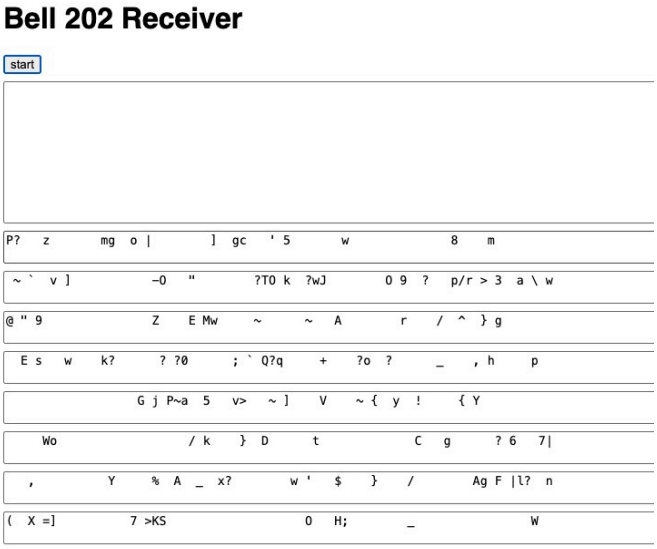


Figure 2. Bell 202 Receiver UI.

Figure 2 shows the receiver UI for the Bell 202 prototype. The bottom eight text boxes scroll to the left as time passes, and incoming messages appear in one of them. Each text box represents a different bitwise alignment of the last 8 bits received. That is, all eight boxes show decoded characters from the same bit stream, but each box starts decoding from a slightly different bit offset, ranging from offset 0 to offset 7. The primary

top large text box displays only received messages that come with the prefix of ‘1010101100’. Using this prefix prevents ambiguity about which bit offset, 0 through 7, correctly corresponds to the message.

Because the Bell 202 prototype was only a proof of concept, we did not bother to implement error correction. As a result, the Bell 202 implementation can sometimes receive signals incorrectly, as shown in Figure 3. These errors are just a result of noise though. In a noiseless environment and with the transmitting speaker and receiving microphone in close proximity, messages are virtually always received 100% correctly.

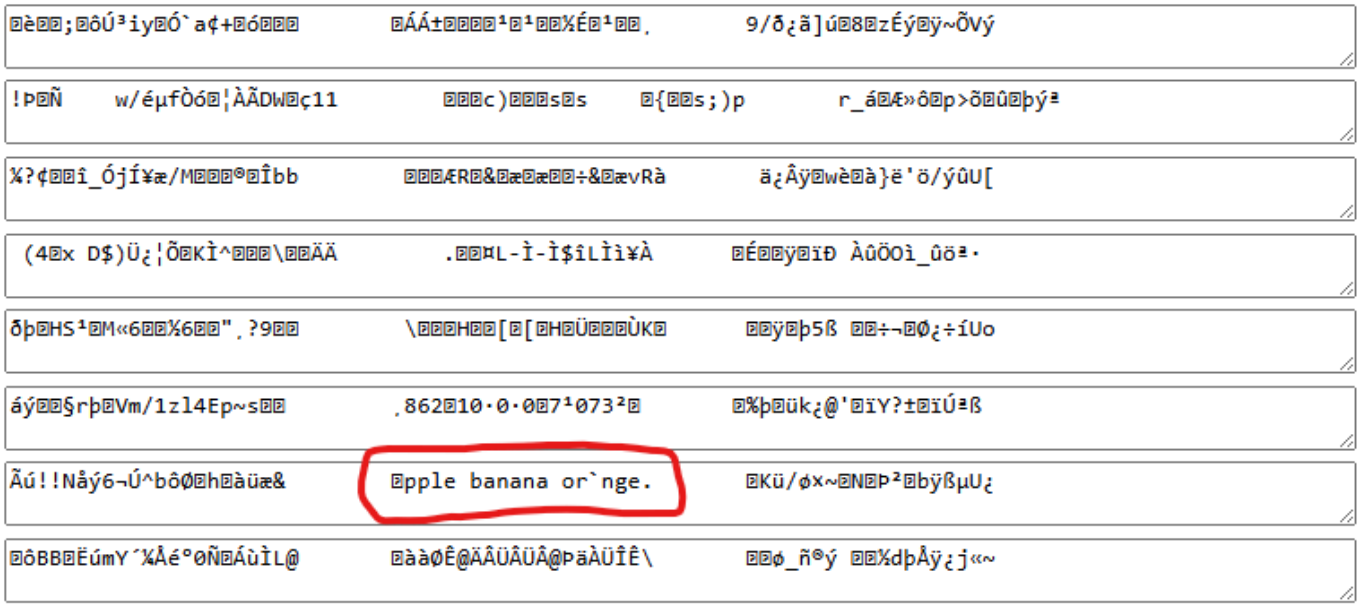


Figure 3. Bell 202 Receiving Message.

VII. System Architecture

The primary deliverable of this project was a library, providing reusable modules for data encoding/decoding and interfacing with radio devices via the WebAudio API. Through project iteration, the architecture evolved to leverage the liquid-dsp [2] library for core signal processing functionality. The final implementation is centered on a WebAssembly (WASM) module built from liquid-dsp, providing configurable and performant encoding and decoding capabilities.

This library is structured around the following functionality:

- Transmission (TX): Arbitrary text data is encoded into binary based on a text encoding scheme of the user’s choice, and then sent to liquid-dsp which then transforms the data into a packet which can be played over audio.
- Receiving (RX): Passively listens to audio from a microphone and if a data packet is received, will demodulate the audio back into binary, then decode the binary data back into text.
- Web Audio Interface: Leverages the WebAudio API to manage audio input and output, allowing communication with radio devices via the device’s audio jack.
- Error Correction: Provides utilities for enabling both Forward Error Correction and Checksums on transmitted and received data.

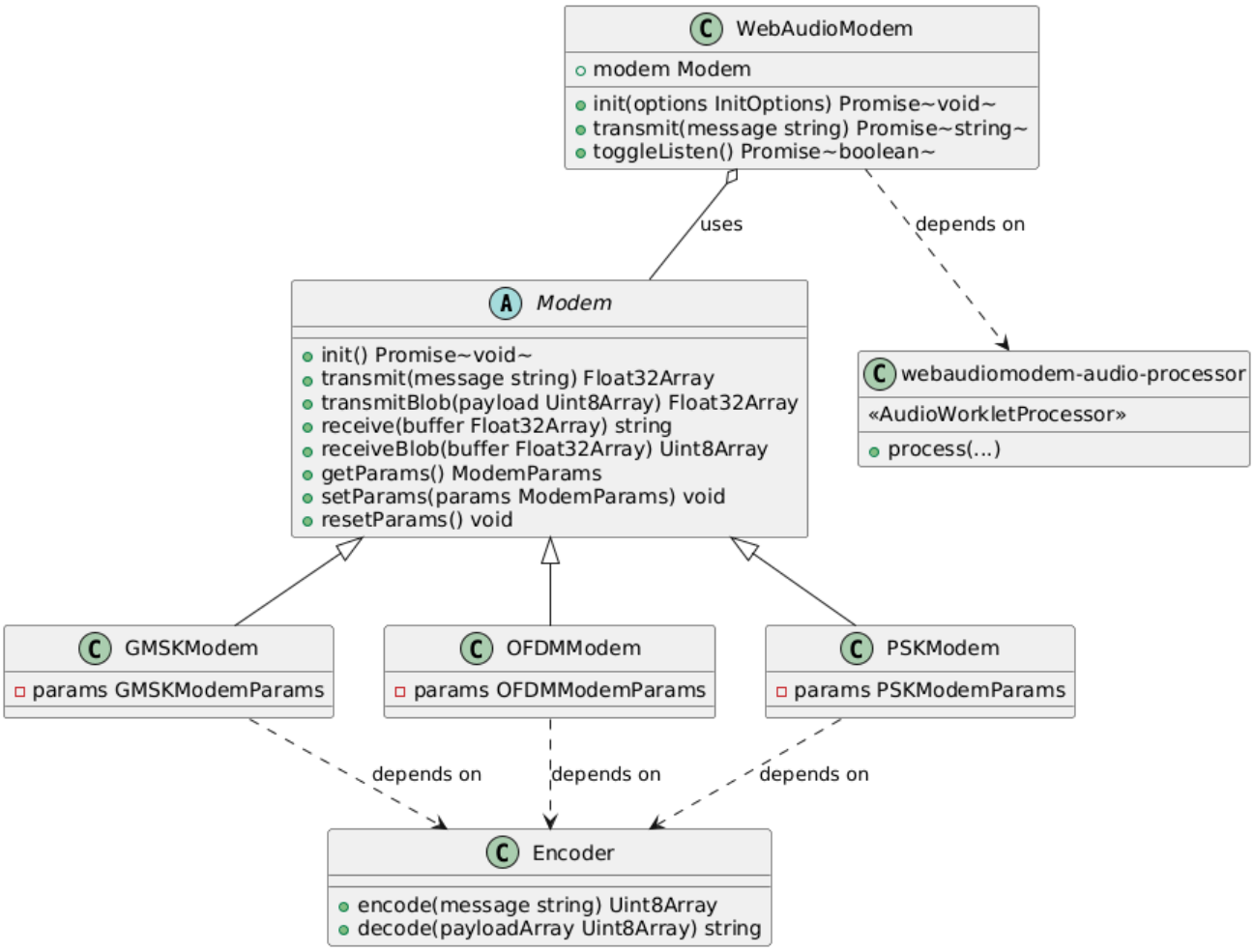


Figure 4. Class Structure, a green filled circle denotes a public method, while a red square denotes a protected member variable.

1. Technical Design & Implementation

Initial prototype challenges included difficulties achieving a reliable FSK decoder in JavaScript, the complexity of PSK decoding algorithms, and performance limitations inherent in JavaScript threading. To address these issues, the team investigated and settled on WASM as an alternative implementation path and explored existing signal processing libraries. As WASM enables efficient execution of compiled code the team determined that it is a better path forward over JavaScript.

Ultimately, liquid-dsp, a C digital signal processing library, was selected as the primary backbone of the project due to its MIT license, built-in error detection, error correction capabilities, and countless modulation schemes. The decision was made to utilize liquid-dsp's "flexframes" for handling packet structure and synchronization. A custom Bell 202 implementation was retained as a fallback, though its stabilization would require significant effort.

2. WASM Integration Details

The core signal processing was handled within a WASM module compiled from liquid-dsp using Emscripten. This module provides high-performance audio encoding and decoding through a C shim. Lower level data types are passed between JavaScript and the WASM module for processing, such as Float32Arrays (for audio) and Uint8Arrays (for data). Through cross-platform testing, low RAM footprint on Firefox Mobile

(Android) and manageable CPU spikes across MacOS, Linux, and Mobile devices were observed and was approved by the client.

3. Shim Architecture Explained

With us now being able to use WASM to compile and use the liquid-dsp library, we needed some C code in order to be able to interface with it. This C shim contains near all the parameters that the library exposed us to, allowing for complete customization. For each type of modulation it had its own custom function to implement its version of flexframe [4], and then each of these functions called the same shared function for necessary digital signalling processing. For example the tx (transmit) for the traditional flexframe had a similar pipeline of: flexframe to multistage arbitrary resampler to DDS and then to a Hilbert transform. The flexframe encodes the package, with optional forward error correction, and cyclical redundancy checks, in addition to parts such as p/n, preamble, and ramp up/down to ensure synchronization. Following this the multistage resampler is just an efficient way to interpolate/decimate in order to convert either between sample rates, or just to add more bits. DDS, allows us to take the audio sample before, and convert it so the spectrogram has the audio reading more in line to 0 to 3125Hz rather than the screeching 0 to 24000Hz. This is then passed into the Hilbert transform in order to take the complex array we had before, and get only a real array. The rx (receive) portion is similar but in reverse.

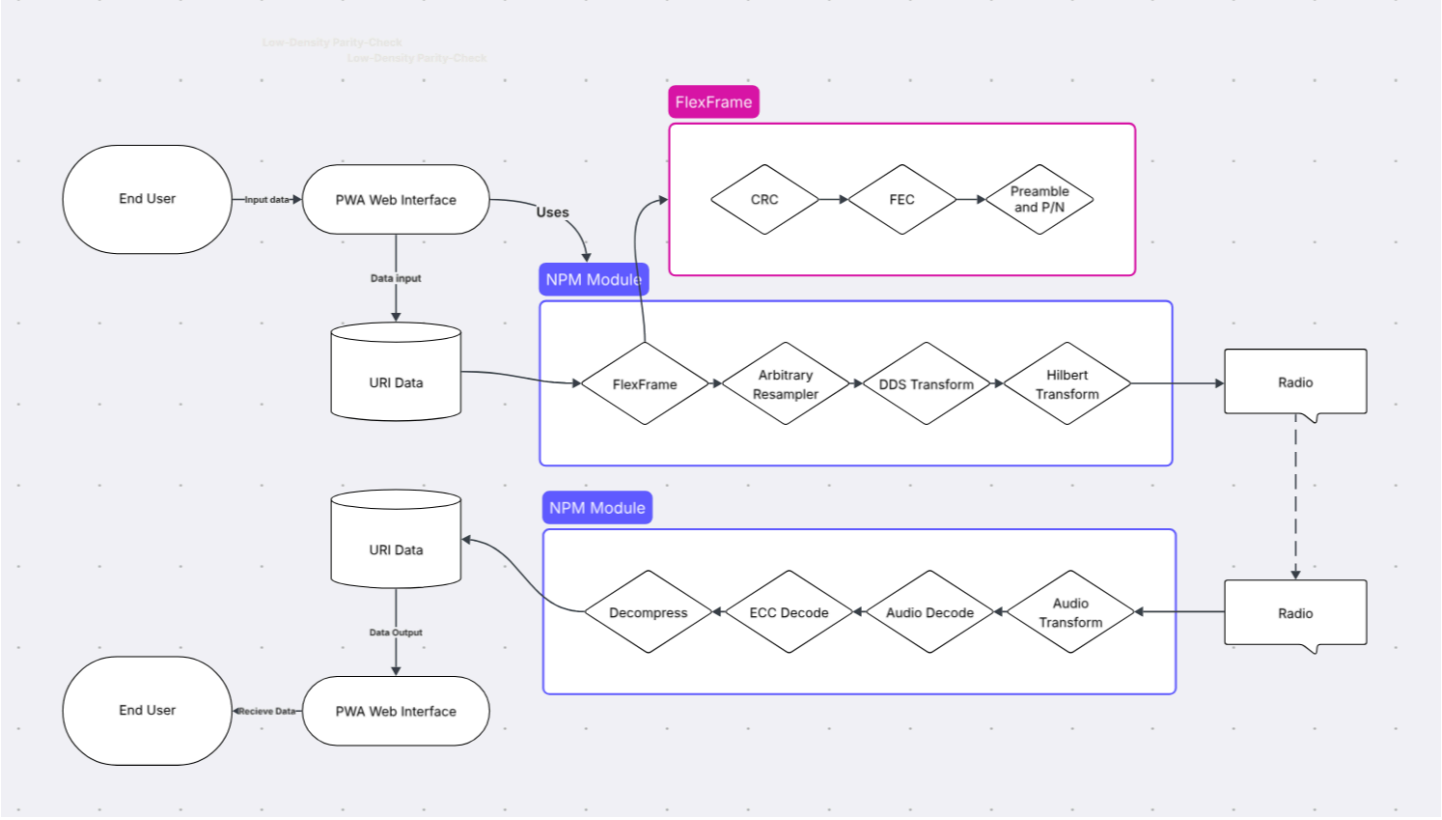


Figure 5. Simplified Architecture Diagram

4. Final Implementation & Deliverables

The completed library delivers a comprehensive WASM flexframe example integration supporting multiple modem types (PSK, GMSK, OFDM), a myriad of modulation schemes within each modem, and text encoding schemes (UTF8, BAUDOT [3]). Successful implementation of p/n sequences [4] for frame synchronization and phase locking was achieved, and noise testing conducted using overlaid noise on

transmission WAV files verified signal integrity. Code review was completed, and integration examples were provided, finalizing the shift from a custom JavaScript implementation to a feature complete, WASM solution utilizing liquid-dsp.

VIII. Software Test and Quality

1. Quality Assurance Approach

Given the research-focused nature of this project, the quality assurance strategy centered around rigorous integration testing and code review. In the end, unit tests were created for each of the abstract modem classes to ensure they behave as expected. This included ensuring that end-to-end each of the PSK, GMSK, OFDM modem would be able to transmit and receive data, along with ensuring that a certain method would perform the correct action when given a set of parameters.

2. Quality Elements & Implementation

Integration Testing:

- Implementation**:** Integration testing was performed throughout the development lifecycle. The team systematically tested the data flow from an input source (direct user input) through the audio encoding/decoding process and recorded the results of these tests. Unit tests were also used to validate the implementation of the modems.
    - This also involved collecting performance metrics for devices, an example graph which can be seen below. These performance metrics demonstrated a low RAM footprint at most reached 150MB, CPU utilization spiked as expected when transmitting.

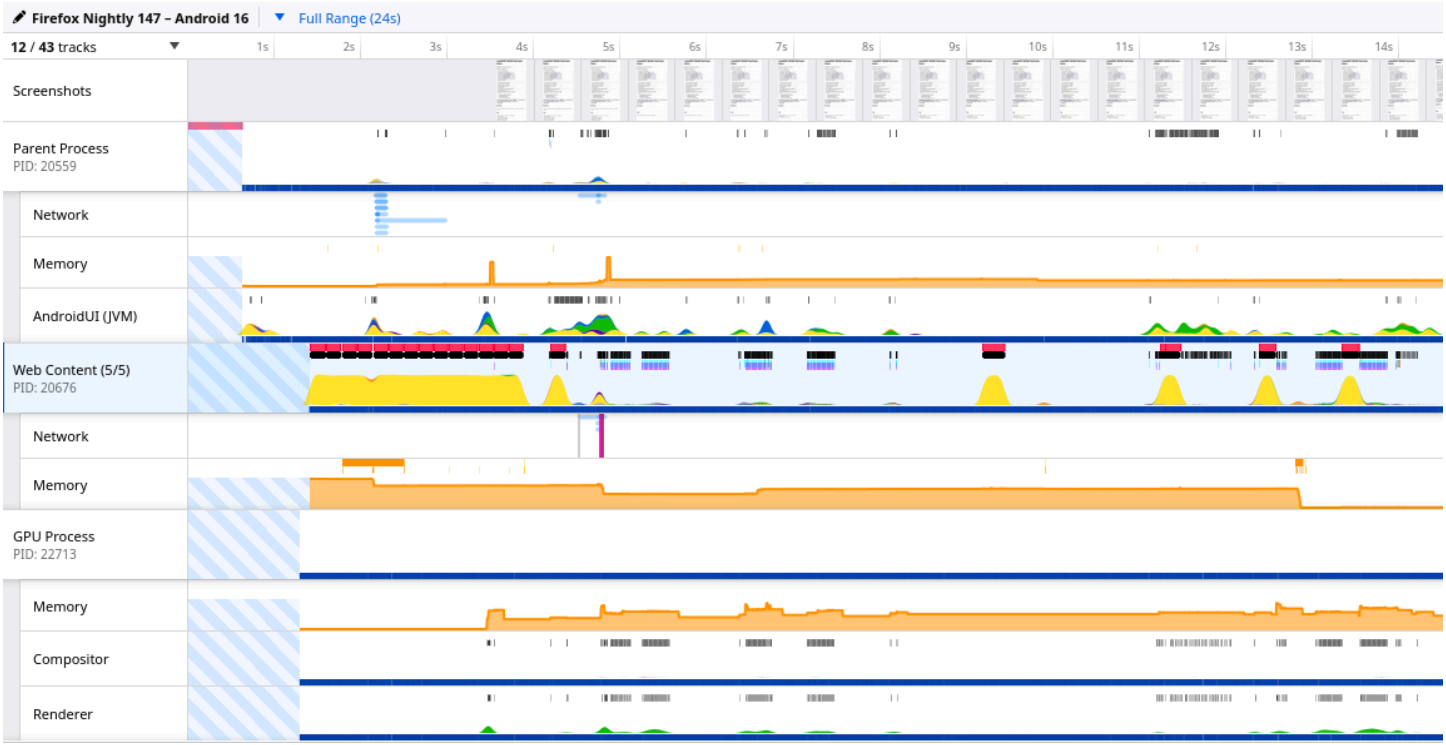


Figure 6. Example performance metrics collected from Firefox Mobile (Android).

- Focus: Validating data integrity and accuracy during transmission remained paramount. Testing results were documented to confirm data remained intact in the scenarios that it was expected to.
- Tools: Vite Unit Testing, Firefox Performance Metrics, Chrome Performance Profiling.
- Aspects Addressed: Error detection, forward error correction, validation of the complete data transmission pipeline, ensuring adherence to functional requirements.

Code Reviews:

- Implementation: All substantial code changes underwent peer review prior to merging into the main codebase. Reviews prioritized code readability, potential bugs, adherence to coding standards, and overall code quality.
- Focus: Preventing bugs, ensuring code maintainability, facilitating knowledge sharing within the team, and verifying the correctness of implemented logic.
- Tools: GitHub's pull request mechanism was utilized for code review.
- Aspects Addressed: Defect prevention, coding standards adherence, improved code maintainability, verification of logic.

FCC Compliance Verification:

- Implementation: Thorough research of FCC regulations was conducted prior to testing. All testing was controlled to ensure adherence to power level limitations, frequency band allotments, and other relevant regulations. The final library has the ability to enable/disable FRS Restrictions [1] to ensure compliance when transmitting over an FRS medium.
- Focus: Preventing accidental violations of FCC regulations.
- Tools: FCC website and resources, consultation with Scott Jensen.
- Aspects Addressed: Legal compliance, avoiding interference with other devices.

IX. Project Ethical Considerations

Given the team's initial limited experience with radio technology, a primary ethical focus throughout the project was strict adherence to FCC regulations. The potential for unintentionally interfering with or jamming emergency communications presented a significant risk. Relevant ACM/IEEE principles, specifically 1.2 (Avoid Harm), 2.3 (Know and respect existing rules), and the IEEE principle prioritizing public safety, health, and welfare, guided development efforts. Non-compliance with FCC regulations would represent a direct violation of these principles and could result in severe legal ramifications.

Throughout the project lifecycle, the team actively mitigated risk by applying Michael Davis's "Legality Test" to all actions, ensuring legal permissibility and compliance with applicable FCC rules. The "Harm Test" further reinforced this commitment, mandating the avoidance of any activity with potential for negative impact, such as radio interference. Early testing focused on range limitations with standard walkie-talkies, which minimized potential broadcast range and interference impact. Rigorous research, documentation of regulations, strict power limitation, and controlled testing environments were essential components of this strategy.

The project scope was intentionally constrained to omit encryption over personal radio frequencies, which requires specific licensing and would contribute to interference for other users. As a result, all data transmission within the library remained in plain text, precluding strong security measures. While this presents inherent vulnerability, the project's focus remained on minimizing unintentional interference and adherence to FCC regulations.

X. Results

1. Feature Implementation Progress

The team successfully delivered an all-encompassing liquid-dsp WASM flexframe demo supporting multiple modem types: PSK, GMSK, and OFDM, along with different text encoding schemes: UTF8 and BAUDOT. This was a significant shift from a custom JavaScript FSK implementation to a fleshed out, library-backed WASM solution. Dawson's implementation of liquid-dsp flexframes was critical to this progress, providing a stable data transmission pipeline.

2. Performance Testing Results

Initial performance testing highlighted constraints within the JavaScript implementation of FSK demodulation. However, the transition to liquid-dsp and WASM has fundamentally changed the project. Comprehensive testing was conducted on the liquid-dsp WASM implementation, revealing a low RAM footprint on Firefox Mobile (Android) (see figure 6) and enabling analysis of CPU spikes across MacOS, Linux, and mobile devices. Testing of noise resistance using overlaid recordings in Audacity yielded promising results (as shown in figure 7 below), validating signal integrity. The team aimed to achieve performance parity with or exceed initial results obtained with Fldigi. The following figure is initial test results obtained using Fldigi:

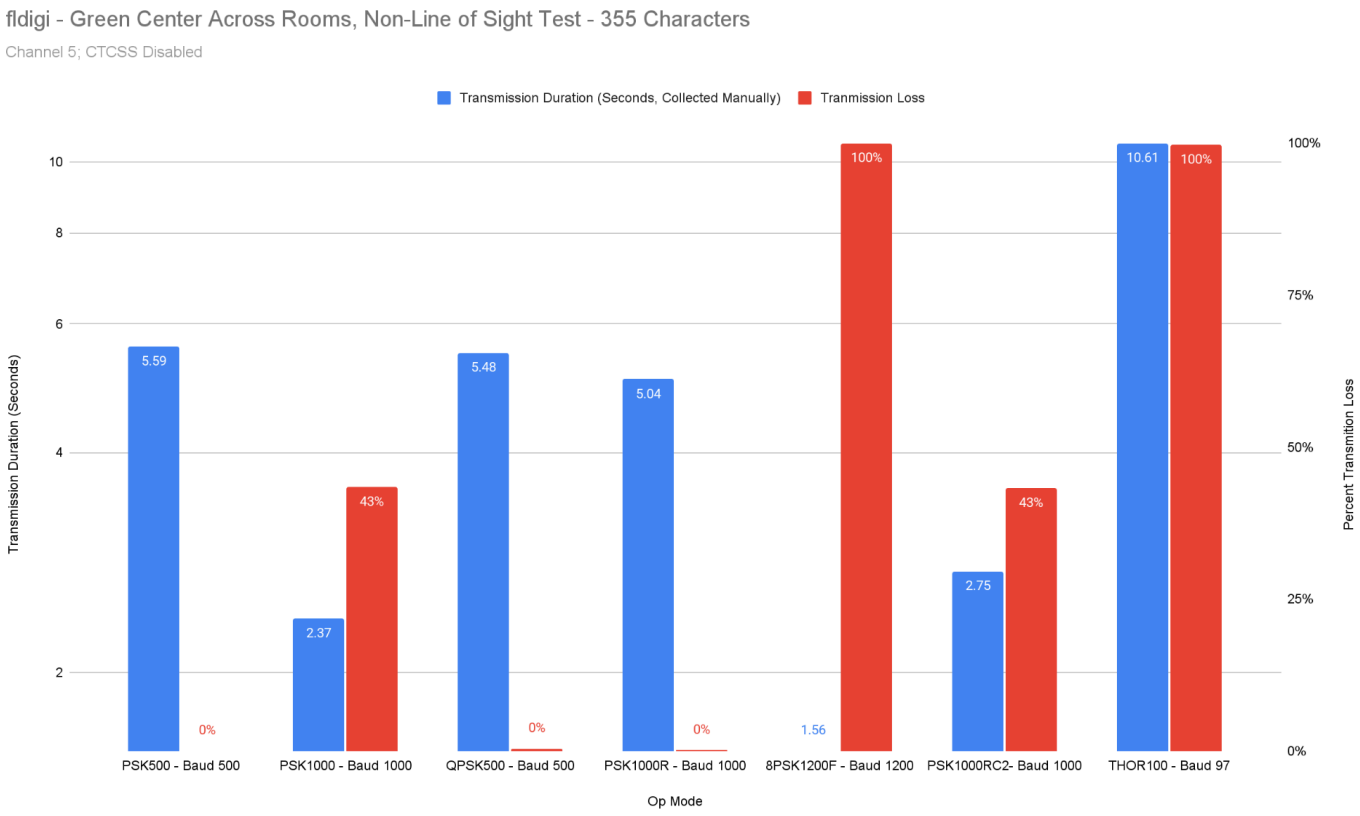


Figure 7. Fldigi initial testing results.

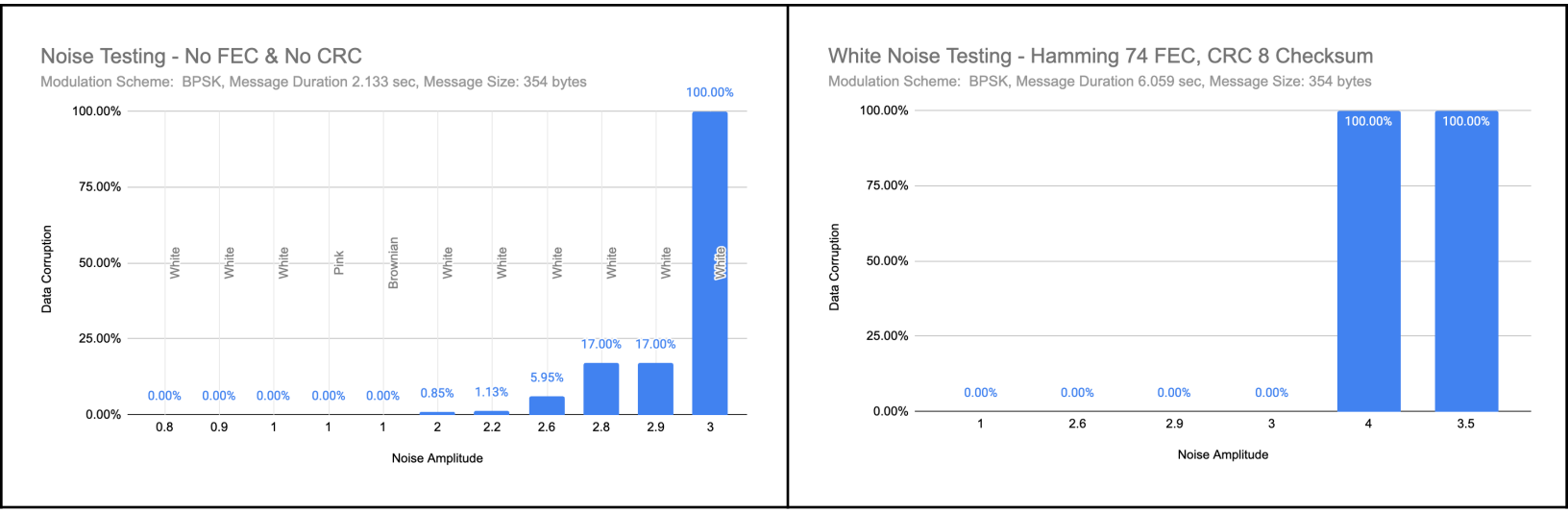


Figure 8. liquid-dsp WebAudioModem noise testing results.

3. Summary of Testing

Cross-platform testing confirmed the functionality of the flexframe example integration on modern web browsers (Chrome, Firefox) and desktop operating systems (Windows, macOS, Linux). Initial tests using the performance throttling feature of the Google Chrome profiler indicated sufficient processing time between messages, though CPU utilization reaches maximum levels during decoding. The client found these results satisfactory. Using the performance throttling of the profiler we achieved these results:

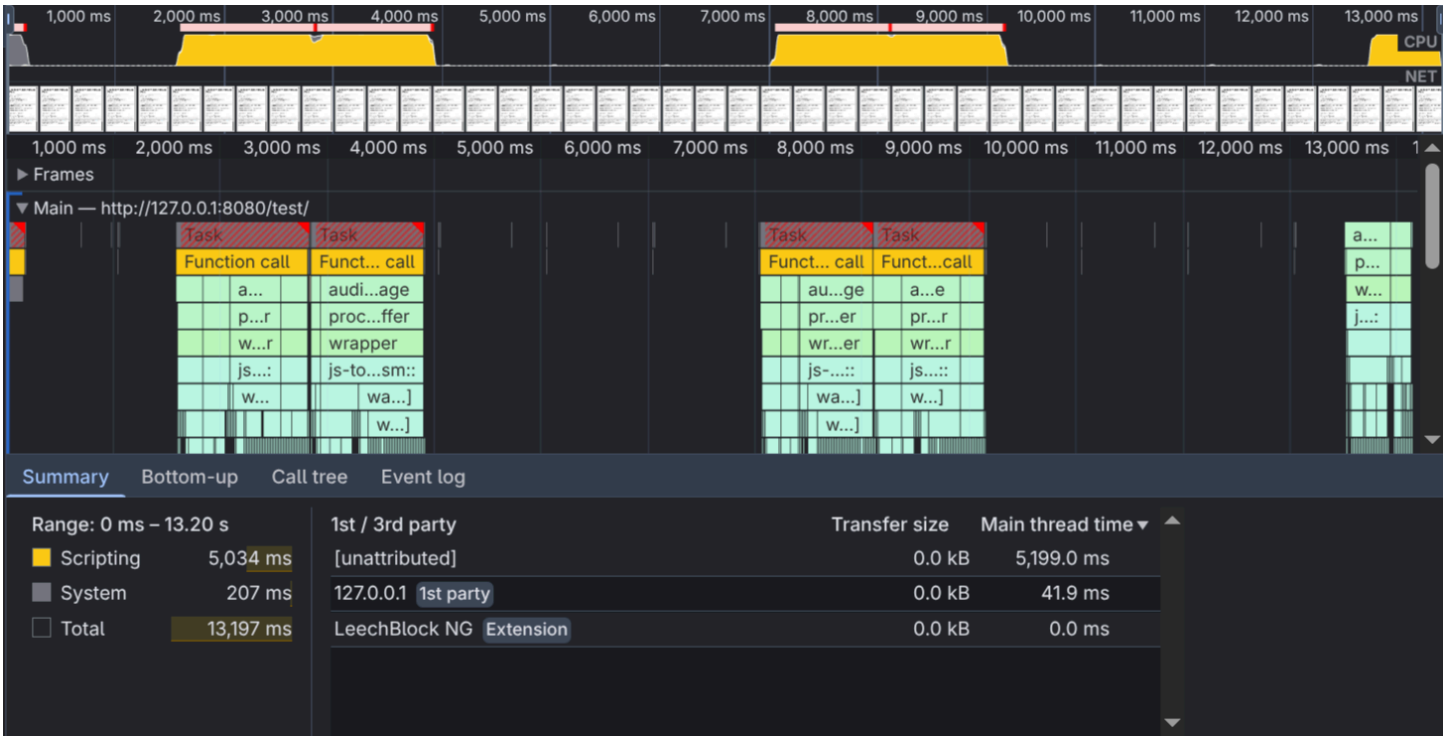


Figure 9. liquid-dsp Chrome performance profiling.

When receiving, the CPU is the main concern for us. As when receiving and decoding, the CPU will be handling multiple layers of undoing the interpolation. The main observation from Figure 5, is that the bars (representing CPU processing time) are not next to each other, indicating that the CPU does have sufficient time

to process the message before the next message comes in. Although, the CPU is maxed out when it is decoding such a message.

4. Usability Test Results

Usability testing was limited due to the ongoing development of core functionality. Preliminary demonstrations of Liam's FSK implementation with Scott Jensen provided valuable feedback on the overall architecture and integration with provided radio hardware. Due to the nature of building a library, documentation also became highly necessary, and was one of the primary focuses after ensuring software reliability. The team achieved what was set out in the problem statement.

XI. Future Work

The project had successfully transitioned to a solution utilizing liquid-dsp, and delivered a feature complete demo supporting multiple modem types and text encoding schemes. Remaining work focuses on refining the user experience and expanding platform support. Building a visually appealing user interface for the PWA is still something that could be addressed, however, that is beyond the scope of the project after consulting the client.

Yet, most critically, testing the liquid-dsp demo over radio was unable to be accomplished. The team encountered an issue where audio output capture from the walkie talkies (provided by the client) inexplicably stopped working. This issue was not attributed to the team's work. Furthermore, the client did not provide additional hardware for testing, and the team lacked access to other radio hardware capable of performing testing ourselves. This area of the project would be an excellent place for future developers and potentially a future field session team, to work on.

Extensive testing across a wider range of web browsers versions and devices (such as older IOS versions and SBCs such as a Raspberry Pi) is also outstanding work. Further optimization of the liquid-dsp WASM module through experimentation with different compilation flags is warranted. Additional components such as bundling the separate FFTW library is a potential area for performance gains, though it is a separate C project that was not investigated during this development cycle. The liquid-dsp WASM module is modular, allowing for relatively straightforward updates based on upstream changes; a developer would need to update the commit lock on the git submodule and then verify the continued functionality of the C shim layer.

For continued development and maintenance, comprehensive documentation has been left in place. All code is documented with inline comments, and a README.md and DEVELOPMENT.md file have been provided to guide future contributors. Several additional features could be added to enhance the developed product. Implementing a more visually appealing UI would improve the user experience.

XII. Lessons Learned

Throughout this project, the team gained significant insights into the practical challenges of real-world radio communication and software implementation. Initial exploration revealed inherent performance limitations when relying on standard JavaScript threading for audio processing tasks. This led to the successful adoption of WASM as a means to achieve the necessary speed and efficiency. While WASM proved effective, challenges were encountered with its implementation, such as ensuring messages are not missed and/or duplicated when handling two buffers, highlighting the need for careful consideration of the architecture of the project.

In addition, the team was in unanimous agreement that we would have greatly benefited from having a person with Electrical Engineering experience as a resource for the project. Whether on the team or not, having someone to consult for the understanding of the underlying digital signal processing required for this project

would have greatly accelerated the timeline of what we could accomplish. Comedically, Dr. Bahar (the team's advisor) informed us near the end of the semester of a senior Capstone project spearheaded by a group of Electrical Engineers who accomplished a similar project, however they wished they had someone with a Computer Science background on their team (go figure).

The project underscored the importance of adaptable planning and strategic pivots. Initial attempts at porting demodulation code directly into WASM encountered roadblocks, necessitating a shift to the liquid-dsp library. This decision, informed by a careful evaluation of licensing, features, and error detection capabilities, ultimately proved crucial to delivering a robust and functional solution. Regular communication with Scott Jensen facilitated valuable feedback throughout the process, ensuring the final product aligned with user needs. Finally, the team recognizes the critical role of meticulous testing, iterative refinement, and a modular, extensible architecture in building a resilient and future-proof communication library.

XIII. Acknowledgments

Team Alamosa would like to thank our client, Scott Jensen of Alamosa Antenna for guidance, wisdom, and incredible flexibility throughout the project. On behalf of the team, we thank you for such an engaging project. Team Alamosa would also like to thank our advisor Dr. Bahar for keeping us on a good trajectory throughout the project lifecycle, and providing insight on resources within Mines, and giving us a nudge in the right direction when it came to deadlines.

1. Attributions

- This project builds off of the liquid-dsp library which has a MIT license. [1]
- This project uses some code snippets from the Baudot.js library for the Baudot encoding, which is licensed under MIT. [3]
- The Radio icon in the example PWA integration is from Wikimedia Commons. [5]

2. References

[1] "Family Radio Service (FRS)," FCC. [Online]. Available: https://www.fcc.gov/wireless/bureau-divisions/mobility-division/family-radio-service-frs

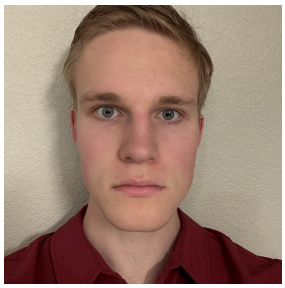
[2] J. D. Gaeddert, *jgaeddert/liquid-dsp*. (Dec. 10, 2025). C. Accessed: Dec. 10, 2025. [Online]. Available: https://github.com/jgaeddert/liquid-dsp

[3] ZEN♡, *zenoamaro/baudot.js*. (Aug. 10, 2021). TypeScript. Accessed: Dec. 11, 2025. [Online]. Available: https://github.com/zenoamaro/baudot.js

[4] "Flexible Framing Structure (flexframe)." Accessed: Dec. 11, 2025. [Online]. Available: https://liquidsdr.org/doc/flexframe/

[5] दुलालशर्मा प्रितम, *English: An Online Radio*. 2016. Accessed: Dec. 12, 2025. [Online]. Available: https://commons.wikimedia.org/wiki/File:Radio_Takchang.jpg

XIII. Team Profile

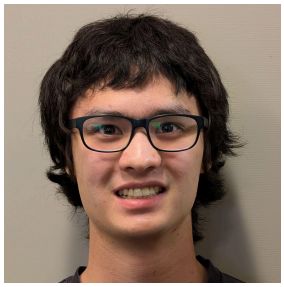


Dawson David
Op Mode Researcher

I worked on the initial testing of various other solutions that existed. From that I managed to take inspiration and created the core code used for our final project. I used my prior experience working on a typescript library to flush it out.

Micah Bird
Client Liaison

Along with being the person of contact for the client, I worked on collecting metrics for existing solutions and testing the final product. Outside of school I like to self-host services and work on my homelab.



Liam Kerrigan
Algorithm Researcher

I developed the initial proof of concept prototype. Additionally, I did research by taking a digital signal processing course to help develop this project. This knowledge allowed me to recognize the need for a Hilbert transform in the final product.

Nate Moore
Project Manager
I was the primary performance analyst, making sure the program is going to run as expected on all expected hardware configurations. I was also able to incorporate my previous knowledge on radios and Amateur Radio to help guide this project.

Appendix A – Key Terms

Term	Definition
PWA	**Progressive Web App**; a website that behaves like a native app.
GMRS	**General Mobile Radio Service**; a licensed land-mobile FM radio service in the US used for short-distance two-way communication.
PSK	**Phase Shift Keying**; a digital modulation process which conveys data by changing the phase of a constant frequency reference signal.
Bell 202	A modem standard specifying Frequency Shift Keying (FSK) tones.
FSK	**Frequency Shift Keying**; a modulation scheme in which digital information is transmitted through discrete frequency changes of a carrier signal.
FEC	**Forward Error Correction**; a mechanism that allows for corrupted data to be corrected.
FCC	**Federal Communications Commission**; the U.S. government agency responsible for regulating interstate and international communications by radio, television, wire, satellite, and cable.
IoT	**Internet of Things**; a network of physical objects embedded with sensors and software for the purpose of connecting and exchanging data over the internet.
OFDM	**Orthogonal Frequency-Division Multiplexing**; a method of encoding digital data on multiple carrier frequencies, widely used in modern Wi-Fi and LTE standards.
GMSK	**Gaussian Minimum Shift Keying**; a continuous-phase modulation scheme that uses a Gaussian filter to smooth signal transitions and reduce bandwidth usage.
WASM	**WebAssembly**; a portable binary instruction format used to run high-performance code, increasingly used in embedded systems and IoT firmware.
C shim	A small library or piece of code that transparently intercepts API calls.