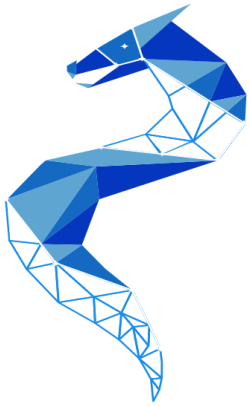


OpenSense v3

An open source, customizable sensor platform

Authors:

Brandon Collins	James Eyler	Daniel Gonzalez	Noah Wilfond
Computer Engineering	*Computer Engineering*	*Computer Engineering*	*Electrical Engineering* *Optical Engineering*

Sponsors:
Design of Resilient Architectures for Computing (DRACO) Lab
Dr. Mike Borowczak

Advisors:
Dr. Aravinda Kar
Dr. Saleem Sahawneh

Reviewers:

Dr. Mike Borowczak	ECE
Dr. Mark Maddox	ECE
Dr. Midya Parto	CREOL

Table of Contents

Chapter 1 - Executive Summary....1
Chapter 2 - Project Description....2
2.1 - Motivation and Background:....2
2.1.1 - Design Features and Functionality....2
2.1.2 - Current Commercial Technologies and Previous Versions....4
2.2.1 - Goals....6
2.2.2 - Objectives....7
2.2.3 - House of Quality Diagram....8
2.3 - Specifications....8
2.4 System Block Diagrams....10
Chapter 3 - Research and Investigation....15
3.1.1 - Component Selection....15
3.1.2 - Micro Controller Unit (MCU)....16
3.1.3 - I²C Mux....17
3.1.4 - Level Shifter....19
3.1.5 - Channel Switch....20
3.1.6 - Analog to Digital Converter (ADC)....22
3.1.7 USB Port....23
3.1.8 Micro-SD....24
3.2.0 Power Supply....25
3.2.1 Battery Considerations....25
3.2.2 Power Distribution....25
3.2.3 Battery Types....27
3.2.4 Battery Selection....27
3.2.5 Voltage Regulation....29
3.2.6 Reflective vs. Transmissive Optics....31
3.2.7 Diffraction gratings....31
3.2.8 Mirrors....34
3.2.9 Sensors....36
3.2.10 Motors....39
3.3.1 Comparison of Software Technology....41
3.3.2 End-to-end architecture: tasks, timing, and composability....43
3.3.3 Sensor interfaces at 10 Hz: I²C throughput, address conflicts, and ADC realities....44
3.3.4 Storage path: SDMMC versus SPI, FatFs through VFS, and CSV strategy....45
3.3.5 Networking and SPA hosting on the device....46
3.3.6 Configuration management, durability, and field ergonomics....47
3.3.7 From theory to implementation: how the pieces fit....47
3.3.8 Charting and UI behavior: pushing only what changes....48

3.3.9 Throughput and latency budget: why 10 Hz is easy to hit..... 49
3.3.10 I²C multiplexing and mixed buses in practice..... 49
3.3.11 ADC calibration strategy and implications for data quality..... 50
3.3.12 Wi-Fi SoftAP behavior and multi-client dashboards..... 50
3.3.13 Why C/C++ usually wins here and when MicroPython is still right..... 50
3.3.14 Testability and reliability: how to prove it works before fielding..... 51
3.3.15 Security posture appropriate for a field instrument..... 51
3.3.16 What "good" looks like for the SPA itself..... 51
3.3.17 Sensor library: interchangeable ports, auto-detection, and robust boot-time discovery..... 52
3.4 Wireless Communication..... 58
3.4.1 Wifi vs Bluetooth Comparison..... 58
3.4.2 WiFi..... 59
3.4.3 Bluetooth..... 59
Chapter 4 - Standards and Design Constraints..... 59
4.1 Industrial Standards..... 59
4.1.1 I²C Protocol Standards..... 59
4.1.2 PCB Design Standards..... 61
4.2 Practical Constraints..... 64
4.2.1 Time..... 64
4.2.2 Money..... 65
4.2.3 Manufacturability..... 65
4.2.4 Remaining Constraints..... 66
Chapter 5 - Application of ChatGPT and other similar platforms..... 68
5.1 - An Overview..... 68
5.2 - Limitations..... 69
5.3 - The Pros of AI..... 71
5.4 - The Cons..... 72
5.5 - Example of AI Use..... 74
5.5.1 Case Study 1..... 74
5.5.2 Case Study 2..... 74
5.5.3 Case Study 3..... 75
5.5.4 Case Study 4..... 75
5.5.5 Case Study 5..... 75
5.5.6 Case Study 6..... 75
Chapter 6 - Hardware design..... 76
6.1 - Power Design Schematics..... 76
Power Management..... 77
3.3V Regulator..... 79
6.2 - Path Switching..... 80
TCA9548A I²C 8-Channel Multiplexer..... 81

TS3A24157 Electronic Switch.....83
TXS0102 I²C Level Translator..... 84
TPS2110A Power Multiplexer..... 86
MCP3221 Analog to Digital Converter.....87
6.3 - Data Storage..... 89
6.4 Sensor Schematics.....92
6.5 Monochrometer Electronics..... 93
6.5.1 Transimpedance Amplifier..... 93
6.5.2 Motor Controller..... 95
6.6 Optical Spectrometer Design Calculations..... 96
Chapter 7 - Software design..... 99
7.1 Software Architecture - Subsystem/Component Level..... 99
7.1.1 WebSocket Implementation Architecture.....99
7.2 Main Program Flow.....101
7.3 Use Case Diagram..... 102
7.4 State Diagram - Sampler Task.....103
7.5 Class Diagram - Driver Abstraction..... 105
7.7 Data Transfer Diagram..... 107
7.8 Data Structures.....108
7.9 Implementation Considerations..... 110
Chapter 8 - System Fabrication / Prototype Construction.....112
8.1 PCB..... 112
8.2 Sensors.....116
Chapter 9 - System Testing and Evaluation..... 119
9.1 - Optoelectronic Feasibility.....119
9.2 - Power Testing..... 121
9.3 - Software Testing..... 127
Chapter 10 - Administrative Content..... 128
10.1 - Budget..... 128
10.2 - Bill of Materials (Tentative)..... 128
10.3 - Project Milestones..... 129
Chapter 11 - Conclusion.....131
Appendix A.....134
Appendix D - Software Code..... 139
Appendix E - ChatGPT Prompts and Outcomes.....172

List of Tables

Table 2.1: System specification table: 8
Table 2.2: Component specification table 9
Table 3.1: MCU Comparison..... 16

Table 3.2: I²C Multiplexer…. 18
Table 3.3: Level Shifter…. 19
Table 3.4: Channel Switches…. 20
Table 3.5: ADC Comparison…. 22
Table 3.6: Comparison of USB Format…. 23
Table 3.7: MicroSD Comparison…. 24
Table 3.8: Power Distribution….25
Table 3.9: Comparison of Battery Types…. 28
Table 3.10: Comparison of Low Power Voltage Regulators…. 29
Table 3.11: Comparison of Grating Technologies…. 32
Table 3.12: Grating Component Comparison…. 33
Table 3.13: Comparison of Mirror Technologies…. 34
Table 3.14: Mirror Component Selection…. 35
Table 3.15: Sensor Technology Comparison…. 37
Table 3.16: Sensor Component Selection…. 38
Table 3.17: Motor Technology Comparison…. 39
Table 3.18: Motor Component Selection…. 41
Table 3.19: Software Technologies…. 43
Table 3.20: Wifi Vs Bluetooth…. 58
Table 10.1: Bill of Materials ….128
Table 10.2: Project Milestones …. 129

List of Figures

Figure 2.1: Previous Iteration of OpenSense ….5
Figure 2.2: House of Quality Diagram…….. 8
Figure 2.3: Hardware Block Diagram …….... 11
Figure 2.4: Software Block Diagram…….... 12
Figure 2.5: GUI Front Page design …….... 13
Figure 2.6: GUI Back Page design…… 13
Figure 2.7: Front View of Prototype…… 14
Figure 2.8: Back View of Prototype…… 14
Figure 2.9: Czerny-Turner spectrometer diagram…… 15
Figure 3.1: Example of code ……... 56
Figure 4.1: I²C Diagram …….... 60
Figure 4.2: I²C Specs chart ……... 60
Figure 4.3: Data message composition …….... 61
Figure 4.4: I²C Process …….... 61
Figure 4.5: IPC-2221 standards …….... 62
Figure 4.6: Minimum clearance values for are board to pass voltages …….... 64
Figure 6.1: USB-C connection (USB-to-UART + Battery Charger) …….. 77

Figure 6.2: USB-C Sense and 5.0V Boost Regulator 78
Figure 6.3: 3.3V Regulator Schematic 79
Figure 6.4: TCA9548A 8-Channel I^2C Multiplexer Schematic 82
Figure 6.5: Basic 3.3V Sensor port paths 83
Figure 6.6: TS3A24157 Electronic Single-Pole Double-Throw switch 84
Figure 6.7: TXS0102 logic level translator for I^2C communication 85
Figure 6.8:TPS2110A Power Multiplexer Schematic 86
Figure 6.9: MCP3221 Analog to Digital Converter Schematic 87
Figure 6.10: Patch Switching System Schematic 89
Figure 6.11: DM3AT-SF-PEJM5 microSD card slot schematic 90
Figure 6.12: MCU Schematic (ESP32-WROOM-32E-N4) 91
Figure 6.13: Temperature Sensor Schematic 92
Figure 6.14: Pressure Sensor Schematic 93
Figure 6.15: Transimpedance Amplifier Schematic 95
Figure 6.16: Servo-motor schematic 96
Figure 6.17: Angle vs Wavelength plot 98
Figure 7.1: Software architecture flowchart 100
Figure 7.2: Main program flowchart 101
Figure 7.3: Use case diagram 103
Figure 7.4: State diagram 104
Figure 7.5: Driver Abstraction Layer - Class Diagram 105
Figure 7.6: User Interface conceptual 106
Figure 7.7: Data Transfer Diagram 107
Figure 8.1: 2D MCU Initial Board Layout 113
Figure 8.2: 3D MCU Initial Board Layout 113
Figure 8.3: 2D Power and Regulator Close-up 114
Figure 8.4: 3D Power and Regulators Close-up 115
Figure 8.5: 2D Close-up of MCU 116
Figure 8.6: 3D Close-up of MCU 116
Figure 8.7: 2D Close-up of Temperature Sensor 117
Figure 8.8: 2D Close-up of Pressure Sensor 118
Figure 8.9: 2D Close-up of Monochrometer PCB 119
Figure 9.1: First Spectrometer Prototype 120
Figure 9.2: Second Spectrometer Prototype 121
Figure 9.3: Breadboard connected to power supply without charger 122
Figure 9.4: Output without Charger 123
Figure 9.5: Breadboard connected to power supply with charger plugged in 123
Figure 9.6: Output with charger and power supply 124
Figure 9.7 - Simulated LT1767 regulator circuit 125
Figure 9.8 - Simulated LT1767 regulator results 125
Figure 9.9 - Simulated LT1302 regulator circuit 126

Figure 9.10 - Simulated LT1302 regulator results ……..126

Chapter 1 - Executive Summary

Sensor platforms are systems designed to collect, process, and communicate data from a variety of connected sensors, making them essential tools for data acquisition in both professional and educational settings. However, many commercially available platforms are prohibitively expensive and inaccessible for most projects. In addition, these systems often function as closed “black boxes,” supporting only predefined sensors and offering little transparency into how they operate or how they might be modified. Our goal is to develop a platform that is significantly more affordable while providing the flexibility to incorporate user-defined sensors and custom configurations.

This project builds on two prior iterations of OpenSense developed under DRACO Lab at UCF. Earlier versions demonstrated the promise of a low-cost educational sensor platform but were limited by pre-coded sensor libraries, fixed functionality, and limited customization. OpenSense v3 expands this foundation by introducing mixed-voltage sensor support, a modular switching architecture, integrated analog-to-digital conversion, customizable sensor definitions, SD-card data logging, and a redesigned graphical user interface hosted wirelessly through the onboard MCU. These improvements directly target the platform’s core engineering and educational goals: transparency, usability, cost efficiency, and extensibility.

Our system is built around the ESP32-WROOM MCU, with communication between the microcontroller and the sensors being handled through I2C connections. Our platform will be able to handle up to five simultaneous sensors, with a combination of power multiplexers, level shifters, and channel-select switches to allow each sensor port to operate at 3.3V or 5V. To support analog devices, a built-in ADC converts analog outputs into digital values. A microSD card provides inhouse storage for collected data, allowing the device to be used remotely off of its battery life which is carefully designed to last for at least 2 hours.

On the software side, the platform uses a lightweight, embedded web server hosted directly on the ESP32 to provide a user-friendly, browser-based GUI. This interface allows users to configure sensor settings, manage voltage selections, visualize real-time data through live charts, and download stored measurements. The software architecture emphasizes reliability, ease of integration, consistent data handling, and the ability to add new sensors through parameter-based definitions rather than firmware modification.

To demonstrate the technical versatility of the platform to accept user defined sensors, the team is designing an optical spectrometer. This subsystem showcases the platform’s ability to integrate advanced domain-specific sensors while maintaining access through modular hardware and intuitive UI tools. The spectrometer highlights the platform’s capability to support complex scientific measurements.

This document records the design process for the OpenSense platform, covering the theory behind the system, the technologies that make it possible, and the reasons

specific components were chosen. It also explains the system architecture, hardware schematics, and firmware design. After that, the report discusses the standards and constraints that shaped the project, and includes information on fabrication, prototyping, and related administrative work. Finally, it presents conclusions and suggestions for future improvements, along with appendices that provide technical references and required permissions.

Chapter 2 - Project Description

2.1 - Motivation and Background:

In today's world, many popular sensor platforms are expensive, closed-source, and not easily customizable. While these systems are powerful, their lack of accessibility creates barriers for those who want to explore how these systems are built and applied in everyday life. This challenge is especially significant in K-12 education, where transparency and affordability are critical for sparking curiosity in technology. To address this, DRACO Labs (Design of Resilient Architectures for Computing) introduced OpenSense, an open-source, customizable, and low-cost sensor platform designed for scalability and educational use. Our project builds on their foundation by expanding the platform's capabilities and developing new features that improve usability and broaden the range of supported sensors.

As students, our team has experienced firsthand the challenges of learning about circuit design and embedded systems with limited, hard-to-access resources. With OpenSense, we want to make these learning materials and tools more approachable and widely available, especially for K-12 educators and students. Our goal is to create a platform that not only lowers technical barriers but also encourages exploration, creativity, and confidence in working with technology. A key part of this vision is enabling users to select their own sensors, making the platform highly customizable to different applications and learning goals. To showcase this flexibility, we chose to develop a custom optical spectrometer as a demonstration sensor. This allows us to highlight how complex measurements can be performed in an adaptable way, reinforcing OpenSense's commitment to accessibility and user-driven design.

Ultimately, our contribution continues the mission of DRACO labs, which is directed by Dr. Mike Borowczak within the University of Central Florida's Electrical and Computer Engineering Department. By improving usability and expanding functionality, we hope to carry forward and expand their vision of making open, educational sensor systems accessible to the next generation of innovators - making their ability to understand, experiment with, and design technology more approachable and engaging.

2.1.1 - Design Features and Functionality

Open Sense is thought to be an open-source sensor platform for a variety of uses, with the ability to customize sensors that you want to read data from, as well as a

user-friendly Graphic User Interface (GUI) that will simplify the addition of new sensors. Users are looking for a simplified way to incorporate sensors of their own, without having to directly change the code of the platform. Furthermore, this prevents users from potentially corrupting the coding of the board. This was also something that was noted by Dr. Mike *(Project Sponsor / Mentor),* who felt needed to be improved on. We plan to meet these requests with our design by including the following functions and features in the platform:

1. **WebServer / User-Friendly GUI:** The platform will be able to display sensor data information through a wirelessly connected WebServer that hosts a User-Friendly GUI. The user will use this GUI to add their sensor, select voltages, turn on sensor ports, view chart data from sensors, and have the ability to download data from a CSV file located in the SD of the board - all in a user-friendly, focused design that makes the platform more inviting for users new to technology.

2. **Long Period Data Collection:** The platform will store continuous amounts of data onto an on-board SD card - *.csv file format*. This allows the user to take in multiple points of data over an extended period of time to then access later. This data will be available to the user for download through the Web Server.

3. **Customizable Sensors:** The platform's versatility is an important part of the market requirements, so it is essential for our board. To increase the variety of sensors that can be used, the platform will take in parameters through the Web Server that will allow the user to implement their own sensor while avoiding the hassle of having to change the main code and reducing the possibility of breaking the code.

4. **Optical Spectrometer Integration:** To further showcase this degree of customizability, our team will design and construct a custom optical spectrometer to utilize with the platform as a demonstration sensor. Spectrometers are typically expensive and inaccessible to beginners, yet they are powerful tools for analyzing light. By including a spectrometer, we bring optics to students and educators and highlight how advanced measurements can be made transparent and customizable with the OpenSense platform.

5. **Sensor Voltage Switching:** Different sensors may require different voltages (3.3V or 5.0V) in order to operate and give back accurate readings. In order to increase customizability, this ability to access both voltages is necessary. Our team's version of the Open Sense will account for this by allowing the user to input which voltage they desire to use for their sensor, and the board will adjust the Vcc of the corresponding sensor port accordingly.

6. **I²C and Analog Compatibility:** Three of the ports will be outfitted with Analog to Digital Converters (ADC) in order to increase customizability. This will allow the user to not be limited to I²C and have the option to include sensors that are 3-pin

analog sensors - i.e., Light Sensors, Humidity Sensors, Temperature Sensors, etc.

7. **System Life-Span:** The platform will be able to be used in both wired and wireless settings in terms of power supply, allowing for more versatility in educational applications.

2.1.2 - Current Commercial Technologies and Previous Versions

There are multiple companies, such as Vernier and Globisen, that offer products such as this one commercially. Vernier offers support for 10 different sensors and is considered to be very user-friendly and educational. This, however, comes with the trade-off of their product being very expensive, as one package can cost anywhere from around $300 - $1400 for a package with a sensor platform [1]. This makes obtaining one of these sensors less practical for lower-level education classrooms with less funding - this is where the need for the board to be low-cost comes into design specifications.

Globisen, on the other hand, is a lot more affordable than Vernier [2]. However, it is also a lot less flexible, as it doesn't allow you to swap out or modify the sensors. This greatly diminishes the versatility of the product and doesn't allow users to implement or add a sensor that is niche to their use. Open Sense, as a concept, tries to solve this problem and add more customization to the platform. Our plan is to make it easy for others to modify our project and set up a custom sensor which would allow for more flexibility and data collection depending on the user's needs.

Another example of limited accessibility in commercial technologies is spectroscopy. Companies such as Ocean Optics and Thorlabs sell compact spectrometers that are widely used in labs, but these devices are often expensive, ranging from several hundred to several thousand dollars depending on resolution and features [3]. Other low-cost options, such as Science Buddies DIY spectrometers use phone cameras or repurposed webcameras to make it more affordable and accessible, but lack the ability to be integrated with a broader sensor platform and are limited in resolution and robustness [4]. By incorporating a spectrometer into OpenSense, we adapt the concept into a transparent, customizable system designed for broader access.

Since the project's initial creation in 2014 - 2015, there have been two prior iterations of the OpenSense platform. This was done at UCF with Dr. Mike Borowczak's oversight. In 2015, the project was able to support 13 sensors, display values to a Google Chrome App, and cost about $60.32 for volume production. Their project was also battery-powered and included micro SD card storage. Obviously, this iteration was powerful, being able to support 13 different sensors, but it lacked the customizability for other sensors that were not pre-coded onto the MCU. In 2024, the project was improved upon by another UCF Senior Design group that managed to bring the cost down to $35 and chose to focus on supporting 3 simultaneous sensors in detail. Similar to the version before it, this project was only able to support sensors that were already coded into the MCU [5], [6].

The previous iterations of this project prioritized keeping the cost low over the customizability of the platform. Adding a new sensor would require changing the code, which can cause unintended issues and prevent the device from working as intended. This made the board more complex in terms of adding your own sensors and went against the goal of creating an accessible, user-friendly device for educational and project-based purposes. This necessity for changing the code to add a new sensor is key when it comes to the versatility and marketability of the platform. The goal of Open Sense, according to Dr. Mike, is for the platform to be as user-friendly as possible, encouraging those who do not have a significant background in Engineering or technology to be able to utilize the device with a sensor of their own choice. The platform's design is intended to be an educational tool for K-12 students and educators - hence the importance of the customizability and accessibility, all while maintaining a low cost for volume production. This is where our team hopes to improve upon previous iterations, focusing on the implementation of universal I²C and Analog sensor compatibility - allowing users to use a wider variety of sensors instead of pre-programmed sensors.

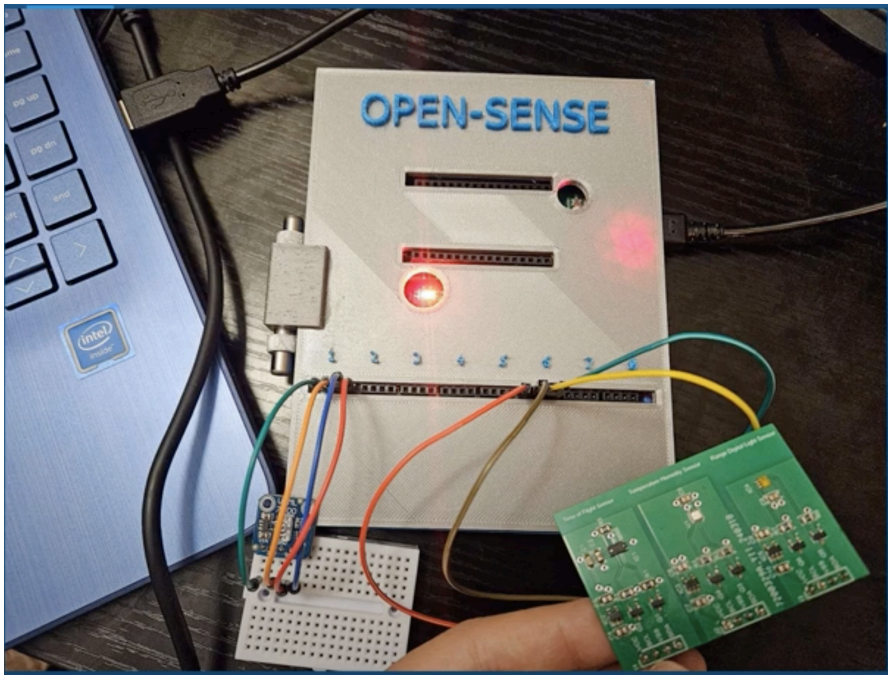


Figure 2.1: *Version 2 of the OpenSense Project*

2.2 - Objectives and Goals

The purpose of this project is to create a low-cost tunable method of collecting sensor inputs in a transparent and reliable manner. The platform is designed to prioritize customizability and user accessibility. The following section outlines the specific goals and objectives our team established for this purpose, split into three categories. Basic goals are the baseline necessities for the project, as detailed by D.R.A.C.O. lab, whereas advanced goals are more complicated but still doable under the scope of the project. Stretch goals would require increasing the scope and necessitate that the basic

and advanced goals be completed before these are considered. Similarly, our basic objectives have been assigned the target values that we find realistic. The advanced objectives increase those target values by raising the expectations and introducing added complexity. Finally, the stretch objectives would challenge the team to push the boundaries of cost and usability.

2.2.1 - Goals

Basic Goals:

- Design a data logger capable of acquiring and recording outputs from multiple sensors simultaneously.
- Be able to store data from sensors onto an SD card, inside of a .csv file.
- Develop a user-configurable conversion interface to present the information in user-readable units.
- Maintain a minimum of 1 hour of onboard storage for collected data.
- Publish the system and associated documentation under an open-source license to enable community access, collaboration, and further development..
- Design an optical spectrometer module to showcase integration of the sensor platform within advanced projects

Advanced Goals:

- Integrate a power supply subsystem capable of sustaining continuous operation for a minimum of two hours.
- Utilize a microcontroller to control the data acquisition subsystem and distribute power to connected components as required.
- Create a housing that enables system portability by minimizing weight, reducing volume, and ensuring robustness during transport and deployment.
- Implement the system for straightforward setup to minimize deployment time and required user training.
- Develop a software framework that supports user-defined sensor models, increasing customization and expanding supported sensor options.
- Demonstrate the platform's flexibility by using it to process data from an optical spectrometer and resolve distinct spectral peaks.
- Incorporate GUI support for visualizing data in real time, enabling intuitive interpretation for educational purposes
- Achieve a minimum wavelength coverage of 400-700 nm (visible spectrum) to ensure relevance for demonstration purposes
- Ensure a straightforward setup to achieve an easy overall user experience.

Stretch Goals:

- Enable wireless data transmission for automated upload to external devices or storage systems.
- Enable the USB port to power the system and also recharge batteries (*Lithium*).
- Incorporate modular sensor ports to support "plug-and-play" integration and replacement of multiple sensor types.
- Implement spectrometer data visualization tools into the GUI to make optics more approachable for students and educators.

- Extend the wavelength of the spectrometer subsystem to 300-800 nm to measure into the near-UV or near-IR to broaden applications.
- Develop a cross-platform interface for the visualization of data to ensure broad accessibility and usability.

2.2.2 - Objectives

Basic Objectives:

- Implement a data acquisition system capable of sampling each connected sensor at a minimum rate of 10 Hz.
- Support simultaneous communication between a central MCU and at least 5 sensors over I²C.
- Establish a baseline battery life that will guarantee a minimum of 2 hours of battery life.
- The system ensures that sensor readings achieve a minimum accuracy of 85% in relaying correct values.
- Ensure that the device is equipped with at least one hour of data storage, independent of the sensor type implemented.
- Implement a diffraction-grating-based spectrometer with a single detector to verify integration of custom sensors with the platform and produce basic spectra for educational demonstration.
- Adjust the angle of the diffraction grating with a servo motor to scan the detected wavelength range

Advanced Objectives:

- Establish a baseline battery life that will guarantee a minimum of 6 hours of battery life.
- Extend the data-acquisition system to include an intuitive interface that simplifies onboarding new sensors.
- Data collection will be able to store up to 4 continuous hours of data.
- The system ensures that sensor readings achieve a minimum accuracy of 90% in relaying correct values.
- We ensure that the device is equipped with at least two hours of data storage, independent of the sensor type implemented.
- Utilize a photodiode within an optical spectrometer setup to demonstrate the customizability of the sensor platform for use in the collection and interpretation of data within advanced projects.
- Evaluate resolution performance by measuring and distinguishing multiple spectral lines from calibration sources.
- Employ reflective optics to ensure broadband compatibility and minimize chromatic aberrations

Stretch Objectives:

- Establish a baseline battery life that will guarantee a minimum of 24 hours of battery life.
- Data collection will be able to store up to 8 continuous hours of data.

- Incorporate a custom library that will allow users to input parameters of their own I²C sensor - effectively making the platform functional with any sensor that utilizes I²C communication Protocols.
- The system ensures that sensor readings achieve a minimum accuracy of 95% in relaying correct values.
- We ensure that the device is equipped with at least four hours of data storage, independent of the sensor type implemented.
- Develop interactive plotting features to display real-time spectral data.
- Select optical components to ensure compatibility with UV/IR operation.

2.2.3 - House of Quality Diagram

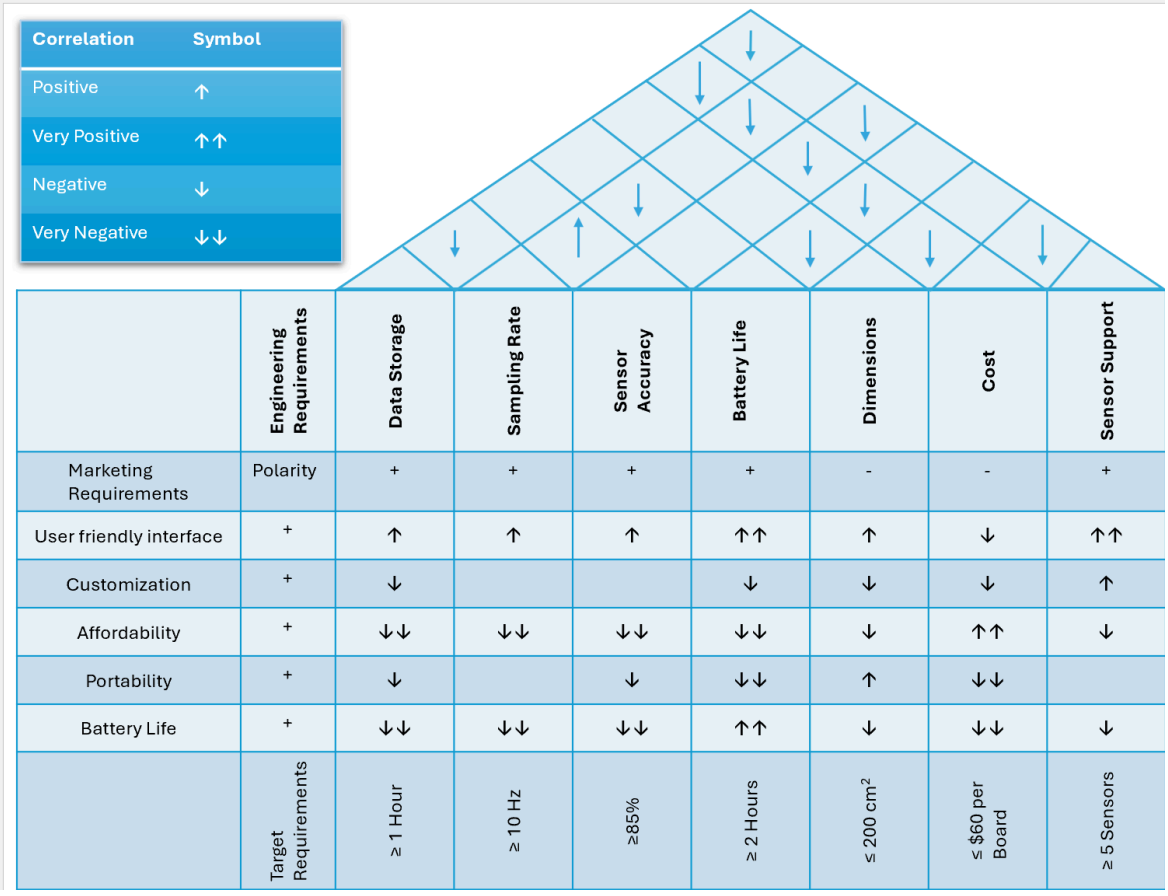


Figure 2.2: *House of quality diagram*

2.3 - Specifications

Table 2.1 - System specifications table

System Specification Table		
Specifications	**Description**	**Target Values**

System Specification Table		
Unit Cost	The platform should have a low cost per unit for volume	< $60
System Lifespan	The device should be guaranteed to support data collection over a set period of time	≥ 2 hours
Sampling Rate	The platform should be able to record multiple points of data over time	At least 10 Hz per sensor
Data Storage	The device should have enough storage to gather significant data for analysis	≥ 1 hour storage
Sensor Accuracy	The sensor should collect data accurately	Accuracy ≥ 85%
PCB Dimensions	The PCB should be made to fit in a compact/portable housing	≤ $200cm^2$
Sensor Support	The platform should be able to support 5 or more sensors	≥ 5 Senors
Spectrometer wavelength range	The spectrometer should be able to plot a spectrum consisting of at least the visible wavelength as well as some UV wavelengths.	300 - 800 nm
Spectral Resolution	To be able to distinctly measure different wavelengths, a spectral resolution of at least 5 nm must be met.	≤ 5 nm

Table 2.2 - Component Specifications table

Component Specifications Table			
Component	**Parameter**	**Description**	**Target Values**
Battery	Discharge Time	The battery should be able to power *X* amount of sensors over the desired time period	≥ 2 Hours
MicroSD	Storage	Data should be able to be stored on a MicroSD card - done so that files are not too bulky, but also store the necessary sensor information.	≤ 1 Gb
MCU	Response Time	MCU should have a good response time in regards to user input from the Web Server, as well as incoming sensor data	< 1 second

Component Specifications Table			
Sensors	Accuracy	Sensors will be able to gather data within a certain margin of error	±5 units of measure
LEDs	Status Accuracy	LEDs should be able to accurately display to the user the current status of the platform (active sensor ports, active voltage, heartbeat of the device)	95%
Power Mux	Vin Switch Accuracy	Power Mux is used to switch Vin from 5.0V and 3.3V - this needs to be done with accuracy to get proper readings from the sensors	95%
Diffraction Grating	Groove density	The groove spacing of the diffraction grating was chosen to provide enough separation between wavelengths without making the optical system excessively large.	1200 lines / mm
	Pitch	The pitch is the spacing between lines of the grating, which determines its diffractive properties.	0.83333 μm
Spherical Mirrors	Focal length	The mirrors should have a focal length of 25 mm .	25 mm
	Reflectivity	High mirror reflectivity is required for efficient transmission of light through the system	0.95
	Diameter	The mirror diameter was chosen to collect the appropriate amount of light.	0.5 inch
Photodiode	Wavelength Range	In order to make measurements across the entire visible spectrum, the photodiode must be able to detect light over a wide range of wavelengths.	300 - 800 nm
	Bias Voltage	To integrate with the platform, the photodiode must be used with a bias voltage of 5V.	5V
Stepper motor	Angular displacement	The angle of each step in the stepper motor determines how fine the resolution of the monochromator will be.	0.1 deg

2.4 System Block Diagrams

2.4.1 Hardware Block Diagram:

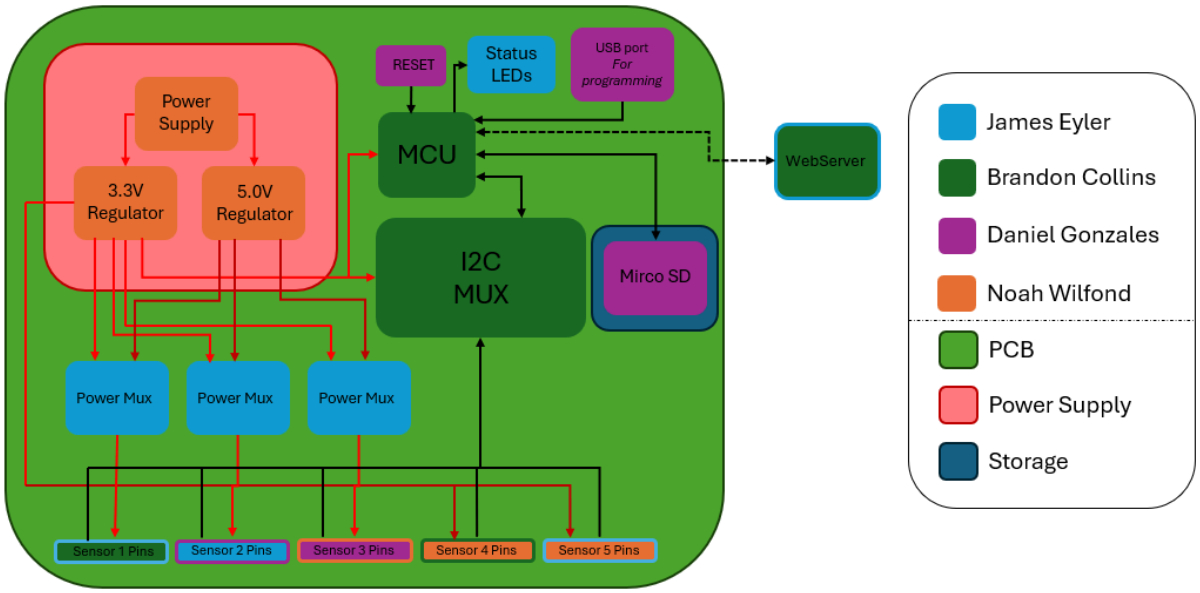


Figure 2.3: *Hardware Block Diagram*

Figure 2.3 illustrates the hardware block diagram for the system. At the center of the design is the MCU, which is the main component for data collection, processing, and communication. The power supply connects to two voltage regulators: a 3.3-volt regulator for the MCU and the lower voltage peripherals, and a 5-volt regulator for components requiring a higher operating voltage. In addition, the MCU relies on a USB port for programming and debugging during development. A reset button and LED indicators are also connected to the MCU to provide user interaction and system feedback. Each sensor will have a green LED to act as a visual aid for sensor operational status.

Sensor integration is achieved through a combination of power multiplexers and an I²C multiplexer. All sensors connect to the MCU through an I^2C multiplexer that allows the system to transmit data over multiple sensor ports without the need to switch addresses on the I^2C bus. Channels 1 - 3 will have their VCC pin (reference voltage pin) connected directly to the 3.3V regulator so that these sensor ports can be used at a 3.3V logic level. The remaining ports (4 and 5) will have their VCC pin run through a channel switching system that allows the user to switch between 3.3 and 5.0V logic levels in order to allow the platform to be compatible with 5.0V logic levels. This channel is a series of switches, translators and a multiplexer that allows the channel switch to be executed by the user from the WebServer.

The MCU also controls communication with other peripherals in addition to the sensors. A microSD module is included for local data logging, enabling sensor values to be stored directly on the system. A USB Type-C port for programming and charging the on-unit battery. The MCU we chose offers wireless connectivity to act as an access point for the web server connection between the board and the client.

2.4.2 Software Block Diagram

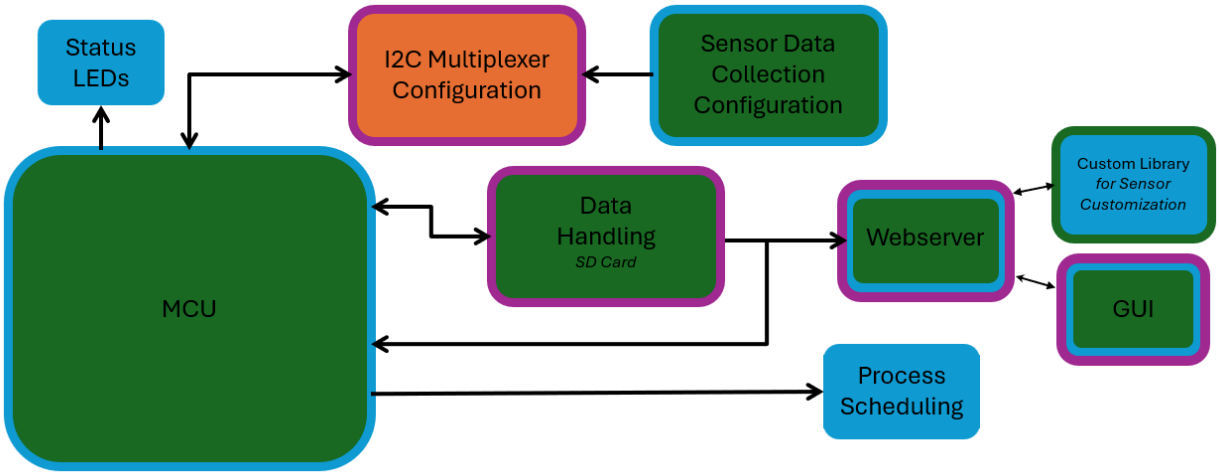


Figure 2.4: *Software block diagram - uses the same legend as hardware block*

Figure 2.4 portrays a proposed software block diagram that establishes a clear and efficient data flow between the system's sensors and the web server GUI. Sensor modules will collect raw values and transmit them to the MCU. The MCU will serve as an access point and forward the data to the data handler, which is responsible for organizing and transmitting the information to the custom web server. To improve system reliability and reduce complexity, the communication pathway uses a direct Wi-Fi connection between the MCU and the server, eliminating an outside factor like an intermediary router. Once the server receives the data, it is processed through a custom sensor library and integrated into the graphical user interface (GUI). The GUI enables users to configure sensor parameters and visualize sensor outputs in real time. This architecture prioritizes reliability, ease of integration, and user accessibility, aligning with our project's goals.

2.4.3 - GUI Design

Our GUI is designed with the main focus of user versatility and customization, built in a way that will allow users to do a lot more with their own sensors than in previous versions. Our GUI offers several important features for customizability, such as the ability to select their sensor's compatible voltage and, most importantly, the ability for the user to input their own sensor's parameters in order to add their sensor to the platform without the need to access the main code. *Figure 2.5* shows an early-stage diagram of what the GUI will look like and also highlights the functionality of the interface.

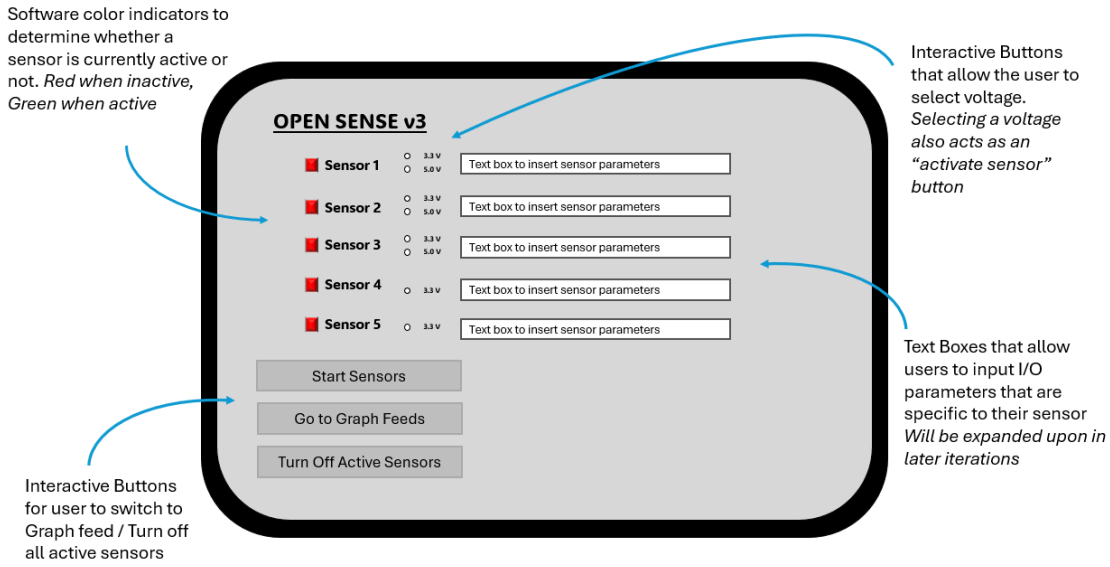


Figure 2.5: GUI Front Page Design

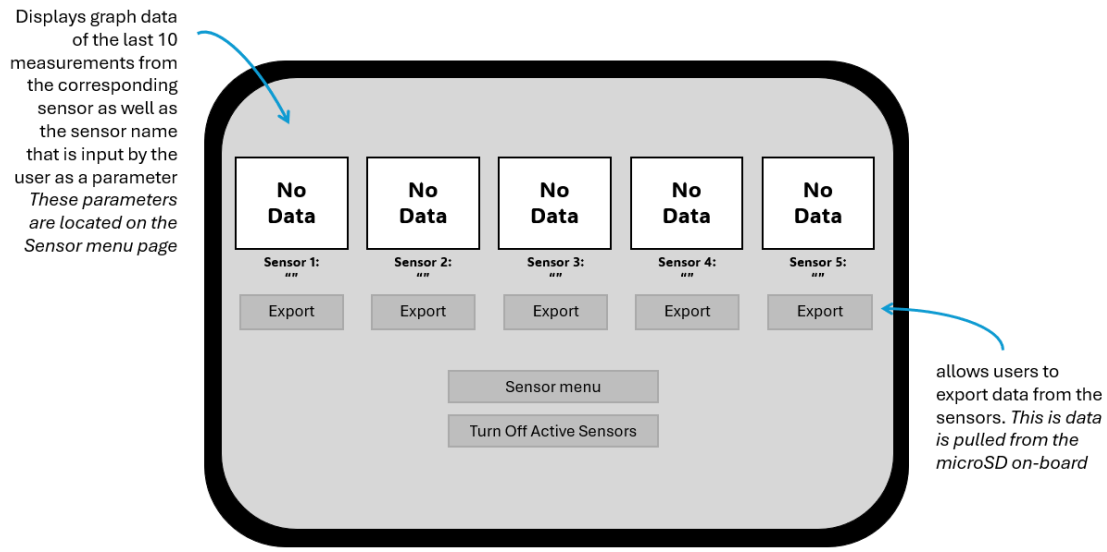


Figure 2.6: GUI Graph Page Design

2.4.4 - Prototype Illustration

Our Prototype illustration highlights the main features that our project offers. There are five available sensor ports, three of which vary in I²C and analog sensor support. In addition, there is a port for programming and charging the device, an On/Off switch, and a MicroSD card port. *Figure 2.7 and 2.8*: The 3D model below is an early rendition of what our finished product will look like.

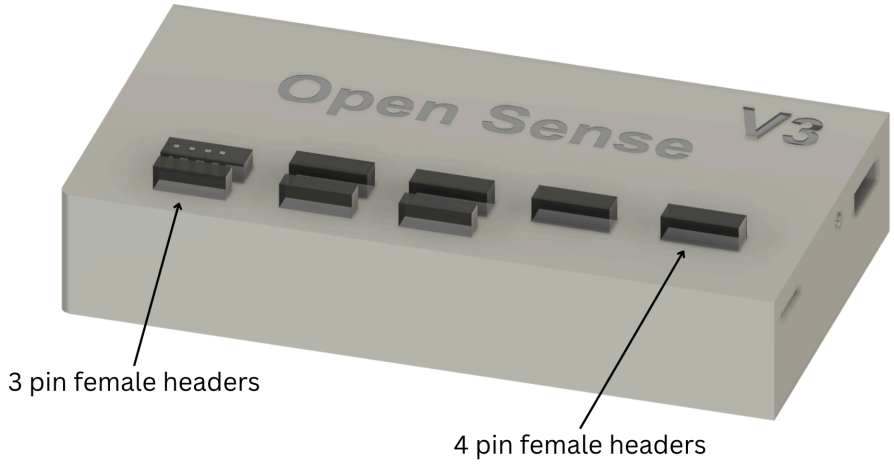


Figure 2.7: Front view of prototype

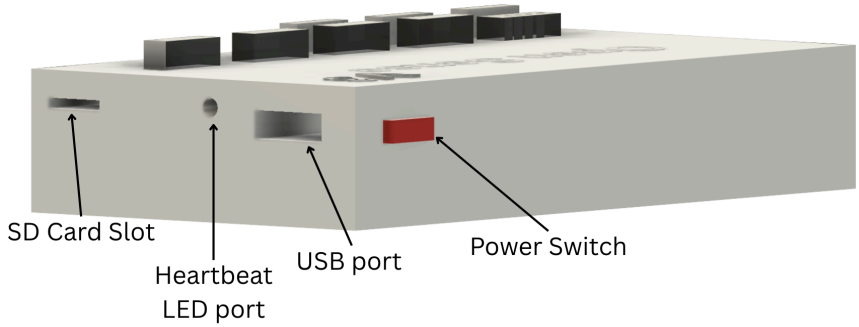


Figure 2.8: Back view of prototype

2.4.5 - Custom Optical Sensor Design

In addition to the other sensors that will be used to demonstrate the functionality of our platform, our group wanted to include a custom-built sensor to showcase the customisability that is central to our goals. For this, we have decided to incorporate our own custom sensor - expanding the requested light detector into a spectrometer to measure the intensity of light over with respect to its wavelength. The design our team has chosen for this spectrometer is the Czerny–Turner spectrometer design due to its effectiveness over a large wavelength span with no chromatic aberrations [7]. This spectrometer uses two mirrors and a diffraction grating. Light entering the aperture is collimated by the first mirror, then dispersed into its wavelength components by the grating. The second mirror focuses these separated wavelengths onto the array detector, where our platform collects and processes the data. This is shown in *Figure 2.9*.

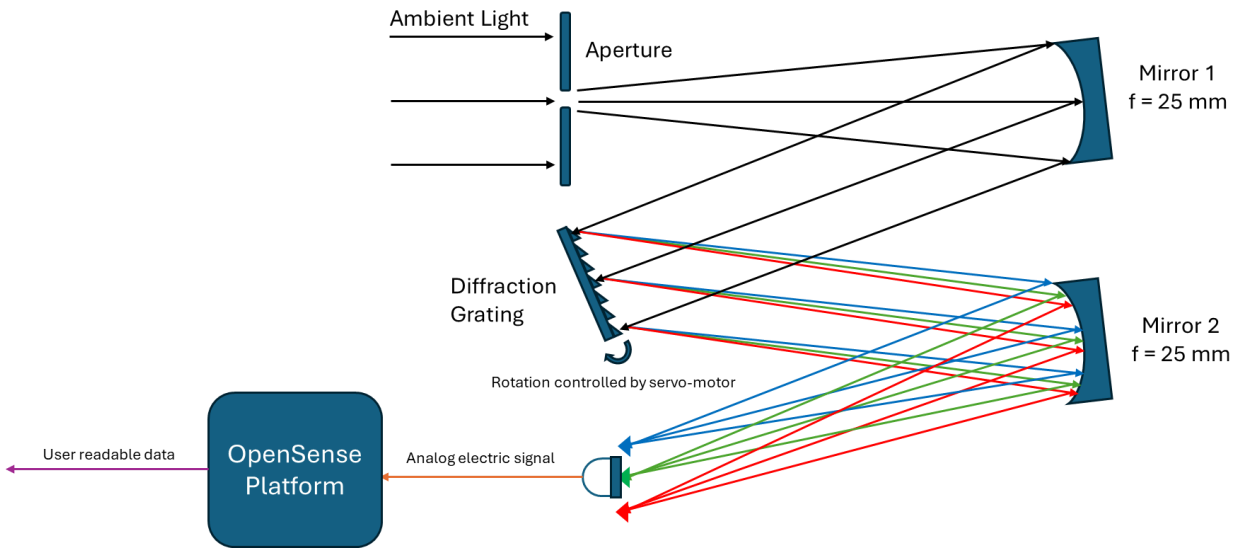


Figure 2.9: *Czerny-Turner spectrometer diagram.*

Chapter 3 - Research and Investigation

3.1.1 - Component Selection

When designing the Open Sense platform, careful consideration must be given to selecting components that balance performance, efficiency and versatility. Each part of the system must work together to achieve not only the engineering goals of low power consumption, accurate signal handling and reliable communication, but also the marketing objectives of user customizability, scalability and long-term durability. The goal is to create a platform that can serve a wide range of users – from developers testing new sensor configurations to engineers integrating the platform into larger systems–without requiring significant hardware redesigns.

To achieve this, the Open Sense platform is designed around several key subsystems that collectively enable multiple-sensor operation, mixed-voltage compatibility, and support for both analog and digital sensors. For instance, the design includes a communication management component that allows multiple I²C sensors to operate simultaneously, as well as a switching mechanism that dynamically routes signals and power between voltage domains. Additionally, a level translation system ensures that the sensors operating at different logic levels can safely communicate with the main controller, while a power control element manages voltages selection to extend battery life and improve system reliability. Finally, to support analog-output sensors, an integrated analog-to-digital converter (ADC) converts analog signals into digital values compatible with the I²C bus, enabling seamless interoperability across sensor types.

Together, these elements form the foundation of a flexible and intelligent sensing architecture capable of supporting diverse user requirements and future hardware expansion. The following sections will discuss each of these core components in detail,

explaining how their selection and integration will help the Open Sense platform meet its technical specifications and align with the broader goals of adaptability, efficiency and long-term usability.

3.1.2 - Micro Controller Unit (MCU)

The MCU is one of the most critical components of our device. It acts as the central processing unit that manages nearly every operational function within the system. It controls the collection of data from the connected sensors, processes user input through the graphical user interface (GUI), and coordinates the timing and communication between modules inside of the system. Knowing its importance, the selection of the best MCU for our platform requires careful consideration of design factors that directly influence performance, scalability, and overall system reliability.There were several specifications that were identified as essential in determining the most practical MCU for the Open Sense Platform. These specifications include processing performance, I/O and peripheral support, connectivity features, power requirements, for factor compatibility, cost and robustness for future expansion. Each of these specifications ensures that the selected device can handle the demands of the system while maintaining flexibility for future design iterations. In Figure 3.1 below, three different MCU options are presented along with their key specifications that were considered during the evaluation process.

Table 3.1 - MCU Comparison

Model Name	ESP32-WROOM-1	STM32UO	MSP430
Manufacturer	Espressif	STMirco	Texas Instruments (TI)
Core / Speed	Dual Core → 32-Bit Speed → Up to 240MHz	Single Core → 32-Bit Speed → 56MHz	Single Core → 16-Bit Speed → 16MHz
Memory	384 kB ROM 512 kB SRAM	256 kB Flash Memory 512 kB SRAM	15.5 kB FRAM
Connectivity	Wifi / Bluetooth	Nothing Listed	Not Listed
Peripherials	36 GPIOs 4 Strapping GPIOs	39 I/Os, most 5V tolerant	60 I/Os on 64 pin-package
Support I²C?	Yes	Yes	Yes
Languages	ESP-IDF, Arduino	C, C++	C
Cost	$6.56	$3.42	$3.14

After evaluating the three MCU options, we determined that the ESP32-WROOM-1 is the best option for the Open Sense Platform. This microcontroller offers a combination of processing power, flexibility and integrated connectivity that is practical for the requirement specifications of our system. Its dual-core 32-bit processor gives a significant advantage over single-core alternatives, particularly in terms of parallel task

execution and real-time responsiveness. This dual-core architecture is especially helpful when implementing FreeRTOS tasks, which our platform uses to manage multiple concurrent tasks such as sensor polling, data logging, wireless communication and user-interface updates. Therefore, the ESP32-WROOM-1 allows us to utilize these cores to distribute tasks efficiently to provide consistent, deterministic performance - which is essential for systems that depend on accurate timing and synchronization among several processes.

Additionally, the ESP32-WROOM's operating speed of up to 240 MHz allows for high-frequency sampling and faster data handling across multiple sensors (a necessary function of the sensor platform) as well as handling other concurrent tasks within the system (storing data to SD card, updating webserver GUI, etc.), allowing for a more deterministic behavior from the system. This also helps our system avoid latency in sensor communication. The MCU also has an embedded 40 MHz crystal oscillator which eliminates the need for an external clock source. One of the most significant factors in our decision was the connectivity of the microcontroller. The ESP32-WROOM was the only microcontroller of the three that includes built-in WiFi and Bluetooth capabilities, which are essential for our platform's wireless communication specifications. Using an on-board antenna, the ESP32-WROOM gives us the ability to connect to local networks or mobile devices allows for remote data monitoring (through the webserver) as well as interactions from the user without additional external communication modules in the system. The ESP32-WROOM's built in features (antenna, oscillator, etc.) not only simplifies our hardware design development but also reduces the need for additional components, lowering our costs and minimizing the physical footprint on the board along with. Because of this, the ESP32-WROOM was the most practical choice for the Open Sense Platform.

3.1.3 - I²C Mux

In order to meet the design requirement of supporting up to five sensors operating simultaneously, the platform needs to be able to extend the I²C communication capability of the MCU. The ESP32's integrated I²C bus can only handle a limited number of devices before running into issues with address conflicts and data timing issues. In order to achieve this, an I²C multiplexer (MUX) is incorporated into the design. This component enables the system to connect multiple sensors to the same I²C bus by selectively switching between them. Each sensor is then able to communicate with the MCU individually, allowing data to be transmitted through a single set of I²C pins without interference. This approach not only simplifies our hardware design and software control, but also reduces wiring complexity and potential errors during development and testing.

When selecting the right I²C multiplexer, several parameters need to be evaluated to verify that the MUX is practical for our platform's design objectives. These include the number of available channels (how many sensors can be supported), the channel selection mechanism (how the MUX switches between channels), the delay/speed of

switching, the operating voltage, and the compatible voltage logic levels. Figure 3.2 shows three different multiplexers and their specification comparisons.

Table 3.2 - I²C Multiplexer			
Model Name	TCA9548APWR	PCA9547PW	LTC4305IGN
Manufacturer	Texas Instruments (TI)	NXP USA Inc.	Analog Devices Inc.
Operating Voltage	1.65V to 5.5V	2.3V to 5.5V	2.27 to 5.5V
Voltage (Logic)	1.8V, 2.5V, 3.3V, 5.0V	1.8V, 2.5V, 3.3V, 5.0V	2.2V,2.5V,3.3V,5.0V
Number of Channels	8	8	2
Clock Frequency Range	0 to 400kHz	0 to 400kHz	0 to 400kHz
Switching Mechanism	I²C Bus	I²C Bus	I²C Bus
Cost	$1.32	$1.95	$5.01

After evaluating the specifications of each multiplexer, we saw that many of the devices offered similar functional capabilities. However the TCA9548APWR from Texas Instruments seemed to be the best option for our needs in the Open Sense Platform. We determined this from the device's technical performance, flexibility, and economic efficiency. Among the three multiplexers, the TCA9548APWR had the widest operating voltage range (1.65V to 5.5V), allowing for it to hold the same operating voltage as the ESP32's operating voltage which ensures that the I²C logic used throughout the system is consistent with the MCU.The TCA9548APWR has eight independent channels that provide a significant advantage over the LTC4305IGN, which only supports two. Having a larger amount of channels inside of the single MUX is advantageous for our sensor platform's needs in terms of accommodating multiple sensors at once. Having this many channels also adds to the robustness of our system - this is because it allows us to add sensors and/or more multiplexers without the need for an extensive redesign of our hardware. Furthermore, this multiplexer allows us to switch channels without having additional wiring for a switching function; the TCA9548APWR allows us to use the I²C pins in order to switch between channels. Allowing the functionality of the multiplexer to be achieved without using external/addition switching circuitry or hardware. Lastly, we come to the economic specification requirement. The TCA9548APWR is the least expensive component out of the three listed in Figure 3.2, which will be beneficial in keeping the cost per unit under the board low.

3.1.4 - Level Shifter

In the open sense platform, I²C is the main method of communication between the sensors and the MCU. So in order for this standard of communication the voltage level logic is important for precise measurements. In order to comply with this constraint, we need to design our sensor inputs (where users plug in the sensors) to communicate back to the multiplexer at whatever voltage logic level the multiplexer uses - we plan to have the I²C multiplexer at a voltage level of 3.3V, due to the fact that most sensors operate or can be operated at a 3.3V VCC. Knowing this we also want to give users the option to use sensors that require a 5.0V VCC - we came up with the solution of using a level shifter, in order to drop the 5.0V logic from the sensor to a 3.3V logic when it reaches the multiplexer. Without proper level shifting, these components can experience communication errors or even permanent damage due to overvoltage. Therefore, selecting an efficient and reliable level shifter is critical to ensure signal integrity, system safety, and consistent operation across all communication lines.

When comparing our options, key parameters to consider include voltage range compatibility, data rate, direction (unidirectional or bidirectional), propagation delay and current consumption. These factors not only influence our performance in terms of data transfer between components but also our power consumption performance.

Table 3.3 - Level Shifter

Model Name	TXS0102DCUR	74LVC1T45GW	TCA9416
Manufacturer	Texas Instruments (TI)	Nexperia USA Inc.	Texas Instruments (TI)
V_{CC}(A) Range	1.65V-3.6V	1.2V - 5.5V	1.08V - 3.6V
V_{CC}(B) Range	2.3V-5.5V	1.2V - 5.5V	1.08V - 3.6V
Number of Channels	2	2	2
Directionality	auto-bidirectional	bidirectional(non-auto)	auto-bidirectional
Transfer Type	Open-drain	Push-pull	Open-drain
T_{Prop}(ns) *A-to-B*	5.4	5.1	23
T_{Prop}(ns) *B-to-A*	5.3	8.2	23
Data Rate	2 Mbps	420 Mbps	1 MHz (*fastest*)
Price per unit	$0.56	$0.29	$0.56

After evaluating several two channel level shifters, the TXS0102DCUR was selected as the most suitable component for the Open Sense platform. This is because this device

is able to meet our design requirements of supporting automatic bidirectional behavior, handle open-drain signals properly in order to work with our I²C setup, and introduce minimal delay to maintain I²C timing integrity. The device also accepts a 1.65V-3.6V V_{CCA} and 2.3V-5.5V V_{CCB} which is great for our translation needs (3.3V ←→ 5.0V). The TXS0102 also allows for the integrated circuit (IC) to be put into a High Impedance (High-Z) state when it is not in use in order to reduce power usage while not in use. According to the datasheet, when in High-Z mode the IC uses a few micro-amps of leakage current (max ± 1µA, *usually much lower*) which is helpful when our specifications require an extensive system lifespan when battery-operated.

In comparison, the 74LVC1T45GW, while compact and equal in cost to the TXS0102, uses a push-pull architecture and requires a direction-control pin, which makes it less suitable for shared I²C lines and increases design complexity and not as practical as the TSX0102. The TCA9416, another Texas Instruments device, functions as a dedicated I²C buffer and translator but introduces a higher 23 ns propagation delay, which is much longer than the TXS0102 and a 1MHz data rate. Where it becomes impractical for our purposes is its available voltage inputs for V_{CCA} and V_{CCB} – it only provides 1.08V-3.6V input range which would work with our 3.3V input but not with our 5.0V input. Because of this, TXS0102DCUR is the best choice out of the three compared in Figure 3.3 and provides the best overall balance of performance, simplicity and reliability.

3.1.5 - Channel Switch

One of the primary goals of the Open Sense platform is to provide users with as much customizability and flexibility as possible when integrating sensors. To support this objective, the design includes an electrically controlled voltage-selections system that allows users to determine which supply voltage (V_{CC}) their connected sensor will use. This feature is implemented through a combination of a power multiplexer and a signal routing switch, both of which are controlled by the system's microcontrol unit. When activated, these components enable users to switch between two distinct operating modes: a 3.3V direct-connection mode, where the I²C bus and power line connect directly to the sensor pins, and a 5.0V translated mode, where both power and communication signals are routed through a level-shifting circuit to ensure safe and reliable operation at the high voltage level.

Table 3.4 - Channel Switches

Model Name	TS3A24157	ADG733	TC4W53FU
Manufacturer	Texas Instruments (TI)	Analog Device Inc.	Toshiba
Operating Voltage	1.65V-3.6V	1.2V - 5.5V	-0.5V-20V
On-State Resistance	0.65Ω Max	2.5Ω Max	1.08V - 3.6V
Switching type	SPDT	SPDT	DNS

Table 3.4 - Channel Switches			
Number of Channels	2	3	2
Directionality	bidirectional	bidirectional	bidirectional
Delay	40 ns - t_{ON}/30 ns - t_{OFF}	40 ns -t_{ON}/12 ns - t_{OFF}	550ns - t_{ON}/160ns - t_{OFF}
Price per unit	$0.71	$5.88	$0.27

This approach provides significant flexibility for end users, as it allows the same sensor port to support a wide variety of external devices without requiring hardware modifications. For example, sensors designed for 3.3V logic can interface directly with the MCU, while 5.0V sensors can be accommodated through the automatic translation path. By integrating this configurable switch network, Open Sense not only expands compatibility across different sensor types but also reduces development time for future users who may need to adapt the platform to different voltage standards or custom sensor modules. Figure 3.4 shows the different electronic switches that were compared.

After evaluating several analog channel switches, the TS3A24157 from Texas Instruments was selected as the most suitable component for our design. The devices considered were analyzed based on their operating voltage range, on-state resistance, directionality, switch delay, and cost. Since the switch is responsible for routing I²C communication between the two system paths, it was essential to choose a device that provided low on-resistance, fast switching, and bidirectional signal flow without introducing noise or signal degradation.

The TS3A24157 offers an ideal balance between performance, simplicity and cost. It operates from 1.64V to 3.6V, matching our 3.3V control logic, and provides an exceptionally low on-state resistance of only 0.65Ω, minimizing voltage drops across the switched path. The device features break-before-make SPDT switching, which prevents short-circuit conditions during transitions - an important consideration when switching between different voltage logic levels. Its fast response time (40 ns t_{ON} /30 ns t_{OFF}) ensures that bus communication remains stable even during dynamic voltage selection events. In addition, the dual-channel configuration conveniently supports both SDA and SCL lines within a single IC package, simplifying PCB layout and control. In comparison, the ADG733 from Analog Devices provides broader operating voltage support (1.2V-5.5V) and includes an additional channel, but its higher on-resistance (2.5Ω) and significantly higher cost make it less practical for our platform’s application. The TC4W53FU from Toshiba is an economical alternative; however it exhibits slower switching speeds (550 ns t_{ON}) and a more limited voltage specification, which could affect timing reliability in high-speed I²C communication. Considering the combination of electrical performance, switching precision, and cost efficiency, the TS3A24157 provides the most balanced and dependable choice for the platform's configurable I²C channel switching system.

3.1.6 - Analog to Digital Converter (ADC)

To increase the overall flexibility and adaptability of the Open Sense Platform, our design focuses on enabling support for both I²C sensors and analog-output sensors through a single, unified interface. This feature allows users to connect a wide range of external devices without requiring any hardware modifications or additional interface modules. To achieve this, each sensor port on the platform provides two compatible connection types: a standard 4-pin I²C header for sensors that already communicate using digital I²C protocols, and a 3-pin analog header for sensors that output a single analog voltage corresponding to a measured parameter, such as a light sensor, temperature sensor, humidity sensors, or gas concentration sensor. In order to integrate analog sensors into the digital communication bus, the 3-pin analog connector is internally linked to an Analog-to-Digital Converter (ADC). This converter samples the analog signal from the sensors and translates it into a corresponding digital value, which is then transmitted through the existing SDA and SCL lines of the I²C bus. By doing so, the platform effectively allows both analog and digital sensors to communicate over a common I²C bus line making data processing from the sensors more simple and fast on the user end, enhancing customizability by making the platform compatible with a broader range of sensor types, while also improving scalability for suture sensor integrations and research applications. Figure 3.5 shows the ADCs that were compared for use in the platform.

Table 3.5 - ADC

Model Name	MCP3221	MCP3021	ADC101C027CIMK
Manufacturer	Mircochip Technology	Mircochip Technology	Texas Instruments (TI)
Number of Bits	12	10	10
Sampling Rate	22.3Kbps	22.3Kbps	188.9Kbps
Number of Inputs	1	1	1
Communication Speeds	I²C Std: 100KHz I²C Fst: 400KHz	I²C Std: 100KHz I²C Fst: 400KHz	I²C Std: 100KHz I²C Fst: 400KHz High Speed M: 3.4MHz
Voltage Supply	2.7V-5.5V	2.7V-5.5V	2.7V-5.5V
Current Usage (ON)	300µA	250µA	When 3.3V → 0.59mA When 5.0V → 1.2mA
Price per unit	$1.91	$1.41	$1.91

After evaluating multiple single-channel I²C compatible analog-to-digital converters (ADC), the MCP3021 from Microchip Technology was selected as the most suitable component for our design. Each device was compared based on resolution, sampling rate, communication compatibility, voltage supply range, and overall power efficiency.

The ADC plays an important role in translating analog sensor outputs into digital data for communication over the I²C bus, so the chosen component needed to balance performance, accuracy, speed, and power consumption while remaining cost-effective. The MCP3021 offers a 10-bit resolution with a sampling rate of 22.3Kbps, providing more than sufficient precision for the types of analog sensors expected to be used on the Open Sense platform. It operates at low current consumption (250μA when active) and supports both standard (100kHz) and fast (400kHz) I²C communication modes, ensuring compatibility with the existing bus configuration. Although the MCP3221 provides a higher 12-bit resolution, it consumes slightly more current (300μA) and comes at a higher cost, making it less ideal for a low-power, cost-sensitive embedded platform - especially since the 10-bit precision (from the MCP3021) is sufficient for our platform's needs. The ADC101C021 from Texas Instruments offers the highway sampling rate of 188.9kbps and additional high-speed I²C modes (up to 3.4MHz), but these features exceed the operational requirements of our system and introduce unnecessary complexity. It also draws significantly more current (up to 1.2mA at 5.0V) and matches the MCP3221 in price, reducing its suitability for a power-efficient platform. Overall, the MCP3021 provides the best combination of performance, low power consumption, and cost effectiveness, which aligns with the design goals of the platform while simultaneously maintaining simplicity in implementation.

3.1.7 USB Port

Table 3.6 - Comparison of USB formats

Feature	USB-C (USB2.0 sink)	Micro-USB (Type-B)	USB-A	2.1 mm Barrel Jack
Data	USB 2.0 (D+ / D-)	USB 2.0	Host side only	None
Power	5 v, default 500-900 mA	5V and <1 A	5V from host	Depends on adapter
Mating Cycles	~10k	~10k but weaker	~1.5k	~5k
Mechanical Robustness	High	Moderate	Bulky	Very robust
BOM complexity	Low	Very low	Very low	Very low
Board Area	Small-medium	Small	Large	Small
User experience	Best	Ok	Awkward	N/A
Cost	$1.19	~$0.30-$0.90	~$0.30-$1	$0.25-$0.50
Pros	Reversible, commonly used, scalable to P D later	Cheap	Simple for hosts	Simple and high current is possible

Table 3.6 - Comparison of USB formats				
Cons	Must implement CC and have current limits	More fragile and non-reversible	Very bulky, mostly for hosts so can't program	Needs second connector

For the sensor we will need a USB port to use when powering it for testing as well as to charge it. In order to determine the best type to use it is best to analyze their strengths and weaknesses to determine which one would be the most efficient. For the sensor, USB-C, Micro-USB, USB-A, and a 2.1mm Barrel Jack have been considered.
[8]

As seen above, the Barrel Jack is probably the least efficient option because a lot of it depends on the adapter and connectors and is also only used for power even if it does have some benefits in terms of size. The next best option would then be the USB-A however it also has its flaws as we wouldn't be able to program our PCB with it since it's intended for host devices. The next best option would be the Micro-USB as it is common and inexpensive and works well for what it is needed for. However the Micro-USB is a lot less durable and easy to damage so it is not worth the risk. Overall the best option seems to be the USB-C which is the more modern USB port and has a higher insertion retention than the Micro-USB.
[9]

3.1.8 Micro-SD

The design requires a Micro-SD connector to be implemented as the Micro-SD is necessary for the device to store the data it has captured over hours of use. Both the Hirose DM3AT-SF-PEJM5 and the Molex 47219-2001 were considered as the most popular types.

Table 3.7 - MicroSd Comparison			
Feature	Hirose DM3AT-SD-PJjM5	Molex 47219-2001	Diligent PMod 410-380
Mechanism	Spring Eject	Friction	Friction
Mount/Profile	Top-mount, low profile	Top-mount, slightly taller	Connector on small board
Card Detect	Yes (switch)	Often No	Yes
Shell / Ground Tabs	Full shield, multiple tabs	Shielded	Shielded metal shell
Insertion Cycles	~10k	~5-10k	~10k cycles

Table 3.7 - MicroSd Comparison			
Ease of use	Very easy; auto eject	Needs fingernail which can damage card	Easy, no auto eject
EMI / ESD Robustness	Good (solid shell)	Good	Good with shielded socket but longer traces
Board Space	Small-medium	Small	Slightly larger
Cost	~$2.50-$3.20	$1.80-$2.80	$7.55

After comparing the 3, the Diligent model was chosen as it is more practical to use and implement onto our design due to the PCB layout and it is also easy to use compared to the Molex option.

3.2.0 Power Supply

In order to power our design, we looked into different power supplies and batteries to determine which one would be best suited for our sensor to power the components while also being portable and safe to use.

3.2.1 Battery Considerations

After analyzing the various components we have selected, we determined that it would be ideal for our source to be able to recharge while also meeting the criteria in which we cannot use lithium polymer batteries because of the safety hazard and risk of causing a dangerous fire. The rechargeability helps as our sensor is designed to be used many times to gather data for as long as we are able to have it run so being able to recharge the batteries is a lot better than having to acquire a new one each time. Another criteria when choosing the battery was making sure it wasn't overly bulky as our sensor is designed to be portable and compact so a big battery would affect our design and where we place the components.

3.2.2 Power Distribution

Table 3.8 - Power Distribution					
Component	Voltage	Current Draw	Qty	Power Consumption (W)	Total (W)
ESP32-WROOM-32E (MCU)	3.3	~80–250 mA (active Wi-Fi range)	1	0.264-0.825	0.825
TCA9548A	3.3	~5 mA	1.	0.0165	0.0165

Table 3.8 - Power Distribution					
I²C MUX					
ADS1115 (ADC→ I²C)	3.3	~150 µA (cont. conv.)	1	0,0005	0.50
Power Multiplexors (ports)	3.3	10 x 3	3	0.033	0.10
MicroSD Socket (DM3AT) + Card	3.3	~50–100 mA (R/W active)	1	0.165–0.330	0.33
HX711 (Load-cell amp)	5.0	1.5	1	0.0075	0.0075
Indicator LEDs (3×Red, 5×Yellow, 5×Green)	3.3 / 5.0	~2 mA each → 26 mA total	13	.005	.065
Array Sensor TSL1401 (2156-TSL1401-ND)	5.0	~15–30 mA (clk/int set)	1	0.075-.150	.150
USB-C Port (sensor ports) (12401610E4 #2ACT-ND)	3.3 / 5.0	20-40 each	5	0.08-0.20	1.00
Optical Spectrometer	5.0	80-120	1	0.40-0.20	0.60
GUI / Wi-Fi Interface	3.3	40	1	0.132	0.132
Total System power Consumption					~2.4 W (avg) / 4.0W (Peak)

We created the table below to outline the power distribution of the system to see how much voltage would be required from our selected battery and how it would be distributed across the different components. This allowed us to better determine the right constraints for the battery as it has to be powerful enough to power all of these while also not needing to be overly powerful, so a value of around 5-10V was settled upon.

3.2.3 Battery Types

When choosing a battery, NiMH, Alkaline, LiFePO, NiCd,Sealed Lead-Acid, Lithium-Polymer, and Lithium Ion were all taken into consideration. Each of them had various pros and cons which are discussed below.

- **Lithium-Ion:** Lithium-Ion batteries are one of the most popular choices in batteries for modern electronics as they offer high energy density, fast charging, and compactibility. Lithium-Ion would have been the ideal choice in a battery due to how easily rechargeable and reusable they are and their high energy output. Lithium Ion batteries are known for being a bit dangerous but with proper casing and handling they could be the best option.
- **Lithium-Polymer (LiPo):** Lithium-polymer batteries are a subset of Lithium-Ion batteries that instead make use of polymer electrodes rather than a liquid one. They can be made a lot smaller than a Lithium-Ion battery and so are more widespread in smaller projects. However they also have the same safety risk that Lithium-Ion batteries do in that they can catch fire or explode and therefore were also not selected.
- **Alkaline AA:** These are one of the most common and affordable batteries and so they were taken into consideration. However upon looking into them further we determined that them not being rechargeable as well as having less effective energy made them inefficient to use.
- **Lead-Acid:** Lead-Acid batteries are a very reliable battery type due to their rechargeability and output. However they are also incredibly bulky most of the time and since the sensor is designed to be portable they would not be a good fit to use despite their benefits.
- **Nickel Cadmium (NiCd):** NiCd batteries are very cost-effective batteries and are typically used in small and low power applications. However they are also known for having discharge issues and losing a lot of charge when they are not being used. They also have some environmental concerns as they contain cadmium which is considered toxic and therefore would not be good for a project like this that is supposed to be an educational sensor.
- **Nickel Metal Hydride (NiMH):** NiMH batteries are somewhat similar to NiCD however they do not contain Cadmium so they are more environmentally friendly. They do tend to discharge when not in use as well however they do provide the wattage we need to power the sensor and are also rechargeable. Their portability also makes them a good fit to incorporate into the sensor without messing with the design.

[10] [11] [12]

3.2.4 Battery Selection

After looking into the various advantages and disadvantages of the batteries above, we made our selection. The Lithium-Ion and LiPO were obviously the most efficient ones however the LiPO's safety concerns made it banned from being used so they were not seriously considered. Alkaline batteries were very cheap however having to dispose of

them each time made them less suited for a sensor such as ours designed to be used a lot for gathering data as you would have to constantly buy new ones. The Lead-Acid battery is a very good choice due to its efficient output and rechargeability however it was determined to be too bulky. The NiCd batteries were a solid contender as they would suit the purposes of the sensor however the Cadmium contained in them makes them too much of a hazard when disposing of them. The NiMh batteries, while not having some of the best memory and discharge rates was a good choice however in the end it just couldn't compare to the Lithium-Ion battery.

Therefore Lithium-Ion batteries were selected to be used as they provide enough power to our sensor and are portable and rechargeable. It also contains straightforward buck conversion to 5V / 3.3V which we will make use of. They are also relatively inexpensive to buy as well so it makes it more cost effective to use. They can also be charged with the USB-C charger along with the BQ2407 battery charger that is a one time purchase so no need to have to buy more batteries like the Alkaline ones.
[13]

Table 3.9 - Comparison of Battery Types

Feature	Lithium-Ion	LiPO	Alkaline	Lead Acid	NiCD	NiMH
Specific Energy (Wh/kg)	150-200	140-200	110-160	30-40	40-60	60-120
Memory Effect	No	No	No	No	Yes	Yes
Weight	Light	Very light	Medium	Heavy	Heavy	Medium
Cost	$75+ per pack	$150+ per pack	$3-6 per set but recurring	$20-30	$18-36 plus special disposal	$12-24 per 6 cells plus $20-30 charger
Self Discharge Rate (% per month)	5-10	5-10	~0.2-0.3	3-20	20	30
Typical Pack	2S/3S 18650	2S/3S Pouch	6xAA (9V)	12V 1.3-3 Ah	6xAA (7.2V)	6xAA (7.2V)
Pros	High energy density, no memory effect	High energy density, no memory effect. Small and light	Safe, cheap, and long shelf life	High surge current and low cost	Long life cycle	Inexpensive, compact, and provides enough Voltage

Table 3.9 - Comparison of Battery Types						
Cons	Can be dangerous if no case is used	Very dangerous and not allowed	Not rechargeable	Very heavy and low energy-to-weight ratio	High self discharge and toxic materials	High self-discharge and voltage sag

3.2.5 Voltage Regulation

For this project the Lithium-Ion battery chosen will supply power to the entire sensor so it is important to select and implement a voltage regulator to ensure that all the components receive a stable voltage to operate properly. This is important as supplying the wrong amount of voltage could lead to some components being damaged as they are meant for a smaller voltage.

For the sensor we are in need of 3.3V and a 5.0V regulator to power the MCU sensors. These regulators do not need to have a very high current capability since they aren't powering anything too powerful. Normally we would consider the trusty TSP63805 and TSP63806 chips as they are very commonly used in boards like these however since we were advised against using some TSP components but it's still worth comparing them to the other options as we are in need of a regulator chip that can handle the input of the battery. It would also be ideal to limit the power that it wastes and also reduce the risk of it overheating.

Table 3.10 - Comparison of Low Power Voltage Regulators							
Parameter	AMS1117 (3.3&5.0)	LM2623	LM27402	MP2315	TPS563200	TSP63805	TSP3806
Input Voltage (V)	4.75-12	0.8-1.4	3-20	4.5-24	4.5-17	1.3-5.5	1.3-5.5
Output Voltage (V)	3.3 / 5.0	Adjustable	Adjustable	Adjustable	Adjustable	3.3V or Adjustable	5.0V or Adjustable
Type	Linear	Switching (Boost)	Switching (Buck)	Switching	Switching	Switching	Switching
Max Output Current	1	~2	~20	3	3	~2-2.5	~2-2.5
Efficiency	~50%	~85-90%	~95%	~90%	~92%	~94-96%	~94-96%
Quiescent current	~5mA	~80µA	~35µA	~600µA	~150µA	~10-20µA	~10-20µA

Table 3.10 - Comparison of Low Power Voltage Regulators							
Thermal Performance	High heat	Low Heat	Low Heat	Low Heat	Low Heat	Low heat	Low heat
Pros	Simple and Cheap	Boosts from a low VIN, high efficiency, low IQ	High current capability, wide VIN, high efficiency	Wide VIN and cheap	Cheap, high efficiency, low Iq	One chip, tiny BOM, high efficiency	One chip, tiny BOM, high efficiency
Cons	Wastes power	Boost only	Controller only	Iq is higher	VIN is less than 17V	Slightly higher cost	Slightly higher cost

The ASM1117 is usually considered a straightforward and simple chip to use as it provides an output of either 3.3V or 5.0V and has a maximum current of 1. However when examining the efficiency of these chips, they suffer from very poor efficiency of 50% or lower when using a voltage of 12V or lower. This in turn generates lots of heat which makes the chip not worth pursuing.

The LM2623 is one of the more suitable candidates as we are in need of a Boost to make the voltage more uniform in the PCB and it also has very high efficiency and low heat and IQ making it a very viable candidate.

The LM27402 regulator is a switching regulator that serves as a buck converter that can supply a wide range of current. It is a strong choice due to its higher efficiency over the previous options and its Quiescent efficiency is impressive as well. As a result, it makes it also a very suitable candidate under the condition that we can’t use the TSP3806.

The MP2315 is a lot more efficient as an option as it offers around 90% efficiency and can also give a very wide range out output voltages up to 24V which makes it very versatile. It also can supply up to 3A of current like the others so it is a viable candidate however other than the increase in output range it still falls behind the next option.

The TSP563200 seemed like one of the best voltage regulator to use for the 3.3V and 5.0V rails in the sensor. Like the others it can provide a current up to 3A and also has an adjustable output ranging from 0.76 to 7V. It is also a lot more efficient since its 96% efficiency is much higher than the others. As a result, it does not generate as much heat as the linear ones and therefore is overall better for the sensor. It also has a quiescent current of 150μA which means that it will lose minimal power when idle so that further increases how useful it will be for our sensor.

However at the voltage that the Lithium-Ion battery is applying it is not that suitable compared to the TSP63805 and TSP3806 which work best with a lower input voltage. And they also have a much higher efficiency as well as a lower Quiescent current so it is worth using TSP63805 for 3.3V and TSP63806 for 5V. However since we cannot use

these regulators under the advisory of Dr. Weeks, we will instead go with the LM2623 to serve as a Boost and the LM27402 to serve as a Buck converter. [14] [15]

3.2.6 Reflective vs. Transmissive Optics

To highlight the adaptability of the OpenSense platform, our group will design and integrate a low-cost optical spectrometer as a proof-of-concept sensor module. The selection of components to build this spectrometer requires careful evaluation of their impact onto the system level parameters that influence its performance and educational utility. Among the most important considerations are the spectral bandwidth and resolution. Additional design factors include optical throughput, dispersion characteristics of the diffraction grating, and the detector array specifications. In the following section, we present an overview of the technologies available for spectrometer construction and outline the criteria guiding our design decisions. When it comes to designing an optical system, most systems can be realized in one of two forms which can be considered analogous to each other – transmissive and reflective optics.

Transmissive optics employ transparent elements to refract light as it passes through them. Their advantages include simpler alignment, straightforward focusing, and often lighter, more compact components for small-scale systems. However, because their behavior depends on the material properties of the medium, transmissive optics are prone to chromatic aberration as well as material-dependent aberrations. Correcting these issues typically requires the addition of compensating optical components, which can add complexity and cost.

Reflective optics, on the other hand, direct light by reflection rather than refraction. Since reflection is largely independent of wavelength, these systems are naturally free of chromatic aberration. This makes reflective designs especially well-suited for broadband applications. For our spectrometer, this property makes reflective optics particularly advantageous, as our goal is to capture data spanning a broad range of wavelengths with minimal distortion.

3.2.7 Diffraction gratings

Comparison of grating technologies

Ruled gratings are produced by mechanically engraving a series of parallel lines onto a reflective or transmissive substrate. They have been widely used in spectroscopy due to their high efficiency and relatively straightforward fabrication methods. However, their mechanical nature introduces periodic surface imperfections, which can create unwanted artifacts such as stray light and ghosting in the output spectrum.

Most ruled gratings today come in the form of blazed gratings – a specialized form of ruled grating designed to direct most diffracted light into a specific diffraction order. This is achieved by shaping each groove with a sawtooth profile, which enhances efficiency around a target wavelength range. Blazed gratings are particularly useful in situations where maximizing light collection is crucial. However, their wavelength-specific

optimization can reduce efficiency outside the blazed region, making them less suitable for broadband measurements. [16] [17], [18], [19]

Holographic diffraction gratings are produced using an interference pattern of laser light, rather than being physically ruled with a mechanical tool. This process creates extremely uniform groove spacing, which minimizes scatter and stray light—two issues that can reduce the quality of spectrometer data. Because of this, holographic gratings are often favored in applications requiring high signal-to-noise ratios and precise measurements. They also exhibit lower ghosting compared to ruled gratings, though they can be more costly to manufacture. For our project, their ability to enhance spectral clarity makes them an appealing option, particularly in educational contexts where demonstrating clean and reliable spectra is important. [17], [18], [19]

Table 3.11 - Comparison of grating technologies

Property	Blazed Grating	Holographic Grating
Cost	Moderate	Moderate
Efficiency	Very high (around blaze wavelength)	Moderate
Surface Quality	Moderate	High
Spectral Range	Limited	Broad

Diffraction Grating Part Selection

After evaluating the available grating technologies, we decided to use a holographic grating for the spectrometer design. The majority of ruled gratings that are available are blazed, making the decision between just the blazed and holographic gratings. While blazed gratings can offer higher diffraction efficiency at a lower cost, they are typically optimized for a narrow wavelength range centered around the blaze wavelength. This limitation makes them less suitable for broadband applications. In contrast, holographic gratings provide a smoother spectral response, making them better aligned with the goals of the OpenSense spectrometer.

GH13-12V Holographic Diffraction Grating

The Thorlabs GH13-12V is a reflective holographic diffraction grating with 1200 lines per millimeter, designed for operation in the visible wavelength range (400–700 nm). Its sinusoidal groove pattern minimizes scattered light and ghosting, producing a clean, high-contrast spectrum. The grating's substrate is made from high-quality fused silica, ensuring thermal stability and low autofluorescence. With its excellent uniformity and low stray light, this grating is ideal for broadband educational or research spectrometers where spectral purity is prioritized over maximum efficiency. The GH13-12V strikes a

balance between optical performance and cost, making it well-suited for the OpenSense spectrometer's goal of accessible, accurate spectral analysis. [20]

Edmund optics #43-215

The Edmund Optics #43-215 is a replicated holographic grating featuring 1200 grooves per millimeter, optimized for first-order diffraction in the visible spectrum. It offers low stray light and uniform groove spacing, ensuring smooth spectral output across a broad wavelength range. The ruled area is mounted on a glass substrate, providing durability and ease of alignment in compact optical assemblies. Compared to traditional ruled gratings, the #43-215 delivers superior spectral quality with minimal flare, though its efficiency is somewhat lower at the spectral extremes. Its availability and consistent performance make it a reliable choice for low- to mid-cost spectrometer systems where optical clarity is a higher priority than absolute throughput. [21]

33025FL01-210H

The MKS/Newport 33025FL01-210H is a premium holographic reflection grating with 1200 lines per millimeter, designed for broadband performance from 350–850 nm. It features a high-efficiency sinusoidal groove profile optimized for first-order diffraction, offering excellent suppression of ghosting and secondary orders. The grating is manufactured using precision replication techniques that ensure exceptional groove uniformity and stability. With its enhanced efficiency and reduced stray light, this model is often used in analytical or research-grade spectrometers requiring high fidelity and consistent performance. While more costly than educational-grade options, its superior optical characteristics make it well-suited for demonstrating the upper limits of achievable performance within the OpenSense framework. [22]

Table 3.12 - Grating component selection

Property	**GH13-12V**	**Edmund Optics #43-215**	**33025FL01-210H**
Groove Density	1200 lines/mm	1200 lines/mm	1200 lines/mm
Wavelength Range	400 - 700 nm	400 - 700 nm	350 - 850 nm
Peak Efficiency Average	45% - 65%	63%	50%
Cost	$106.47	$142.04	$155.00

After comparing the available holographic diffraction gratings, the Thorlabs GH13-12V was selected for the OpenSense spectrometer. This grating provides an optimal balance between performance, cost, and availability, aligning well with the project's educational and modular goals. Its 1200 lines per millimeter groove density and broad operating range from 400 to 700 nm make it ideal for capturing visible light spectra with high clarity and minimal stray light. While higher-end options such as the MKS 33025FL01-210H offer slightly greater efficiency, their cost and precision exceed the

needs of this demonstration platform. The GH13-12V's clean spectral output, durability, and affordability make it the most practical and effective choice for achieving reliable, high-quality optical measurements.

3.2.8 Mirrors

Comparison of Mirror Technology

Metal-coated mirrors use thin films of reflective metals such as aluminum, silver, or gold. They offer broad spectral coverage and are relatively simple to manufacture, making them widely used in general-purpose optical systems. Aluminum, for example, provides consistent reflectivity from the ultraviolet through the visible spectrum, while gold is more effective in the infrared. However, metal mirrors typically have lower peak reflectivity than dielectric alternatives, and their coatings are more prone to oxidation and wear over time. Despite these limitations, they remain cost-effective and versatile, making them a strong candidate for our low-cost spectrometer design.

Dielectric mirrors achieve reflectivity by stacking alternating layers of materials with different refractive indices. This structure creates constructive interference for selected wavelengths, enabling extremely high reflectivity within a designed spectral band. Compared to metal mirrors, dielectric coatings can provide superior performance in terms of efficiency and durability, with reduced absorption losses. The trade-off, however, is that dielectric mirrors are typically optimized for a narrower spectral range and can be more expensive. For applications like ours, where broadband coverage is desirable, dielectric mirrors may be less versatile unless designed specifically for the required range. [23], [24]

Table 3.13 - Comparison of mirror technologies

Property	**Metal Mirror**	**Dielectric Mirror**
Cost	Lower	Higher
Reflectivity Range	Broadband	Narroband (designed range)
Efficiency	Good	Excellent (in Range)

Mirror part selection

After comparing both mirror types, our team selected metallic mirrors for the design. Their broad spectral reflectivity and relatively low cost make them well-suited for a broadband system like the OpenSense spectrometer demo sensor.

CM127-025-G01 Protected Aluminium Mirror

The CM127-025-G01 is a half-inch diameter concave mirror with a protected aluminum coating and a 25 mm focal length. This mirror offers an average reflectivity above 90% across a wide spectral range from 450 nm to 20 µm. The protective overcoat enhances

resistance to oxidation and environmental degradation, ensuring long-term stability even under frequent handling or in open-air setups. Within the OpenSense spectrometer, this mirror serves as a robust and cost-effective option for general optical reflection, particularly in the visible and near-infrared regions. Its balance of performance, durability, and affordability makes it ideal for an educational or research-oriented instrument that prioritizes accessibility without sacrificing optical quality. [25]

CM127-025-P01 Protected Silver Mirror

The CM127-025-P01 is a half-inch diameter concave mirror featuring a protected silver coating and a 25 mm focal length. Compared to the previously mentioned aluminium mirror, silver coated mirrors are known for their exceptional reflectivity. This particular mirror has an average reflectivity typically exceeding 97% across the wavelength range of 450 nm to 20 μm. However, compared to aluminum coatings, silver surfaces require more cautious handling to prevent scratching or chemical degradation. This mirror provides enhanced reflectivity, improving optical throughput through the system; however, silver surfaces require more cautious handling to prevent scratching or chemical degradation. [26]

Edmund Optics #43-465 Protected Aluminium Mirror

The Edmund Optics #43-465 is a 1-inch diameter concave mirror with a protected aluminum coating, optimized for broadband use between 400 nm and 2000 nm. This mirror features a larger diameter than the others, allowing for an improved light collection and alignment tolerance at the cost of the compactness of the system. [27]

Table 3.14 - Mirror Component Selection

Property	**CM127-025-G01**	**CM127-025-P01**	**Edmund Optics #43-465**
Manufacturer	Thorlabs	Thorlabs	Edmund Optics
Diameter	0.5 in	0.5 in	1.0 in
Coating	Protected Aluminium	Protected Silver	Protected Aluminium
Focal Length	25 mm	25 mm	25 mm
Wavelength Range	450 nm - 20 μm	450 nm - 20 μm	400 nm - 2000 nm
Durability	Higher	Lower	Higher
Average Reflectivity	90%	97%	85%
Cost	$46.40	$46.40	$52.47

We chose the CM127-025-G01 mirror, due to its durability and practicality. Its protected aluminum coating offers greater resistance to oxidation and handling damage compared to silver, ensuring long-term stability. Because the incoming light is already confined by

the entrance slit, the larger 1-inch mirror provides no optical advantage. The 0.5-inch mirror, therefore, offers a more compact and cost-effective solution while maintaining excellent broadband reflectivity and sufficient collection efficiency for the system's optical path.

3.2.9 Sensors

Comparison of Sensor Technology

The choice of detector is fundamental in determining the sensitivity, resolution, and practicality of a spectrometer. Detectors vary not only in cost and complexity but also in their ability to handle low-light conditions, capture fast-changing signals, and interface with microcontrollers or external systems. For OpenSense, we focus on compact, affordable sensor types that balance usability with educational impact. Among the most relevant are linear array detectors, charge-coupled devices (CCD), and complementary metal–oxide–semiconductor (CMOS) sensors.

Linear array detectors consist of a row of photodiodes arranged in sequence, each element corresponding to a particular wavelength dispersed by the grating. These sensors are relatively inexpensive and straightforward to integrate with microcontrollers, making them an attractive choice for entry-level spectrometer designs. While they generally provide lower resolution compared to CCD or CMOS sensors, their affordability and simplicity align well with our project's goals of accessibility and educational use.

Charge-coupled devices (CCD) have been a standard in scientific imaging and spectroscopy due to their high sensitivity and low noise characteristics. In spectrometers, CCDs allow for precise measurement of weak light signals across a wide dynamic range, making them highly effective for detailed spectral analysis. However, CCDs typically require more complex readout electronics, consume more power, and are generally more expensive than simpler detector options. For a low-cost educational platform, CCDs may be less practical, but they serve as a valuable benchmark for performance comparison.

Complementary metal–oxide–semiconductor (CMOS) sensors are popular due to their lower power consumption, faster readout speeds, and integration-friendly design. Unlike CCDs, CMOS sensors can incorporate on-chip electronics, simplifying the overall system architecture and reducing cost. While earlier generations of CMOS sensors had limitations in noise and sensitivity, modern CMOS designs have closed much of that gap, offering competitive performance in many spectroscopic applications. For OpenSense, CMOS detectors represent a promising balance between performance, affordability, and scalability, making them particularly relevant to our demonstration spectrometer.

Photodiodes are semiconductor devices which convert light into electrical current, serving as the foundation for many optical detection systems. Photodiodes are available

in various types, such as PN, PIN, and avalanche, each offering different levels of sensitivity and speed. While photodiodes offer only a single-point measurement compared to the array detectors we looked at earlier, combining them with a scanning mechanism (such as a rotating diffraction grating) enables wavelength-resolved measurements. For the OpenSense spectrometer, photodiodes are particularly appealing because they are compact, inexpensive, and better showcase the type of sensors which our platform was created for. By using a photodiode within our design for the demo-sensor, we showcase how the platform can be used to interpret data from sensors in complex projects.

Table 3.15 - Sensor Technology Comparison

Property	**Linear Array**	**CCD**	**CMOS**	**Photodiode**
Cost	Low	High	Moderate	Very Low
Sensitivity	Moderate	High	High	Moderate
Noise	Moderate	Very Low	Low	Low
Response Time	Fast	Slow	Very Fast	Very Fast
Power Consumption	Low	High	Low	Very Low
Area	Area	Area	Area	Single-point
Compatibility with Platform	Good	Difficult	Difficult	Best

Sensor Part Selection

Since the core purpose of our project is to develop and demonstrate a modular sensor platform, selecting the appropriate sensor type is a critical design decision. While array-based detectors such as linear, CCD, and CMOS arrays are popular choices for spectrometers for capturing full spectrums simultaneously and without moving parts, they differ from the type of sensors that the OpenSense platform is designed to support. To better demonstrate how OpenSense can be used with sensors in complex projects, our team has decided to use a photodiode-based detection system.

FD11A PIN Photodiode

The Thorlabs FD11A is a high-performance silicon PIN photodiode housed in a TO-can package, designed for general-purpose optical detection across a broad wavelength range of 320–1100 nm. Its 1.21 mm² active area provides a good balance between sensitivity and response speed, making it suitable for visible and near-infrared applications. The hermetically sealed package offers durability and protection from environmental factors, supporting stable long-term operation. The FD11A's broad spectral response and robust construction make it well-suited for the project, however it is one of the more expensive options. [28]

TEMD5080X01 PIN Photodiode
The Vishay TEMD5080X01 features a wide spectral response from 350 - 100 nm, with its peak sensitivity at 940 nm. Of all the photodetectors looked at, it features the largest active area of 7.7 mm², allowing it to capture all of the light which comes through the system. It is also packaged as a surface mount device (SMD), making it easy to work with. Overall, this sensor is a good fit to integrate with the OpenSense platform, making it a good pick to use for the spectrometer demo sensor. [29]

SFH 2716 A01 PIN Photodiode
The SFH 2716 A01 from OSRAM is a compact, surface-mount photodiode optimized for visible light detection. With a spectral range of 350–1000 nm, peak sensitivity near 620 nm, and an active area of 0.35 mm², it offers good responsiveness within the visible spectrum while maintaining a very small footprint. Its 0805 SMD package makes it ideal for lightweight and integrated designs where space is limited. However, the small active area limits total light collection, making it better suited for applications with sufficient illumination intensity. [30]

T1670P6 PIN Photodiode
The T1670P6 is a PIN photodiode with a 0.27 mm² active area and peak sensitivity at 560 nm. It provides high-speed, high-sensitivity detection for visible light and is well-suited for applications requiring photometric accuracy or human-vision-related measurements. [31]

Table 3.16 - Sensor Component Selection

Property	**FD11A**	**TEMD5080X01**	**SFH 2716 A01**	**T1670P6**
Manufacturer	Thorlabs	Vishay	OSRAM	Vishay
Package Type	TO-can	Surface Mount	Surface Mount 0805	Chip
Spectral Range	320-1100 nm	350 - 100 nm	350 - 1000 nm	390 - 800 nm
Peak sensitivity	~900 nm	~940 nm	~650 nm	~560 nm
Response Speed	Moderate - Fast	Fast	Fast	Very Fast
Active Area	1.21 mm²	7.7 mm²	0.35 mm²	0.27 mm²
Cost (unit)	$17.12	$1.9	$0.86	$1.20

After comparing the available photodiodes, the Vishay TEMD5080X01 was selected for the OpenSense spectrometer. Its broad spectral response from 350–1100 nm provides excellent coverage across both the visible and near-infrared regions, aligning well with

the intended operating range of the optical system. The device's fast response time, low capacitance, and compatibility with a 5V reverse bias make it highly suitable for a scanning-based setup using a single photodiode detector. Additionally, its compact and affordable design supports the OpenSense project's goals of accessibility, modularity, and educational value. The TEMD5080X01 provided the best overall balance between performance, speed, cost, and ease of integration for the spectrometer's proof-of-concept implementation.

3.2.10 Motors

Comparison of Motor Technology

Because the spectrometer design utilizes a single photodiode as the detector, a rotational mechanism is required to scan across different wavelengths of light. This rotation enables the photodiode to sequentially detect intensity at various spectral positions, effectively building a full spectrum over time. To achieve this controlled motion of the diffraction grating, an electronic motor will be employed. For this purpose, our team evaluated two viable motor types – servo motors and stepper motors – to determine which best balances precision, control, and cost for the spectrometer showcase sensor.

A servo motor is a type of motor designed for precise control of angular position, velocity, and acceleration. It operates as part of a closed-loop system, continuously monitoring its position through feedback from an encoder or potentiometer and adjusting its movements to match the target input. This feedback control makes servos a good choice in applications that demand high positional accuracy and smooth movements. However, the inclusion of feedback electronics and control circuitry increases both system complexity and cost. [32], [33]

A stepper motor is a device designed to rotate in discrete, fixed angular increments, allowing for precise precision control without the need for feedback sensors. Each step corresponds to a defined rotation angle, determined by the motor's internal coil configuration.This open-loop operation simplifies control and reduces system cost while still offering sufficient precision for many applications. In cases where the individual step size of the motor is too large, different operation modes such as half-stepping or microstepping can be used to reduce the stepsize at the downside of requiring a more complex motor driver. Stepper motors provide a balance of accuracy, simplicity, and low cost which make it an ideal choice for our platform. [32], [34]

Table 3.17 - Motor Technology Comparison

Property	**Servo Motors**	**Stepper Motors**
Cost	Higher	Lower

Table 3.17 - Motor Technology Comparison		
Control Type	Closed-loop	Open-loop
Position Accuracy	Very High	High
Speed Control	Continuous	Incremental
Torque	Higher	Lower
Response Time	Fast	Moderate

Motor Part Selection

After evaluating the motor options, a servo motor was selected to control the diffraction grating in the spectrometer. While a stepper motor can provide sufficiently small angular increments and does not require a feedback system for basic positioning, the OpenSense spectrometer must also measure and record the precise angular position of the grating during operation to correlate rotation with detected wavelength. Because this information is essential for accurate spectral measurement, a feedback mechanism becomes necessary regardless of motor type. A servo motor inherently includes a built-in position feedback element, typically a potentiometer, allowing both controlled rotation and real-time verification of its angular position without additional sensing hardware. This simplifies system integration, reduces component count, and ensures that the grating position can be reliably tracked throughout the scanning process, making the servo motor the most practical and efficient choice for the final design.

Adafruit 1404

The Adafruit 1404 is a micro analog servo motor commonly used in lightweight robotic and positioning applications. Operating from 4.8–6 V, it offers a typical rotation range of about 180°, controlled through standard PWM signals. Thanks to its compact size and straightforward electrical interface, the 1404 provides smooth, moderate-torque motion suitable for small optical or mechanical assemblies. While it does not feature continuous rotation or high torque output, its integrated positional feedback enables accurate and repeatable angle control without the need for additional sensors. This makes it well-suited for spectrometer systems requiring reliable grating positioning with minimal system complexity. However, its smaller size limits load capacity, making it less suitable for larger or weight-bearing optical components.

Adafruit 154

The Adafruit 154 is a larger and more robust analog servo motor designed for applications requiring higher torque and improved mechanical strength. Operating from a standard 5–6 V supply, it offers precise positional control and a typical 180° travel range. Compared to smaller micro servos, it can drive heavier loads with reduced jitter, ensuring stable and repeatable angular movement even under moderate mechanical stress. This makes it well-matched for spectrometer configurations where the diffraction grating mount or mechanical interface adds weight or requires steadier actuation. While it is physically larger and more expensive than micro-class devices, the enhanced torque and mechanical reliability allow it to deliver more consistent motion in laboratory or demonstration environments.

DFRobot SER0046

The DFRobot SER0046 is a compact servo motor designed for low-cost and embedded applications, operating at 5 V with standard PWM control. Similar to other small analog servos, it

features integrated positional feedback through an internal potentiometer, enabling closed-loop angular control without external sensing hardware. Its small size and modest torque output make it ideal for lightweight optical components, particularly in prototypes or compact spectrometer assemblies where space and budget are limited. While it does not offer the torque or structural rigidity of larger servos, its ease of integration and stable operation provide a practical solution for modular instrument designs, such as the OpenSense spectrometer, where accessibility and simplicity are prioritized.

Table 3.18 - Motor component selection			
Property	**Adafruit 1404**	**Adafruit 154**	**DFRobot SER0046**
Manufacturer	Adafruit	Adafruit	DFRobot
Operating Voltage	4.8-6V	5-6V	5V
Rotation range	180 degrees	180 degrees	180 degrees
External Feedback	Yes	No	No
Cost	$14.95	$11.95	$6.90

The Adafruit 1404 servo motor was selected for the OpenSense spectrometer due to its compact size, sufficient torque for the lightweight diffraction grating assembly, and integrated potentiometer feedback system. Unlike stepper motors, which require additional sensors or encoders to confirm their angular position, the 1404 provides a direct analog voltage output from its internal potentiometer that varies linearly with shaft rotation. This feature allows the system to accurately monitor the grating angle in real time, ensuring correct wavelength association during scanning without adding extra hardware or circuitry. Its low operating voltage, simple PWM control, and minimal integration overhead make the 1404 a practical and efficient solution.

3.3.1 Comparison of Software Technology

In looking at what programming environment will work best with ESP32 for our project, there are really three possibilities that are quite viable. The first is Espressif IoT Development Framework (ESP-IDF). The second is the Arduino Core for ESP32. The third is MicroPython. The ESP-IDF is the native programming environment provided by Espressif for the ESP32. It has the built-in components to do things like embedded Hypertext Transfer Protocol (HTTP) servers, FAT file system module (over Secure Digital/Multi-Media card (SD/MMC)), and a virtual file system layer to output/input (I/O) like POSIX, non-volatile storage (NVS), and drivers for I²C and analog/digital converters (ADC), all running on top of FreeRTOS. One of the attractive benefits of the ESP-IDF is the uniform way in which all of these components work together. If the design requirements are such that there is the necessity of having fine control over memory, scheduling, and driver behavior, and/or avoiding layers of software that prevent such

control, then the ESP-IDF is likely the best choice. In fact, many engineering groups will start using Arduino for prototyping purposes and later will convert to ESP-IDF for production. Specifically, one of the nicer aspects of the HTTP server component of the ESP-IDF is that it is lightweight, uses URI handlers, and is rather easy to create both static assets and dynamic endpoints.[38] The use of the microcontroller will thus allow the user to run a single-page application directly on the host. As the server is run in process and handlers are simply C functions, the user can expect consistent request latencies with no blocking of the sampling operations, while being able to expose to their single page application (SPA) an updated browser interface.

The main advantage of the use of Arduino is the ease of use and the size of the community-built library and example implementations that are made available to the developer. An example of this is for a local SPA with real-time multiple channel visualization and configuration management (i.e., showing five simultaneous data streams with a 10-second rolling window), the asynchronous webserver library for the ESP32 exposes a solid HTTP and WebSocket functionality via a callback-based API [39]. Therefore, the Arduino Framework works as an abstraction layer over the ESP-IDF components, so all I/O management and wi-fi control will still use the native IDF drivers and FreeRTOS kernel. The structural relationship between Arduino and IDF thus allows for rapid prototyping and primary integration, while providing the performance characteristics of the IDF foundation. The Arduino implementation is thus, in its base form, a FreeRTOS implemented firmware, but the framework is encapsulated within its IDF functionality in higher-level C++ classes and functions, which lessens the initial development friction.

MicroPython is a good base as it's good for fast iterations in time, and also allows for code to be kept short and readable. The newest versions also support cooperative multitasking with uasyncio, and now users are able to use WebSocket support and routing for an embedded web server completely in Python, which makes the control flow much easier to read than going down to the raw FreeRTOS primitives. However, there are downsides. Normally, the user will have timing misses that are tighter and will find the throughput less than satisfactory and less deterministic than with C/C++. This notwithstanding, it is possible from a technical view for MicroPython to be used as the operating environment for our implementation (sampling five channels at 10Hz, continuously writing to microSD, and also streaming a real-time graph over websockets); however, the user has to be very careful with buffering, etc., and avoiding blocking I/O for the main pair. However, in practice, the extra performance headroom and predictable behavior from the C/C++ options under IDF or Arduino is far more acceptable to the user once writing to SD cards and streaming to multiple clients begins. As expected, these conclusions are backed up by peer-reviewed studies comparing language performance on ESP32 microcontrollers [40]. Recent studies have also examined FreeRTOS evolution and performance characteristics, confirming its suitability for embedded real-time applications [41]. Additionally, research on IoT sensor data acquisition frameworks validates the architectural patterns we have adopted for our multi-sensor system [42].

Table 3.19 - Software Technologies			
Language/IDE	ESP-IDF	Arduino Core	MicroPython
Implementation Languages	C/C++	C/C++	Python
Libraries & Example implementations	Good Amount	Most	Least
System Control	Full Control	Good Access	Moderate
Prototyping	Medium	Fast	Fastest
Development Friction	Some	Minimal	Most

Recent studies have also examined FreeRTOS evolution and performance characteristics, confirming its suitability for embedded real-time applications [41]. Additionally, research on IoT sensor data acquisition frameworks validates the architectural patterns we have adopted for our multi-sensor system [42]. Based on these considerations and the need for deterministic behavior, fine-grained control, and production-ready stability, we selected ESP-IDF with Arduino Core compatibility as our primary development framework.

3.3.2 End-to-end architecture: tasks, timing, and composability

The above method would use a steady 10 HZ sample rate with a web application writing to an SD card using two different, non-blocking, task-based queuing systems. One of the two systems is called a “sampler” task and is used to read the samples from the I²C register or trigger the ADC conversion every 100 ms by running a timer for each sample. The only responsibility of the “sampler” task is to append a time-stamp to the sample data and append it to a struct. The “struct” is then appended to the “non-blocking” queue. Another one of the “task” systems is called a “logger” task and is used to retrieve the “structs” from the “queue”, and append them to a ring-buffer; the “structs” are then written to a CSV file in “sector-aligned” chunks so that the SD-card latency does not affect the sample frequency. If desired, a “telemetry” task is used to create a compact JSON frame with the last measured values and send them at least every ten or twenty packets/second over a single WebSocket port to all connected browsers. Because freeRTOS “queues” are perfectly designed to pass information from task-to-task and/or from an interrupt service routine to a task without requiring busy-waiting, this allows for continuous data transfer, regardless of how long the “logger” and “network” tasks require to complete.

The FreeRTOS task-based architecture has been extensively studied and proven effective for real-time embedded systems [41], providing confidence in our queue-based sampling and logging design.

There were two decisions made to enhance this design further. The first was to limit the amount of work required of the “sampler” task. The second was to treat the “writer’s” buffer as a shock absorber for the file system. SD-card controllers typically write data to flash in blocks, and may periodically pause for hundreds of milliseconds. Since the sampling frequency of 10 Hz, and the five available channels result in a relatively low data-rate, the most effective approach to writing the data is to write data in large chunks (i.e., 1 second worth of samples at a time), and to write these large chunks such that they are aligned to the 512-byte sector-bounds of the SD-card. The VFS simplifies the file-I/O aspect of things as well, because once the FatFs file system is mounted, the standard C file functions will function properly and allow the firmware to be written to operate within the IDF ecosystem[43], [44].

3.3.3 Sensor interfaces at 10 Hz: I²C throughput, address conflicts, and ADC realities

While I²C is able to accommodate a cluster of five sensors at 10 Hz, this can be done because both I²C controllers on the ESP32 have been flexibly mapped to allow either controller to act as the Master. As well as being fast enough at 400 kHz Fast Mode, this is also a common frequency used in the area of Environmental Sensors and Motion Sensors. In situations where there could be address conflicts between multiple devices (for example, two identical sensors connected to the same I²C Bus), a simple solution would be to connect an I²C Multiplexer (such as the TCA9548APWR). The TCA9548APWR has eight downstream channels and can be configured using just one byte of writing to the Control Register. Each Channel can then be selected by writing one byte to select that channel, then the downstream I²C device will appear to respond as the only device on the bus. This can save time compared to the alternatives of doing boardwork or strapping pins together. This is particularly useful in a situation where you may have a cluster of the same boards that require access to the same address space [45], [46].

The analog inputs present a different set of requirements. The reference voltage for the original ESP32 chips varied from chip to chip, about a nominal value of 1.1V, and had a non-linear transfer function. Therefore, if accurate measurements are necessary, each unit must be calibrated individually, or an external ADC must be used to measure the signals with a high degree of accuracy. However, the good news is that there is an acceptable ADC calibration module in the ESP-IDF, which compares the different properties of each device and eliminates the differences that were found. Another option in the documentation for the ADC shows how attenuation can be used to increase the maximum measurable signal level without exceeding the saturation point of the ADC's conversion core. While it may be difficult to obtain very accurate engineering data at a rate of 10 Hz, good results can still be obtained if the appropriate attenuation values are selected, calibration is performed at start-up, and sampling is performed while Wi-Fi is

active, it is reasonable to sample the ADC1 channels first since they share resources with the Wi-Fi system[47], [48]. Recent characterization studies of ESP32 sensor networks have identified the ADC's non-linear behavior and noise characteristics [49], reinforcing our decision to use calibration and potentially external ADCs for high-precision requirements.

3.3.4 Storage path: SDMMC versus SPI, FatFs through VFS, and CSV strategy

The ESP-IDF storage stack has two ways to interface to microSD cards: the SDMMC host (that supports native 1-bit and/or 4-bit modes) and the SDSPI host via the SPI peripheral. While the SDMMC host is preferred because it is less CPU-intensive and could possibly give you a faster throughput than the SDSPI host, if your board's wiring cannot accommodate a connection to the SDMMC, then the SDSPI host will likely give you a sufficient data transfer rate for the moderate data rate that is being used in this project. In either case, the higher-level SD/SDIO/MMC driver handles all the low-level operations that relate to initializing the card as well as transferring data. Thus, your code for your application can focus on mounting a file system and creating/writing files [50].

The VFS, along with the FatFs library, is integrated at the file system layer above the block device layer in ESP-IDF. Because of this, the standard file open/fwrite/file close functions work just like they do on your desktop, so the logging task can write a second's worth of readings to disk in a single operation, avoiding making multiple fsync calls between file rotations and/or the creation of periodic checkpoints. FatFs also allows you to rotate files based on either their size or their age. This would be helpful if you need to download the log file(s) via the web browser or swap out the SD card. Since your data stream is sparse relative to what most people consider typical usage of SD cards, the performance degradation you experience is probably due to writing line-by-line or using small buffers and not due to bandwidth constraints. Using the one-second, sector-aligned write method outlined here will help you avoid both of these potential performance problems[43], [44].

The CSV format should be intentionally simple, self-documenting, and include: a header row that lists the channel name, units, etc., and rows of monotonic time-stamp and value pairs that are formatted in a consistent numeric format. Whenever you want to record a setting change from the web-app, write a comment line that describes the change to make the processing of the data easy after the data is recorded and to enable auditors to associate the data sets with the field action changes made to the device without needing to access the device's internal state. Since this device can operate in SoftAP mode with no upstream connectivity, use "milliseconds since boot" for both real-time streaming and for the CSV file. If you require human-readable timestamps, you will need to add a battery-backed RTC or have the operator set an epoch via the web-app. This approach of using ESP32 with SD card storage for data logging has been successfully demonstrated in similar real-time monitoring applications [51], validating our storage architecture choices.

3.3.5 Networking and SPA hosting on the device

The simplest way to use the web app while using a minimum of resources (i.e., typically, an index.html file, a minified JavaScript bundle, and a CSS/stylesheet) is to run the web app directly on the device; i.e., as if it were a native application.

Since the IDF HTTP server serves both static files and REST endpoints through URI handlers, you can simply add a WebSocket endpoint to it without needing to add a separate web framework. Arduino's server library is asynchronous, and like with most asynchronous server libraries, it provides a high-level abstraction layer over similar concepts and also provides built-in WebSocket support for rapid development.

So long as you keep your handlers blocking, push any sampling and/or writing into background tasks, and allow the main loop to get back to the event loop as soon as possible, the web stack will be able to respond to any amount of load [38], [50].

On the browser side, you should choose a lightweight charting library that can efficiently handle real-time time series data. As such, Chart.js is a good option since it offers a time-series axis, clearly shows how to do incremental updates, and is capable of providing fast performance even for large fixed-size circular buffers. For example, a "last 10 seconds" display at 10 samples/sec would need a 100-sample buffer for each chart. When the client receives a WebSocket message, it adds a new sample to the end of the buffer, removes the oldest sample, and calls the update method of the library. With HTTP compression and cache headers, most modern web applications are small enough in size, so their load times are quick when users connect to the SoftAP in the field [52]. Hassan and Hassan demonstrated that ESP32 web servers can effectively handle real-time data acquisition while simultaneously serving data via Wi-Fi to multiple clients [51], validating our architectural approach. Similarly, Labib et al. showed that ESP32 devices can effectively operate as web servers in access point mode while managing sensor data acquisition [53], confirming the viability of our design.

In SoftAP mode, the ESP32 creates its own Wi-Fi network and runs a DHCP server to assign IP addresses to connected browsers. In previous releases of the ESP32, there was some confusion regarding the number of clients that could be supported. While older sources claimed limited client support, the ESP-IDF Wi-Fi documentation confirms that the default maximum for SoftAP mode allows up to 10 simultaneous station connections [54], which is sufficient for our multi-client dashboard requirements. Devices that send out five different variables per second at 10-20 FPS to several browser-based dashboards will have plenty of room, and you may see no more than one or two simultaneous connections in practice. If you expect more, you can either decrease the transmission rate, decrease the payload size, or restrict access to only a few users. A basic captive portal setup will map all of the hostnames to the device and then send them to the /index.html page to provide a simple and seamless experience for the technician when they are in the field and cannot access the Internet .

3.3.6 Configuration management, durability, and field ergonomics

These configurations that can be changed by the operators, i.e. type of sensor, I²C address, ADC attenuation factors, calibration factors, units of measure, rates of logging, etc. and anything to do with web page facility should be placed in NVS (non-volatile storage) which is a key-value store with wear leveling that is designed for embedded units, and will permit atomic writes. A general procedure would be to read the present JSON blob from NVS into system RAM at boot time, edit it in memory, but only when the white conditions against some firmware thing are satisfactory, and write it back with a new version number. Thus, in case anything when the write fails, on the next boot, the unit will come back cleanly to the previous version, and the new data will come into operation when the software comes up again. This is particularly useful in field instruments where casual cycling of the power supply can be anticipated; these atomicity and roll-back features would be of greater importance than raw write speed[43]. There are also aspects to consider in relation to the durability of the way faults are recorded and reported. The logger task should keep track of whether the internal section involved in writing is “buffered but not yet flushed” or “flushed” and write a short footer in the CSV table when in the file rotation due to size, time, or error. If there are repeated errors in writing the firmware should declare some status that is visible to the user of the web app, and is of importance, and the streaming of the measurement data should continue even if the logging has been temporarily switched off. This separation of work is the result of the queue and task architecture. For it is that both Wi-Fi sends and SD writes draw current, and intrigued by the integrity of the supply, it is interesting to consider the effects of short transmissions and so not causing a brown-out on the card rail, for instance, during the write operation. The design details of power supplies will not be covered in this literature review, but the storage units and the Wi-Fi in IDF work well regarding voltages of working, and it is up to the application to ensure that enough priorities and buffer sizes are chosen when processes are written[38], [50].

3.3.7 From theory to implementation: how the pieces fit

In contrast to the many other forms of embedded IoT platforms, an IDF implementation is unique in that it typically has four main parts. The first part is called the "sampling" element,which determines what ports should be configured for I²C, what addresses should be configured, and how to map ADC channels to addresses based upon information stored in the NVS-backed config, and then uses that to determine a run plan for the sampling process to take place every 100 ms as a high-priority task. The second part is referred to as the "logging" element, which mounts the VFS (Virtual File System) and houses the FatFs file system, and serves two purposes: to provide a queueing service that the sampling element can use to invoke and to provide a flushing service (which writes out 512-byte chunks) to manage file rotation. The third part is the "web" element, which sets up the HTTP server and provides URI handlers for static content and for GET/POST operations to configure the device, and a WebSocket handler for the device to stream data to clients. The fourth part is the "configuration" service, which converts JSON schema to and from NVS, validates changes to the configuration, and notifies the sampling task when something like the sensor's register map or the ADC

attenuations has changed. Because the URI handlers for the HTTP server are simply C functions, they can easily validate and copy the user-provided configuration values to the configuration service and then immediately return, thus allowing the sampling and logging elements to perform their necessary operations based upon the updated configuration values without having to perform a reboot. The division of responsibilities in an IDF also naturally correlates with the module boundaries of the IDF, which enables the testing of each individual componts[38], [43], [44], [55].

The Arduino version of the implementation follows the same conceptual design as the previous versions of the implementation. However, unlike the previous versions, the Arduino version of the implementation differs in that the entry point of the web server is asynchronous. The asynchronous web server sets up the routes and WebSocket connections during the setup() phase of the program, and a timer or cooperative loop invokes the sampling operation every 100 ms, and the SD writer utilizes the same FatFs and SD drivers as in the IDF version, but through the Arduino Core. This version of the implementation is often used by teams because of the many examples of applications that are available in the library ecosystems that utilize WebSockets, file serving, and JSON parsing, which allow for the quick initial integration of the web application and the display of real-time graphs before optimizing the application. The advice still applies: try to avoid performing blocking I/O operations within the high-rate sampling path and instead move as much of the work as possible to background tasks or callback functions [39].

A MicroPython implementation offers different trade-off considerations than the IDF and Arduino implementations. With the uasyncio library, you can create a coroutine that waits about 100 milliseconds between samples, another coroutine that packages and sends WebSocket frames, and a MicroWebSrv2 server that handles both static files and REST write handling. The structure is very elegant; however, the cooperative scheduler requires more responsibility from you to ensure that your coroutines are short, minimize JSON encoding overhead, and keep the buffer sizes small so as to prevent any delays that occur during SD writing from affecting the timing of the sampling operations. Due to its cooperative nature, this is best suited for proof-of-concept and educational-type projects. For longer-term logging and multi-client streaming applications, the deterministic nature of the IDF and Arduino C path will provide greater flexibility. Research comparing the relative performance of various programming languages on the ESP32 supports the idea of the raw speed differences between the IDF/C and MicroPython paths [40].

3.3.8 Charting and UI behavior: pushing only what changes

The Web Application must perform as little "work" as necessary to stay alive. The device publishes one WebSocket endpoint (like /ws). It provides the last values and a monotonic timestamp in a compact frame. Each of the five graphs that the web application keeps has a ring buffer of 100 values. Each time the device provides a new frame to the client, the client adds the new values to the ring buffer, deletes the old values that are no longer visible to the user as they are now at the end of the buffer, and

calls the Chart.js standard update function. Thus, the web application does not continually render large amounts of data, and regardless of whether the user is viewing the web page via a mobile or desktop browser, the web application's memory usage will be predictable. Since the device may be deployed in a remote area that does not have access to an upstream clock, the web application shows elapsed time on the x-axis instead of showing wall-clock time. Chart.js' time series scales will work fine for rendering elapsed time on the x-axis. Also, since the web application can send the same small JSON payload to all connected clients, the device does not have to keep track of per-client state on the firmware side [39].

3.3.9 Throughput and latency budget: why 10 Hz is easy to hit

A rough estimate tells us what overhead is present. A few register reads off of an I²C bus running at 400 kHz will take tens of microseconds to read. Performing five of those reads, plus adding overhead for functions (which is probably going to be in the low milliseconds) and performing ADC conversions and scaling on two channels (staying in integer space, so no floating point conversion needed) should take approximately another millisecond of CPU time. Placing data in a FreeRTOS queue is going to take less than a millisecond of CPU time. Your network framing time (along with the time it takes to do a little bit of WebSocket pushing) should be a few milliseconds of CPU time and a few milliseconds of airtime at reasonable WiFi speeds. The logger task will run independently of the sampling task and wake up every second to write a sector-aligned block to the CSV file. The only real way to risk maintaining a 100 ms period on this task is by having a blocking function call in the sampler or you have a ring buffer size that is smaller than the amount of time it takes to stall an SD card; both are easily avoided using the architecture as described above [30]. These timing estimates align with performance characterization studies of ESP32 in real-time sensor applications [49], which confirm that our 10 Hz sampling rate provides significant timing margin for reliable operation.

3.3.10 I²C multiplexing and mixed buses in practice

When working in a lab setting or in the field, conflicts among sensors occur frequently for team members who have chosen to use a single type of sensor. An ideal solution to this issue is the TCA9548A, which provides eight additional downstream ports to enable/disable using a single byte of control information; therefore, implementing this function mechanically in your sampler program will be straightforward. In addition to providing additional downstream ports to enable/disable, the TCA9548A also provides level shifting of the signals from the different busses, enabling devices connected to each bus to operate at different voltage levels if required. In addition, all channels are disabled upon power-up, thus eliminating the possibility of bus contention during startup. If there are more than five sensors being used in a project, they should be divided between the two I²C controllers of the ESP32, and a multiplexer placed on the bus with the most conflicts. This method will minimize the amount of data traffic that occurs on each bus cycle, minimize the longest possible transaction time, and provide sufficient bandwidth to increase the sampling rate in the future [56].

3.3.11 ADC calibration strategy and implications for data quality

To ensure that analog measurements are done consistently, calibration should be used as a regular procedure during the complete lifetime of the device. The IDF's ADC calibration driver is built from APIs that calculate the device's specific characteristics from a 1.1 volt reference point (which varies from chip to chip from approximately 1.0 to 1.2 volts) and which will adjust the readings made for errors, and will give both unadjusted and adjusted readings to the CSV file for historical purposes. Innate calibrating on boot-up (or some "maintenance" option) enables a adjustment of readings and update of the CSV files to take place, whereby adopting a suitable level of attenuation for expected input voltage ranges, then this soon solves any measurement errors which occur except for cases where they are of high precision type (such as those taken with a frequency greater than 100 Hz), in which case if some smaller absolute measurement error, or large dynamic range are required in future, the I²C reads can be substituted from the internal ADC to take I²C reads via the external ADC driver (without any change in overall architecture of the system) and work continues via same queues, logger and web app [48], [57].

3.3.12 Wi-Fi SoftAP behavior and multi-client dashboards

With ESP32 running as SoftAP, the ESP32 will take care of the SSID, channel, and DHCP lease area. The IP server, WebSocket endpoint are combined, so clients can connect and start getting telemetry. A frequent question comes up: “How many simultaneous Dashboards can we provide?” According to the ESP-IDF Wi-Fi Driver documentation, the default maximum number of stations that can associate with the SoftAP is 10 [47], which exceeds earlier informal reports and provides sufficient capacity for our multi-client telemetry requirements.

3.3.13 Why C/C++ usually wins here and when MicroPython is still right

Even though this workload is very I/O intensive, C/C++ on FreeRTOS is deterministic (and thus has an advantage) with respect to SD logging and multi-client networking because of the C/C++ nature. The comparison of CPU bound task performance among C/C++, MicroPython, Rust, and TinyGo, all running on ESP32 hardware, shows that there are substantial differences in performance; although one would not normally expect one of these to have significantly better or worse performance than another, it does make sense that one should consider the differential in their performance with respect to I/O performance, which leads to the fact that regardless of the function(s), the application(s) that utilize the service(s) could likely be changed and/or modified to provide the same function(s). C/C++ on FreeRTOS has a steep learning curve(s); therefore, we will discuss several methods of testing that may be useful to you. While MicroPython should not be discounted for use in rapid prototyping, classroom projects or demo's, in applications where uasyncio and MicroWebSrv2 are applicable, if you are developing a field instrument that you plan to sell, and you need your instrument to

operate unattended for extended periods of time, the IDF and/or Arduino on IDF will provide the best roadmap to develop a reliable, robust product [40].

3.3.14 Testability and reliability: how to prove it works before fielding

Before shipping equipment to remote sites, you may have the opportunity to test each layer of firmware in the hardware before it is sent out. This is done through a self-test for each layer of firmware that runs on the sampler. A hardware sampler can run in a "synthetic" mode where it generates a known waveform (a specific pattern) on each channel and checks if the logger has retained its timing and if there are any missing samples in the CSV rows. In addition, the logger can perform a "stress test" on the SD cards by using the writer function, which alternates short and long flushes and validates that the queue will not become full while writing at 10 Hz. The web layer can also create a loop-back client that creates a Web Socket connection and measures the round-trip time of each of the five data streams (with both one client and with multiple clients). Since each interface between the different components in this system is either a queue or a socket, the tests above will naturally fit into the architecture of the system and do not need special test-only hooks. The HTTP server is also easy to instrument because the URI handlers are just functions, and the VFS allows your tests to work equally as well when you use a physical SD card mounted as a virtual file system versus a RAM-based file system during continuous integration testing [38], [55].

3.3.15 Security posture appropriate for a field instrument

When an instrument is running as a SoftAP, a good place to start would be using WPA2 with a non-standard passphrase. Limiting the number of SoftAP clients to what you actually need limits your potential target space. Using plain HTTP for the dashboard within the local access point will significantly lower the amount of CPU & Memory needed; also, eliminate the requirement to renew certificates in completely air-gapped environments. The IDF's HTTPS Server can be used if this is required by your threat model, but there is some additional memory use and added complexity for the added security. Using NVS for storing credentials and other operational preference values stores those values outside of the firmware images, and could potentially allow for supporting Factory Reset operations through the NVS Namespace Erase command [38], [43] [8].

3.3.16 What "good" looks like for the SPA itself

User experience will be shown through five real-time charts with the same x-axis, a small configuration menu, and a status indicator, easy to read, showing the status of the SD writer. Due to telemetry data format, the device only sends the last values of data; the browser will create and update a ten-second window of the previous values. Using the time-series scale of chart.js to include elapsed time in your chart is helpful when deploying via the softAP method, since the softAP method does not require access to NTP. The configuration panel will make one GET request to receive the current JSON object, allow the operator to edit fields such as "Sensor Type", "I²C Address", and

"Units", then make one POST request to send the validated object back to the device. When the device receives the object, it will immediately update the configurations in real-time and save the config to the NVS in an atomic fashion. This will result in a quick and simple to use tool that users can use both on their phone and laptop without needing any upstream infrastructure [43], [52].

3.3.17 Sensor library: interchangeable ports, auto-detection, and robust boot-time discovery

Constraints
The end goal is that the users turn the unit off and simply swap any of the supported sensors with any of the MUX ports, then turn the unit back on, and they will work correctly. The firmware will determine which sensors are currently plugged into the MUX ports, load the correct drivers for those sensors, publish their normal data streams, and store the configuration map for any potential troubleshooting in the future.

As our I²C master, we are using an ESP32-WROOM. As our multiplexing device, we are using a TCA9548A 1-to-8 Multiplexer. All of the sensors have been connected to the individual downstream ports of the TCA9548A at 3.3 volts.

Since the ESP32 has two I²C controllers, we can use one controller to act as a traffic divider or maintenance bus if we choose to do so in the future. Additionally, since the TCA9548A is a FET-based isolator, each port is physically separate from the others and can have its own pull-up resistor and voltage rail without affecting the operation of the other ports. This physical separation of each port allows us to accomplish a "swap while off" type of replacement of sensors without any potential conflicts during the boot-up process due to address collisions. In addition, a predictable channel scanning sequence is used to prevent any potential conflicts during the boot-up process. For example, if there was a failure on port #3, it would not affect the proper functioning of ports #1 through #8 [56], [58], [59].

Physical layer considerations: pull-up resistors, mux behavior, and bus recovery
Using N-channel pass FETs to decouple each channel downstream from the main I²C bus, the TCA9548A requires that you take into consideration both the positioning and sizing of pull-up resistors. The Application Notes provide the equations for determining your minimum. pull-up and max. pull-up resistor requirements. Your min. Pull-up is based on how much current the specification allows you to sink for I²C (3 mA for Standard, Fast, and Fast-mode Plus) and your max. Pull-up will be determined by the bus capacitance and rise time you want to achieve for your desired speed [59]. With a typical Fast-mode speed of 400 kHz and 100-200 pF of capacitance present on most sensor boards, values of pull-up resistors ranging from 2.2kΩ to 4.7kΩ should work fine for you [56], [59].

This is where bus recovery comes into play. Per the specification, there are defined recovery sequences for when SDA becomes stuck low, generally due to power coming on in the wrong order, static discharge, or firmware leaving a device in the middle of

transmission. Send up to nine clock pulses on SCL while looking at SDA, then properly transmit a STOP command after SDA returns high [60]. TI has an entire App Note dedicated to bus recovery, and since stuck buses are one of the primary reasons field failures occur [61], it is worth the read. The TCA9548A also provides a hardware reset pin to provide you with another exit strategy. When the reset pin is asserted, all channels will disconnect, providing isolation to the troublesome sensor [56].

Driver abstraction: a standard interface for different sensor hardware

At the firmware level, we've created a thin driver model, which means each sensor driver talks the same language. This idea is not a new one since Linux and Zephyr do likewise with their sensor subsystems. Each driver implements four functions. The probe function reads a WHOAMI or chip ID register without changing any state of the device and indicates its confidence that it knows what it has found. The init function sets up the ODR, filters, and measurement ranges, returning an opaque handle. The read function takes that handle and fills in a sample struct with its data converted to standard units: Celsius for temperature, Pascals for pressure, percent for humidity, and m/s² for acceleration. The meta function simply indicates what the sensor can do, channels, units, ranges, and default rates. This standardization is a huge win because the rest of the system would not care what sensor was plugged into port three. The sampler task simply calls read on whatever handles it got from discovery, and the logger and telemetry code never branches on the sensor types. Everything is already in the correct units and format.

Automated discovery across the mux channels

On power up or as part of a user-initiated event such as selecting "Rescan" in the web app, the firmware sequentially scans all 8 of its Mux Channels. The firmware selects the current mux channel k by writing a byte into the mux control register of the TCA9548A mux.

The firmware then attempts to discover any devices connected to each mux channel by sending a ping (i.e., attempting to read) to an I²C address on the channel. However, not all devices respond correctly to pings. Many devices require the firmware to read a specific register (to receive an ACK), which is why I²Cdetect supports multiple probing mechanisms [62].

Therefore, we iterate through all of our registered drivers. Then we attempt to read the minimal ID of each driver (usually referred to as the WHOAMI register). If we get a response, we assume the driver associated with the current I²C address on the channel is the correct driver.

It is possible that we will encounter scenarios where two drivers are claiming the same I²C address. This could occur for a variety of reasons, including duplicate devices (cloned) or overlapping sensor families. In these cases, we choose the driver with the higher confidence level. Confidence levels usually correlate with the completeness of the test. For example, reading a single byte is generally sufficient to uniquely identify

many devices. Reading a multi-byte ID is slightly more reliable than reading a single byte, and reading a checksum is the most reliable.

Once we determine the correct driver for the given I²C address, we call the driver's init() function and store the results in NVS along with the following information: the channel selected, the I²C address used, the driver selected, and the raw probe data so that we can troubleshoot if necessary. Also, the Web App allows a user to manually override the automatically detected device in situations where the device is confused by a clone device.

Examples from real sensors
There are a number of ways to utilize the benefits of SMBus. Some examples include, after power-up, a Bosch BME280 humidity/temperature pressure sensor will always return 0x60 (the "who am I") when read from one byte of register 0xD0 during polling [63], therefore it is a quick method to verify that the sensor is functional. An additional example would be probing the InvenSense MPU-6050 motion sensor; as the WHOAMI (at 0x75) will always return 0x68 (which is the same as its default I²C device address), but you need to compare the values returned from each function call to ensure they match [64]; in fact the InvenSense documents specifically mention to do this before setting any of the DLPF or scaling parameters.

Other than the device ID block, most devices also have SMBus extensions, like Address Resolution Protocol/Unique Device Identifier, which are strong, cryptographic-style identifiers for the device [65]. Therefore, if they exist, use them; however, they are not mandatory, so you cannot count on all sensors supporting them. A good place to get started with performing these reads within an ESP-IDF I²C example is the ESP-IDF I²C example library, along with sample code that demonstrates how to prevent accidental writes to the sensor, and/or put the bus in a bad state[45].

Why port swapping works (while powered off)
Moving sensors from Port 2 to Port 5 (and turning off all of them) can be done without any issues because the mux puts a different pull-up on each port, which means there will be minimal capacitive loading when Discovery brings up only one channel at a time [56]. By using the I/O matrix of the ESP32 to remap I²C to whatever GPIO pins are best for your board design [58], and by putting the reset pin of the mux on a GPIO pin that is currently unused, you can put every channel into a known state before scanning to prevent leftover data from the previous boot cycle or a brown-out event.

Manual vs. automatic mapping in the web UI
The same 5-tile grid that is shown on the website, as well as the number of MUX channels, is used in the web application for the status display for each tile: empty, detection was found (BME280 at 0x76), or manually override (MPU-6050 at 0x68). The user can either accept what the auto-detecting has found, manually override because of being confused by a clone board, or re-scan just the one channel that had an issue without having to do a reboot. The re-scan operation is very cautious: stop the channel driver, delete the channel from the MUX selection, flip the reset line to see if there is

anything wrong with the bus, find the channel again, then put this channel back into the sampler's handle list when the first data sample is found. At the same time, the other seven channels continue to sample at 10Hz.

Any changes to the configuration are saved in a versioned JSON file in the NVRAM memory with atomically written operations [43]. Every time you make a change to the configuration, the version # increases. If the power goes out while the config is being written, when you boot up the next time, the config will revert to the previous version. The current mapping of sensors to probes and the unprocessed probe data is part of the CSV header and can be obtained at /config for auditing purposes for anyone asking questions about the sensor data looking strange 6 months down the road.

Handling failures gracefully
In cases where the bus-clear cannot "break loose" the SDA signal from a specific channel - indicating a valid hardware problem rather than a simple communication issue - the firmware will designate that channel as Faulted and log the fact, but it will continue to function with all of the remaining channels. Each time a fault occurs or is resolved, the CSV receives a timestamped footer message to allow clear identification of the time gaps between successive samples of data from the sensors.

Channels with identical sensors connected to them will have collision issues if you use a single sensor interface. However, the MUX will resolve these collisions because you are able to place two BME280s on different channels, creating two separate driver instances for each [56]- essentially providing two separate interfaces to the sensor data.

Cloning a sensor that has an uncertain ID requires a two-stage scan: First, identify the sensor ID via its ID register, and then perform at least one additional sanity check against a known power-up value on another register. If the auto-detect process continues to fail to properly identify the cloned sensor, the user can assign a driver to a port manually via the web application. This is essentially duplicating the work that I²C detection has performed for many years, attempting to address weird hardware issues that may exist in the field [62].

Keeping discovery out of the real-time path
The discovery occurs at the beginning of the program (at start-up) before the high-priority sampler task begins its 10Hz cycle to prevent the probe operations, even slow ones such as bus-clearing, from interfering with sampling timing. When an user resamples a single channel from the field, only that one channel will display "NA" within the telemetry data stream. The sampler continues to read all the other handles at full rate; therefore, the WebSocket Frames remain intact and the DashBoards continue to function properly.

What the code actually looks like
Once you decide to use the "driver abstraction," the implementation will be easy. In order to implement the driver abstraction, we have a "sensordriver" struct that contains four function pointers for probe, init, read, and meta, along with an array of possible I²C

addresses to attempt to connect with. We also create a compile-time registry of all the drivers in our system. In the "discoverchannel" method, we pick a MUX channel, iterate over the registry (trying each driver's probe) to find which driver has the highest score, and then call the "init" method on the best-performing driver:

Figure 3.1: Example of code

example.c

```c
typedef struct {
    const char *name;
    uint8_t     addrs[4];     // common addresses to try
    int       (*probe)(i2c_port_t, uint8_t, uint8_t mux_ch); // return score
    esp_err_t (*init)(...);
    esp_err_t (*read)(..., sample_out_t*);
    const meta_t* (*meta)(void);
} sensor_driver_t;

static const sensor_driver_t *REGISTRY[] = { &drv_bme280, &drv_mpu6050, /*...*/ };

bind_t discover_channel(uint8_t ch) {
    mux_select(ch); // write 1<addrs) {
            int score = REGISTRY[i]->probe(I2C_NUM_0, addr, ch);
            if (score > best.score)
                best = (bind_t){ .drv = REGISTRY[i], .addr = addr, .score = score };
        }
    }
    if (best.score) REGISTRY[best.idx]->init(...);
    return best;
}
```

It’s as simple as that. The probing process has been made completely transparent, so the driver is simply a small C file that defines some basic functions. The mux channel is another function call argument, but everything else about your architecture remains unchanged (queues, logger, web server).

Testing the discovery logic

You really need to test this extensively before deploying it into the field. We plan to run at least 100 different boot processes with randomly assigned sensor-to-ports and verify that the binding was always 100% correct and the NVS is always stable. This will give us numerical evidence of the probe scoring mechanism and also ensure the NVS atomicity is robust enough to handle repeated write cycling.

To perform fault injection, we can intentionally short SDA on a single channel during discovery and then verify that all other channels continue to bind properly and that the firmware has reported the fault properly in both the CSV footer and the Web UI. Another type of testing is clone sensor testing, and it is where you have multiple sensors using the same address, and you test to ensure that the two-stage probe logic selects the appropriate sensor based on confidence levels.

To verify the Pin/Unpin workflow through the web application survives multiple power cycles, to ensure manual override persists after each reboot, and is displayed properly

in CSV headers. Lastly, measure discovery time per channel at 400 kHz Fast-mod, and it should be less than one second. Performance regression testing can then be performed by measuring this again whenever you modify anything in the future.

How it fits into the rest of the system
Discovery attaches to the wider firmware in three ways. The sampler task just has an array of driver handles from discovery and calls read on each one every 100 ms. It doesn't care what's underneath for hardware, just goes through the handles and populates sample structs. The logger and telemetry code operate with those generic samples with no sensor-specific branches.

The NVS-based config service keeps binding stuff with port, address, driver, units, scales, version, and raw detection evidence. This allows both auto-binding from discovery and manual overrides from ops. The HTTP server exposes /detects (rescan a channel) and /ports (get/set the full mapping). The web front end renders the 5-port grid and distinguishes between empty ports, auto-detected sensors, and manual overrides. It also distinguishes between I²C faults (can't talk to a sensor that's physically present) and empty ports (no response), thus allowing operators to utilize useful diagnostic info when troubleshooting in the field.

This whole design keeps the 10 Hz timing guarantees valid, makes field maintenance brain-dead simple (no manual config), and creates a complete audit trail in the CSV logs. The design works because it preserves three strict separations: physical separation with the mux, logical separation with the driver abstraction, and persistent state with NVS atomic writes. When those three pieces work, you get both flexibility for users and determinism for the real-time data path.

Related Work and Competitive Analysis
Several studies have demonstrated similar architectural goals using ESP32-based IoT sensor systems; i.e., Hassan and Hassan used an ESP32-based low-cost web-server to demonstrate the possibility of implementing an ESP32-based web interface for real-time photovoltaic system monitoring along with using the SD card for storing data logged from sensors [51]; They validated the fact that the ESP32 can perform concurrent tasks of sensor data sampling, writing data to the SD-card, and operating a web-server without having timing conflicts between these tasks.

Labib et al. performed investigations on efficient networking solutions for wireless sensor networks using ESP32 in both access-point and station modes [53]. Their performance evaluation of ESP-NOW and web-server modes provides empirical evidence for our selection of soft-AP mode for field-deployment scenarios. Most recently, Bautista et al. completed a comprehensive characterization of ESP32 for real-time synchronized sensor networks [49]; They identified key performance characteristics for our design decisions, including ADC limitations, network latency bounds, etc.

Additionally, from a software architecture viewpoint, Balakrishnan et al. proposed a framework for IoT sensor data acquisition and analysis [42]; They demonstrated

patterns for collecting, processing, and displaying multiple sensor data streams, which is similar to our queue-based sampling and telemetry architecture. Additionally, the development and reliability of FreeRTOS, our underlying RTOS, was extensively studied by Guan et al. [41]; They studied ten years of FreeRTOS releases and confirmed the deterministic behavior of FreeRTOS and its suitability for use in real-time embedded applications.

Although these works addressed individual aspects of embedded sensor systems (i.e., web serving, data logging, real time operation, or sensor networking) OpenSense is the first integrated solution to combine all of these capabilities in one single, field-deployable platform; OpenSense includes features that are not available in existing solutions, such as automatic sensor detection, hot-swapping support through I2C multiplexing, and atomic configuration management, etc. Therefore, the major contributions of our study lie in the integration of these features in a field-operable manner and the addition of additional features to improve the robustness of field-operation, including sensor auto-detection and graceful fault recovery.

3.4 Wireless Communication

Wireless communication is necessary for the OpenSense sensor to send and transmit data, accept configurations, and update firmware without having to be plugged in. Thanks to the capabilities of the ESP32's built in radios, the device can publish temperature, brightness, pressure/weight, and optical channel reading to a phone or laptop to check them without having to plug in the device itself. When a measurement session starts, the ESP32 streams values over the local network and mirrors them to the microSD card for reliability. Users can view live plots, start/stop logging, set sample rates, and push calibration constants from a browser—no app install required.

On first power up, the sensor will attempt to join a Wi-fi and then run in station mode and expose a small UI for dashboards, CSV download, and OTA firmware updates. When in the field, the unit can use access-point mode to connect to a phone.

3.4.1 Wifi vs Bluetooth Comparison

Table 3.20 - WiFi vs Bluetooth

Aspect	Wi-Fi (ESP32)	Bluetooth LE (ESP32)
Power	Higher ~80-200mA avg active 10-20mA sidle	Very low: ~2-15mA active; 0 in sleep
Throughput	High: 10-30+ Mbps	Low-Moderate: 0.27-1.4 Mbps
Range	20-50 m typical indoors	10-30 m typical
Multi-user	Easy with standard IP	Limited concurrent connections

Table 3.20 - WiFi vs Bluetooth		
Features	Web UI, OTA, TCP/UDP, NTP, mDNS, file/log serving	GATT services for config/telemetry/ No native web/files
Provisioning	~5mANeeds SSID/PSK	Simple Pairing
Interference	2.4 GHz crowded	2.4 GHz
Security	WPA2/WPA3, HTTPS possible	LE Secure Connections

As shown above, Wi-Fi and Bluetooth each come with different capabilities and strengths. WiFi will definitely be used a lot but BlueTooth will see some use as well.

3.4.2 WiFi

WiFi will serve as the primary link. It provides the bandwidth needed for multi-sensor streams and optical spectra and it supports standard IP tools. Typical operation uses station mode with AP fallback for provisioning. Power is usually managed via modem-sleep between bursts to keep the average draw low.

3.4.3 Bluetooth

Bluetooth is less necessary for the sensor but since the ESP32 already provides capabilities for it then it is important to keep the option open as other things that could be implemented in the future could rely on Bluetooth. Bluetooth Low Energy is more for quick set up and low power accessories. For the most part, for bulk data, dashboards, and firmware updates the system will prefer Wi=F

Chapter 4 - Standards and Design Constraints

4.1 Industrial Standards

Our OpenSense sensor makes use of many different industry standards and so some of them will be listed below.

4.1.1 I²C Protocol Standards

The Inter-Integrated Circuit (I²C) protocols are a standardized method of connecting integrated circuits (ICs) together. As our design focuses on using a variety of sensors, it relies on these I²C protocols a lot.

I²C works on a master and slave system in which you have a master device that initiates and controls data transfers with one or more different slave devices. There can also be

multiple master devices under this protocol if needed which is useful when using more than one microcontroller. Below is a diagram of how it works. .

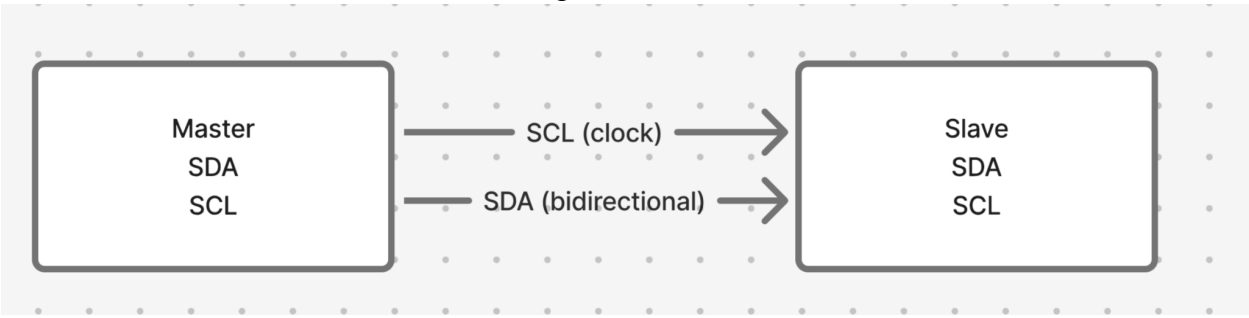


Figure 4.1: I²C Diagram

The master and slave both send and receive data from each other through the SDA (Serial Data) line although the master has control over starting and stopping the data transfer. The master also sends the clock signal to the slaves through the SCL (serial clock) line which gives the system synchronization. The I²C also uses an open-drain system which functions in a way to prevent slaver devices from pulling the devices to high voltage and can only drop them to low. However the system can still be pulled up when no device is being used through pull up resistors implemented inside the system.

Parameter	Typical I2C Specification
Number of signal wires	2 (SDA, SCL)
Bus Type	Serial, multi-drop
Signaling	Synchronous (clocked by SCL)
Standard-mode speed	Up to 100 kbps
Fast-mode speed	Up to 400 kbps
Fast-mode Plus speed	Up to 1 Mbps
High-speed mode	Up to 3.4 Mbps
Ultra-fast mode	Up to 5 Mbps
Number of masters	Multi-master supported
Maximum number of slaves	Up to 1008 devices with 10 bit addressing

Figure 4.2: I²C specs chart

I²C transfers data through messages sent from the master to the slaves. Each of the messages is broken into frames of data that contains the binary address of the slave as

well as the start condition, stop condition, address frame, read/write bit, and the ACK/NACK bit as shown below.

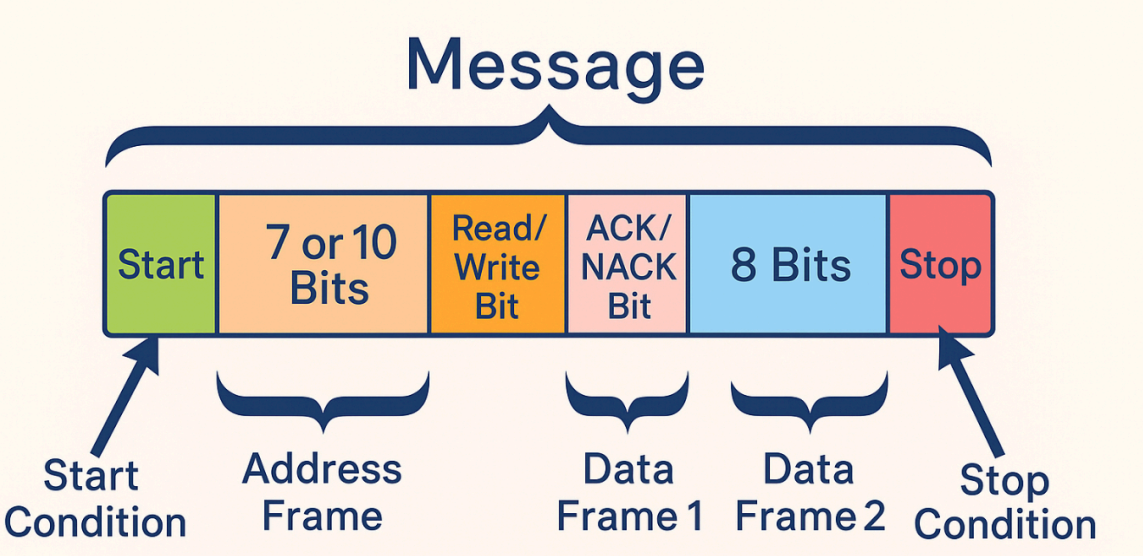


Figure 4.3: Data message composition

I²C data transmission begins when the master sends the start condition to all the slaves connected to it by switching the SDA voltage from high to low and then switching the SCL line from high to low. Then the master will send a 7 or 10 bit address to each slave that it wants to talk with as well as a read/write bit. The slaves then compare the master address to each of their own addresses and if they match then they return an ACK bit by pulling the SDA to low for one bit. If they do not match then the SDA line is left on high by the slave. Then after the master receives the data frame, the receiving device returns another ACK bit to show that it has successfully received the frame. Once it's done, the master then switches the SCL to high and then switches the SDA high to trigger a stop to the slaves.

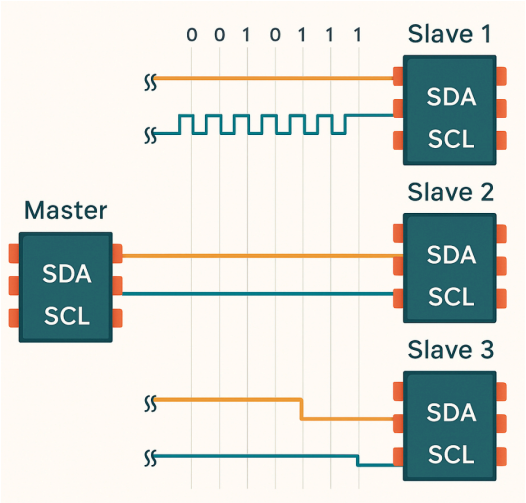


Figure 4.4: I²C Process

[66] [67]

4.1.2 PCB Design Standards

As our project requires a PCB, we must adhere to the standards of the manufacturer which in this case will be JLC PCB as they are the most reliable for quality and shipping. Therefore we must follow JLC PCB's standards for our board design which means it must meet criteria such as only being able to use 1 to 2 oz copper for the outer layers. The board also requires pads to be placed .0127 mm distance from different pads and .2 mm away from tracks. The holes must also be between 0.3 and 6.3 mm and cannot be bigger than 0.3 mm holes and 0.55 pads. Also it must be a 0.3 mm distance between the track and the edge of the board.

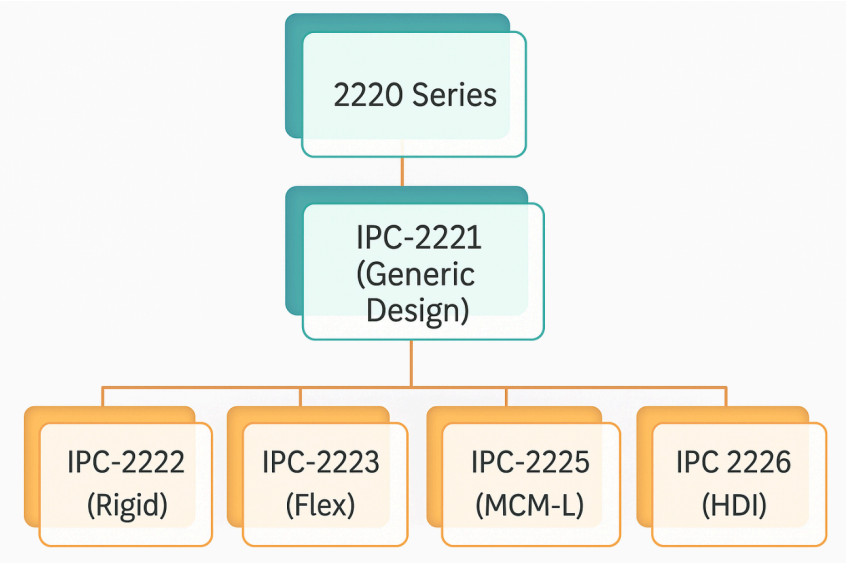


Figure 4.5: IPC-2221 standards

The PCB must also make sure to follow the IPC-2221 standards for PCB's. These standards were established for the electronic manufacturing industry and lay the foundation for the design, assembly, packaging, interconnection, material, performance, and inspection specifications for the electronic industry. Although IPC-2221 is the standard for most PCB's, it is also used in conjunction with other IPC's that are a subset of it.

IPC-2222 is used for rigid boards and focuses on:

- Holes and interconnections specifics
- Dielectric Spacing
- Selection of materials
- Routing Parameters
- Board Thickness Tolerance
- Mechanical Parameters

[68]

IPC-2223 is the sectional standard for flex circuits and focuses on:

- Material selection and construction
- Selective plating
- Minimum bend radius for flexible circuits
- How to achieve controlled impedance
- Unsupported edge conductor design
- Filling of via
- Pad placement

[68]

IPC-2224 consists of the standards for designing PWBs with PC Cards and focuses on the following types of boards:

- Type 1: Single-sided circuit board
- Type 2: Double-sided circuit board
- Type 3: Multilayer board without blind or buried Vias
- Type 4: Multilayer board with blind or buried Vias
- Type 5: Multilayer metal core board without blind or buried Visas
- Type 6: Multilayer board with metal core or buried vias

[68]

IPC-2225 is for designing organic modules (MCM-L) and MCM-L assemblies and incorporates adhesive details, attached dies, and microvia materials
[68]

IPC-2226 are the standards for designing HDI boards and consists of:

- Type 1: Through vias connecting the outer layers
- Type 2: Buried vias in the core and may have through vias connecting the outer layers
- Type 3: The core may have buried vias or through vias connecting the outer layers
- Type 4: Where P is a passive substrate with no electrical connections
- Type 5: Constructions without a core
- Type 6: Other constructions

[68]

IPC-2221 matters in high-voltage design because it addresses the issues of the possibility of anodic filamentation as well as the risk of dielectric breakdown in a strong high-electric field. The strong field can be eliminated by choosing the correct material, spacing, and insulation as well as by keeping it clean. Also the standard specifies the minimum spacing details that will allow high voltage to pass through two conductors and they are evaluated with respect to peak DC or AC voltage. IPC-2221 clarifies the exact minimum spacing needed for voltage to pass through at around 500V as well as calculate what you need when it passes 500.

Voltage range between conductors (V)	Internal Traces (mm)	External, uncoated ≤ 3050 m (mm)	External, uncoated > 3050 m (mm)	External, coated (any altitude) (mm)
0-15	0.05	0.10	0.10	0.05
16-30	0.05	0.10	0.10	0.05
31-50	0.10	0.60	0.60	0.13
51-100	0.10	0.60	1.50	0.13
101-150	0.20	0.60	3.20	0.40
151-170	0.20	1.25	6.40	0.40
171-250	0.20	1.25	6.40	0.40
251-300	0.20	1.25	12.5	0.40
301-500	0.25	2.50	12.5	0.80
> 500	0.025 mm/V	0.005 mm/V	0.025 mm/V	0.00305 mm/V

Figure 4.6: Minimum clearance values for bare board to pass voltages

4.2 Practical Constraints

4.2.1 Time

One of the biggest constraints for this project is the time factor. We are given 2 semesters to fully construct a deliverable that can be demonstrated and so we have to use our time wisely to make that happen. For the research portion we divided our assigned tasks by week so that we knew what was expected of us and it helped with time management. We also made sure to have weekly check-ins or meetings if we were able to schedule them and often met up at the library or a spot that was easy for all of us to meet up at.

One of the major disadvantages of having limited time is that it will be difficult to do extensive testing and debugging. We will need to design a PCB for the sensor which takes time and we also need to order it from JCL PCB which can also take weeks to arrive. This also doesn't take into account that we will most likely have to make adjustments and then order another revised board so it is important to do the design and order the first board early so we have more time for testing and revisions. Each error and setback will cost us precious time if we don't catch it in time and so it is important to always double check the designs and receive feedback on them before ordering the PCB in order to try and save as much time as possible.

Another big delay is the time lost in coordination. During this limited time, a lot of team members lead busy lives working or preparing for graduation with interviews. As a result it can be hard to coordinate a time in which everyone is free to meet in person or via discord call to do a meeting and check progress. Our team has been good in communicating our availability so far however as the project approaches its final stages and graduation looms closer things might get more difficult. This also impacts communicating with our advisors as many of them are very busy and also have to deal with other students and so finding a time to schedule a meeting and receive feedback on the project can be a bit tedious, especially if the feedback affects our direction on the project.

Each of these factors results in time loss and if not properly handled could lead to a rushed project and will negatively impact our quality or cause drastic measures to be taken. As this is the senior year of everyone taking this class, it is important we try and avoid this result as much as possible as another delay in graduation could negatively impact our future careers with the way the job market is going.

In order to prevent this, our team will continue to coordinate and plan to make sure we make the most of our time and prevent any situations where we fall behind. By making soft deadlines, we can motivate each other to stay on task. Another big way of doing this is to make sure to attend every lecture together as we would ensure we are all on the same page and is also a way to make sure we are all on campus so we can meet up in person to further increase productivity.

4.2.2 Money

Another big constraint for this project is an economic one. Although our project is generously sponsored by D.R.A.C.O Labs and Dr. Mike, our team has vowed to try and keep the project on budget. As part of the project proposal, the sensor is also supposed to be an educational, affordable and open source alternative to some of the other existing sensors. As a result, our team has decided to try and keep the project budget to be around $600 and for the production cost for each sensor to be no more than $60 in order to try and lessen the financial strain for our sponsor as he has set no formal limit but we still don’t want him to have to spend too much of his own money when he has other groups he is sponsoring as well.

In order to ensure we keep the project on budget, we are making sure we pick affordable components as well as ensuring the design is as sleek as possible so that it won’t cost too much to produce. For the most part this hasn’t affected the project too much as we were already planning for a smaller scale project and so most of the components we need are luckily on the affordable side. However we will also be dealing with optical components and sensors that are a bit more expensive and so we must ensure that we have money to spare in case one of the components breaks and we have to replace it. One of the biggest cost factors will be ordering the PCB as our design will have some things that will need to be built into it like the USB-C port which will cost extra and we will of course need to order multiple boards at once as insurance

to ensure that if one breaks then we have backups and also for testing. However when it comes to ordering the boards, the shipping costs will be what eats up some of the budget. JLC PCB is based in China so depending on the size of the order the shipping cost can get quite expensive, and if we end up needing to adjust the size and have to order one or two more batches of prototypes, that exponentially increases the cost.

To try and cut costs, we will also try to solder and assemble as many parts of the board and sensor as possible rather than paying someone else or having it custom ordered. This does require a bit more skill but it beats the alternative of wasting a portion of the precious budget on things that are not completely necessary as well as improving our own skills as well as making do with what we have and trying to do things ourselves. Other than for Dr. Mike's sake, reducing our spending on the project is also important for making sure we have enough of the budget left over in case complications arise as things tend to happen regardless of how much we plan and prepare. Whether it's an important component breaking or receiving the wrong component in the mail, money is the only way to fix these issues in a timely manner and so properly budgeting our costs to make sure we have an emergency fund for this type of situation will help us out a lot in the long run.

4.2.3 Manufacturability

Another constraint our team needs to deal with is that as undergraduate students we need to ensure that our project is something we can actually feasibly make. The group mostly consists of Computer Engineers and so not all of us are experts when it comes to the electrical side of things as well as processes like soldering. However we have at least learned the basics due to our Junior design lab and also one of the members is an electrical engineer.

Part of this means making a PCB that isn't too complex considering all the components we will need that make up our sensor. Making a small mistake in the design such as using the wrong type of trace or not having the power source correctly connected could lead to big problems if not caught in time as we would lose a lot of time and money having to order another batch to fix such a small error.

Another issue is that many popular components are often unavailable. For example the BQ2704 that we plan on using has 3 variations on Digi key and yet one of them is currently unavailable. Luckily we have the alternatives and they work fine but something like that could happen at any time to a critical component which could cause a major issue if we are not able to find an alternative in time.

Also due to the complexity, most of our components will most likely be ordered already built into the board from services such as JLC PCB. These components would be too small or complex to solder onto the PCB ourselves so as much as we would like to cut costs in some places, it is important to know our limitations and keep things feasible. By relying on services such as these as well as asking for help in the senior design lab or from advisors we can ensure that we will be able to properly manufacture our project without any issues.

We must also ensure that our design remains durable as we plan on our sensor being able to be used for a long time. If it were too easily damaged either externally or by the components inside not being secure then it could lead to big problems down the line considering we plan on making this an open source project that people will be able to use for educational purposes. In order to prevent this we need to make sure the wiring and design meet industry standards as best as they can to help them last long. In order to address these constraints as best as we can, we will make sure our design is as solid as it can be and double check that the PCB layout has no obvious mistakes and have our advisors review it as well before even considering shipping it out to be manufactured.

4.2.4 Remaining Constraints

Safety
In order to meet the safety criteria for this project , we were unable to make use of some ideal components such as the Lithium Polymer Battery. This battery would have been more flexible to use due to its high quality as well as how much more compact and flexible it is, however this type of battery also poses severe safety concerns. The LiPo battery is known for being responsible for many dangerous fires due to exploding or being misused and so as a result it has been banned from the lab which caused us to go with a Lithium Ion battery instead which is safer.

We are also asked to make sure the Lithium ion batteries that we will use have the proper cover attached to keep them safe as well since otherwise they can be just as dangerous as Lithium Polymer batteries. We also have to be extremely careful when bringing them into the lab and also have the proper safety equipment when handling them or else will suffer the consequences. We also have to be careful with some of the equipment from the Spectrometer as if handled improperly the laser could lead to some damage to eyesight or other harm. Therefore we need to take care to make sure to prevent any potential hazards and keep the entire project as a whole safe.

Scalability
Another constraint our team needs to deal with is the scale of our project. As a sponsored project and one that is succeeding a previous version, it is a serious possibility that if our project is a success we may end up with a popular sensor used by many. As a result we need to take into consideration how accessible we make it in terms of cost to produce as well as implement ways for it to be able to expand in the future. For a project like this, an easy way to try and achieve that is to leave the possibility open to be able to implement other custom sensors so that if the user needs to they can plug in their own sensors to measure what they need while using our platform to record the data. We could also see how possible it is to increase the maximum number of sensors that can be supported at once which would certainly make it easier for the project to expand in the future.

Ethical
Another very important aspect of this project is to make sure we keep things ethical. One of the best ways to do this is to make sure we get our components and information from ethical sources that do not exploit their workers as that would mean we are contributing to that. Therefore we shall keep this in mind when selecting components and also make sure that we are keeping a balance with our other goal to keep the project affordable. Another source of ethical constraints is that our sensor is meant to be used for educational purposes. This means that we have to ensure that the results of the readings will be as accurate as possible in order to make sure we are not misleading others with the data they gather as it is our ethical responsibility to make sure our product functions correctly the way it is intended. It would be very unethical to knowingly create this sensor knowing it would not give others the best and accurate results as it possibly can.

Power
One of the constraints given to us by our sponsor was to try and have the battery life of the sensor last as long as possible. This constraint is an important one as it leads us to many of our design decisions such as what battery to use and what components are the most energy efficient. As we were unable to use the Lithium Polymer battery we instead chose the next most efficient battery as the Lithium Ion battery should be able to supply our sensor with all the power and voltage it needs.

As a result of this it also impacted some of the other choices such as which MUX and regulators to choose to incorporate. It also led to us choosing to incorporate the BQ24074 battery charger that would allow us to connect the USB-C and charge the battery when not in use to help it retain more battery life.

With as many optimizations as possible, we believe our goal of the battery lasting more than 2 hours should be achievable although it might require some trial and error on how we arrange the battery and the other components to reduce battery drain as much as possible. This was also why we didn't go with some of the worse battery options such as the NiMH which has a lot of voltage loss over time so the battery would not have lasted as long as we would have wanted it to.

Chapter 5 - Application of ChatGPT and other similar platforms

5.1 - An Overview

Over the past few years, Artificial Intelligence (AI) has developed at an extraordinary pace, evolving from a few niche areas of computer science into one of the most influential technologies of the modern era. It is now being used across nearly every field, from healthcare and engineering to education and business, as a tool that

enhances productivity, understanding and problem-solving. At its core, AI is the ability of a computer system to imitate human cognitive functions such as reasoning, interpreting information, learning from experience and responding to new situations. These capabilities are built upon large datasets and complex algorithms that enable the system to improve over time through self-teaching and training. One of the most significant turning points in the widespread adoption of AI came in 2022 with the release of ChatGPT by OpenAI. This platform popularized the concept of prompt-based interaction, where users input requests or questions (similar to how a search engine would work), and the AI generates meaningful, context-aware responses. Through this interaction, users can engage with the technology almost as if they were conversing with another person, which makes it accessible to both professionals and casual users.

According to a blog post from the University of Cincinnati Online, AI's adaptability has made it a valuable resource in numerous industries, including education and scientific research. The post explains how "we can train an AI program to analyze data with remarkable speed and accuracy," emphasizing the way AI amplifies human potential rather than replacing it[69]. This perspective aligns closely with how students and researchers can benefit from the technology in academic settings. For example, students can use AI to explore complex subjects more efficiently by generating explanations, comparing theories, or summarizing key concepts from various sources. Instead of spending hours sifting through material, they can quickly obtain a structured overview and then dive deeper into areas that require critical thinking or verification. Furthermore, AI can strengthen the research process by providing citations or references that allow users to verify the credibility of information. This encourages active learning, not blind reliance, as students can confirm, question, or refine the responses they receive. The result is a more dynamic learning process that combines speed with comprehension. Similarly, in scientific research, AI accelerates discovery by helping researchers analyze massive datasets, identify patterns , and form new hypotheses in a fraction of the time it would take through manual effort. In essence, AI acts as a bridge between information and understanding. When used responsibly, it enhances our ability to learn, create and innovate. Whether in classrooms or in a variety of market areas, AI's strengths lie not only in processing information rapidly but in empowering humans to think more critically and achieve results that once seemed out of reach while being significantly faster.

5.2 - Limitations

While AI has become an incredibly powerful and effective tool in recent years, it is not without its limitations. One of the most critical of these which is often overlooked by casual users, is the accuracy and reliability of the information it provides. While AI systems—especially large language models (LLMs)—are able to provide highly accurate and insightful answers to many types of questions, there are many times when they are inaccurate—sometimes significantly so. And because of the nature of AI technology, this can be an extremely serious issue for people who don't have a background in evaluating the accuracy of the information provided by AI systems. One

of the main concerns regarding AI is that it can provide inaccurate information with the same degree of confidence as it provides accurate information. As a result, users may view flawed or even entirely false information as fact solely because the AI sounds so sure of the information. Authors Bruce Schneier and Nathan E. Sanders discusses this topic in depth in their article *"AI Mistakes Are Very Different From Human Mistakes"* posted on IEEE.org. They explain that the primary differences between human mistakes and AI mistakes is that human mistakes are typically accompanied by some type of hesitation or visible uncertainty while AI mistakes are always presented with complete confidence. Therefore, AI mistakes can be particularly damaging to users, as users are unaware that they are being misled. The article explains, "A LLM will be just as confident when saying something completely wrong." This represents a fundamental distinction between human fallibility and machine-generated error: AI generates errors without providing any obvious indications of the user that the information could be uncertain[70].

Additionally, the increasing use of AI in academic, professional, and research settings exacerbates the problem of errors generated by AI. Many users currently depend on AI tools to assist in summarizing complex topics, draft emails and essays, write code, and answer technical questions. Since many users treat AI-generated responses as the final authority without cross-checking them against reliable sources or subject-matter experts, users may inadvertently spread misinformation or make poor decisions based on erroneous data. Therefore, responsible usage of AI is vital for using the tool efficiently. AI should never be used as a replacement for critical thinking or domain expertise; instead, AI should be used as a tool that supports both. While AI can be a useful resource for brainstorming, preliminary research, and breaking down complex concepts into simpler terms, users should verify any conclusions or outputs produced by AI against credible, human reviewed resources. Users should also maintain a healthy amount of skepticism about the information provided by AI and recognize the purpose of AI as a tool.

An essential part of using AI responsibly is recognizing how AI models similar to ChatGPT gather information. Unlike accessing a live, real time database or the internet, AI systems are trained on massive databases of pre-existing content. The databases contain items such as books, articles written by scientists, websites, Wikipedia entries, open-source coding repositories, and digitized texts of various subjects. By analyzing the patterns, relationships between words, grammatical structures, and contextual structures contained within the databases, AI systems are trained to develop predictive capabilities based on probability and the most likely sequence of words based on a user's input. Although this method has several benefits, there are also many limitations associated with the process.

One limitation is that the training date can introduce bias into the system. Because all of the content included in the databases were created by humans, the content can reflect a variety of different human biases—whether they are intentional or not. For example, if the database contains disproportionate amounts of viewpoints that are biased toward one side or another, or if the database contains offensive stereotypes, the AI system

may unintentionally replicate and continue those patterns in its outputs. In the article *“AI Demystified: Introduction to Large Language Models”* published by Stanford University IT, the author indicates that "the model might produce outputs that reflect these biases, leading to ethical concerns and potentially harmful outcomes." These potential problems can range from very subtle examples of stereotyping in responses to the promotion of false or damaging narratives [71].

Another limitation is that AI models are trained on static databases and therefore are unable to automatically update themselves with new information once they have been trained. Therefore, AI models lack real-time awareness of the world and are unable to expand their understanding of the world past the limits of the data they were initially exposed to during training. Therefore, AI models can provide outdated, obsolete, or even factually incorrect information related to rapidly changing topics such as current events, scientific developments, or public policy. For example, if an AI model was trained on data up until 2023, it would not have the capability to reference or draw upon any materials that have been developed or published since 2023 unless the model's developers specifically train or update the model after 2023. Additionally, AI models do not access the internet in the same manner as a human researcher does. Instead, they are generative models that create new text based on statistical patterns that the model develops during the training process and not based on fact checking.

5.3 - The Pros of AI

Although AI comes with notable limitations, it would be a disservice to focus solely on its shortcomings while ignoring the profound and growing benefits it offers. Large language models (LLMs) like ChatGPT have already made a significant impact on modern life and society, particularly in the realms of productivity, innovation, and the future of work. These tools are not only widely adopted but are increasingly encouraged, and in some industries required, as part of everyday workflows. Their influence spans a wide range of sectors, from business and finance to education, healthcare, research and more.

One of the most immediate and measurable ways AI improves workplace productivity is through automation of repetitive tasks, generation of code, organization of data, synthesis of information, and assistance with writing and communication. All of these activities allow workers to complete more in a shorter amount of time. Improved productivity enables workplaces to operate more efficiently. The Federal Reserve Bank of St. Louis Blog Post “The Impact of Generative AI on Work Productivity,” authored by Alexander Bick, Adam Blandin and David Deming, provides the results of a survey that clearly illustrates the extent to which AI is currently being utilized throughout the workforce. Approximately one third (31.9%) of respondents reported utilizing generative AI for one or more hours per workday during the past month, while another 47.0% reported utilizing generative AI for between 15 and 59 minutes each workday. These statistics illustrate that a significant segment of the workforce utilizes AI tools regularly. The study also examined the effects of that utilization in terms of the amount of time saved. Twenty five percent (25%) of respondents reported that AI saved them 4 or more hours in the last week, 20.1% reported that AI saved them 3 hours, 26.4% reported that

AI saved them 2 hours, and 33.0% reported that AI saved them one hour or less. Overall, more than half of the study participants reported that they saved at least two hours each week as a result of utilizing AI tools. The results of the study demonstrate that AI can assist in enhancing productivity and efficiency within the workplace and decrease the amount of time spent completing repetitive tasks, thus permitting employees to complete high-level tasks in addition to low-level tasks, in a timely manner. Furthermore, the increased productivity generated by time-saving generative AI tools empowers employees with increased job satisfaction, quicker turnaround times, and greater quality of output, ultimately creating a more agile and competitive work environment [72].

In addition to its productivity-enhancing capabilities, AI (specifically LLMs) plays a vital role in the acceleration of innovation. AI's capability to rapidly analyze, synthesize, and derive insight from massive amounts of data facilitates the rapid creation of new ideas, testing of hypotheses, and development of new products at a pace previously unthought-of. The capability of AI to identify relationships and trends within complex datasets is a powerful catalyst for discovery and advancement. The article "The Effects of AI on Firms and Workers," published by the Brookings Institution, examines the relationship between AI investments and real-world innovation outcomes. Utilizing firm-level data collected from 2010-2018, the authors of the study found that companies that increased their AI investments during this period realized substantial increases in innovation metrics. Specifically, the study found that companies increasing their AI investments realized a 13% increase in new trademarks and a 24% increase in product patents. The authors interpret these results as evidence of what they refer to as "AI-fueled growth" — where the primary purpose of the utilization of AI is not merely to optimize internal operations, but to create new products and services. The article states that the findings support the notion that companies utilize AI to innovate primarily in the product space, rather than the process of innovation and improved efficiency." This is especially important because it demonstrates that AI's impact is not limited to efficiency, but rather it is facilitating companies to create entirely new offerings, design more intelligent systems and bring new products to market faster. Rather than merely enhancing existing processes, AI acts as an intellectual partner in the innovation process, enhancing the creative and analytical abilities of human teams. As AI tools continue to grow in accessibility and usability, they provide users with the ability to innovate. Small groups of individuals and even individual innovators can now utilize the computational power and analytical reach of AI to accomplish tasks that previously required multiple departments of specialists. From developing new software applications to designing more intelligent consumer products, AI has created a new pathway for innovation in various marketable areas [73].

5.4 - The Cons

In a previous section, we explored the limitations of AI and the negative consequences these limitations can have on users, particularly when misinformation is accepted without verification. However, the reach of AI’s impact extends far beyond individual users. It also affects those who do not interact with AI directly, as well as the broader

societal and environmental systems in which AI technologies operate. As language models and AI systems become increasingly embedded in the infrastructure of modern life, it is crucial to examine the secondary and systemic drawbacks they may introduce.

An important, and increasingly visible, environmental concern related to AI is the environmental "footprint" of the AI technology itself. Unlike software applications which are hosted on mobile or desktop devices, or lightweight cloud-based processes, advanced generative AI systems require vast computational resources to function. As such, these systems are usually trained on large-scale GPU clusters located within massive data centers. Data centers, in turn, represent the physical environment in which large banks of servers, large volumes of cooling equipment, extensive networking infrastructure, and large storage systems are contained. These facilities require substantial amounts of electricity to provide a constant operation. Adam Zewe's MIT News article, *"Explained: Generative AI's Environmental Impact,"* provides detailed information regarding the sizable amount of energy required to support AI systems, specifically large language models including, but not limited to, ChatGPT. Specifically, the article outlines that one of the primary drivers of the environmental impact of AI is the explosive growth of data center infrastructure upon which modern AI research/development is built. The climate-controlled environments described above are necessary to house the server-heavy hardware utilized in the training and operation of LLMs, while consuming a staggering amount of electricity in the process. According to the article, data center power consumption in North America grew from an estimated 2688 MW to 5341 MW between the end of 2022 and the end of 2023 — with this growth being partially attributed to the demand of generative AI. While this represents an additional 2,653 MW in just one year alone, scientists estimate that global power consumption of data centers is currently 460 TWh and by 2026 will be approximately 1050 TWh — representing nearly a doubling in just four years. For perspective, if realized, this would equate to greater than the total annual electricity consumption of several developed countries and thus has a profound potential for negatively impacting the planet in terms of sustainability [74].

Additionally, to gain an understanding of the computational resource utilization associated with training individual models, the article references a 2021 study authored by researchers at Google and the University of California, Berkeley. The authors of the referenced study found that training the GPT-3 model (which is one of the most well-known language models) resulted in an estimated 1287 MWh of electrical energy use during the training process. This estimate includes the energy utilized to process the hundreds of billions of parameters for weeks or months. Additionally, the environmental costs of this energy utilization were estimated to be in excess of 552 tons of CO2. A comparison can be made to a typical gasoline-powered automobile which generates only approximately 4.6 tons of CO2 annually (as stated by the U.S. Environmental Protection Agency). These numbers illustrate the environmental impact of AI in a way that is both tangible and concerning, and will likely become even more challenging to manage as the data centers supporting these AI models become more technologically sophisticated (as indicated by the data presented in the article). Furthermore, the increasing number of AI models and services being adopted across

multiple sectors (such as healthcare, education, and finance) implies that an increasing number of organizations, both public and private, will utilize AI-related infrastructure and therefore contribute to the increasing trend of the environmental impact of AI. If the environmental impact of AI continues unabated, the carbon footprint of AI may ultimately hinder the ability of organizations to meet the environmental goals outlined in international agreements (such as the Paris Accord), and/or national clean energy initiatives [74].

As an additional economic implication of AI, is its broader economic influence that goes far beyond just the creation of new technologies or increases in productivity. With the ability to provide businesses with efficiencies, through the use of automated processes, and reduce their labor costs through the use of machines and technology, AI has created many economic issues that could negatively affect individual households and industries as a whole. For example, one of the biggest negative aspects of AI is the possibility of large-scale job loss due to the fact that AI systems will perform tasks that were traditionally performed by humans. Another issue is the increased cost of energy that comes from the high demand for electricity needed to power the equipment and infrastructure used to support AI systems. Ultimately, this could lead to higher energy bills for all consumers. These two issues create a larger question about the long term economic viability and fairness of the development of AI. In a CBS news article titled *"The AI revolution is likely to drive up your electricity bill. Here's why."* by Mary Cunningham, the author discusses a recent increase in electrical costs. According to Cunningham, the price increases of electrical costs are primarily due to the growth of Data Centers in the United States. Between 2021 and 2024, the number of Data Centers in the United States grew significantly from a report released by Environment America (a coalition of environmental organizations). As reported earlier in this chapter, the growth of Data Centers directly resulted in an increase in electrical usage, resulting in higher electrical costs to the consumer. The article continues to explain the effects of this growth of Data Centers and how they have affected the electrical costs of consumers. According to the article, the cost of electricity rose 4.5% during the past year (according to data provided by the U.S. Bureau of Labor Statistics) and in the state with the highest number of Data Centers, Virginia, the average electricity bill increased by approximately 44%, from $124.53 in 2017 to $179.73 in 2025. In addition to the increase in the cost of electricity for consumers, the other concern is how AI affects the incomes of consumers by replacing jobs and potentially eliminating the workforce. Although we do not know exactly when, it is possible that over time, AI will completely eliminate the need for workers. This raises the question: What will happen to the people who lose their jobs as a result of AI? We simply cannot predict the future. However, given the fact that AI is already beginning to displace some workers in various occupations, the outlook is bleak and the negative consequences of developing AI for society are apparent [75].

5.5 - Example of AI Use

5.5.1 Case Study 1

Please refer to Appendix E, [1a] and [1b] for the Case Study 1 given prompt and outcomes from ChatGpt. ChatGpt gave me a very helpful answer on figuring out how to choose the best battery for our project. I like that it gave me a large list of different factors while also looking up and giving basic information on some of the more popular types.

I did not tell Chat GPT that Lithium Polymer batteries were potentially banned and yet it gave its answer under the assumption that because of their danger that it was a possibility and yet still gave me information on it. The Average Power equation it gave out was also accurate and a good way to determine if the battery was suitable for the sensor. Overall the information it provided in this case was more than satisfactory and very helpful.

5.5.2 Case Study 2

Please refer to Appendix E, [2a], and [2b] for the Case Study 2 given prompts and outcomes from ChatGPT.This question was meant to see if ChatGPt was capable of giving a detailed definition of what I2C was as well as if it knew what I meant by standards.

It did a good job at gathering all the relevant information and simplifying it rather than spitting out a big blurb of text and even gave me extra information on things to take into consideration when choosing I2C for a sensor even though I hadn't asked it to. However I do wish that by default it would give me citations on where it gathered its information from.

5.5.3 Case Study 3

Please refer to Appendix E, [3a], and [3b] for the Case Study 3 given prompts and outcomes from ChatGPT. This question tested ChatGPT's ability to explain and visualize something with a diagram. It actually did a pretty good job at visualizing how the connection would look like in a way that seemed pretty accurate while also providing a good explanation on how the USB-C would connect to the battery and even gave recommended chargers to use for the task without needing to be asked to do so which was pretty helpful of it.

5.5.4 Case Study 4

Please refer to Appendix E, [4a], and [4b] for the Case Study 4 given prompts and outcomes from ChatGPT. This question helped me test ChatGPT's ability to research and find prices and also to compare them. Unexpectedly, it also did a good job in researching and pulling up sources while also linking the website it got it from to cite it which helps the user verify for themselves if the information is accurate rather than blindly trusting ChatGPT.

It also gave different categories in which the different PCB suppliers might be more helpful and as a result made it easier for our group to decide where to order from to be the most efficient while considering other options.

5.5.5 Case Study 5

Please refer to Appendix E, [5a], and [5b] for the Case Study 5 given prompts and outcomes from ChatGPT. This question was designed to see how well ChatGPT can generate a pros and cons list for a variety of options as well as to see if the pros and cons made sense.

Overall it did a pretty good job although it was unfortunately pretty vague for some of the pros and cons so you couldn't really tell how much better one option is from the other. It at least also offered to give more information and maps a power budget if I gave it more information which was pretty interesting.

5.5.6 Case Study 6

Please refer to Appendix E, [6a], and [6b] for the Case Study 6 given prompts and outcomes from ChatGPT. This last prompt tested to see if it could find various valid constraints and standards related to optics and apply it to the scenario of a sensor. It surprisingly ended up giving some good information while once again providing links to its sources so I could find more information for myself on these various standards and constraints. It was also interesting that it found a way to separate the various constraints into different categories and yet managed to find a good number for each one to highlight just how many different possible constraints were applicable for this situation. This serves as a good starting point when researching these constraints and standards for the report.

Chapter 6 - Hardware design

6.1 - Power Design Schematics

USB-C Connections

This is an overview of the area of the design where the USB-to-UART and Li-Ion charging functions take place. Beginning at the left side of the diagram is the USB-C connector (P1), configured as a USB 2.0 with VBUS on A4 that supplies the main 5.0V input; and D+ (A6) and D– (A7) supply the data lines for serial communication. These two lines connect to the MCP2221A, a USB-to-UART converter, that translates the USB data to UART level serial data signals for microcontroller communication. The MCP2221A is powered by VBUS from the USB line and has decoupling capacitors to ground (such as C12, 0.47 µF) for bypassing. The RX and TX lines from the MCP2221A are connected to the MCU via series 1 kΩ resistors R31 and R32 respectively, to ensure

signal integrity and some current limiting. SDA and SCL lines are present for optional I²C communications however they are not being utilized within this schematic section.

The second section of the diagram is for the MCP73831, a single cell linear Li-Ion/Li-Polymer battery charger IC. Like the MCP2221A, it's powered by VBUS, with its VDD pin tied to the USB 5 V input and a decoupling capacitor (C1, 0.1 µF) located nearby. The battery is attached to the VBAT pin of the MCP73831 (Pin 5), and like all other connections this has an output capacitor (C3, 4.7 µF) for stability. The charging current is determined by the value of resistor R4 (2 kΩ) connected to the PROG pin and the ground plane, using the formula described in the datasheet

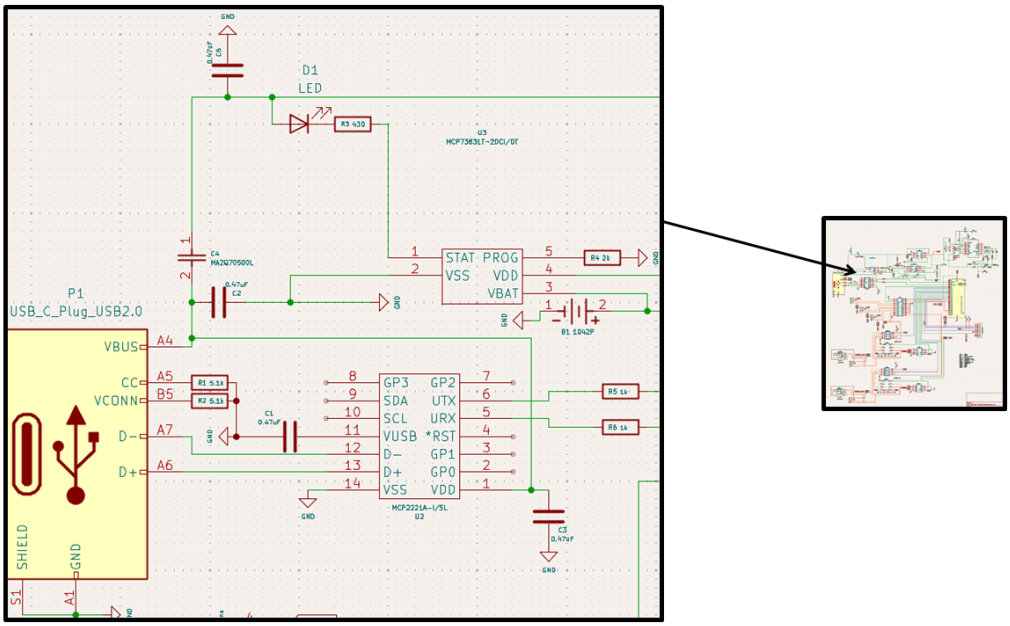


Figure 6.1: USB-C connection (USB-to-UART + Battery Charger)

(approximately 500 mA for 2 kΩ). There is an LED (D1) connected in series with a current limiting resistor (R30, 1 kΩ) to the STAT pin (Pin 1), which will illuminate the LED during charging because this pin is pulled low when charging. The positive terminal of the battery connects directly to the drain of Q1 (a SOT-23 P-MOSFET), which is a component of the system's power switch logic and is a part of the overall power management circuit that is not visible in this enlarged view. Overall, this section allows the USB-C power source, the onboard battery, and UART communication to the microcontroller to be interfaced together for both system recharge and serial communication over a single USB-C port.

Power Management

The TPS2111A is used to switch sources of power to provide a 5.0V supply to the system. Power is derived from either the USB port, via the USB-C VBUS rail, or from the internal lithium ion battery via a battery-powered 5.0V boost converter. The boost

converter increases the internal battery voltage to the required 5.0V level to provide a stable 5.0V supply to the system when the USB port is disconnected.

The logic controlling the source selection is implemented via the use of two control pins (D0 and D1) on the TPS2111A. The state of D0 determines if the boost converter is enabled or disabled; if D0 is low (i.e., logic low), the boost converter is enabled; if D0 is high (i.e., logic high), the boost converter is disabled. D1 is used to determine the active input to the TPS2111A; if D1 is high (i.e., logic high), the TPS2111A selects the input associated with IN1; if D1 is low (i.e., logic low), the TPS2111A selects the input associated with IN2. Since there are no microcontrollers in this implementation, the selection of the active input to the TPS2111A is determined by the state of D1; when a USB connection is made and the USB-C VBUS rail is present, D1 is high, and the TPS2111A selects the input associated with IN1, which is the USB-C VBUS rail. When the USB connection is broken and the USB-C VBUS rail is absent, D1 is low, and the TPS2111A selects the input associated with IN2, which is the output of the boost converter. The boost converter consists of the LM2623 and a number of passive components including an inductor (L1), a diode (D2), and capacitors (C6 and C7). The output of the boost converter is filtered by the capacitors to reduce the switching ripple associated with the boost operation, and the output of the boost converter is stabilized by the capacitors. The output voltage of the boost converter is controlled by a voltage divider formed by the feedback resistors (R6 and R7) connected to the FB pin of the LM2623. The voltage divider sets the output voltage to 5.0V by controlling the internal regulation loop of the LM2623.

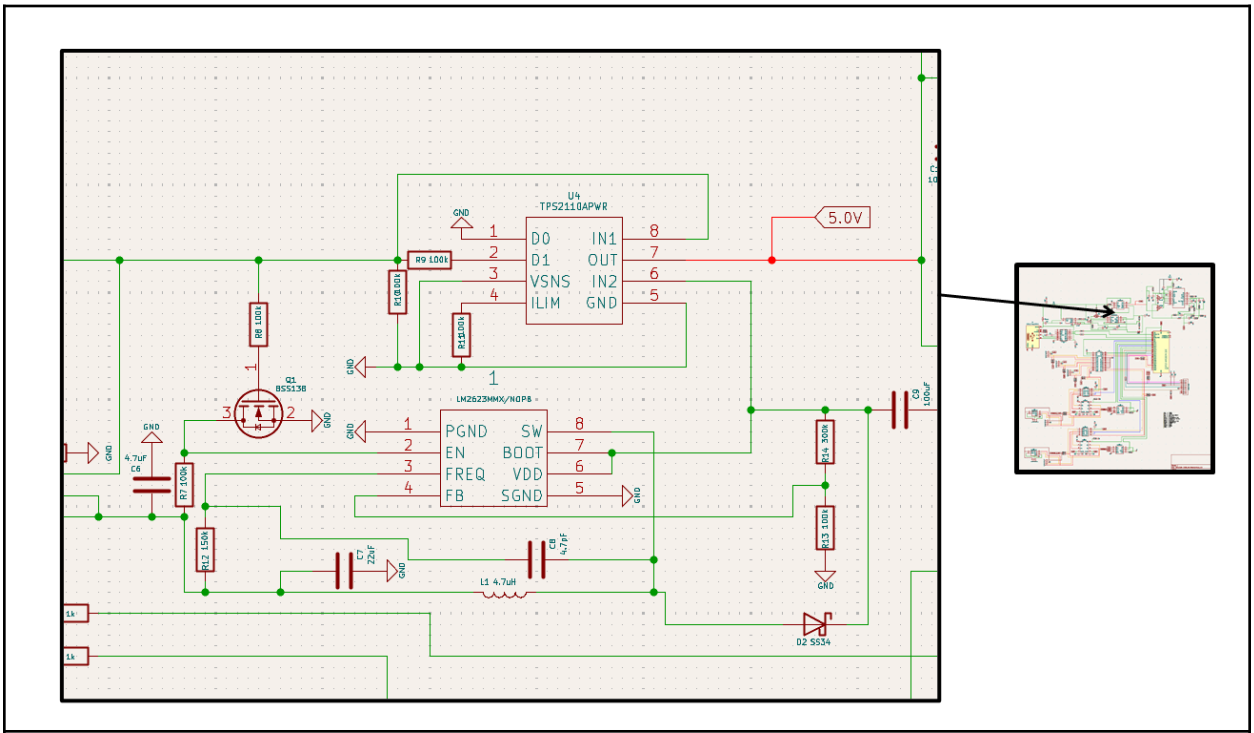


Figure 6.2: USB-C Sense and 5.0V Boost Regulator

The enable/disable function of the boost converter is performed by a P-channel MOSFET (Q1) connected to the battery voltage and the EN pin of the LM2623. When the MOSFET is turned on, the EN pin is pulled low and the boost converter is disabled; when the MOSFET is turned off, the EN pin is pulled high and the boost converter is enabled. When a USB connection is made, the USB-C VBUS rail is present, and the gate of the MOSFET is pulled low, turning the MOSFET on and disabling the boost converter. When the USB connection is broken, the USB-C VBUS rail is absent, the gate of the MOSFET floats high, the MOSFET is turned off, and the boost converter is enabled. The purpose of the enable/disable function of the boost converter is to prevent the boost converter from operating unnecessarily and drawing current from the battery when USB power is present. The logic controlling the enable/disable function of the boost converter is simple and does not require the use of a microcontroller. The TPS2111A has two inputs, IN1 and IN2, and selects the higher-priority input to the VSNS pin and passes the selected input to the output labeled "5.0V." The VSNS pin of the TPS2111A is connected to the output and is used to provide internal feedback for current-limit and switchover logic. The ILIM pin is connected to ground, which enables the current-limit function of the TPS2111A to pass full available current from both the USB-C VBUS rail and the output of the boost converter to the load without reducing the available current to the load. Therefore, the TPS2111A provides an automatic source-selection capability that allows the system to be powered by the USB-C VBUS rail when it is present and by the battery via the boost converter when it is not present. Overall, the implementation provides an effective means of providing a clean and uninterrupted 5.0V supply to downstream components in a variety of applications such as portable or USB-connected sensor platforms, data loggers, and IoT devices requiring power autonomy and USB connectivity.

3.3V Regulator

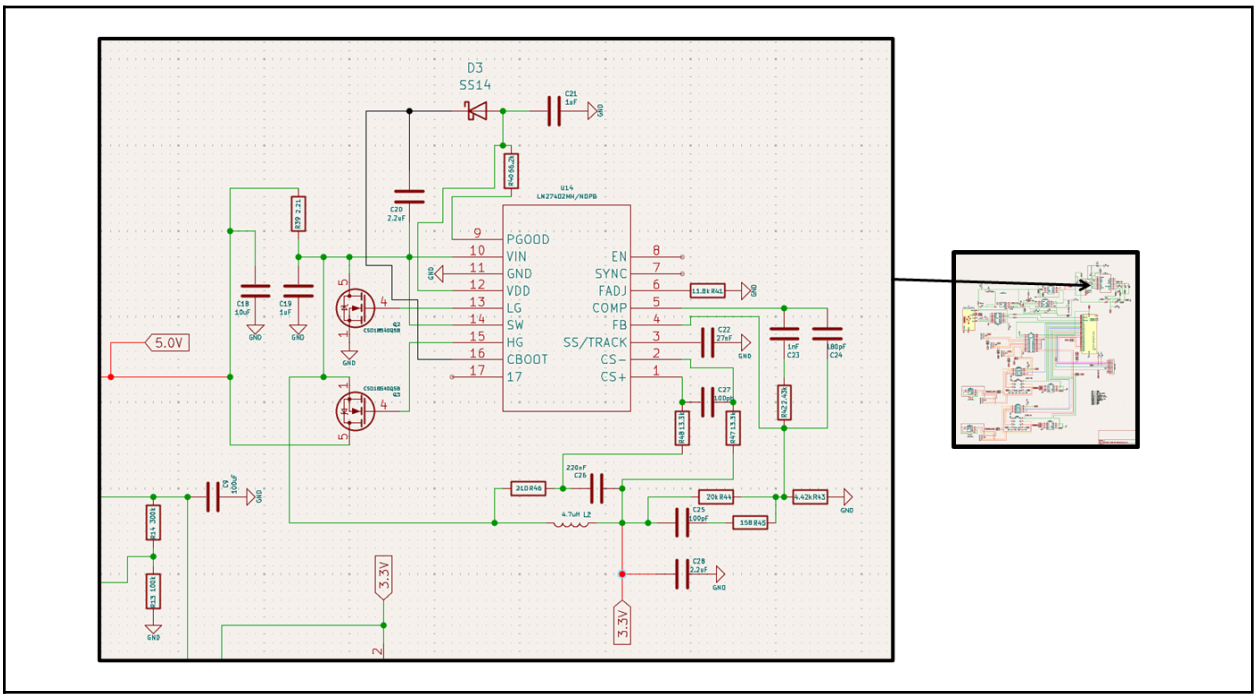


Figure 6.3: 3.3V Regulator Schematic

As illustrated by this schematic diagram, the use of a Texas Instruments LM27402 integrated power module greatly simplifies the design of a 3.3V power rail. The LM27402 is an integrated power module that has a controller, power MOSFETs, inductor, and all passive components combined into one compact footprint. It regulates the 3.3V power rail that the Microcontroller Unit (MCU) and most digital Integrated Circuits (ICs) require for operation at a lower logic voltage level.

It provides a highly integrated synchronous buck converter function. The 5.0V input voltage is fed to pin 12; after filtering the input by a group of two ceramic bulk capacitors (C18, C19) and a 2.2 µF capacitor (C20) for high frequency noise filtering; a Schottky diode (D3) is placed between the input and the power module to block any reverse current or back power into the 3.3V system should the 5.0V rail drop. A power good (PGOOD) signal is taken out to a pull up network for system monitoring purposes. When the 3.3V output is within acceptable limits, the PGOOD signal will be asserted.

A precision feedback loop is provided on the FB pin (pin 4). Precision resistors (R43, R44) provide a precise voltage divider that establishes the desired output voltage based upon the relationship between the voltage divider output voltage and the internal reference. The stability of the feedback loop, as well as the transient response, are established using the capacitors C23, C24, and C25 in conjunction with the resistor-capacitor (RC) compensation network associated with the COMP and SS/TRACK pins. The RC compensation network allows for soft start behavior (established by capacitor C27) and also provides the ability to sequence or track functions if needed. After buffering the 3.3V output by means of more ceramic capacitors (C26, C28); the output is passed through a Low Pass Filter (LC filter) composed of a 4.7 µH inductor.

The LC filter is used to establish a low ripple 3.3V output that is suitable for sensitive digital circuitry. The 3.3V output voltage is monitored and regulated internally so that tight load regulation can be maintained. The addition of the LM27402 module to the design simplifies the power layout. Therefore, it is very beneficial when designing small and/or dense, mixed voltage, embedded systems where board real estate may be limited, such as the Open Sense platform.

6.2 - Path Switching

In an effort to increase the flexibility of the OpenSense platform to accommodate a variety of configurations for the end-user, we have included a dynamic voltage path selection system which will allow users to rapidly switch between 3.3V and 5.0V sensor operation. As there is a wide variety of available sensor modules each using a different logic and supply voltage level for either digital and/or analog functions, this was a logical choice to provide maximum versatility in supporting various types of sensor modules. Using a combination of power multiplexers, level shifters, and SPDT (single pole double

throw) switches, the platform has been designed to dynamically route both the power and communication paths based upon the selected voltage domain.

When the 3.3V mode is chosen, the system has been designed to be compatible with low-voltage digital sensors and microcontrollers such as the ESP32-WROOM-32E (the microcontroller that we have selected), which has a native 3.3V operating voltage. On the other hand, when the 5.0V mode is selected, the platform has been designed to provide the required supply and logic conversion to allow interfacing with high-voltage sensors without risking damage to the MCU (microcontroller) or communication bus.

The flexible design of the OpenSense platform does not stop with the power rails; the flexible design extends to the entire sensor path. The I²C lines (SDA and SCL) are routed through bi-directional level shifters while the analog sensor inputs are routed through configurable routing stages to ensure the correct voltage scaling prior to being input into the ADC (analog-to-digital converter). In addition, the control logic of the level shifters, power muxes and SDPT switches can be individually controlled via individual GPIO pins to ensure safe transition between the two logic levels. Overall, the ability of the OpenSense platform to operate at multiple voltages will enable OpenSense to act as a universal development platform that can be used with sensors from multiple manufacturers and voltage families without requiring hardware modifications. Therefore, the OpenSense platform is highly suitable for use in both educational/research environments and field environments where different sensors may require to be hot-swapped or reconfigured "on-the-fly".

TCA9548A I²C 8-Channel Multiplexer

One of the main features of OpenSense's platform is the addition of the TCA9548A I²C multiplexer, which allows for dynamic control of multiple busses of digital sensors using a single I²C master port. This will allow for the system to grow as new sensors are added to the system while maintaining I²C compliance and avoiding possible I²C address collisions. The schematic representation of the TCA9548A I²C multiplexer is represented in Figure 6.4. The TCA9548A I²C multiplexer has the capability to route multiple I²C ports as a single downstream I²C port. Therefore, multiple devices with the same I²C address can reside on different buses and therefore eliminate bus contention. As illustrated in the schematic, there are eight total I²C buses available (Channels 0-7). However, only six of these channels were enabled (Channels 0-5). Each channel has a pull-up resistor on both the SDA and SCL lines, so that the logic levels on the bus are properly established for each channel. Standard-mode and Fast-mode I²C typically utilize a 4.7 kΩ pull-up resistor for each channel.

Each I²C slave port is connected to the microcontroller via the MCUSDA and MCUSCL pins, which are located at pins 23 and 22 respectively. The power to the multiplexer is supplied via pin 24 (VCC), which is connected to the 3.3V system rail, and pin 12 (GND), which is a stable reference voltage. The three address select pins for the multiplexers are A0, A1, and A2 (located at pins 1, 2, and 21) and are connected to GND in this example. When configured in this manner, the TCA9548A multiplexer has a fixed I²C address of 0x70 and can therefore be commanded by the microcontroller to

switch between different channels. The control logic of the TCA9548A I²C multiplexer is performed by the microcontroller via I²C commands that enable or disable different channels. Each downstream channel consists of two lines; SDx for data and SCx for clock, where x is an integer ranging from 0 to 7. As illustrated in this schematic, channels 0 through 5 (pins 5-17) are connected to external sensors. Channel 6 and 7 (pins 18 and 19) are currently unused and have been shorted to GND to prevent any unwanted noise being picked up on the lines that would potentially cause unexpected behavior on the bus. A GPIO line from the MCU is connected to the RESET pin (pin 3), which allows the firmware to perform a hard reset of the multiplexer if necessary. This

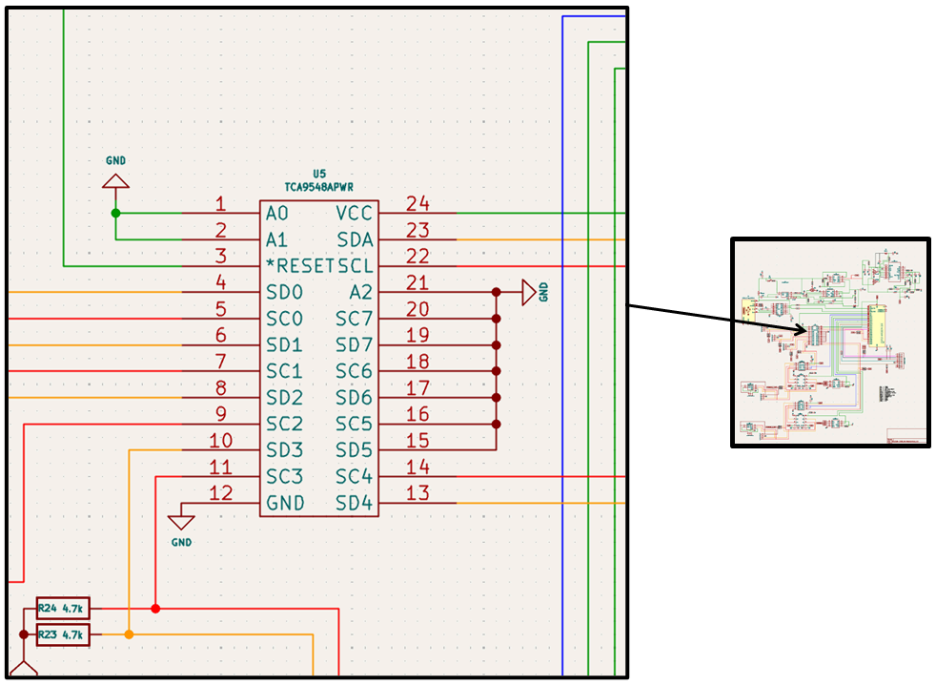


Figure 6.4: TCA9548A 8-Channel I²C Multiplexer Schematic

could be useful if a sensor begins malfunctioning or causes the bus to become stuck in a low state, as the MCU could then reinitialize the multiplexer and restore normal operation of the bus.

Basic Sensor Port Paths

This area of the schematic demonstrates the basic 3.3V-only sensor interfaces, utilizing three four-pin female connectors labeled J1, J2, and J3. The connectors provide power (3.3V), ground, and the SCL and SDA I²C lines to allow direct connection of standard I²C sensor modules. The connectors were developed to connect directly to the I²C multiplexer to enable the microcontroller to selectively address and talk to one sensor at a time. Due to the fact that the sensors operate at 3.3V, no level-shifting or voltage-select circuitry is necessary to make the connections easy and dependable.

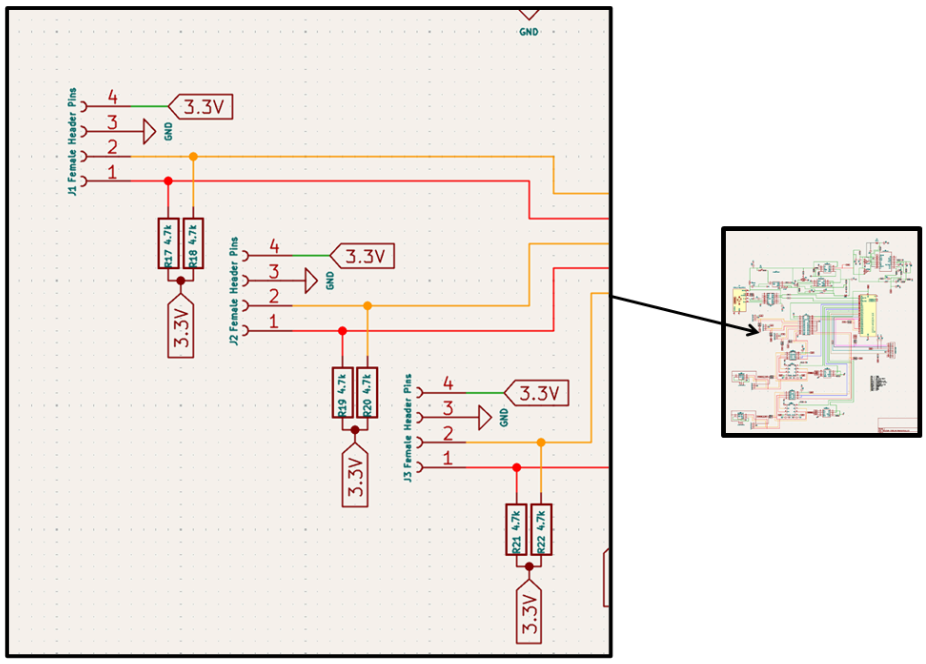


Figure 6.5: Basic 3.3V Sensor port paths

Each connector's SDA and SCL lines have been properly pulled-up to the 3.3V supply by 4.7kΩ external pull-up resistors to ensure reliable I²C communications. The pull-up resistors are located in close proximity to each connector to protect against unwanted noise on the I²C bus lines and to preserve the signal quality of the signals transmitted over those lines. The isolation of these sensor ports as 3.3V-only, helps to prevent damage to the sensor and ensures that there will be no voltage mismatch from 5.0V peripherals. The cleanliness of the separation between the 3.3V fixed-voltage domain and the 5.0V dual-voltage domain enhances the flexibility and modularity of the OpenSense system.

TS3A24157 Electronic Switch

The first part of this path switching design uses an analog switch to direct the two I²C bus lines (SDA and SCL), as shown in Figure 6.5. The TS3A24157 SPDT analog switch allows routing of the two I²C bus lines to two selectable voltage domains. The TS3A24157 SPDT analog switch has a built-in switch network that will allow the routing of both the SDA and SCL I²C bus lines between the 3.3V rail and 5.0V rail paths. The TS3A24157 is a logic level switch and operates off of 3.3V logic levels allowing the direct interfacing to the systems dual voltage regulator supply of 3.3V and 5.0V rails. Control input pins 7 and 8 determine which I²C bus line is switched on. In order to ensure simultaneous switching of the SDA and SCL bus lines, and thus preserve bus integrity and eliminate timing differences between the two bus lines, pins 7 and 8 are connected together and driven by a single MCO GPIO pin. The GPIO pin is then programmatically switched via the WebServer interface by the user, allowing for direct

GUI selection of either the 3.3V path or 5.0V path for the I²C bus lines. By automating the selection process, the user has flexibility to configure different sensors without having to manually set jumpers or physically change hardware.

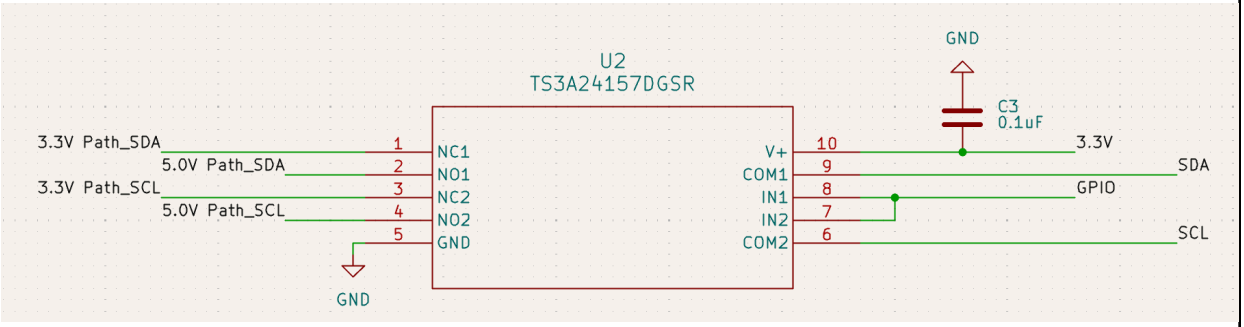


Figure 6.6: TS3A24157 Electronic Single-Pole Double-Throw switch

Within the OpenSense system, the 3.3V path is established as the default voltage domain since it represents the native logic level of the ESP32 microcontroller. The SDA and SCL lines associated with the 3.3V domain are therefore connected to the normally connected (NC) terminals on the SPDT switch, thus guaranteeing they are activated by default. The 5.0V path, on the other hand, is wired through the normally open (NO) terminals, which are not connected unless intentionally selected by the end user with the use of the WebServer. This design choice eliminates the possibility of exposing 5.0V signals inadvertently to 3.3V components and, consequently, the microcontroller, or any other sensitive components of the circuitry are never exposed to the effects of possible overvoltage. By utilizing the lower voltage levels in the default condition, the system guarantees an innocuous power up state allowing the full functionality of the system where higher voltage sensors are applied.

For stable operation of the SPDT switch, a bypass capacitor (C3) is located at the power input to the device. This capacitor is critical in power integrity maintenance and stabilization of the supply voltage, providing a filtering function to eliminate high frequency noise, absorbing the currents of transience. In effect, the capacitor allows local storage of energy to avert dips in voltages due to rapid switching and/or bursts of communication which is critical in the maintenance of reliable I²C bus communications. Transients on the power rail can easily propagate into the logic where bus errors or degradation of signals and/or instability of communications can result. The addition of this bypass capacitor greatly aids in provision of clean transitions and a solid performance under varying load conditions.

TXS0102 I²C Level Translator

Following the sensor path on the OpenSense platform will be the employment of a dedicated logic level translator that permits secure and dependable exchange of data among circuit elements running at diverse logic voltage levels. The TXS0102 shown in

Figure 6.6 is an example of a bi-directional level shifter that was designed particularly for open-drain signal lines like those employed with I²C (Inter-Integrated Circuit) communications. This device is important in translating the 3.3V logic level that the microcontroller operates at into the 5.0V logic levels used by some sensors, including those located along the high voltage path that are selected based upon the power multiplexer and SPDT (Single Pole Double Throw) switch configurations explained previously.

The SDA and SCL lines from the microprocessor are brought to the B side of the level translator, specifically pins B2 and B1, which match pins 1 and 8 of the board. They come from the output of the I²C multiplexers and represent the 3.3V logic domain of the ESP32 microprocessor. Pins A2 and A1 (pins 4 and 5) are the other side of the translator with the corresponding outputs to the sensor path which operate at 5.0V logic levels. Each side of the translator has its own separate power input for defining the logic levels on each side. VCCB is tied into 3.3V to determine the MCU side logic domain and VCCA is connected to 5.0V for the sensor side voltage. The setup allows the part to safely and properly pass logic level signals in both directions across the I²C bus. The output enable pin (OE) is located at pin 6 and is used to either turn the translator ON or

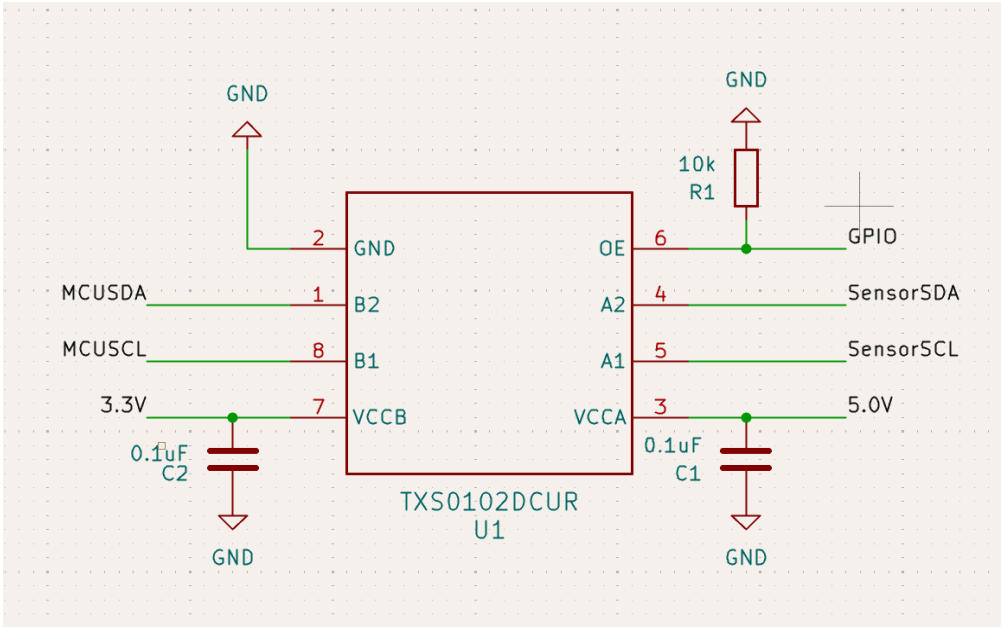


Figure 6.7: TXS0102 logic level translator for I²C communication

put it in a high impedance state. It is connected to one of the microprocessors GPIO lines and is tied low to ground by a 10kΩ resistor so that the translator will always be turned off until the system is configured and ready to go. When you want to select the desired sensor voltage path, the microprocessor drives this pin high, turning the translator ON and allowing bi-directional signal flow between the MCU and sensor sides. This sequence is important to prevent logic contention and protect the system from undefined voltage states during transition periods between power domains. The circuit also includes two 0.1uF ceramic bypass capacitors for ensuring stable operation.

One is placed between VCCA and ground on the 5.0V side and the other is placed between VCCB and ground on the 3.3V side. These capacitors store localized energy to quickly absorb transients due to rapid current changes caused by digital switching and they help to suppress high frequency noise. The capacitors are crucial to providing reliable logic translation; especially in applications such as OpenSense, in which there are many I²C devices that are switching and communicating simultaneously.

TPS2110A Power Multiplexer

The voltage supply routing within the OpenSense open source platform employs a means to allow dynamic switching between 2 separate voltage supplies (3.3V and 5.0V) to accommodate the needs of various sensors or sub-systems. For the purpose of controlling and safely managing this voltage routing path selection, a TPS2110A power multiplexing IC is employed (refer to Figure 6.7). The TPS2110A allows for seamless transitions between these two supply paths with the use of a single command from the

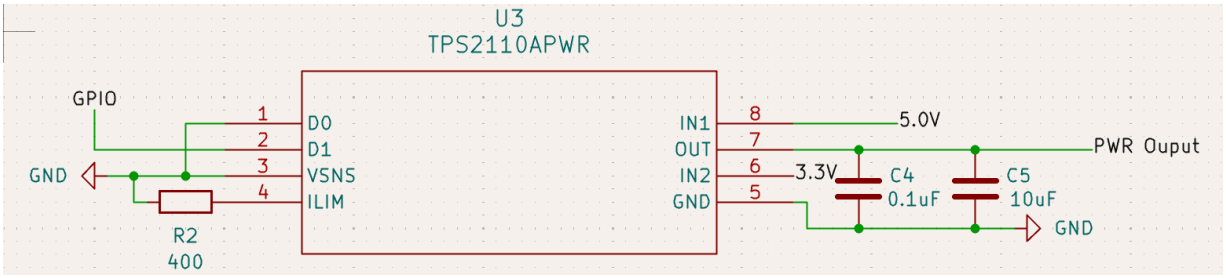


Figure 6.8: TPS2110A Power Multiplexer schematic

microcontroller. In the circuit diagram shown above, the TPS2110A accepts both input voltages at pin 8 (IN1) and pin 6 (IN2); the input to IN1 is connected to the 5.0V rail and the input to IN2 is connected to the 3.3V rail. The output of the TPS2110A is the active voltage rail chosen by the microcontroller's external logic commands and is represented by the signal present on pin 7 (OUT) and is referred to as "power output" to the rest of the sensor circuitry. The output of the TPS2110A is then routed to the VCC pins of the analog and I²C female header as well as the pull up resistors for the I²C communication lines (SDA and SCL) that have followed the logic level translator so that when the voltage path switches to the 3.3V route, the pull-up resistors from the 5.0V logic switch will not interfere.

The logic that controls the selection between the two inputs consists of a combination of internal logic and external logic pins. The D0 and D1 pins (pins 1 and 2) are utilized to control the selection of output. As indicated in the truth table found in the datasheet, if the logic on pin D0 is low, the logic on pin D1 is able to select the desired input using the high and low states provided by the microcontroller (states for D1: LOW = IN2; HIGH = IN1). This GPIO signal is utilized to provide software-controlled logic to determine whether the multiplexer should utilize the 3.3V or 5.0V input. Additionally, the TPS2110A utilizes pin 3 (VSNS) to sense the input voltages and makes intelligent selections based on valid voltage levels while pin 4 (ILIM) is utilized to set the current limit of the output channel utilizing an external resistor. Within this design, a 400Ω

resistor has been connected between ILIM and ground to establish the maximum allowable current to prevent overload conditions of the circuit during brownout or voltage switching events. Finally, to provide clean and stable power delivery, the output of the multiplexer has been filtered by placing two bypass capacitors proximate to the output pin. A 0.1µF ceramic capacitor (C4) has been installed to filter high frequency noise and rapid transients, while a 10µF electrolytic capacitor (C5) provides bulk capacitance to assist with supporting larger or slower changes in current. The two capacitors function in conjunction to reduce the ripple of the voltage supplied to sensitive downstream components and prevent brownouts occurring during the switching process.

MCP3221 Analog to Digital Converter

At the last stage of the sensor path, OpenSense's sensor path includes an analog-to-digital converter (MCP3221A5T-E/OT) that interfaces with digital sensors' analog outputs as well as with the digital sensors themselves using a digital interface (I²C compatible). It is designed to allow users to connect their analog sensors to the same I²C bus that they are currently connecting their digital sensors to; therefore, it allows for a unified communication protocol across the system. In Figure 6.8, you will find a schematic showing how to wire the MCP3221 for both its power and connection within the sensor architecture.

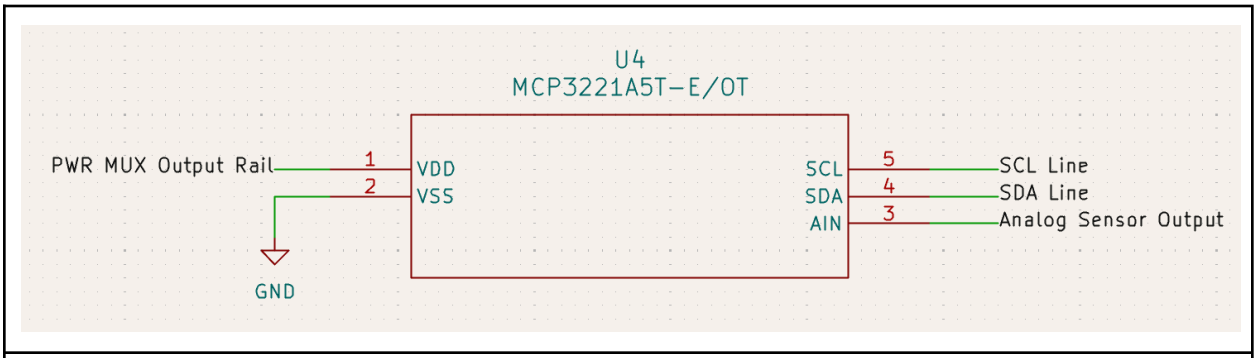


Figure 6.9: MCP3221 Analog to Digital Converter Schematic

The MCP3221 gets its power from the PWR MUX Output Rail, which is determined based upon which voltage has been chosen from the upstream TPS2110APWR power multiplexer. The PWR MUX Output Rail is wired to pin 1 (VDD) of the ADC; therefore, the voltage at pin 1 (VDD) is 3.3V or 5.0V, depending upon whether 3.3V or 5.0V was selected as the user's desired voltage. Pin 2 (VSS) of the ADC is wired to GND and represents the ground reference for the MCP3221; thus, there is proper voltage regulation across the entire chip. As a result, this flexible power arrangement allows the MCP3221 to function correctly for a wide variety of sensor voltage ranges without changing the circuitry.

As far as signals are concerned, the ADC has one analog input located on pin 3 (AIN). Pin 3 (AIN) is where the voltage output from the analog sensor is fed into the ADC for conversion. The pin is wired directly to the analog output of the sensor, so when the

analog output of the sensor changes due to an increase/decrease in temperature, etc., the voltage output from the sensor also changes. When the voltage output from the analog sensor is received at pin 3 (AIN) of the MCP3221, the voltage is captured and digitized for transmission to the microcontroller over I²C. The converted digital value is transmitted to the microcontroller using the SDA and SCL I²C lines that are connected to pins 4 and 5 of the ADC, respectively. Because these lines are part of the same I²C bus used by the rest of the digital sensors, analog sensor converters, level translators, and multiplexers on the board, the MCP3221 does not require a separate communication line to send/receive data to/from the microcontroller. Using the MCP3221 as a dedicated ADC offers advantages in this application. By providing an I²C interface, the MCP3221 does not require a separate SPI or parallel bus; therefore, it minimizes the number of microcontroller pins required and reduces the total complexity of the board. Additionally, since the MCP3221 supports standard I²C addressing and bus arbitration, it can be accessed using the same driver code as all the other I²C peripherals on the board, making software integration simpler. Additionally, by positioning the ADC immediately after the power multiplexer, the design ensures that the analog sensor is powered with the correct voltage prior to receiving the analog signal, thereby ensuring that the analog sensor is properly matched to the input range of the ADC and ensuring accurate measurements. Due to its ability to provide a wide range of operating voltages and its simple pinout, the MCP3221 makes a good fit for the OpenSense platform, allowing users to easily add analog sensors to their systems while maintaining the overall modularity of the OpenSense platform.

Overall Path Switching System

The schematic gives an overall picture of the complete voltage path switching system on the OpenSense platform that uses the TS3A24157 analog switch; the level shifter (TXS0102); the power multiplexer (TPS2110A) and the analog-to-digital converter (MCP3221) in order to dynamically route between the two different 3.3V and 5.0V sensor domains; and, ensure safe and reliable I²C communications.

At the input side of this system, the TS3A24157 analog switch routes the SDA and SCL lines to either the 3.3V or 5.0V I²C domains depending on a GPIO signal. 3.3V is the default setting at this point. The SDA and SCL lines are then routed through the TXS0102 level shifter to handle bi-directional level shifting of the signals between the MCU and sensor logic levels. On the other hand, the TPS2110A selects one of the two possible output voltage rails available for powering the sensors using the microcontroller's input logic to select either the 3.3V or the 5.0V rail. Both sensors and the MCP3221 analog-to-digital converter receive their power from the rail selected by the TPS2110A. The analog sensor can therefore be integrated into the system using the same I²C bus. The stable transition of control logic; the programmable GPIOs and the bypass capacitors work together to provide robust operation of the system over a very large number of sensor configurations.

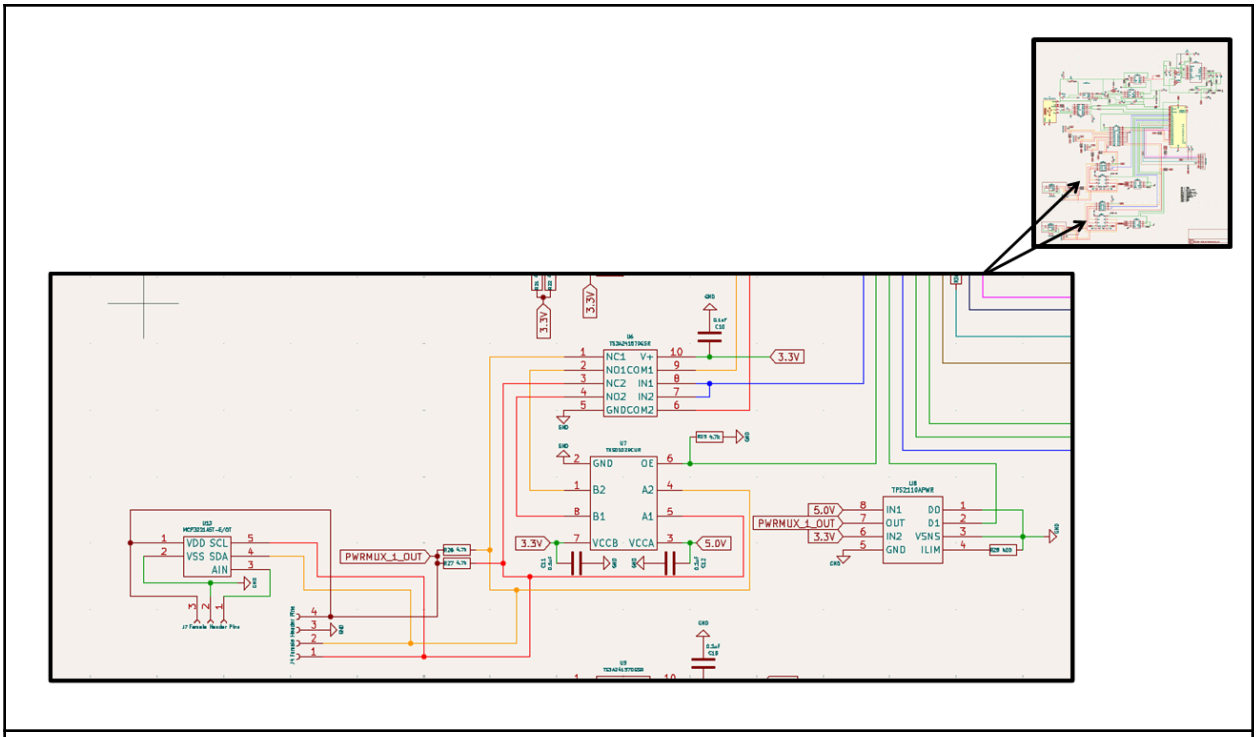


Figure 6.10: Path Switching System (3.0V⇋ 5.0V)

6.3 - Data Storage

To facilitate persistent local data logging, the OpenSense platform has a microSD card interface, which allows data from sensors to be retained for future usage such as offline processing or storage (i.e. back-up). Figure 6.9 demonstrates the schematic implementation of this subsystem which is based on the DM3AT-SF-PEJM5 microSD card socket, a surface mount push-pull connector facilitating full SPI communication with the microcontroller. This storage method is crucial to the versatility of the platform, especially in cases of remote or disconnected deployments where real-time communication via the cloud is either impossible or impractical.

The microSD interface uses the SPI protocol with all communication signals routed between the MCU and the SD card through a standardised nine-way connector. The four major SPI signals, Chip Select (GPIOCS), Master Out Slave In (GPIOMOSI), Master In Slave Out (GPIOMISO) and Serial Clock (GPIOSCK), are connected to the relevant pins on the DM3AT connector. These signals allow the MCU to initiate read/write transactions, give block commands to the memory and receive data from the microSD card. The SPI mode is preferred in embedded systems of the type used here (OpenSense) due to its simplicity, widespread support and compatibility with common filesystem libraries (e.g. FAT32).

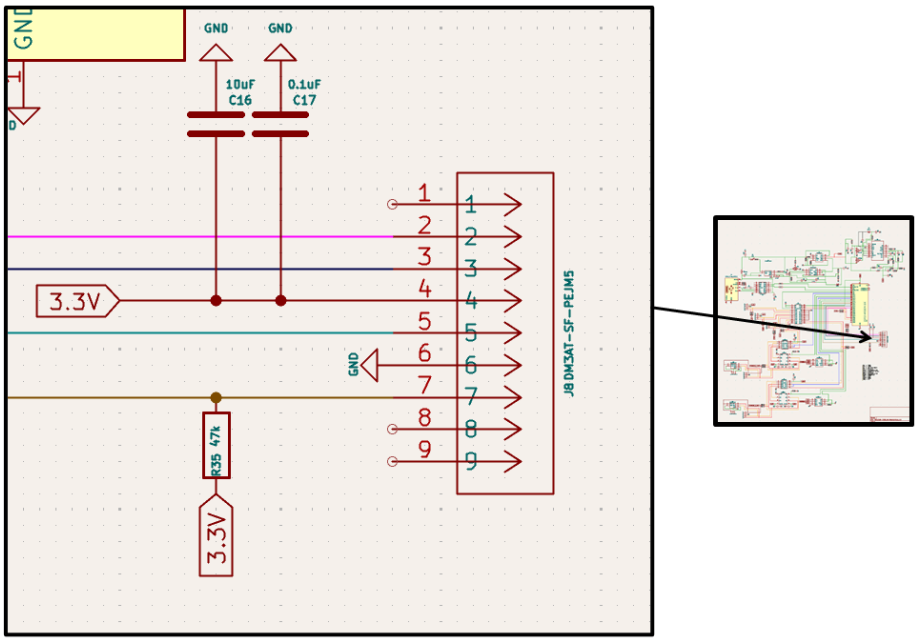


Figure 6.11: DM3AT-SF-PEJM5 microSD card slot schematic

In order to improve integrity of the SPI signals and to prevent reflections or overshoot at faster data rates, series resistors are used at various key locations. In order to prevent ringing on the SCK line a resistor of 22Ω has been chosen. In order to provide correct logic levels for the period of boot up before and during idle operation 47kΩ pull up resistors have been employed on the CS, MOSI and MISO lines. These resistors will do much to eliminate the chance of inadvertent initialization, or running into bus contention during power up and system reset. Power to the SD card is fed from the regulated 3.3V rail, which adheres to the requested logic levels for the card type. Two decoupling capacitors (C6) (0.1 uF), (C7) (10 uF) used in parallel and placed close to the power input pins of the connector will give benefits of high frequency noise suppression, and storage for bulk power requirements. Such a clean stable supply must be provided to the card, especially during WRITE operations which call for high instantaneous current drain. The bypass capacitors are placed immediately adjacent to the connector so as to minimize the parasitic inductance involved and maximize their efficiency. The DM3AT-SF-PEJM5 connector is a push pull mechanical socket using gold plated contacts with an excellent insertion life cycle. The connector is capable of giving the required standard SD card mounting and pinout conventions where pin four and pin six are tied to GND in accordance with card definitions. The footprint is compact overall so as to allow its usage in space restricted sensor platforms such as OpenSense. The inclusion of a microSD card interface will allow the OpenSense platform to timestamp sensor loggings, system logs, or raw data block, thus not totally relying on wireless transfer for this. This will open the platform to remote application, environmental monitoring, industrial sensing, offline data logging etc where reliability and integrity of the data are very important. The SD Card is easily capable of being unlocked and

removed for insertion into a PC or mobile reader for data retrieval enabling users and researchers alike to analyze and store elsewhere the sensor outputs at their convenience.

MCU Schematic

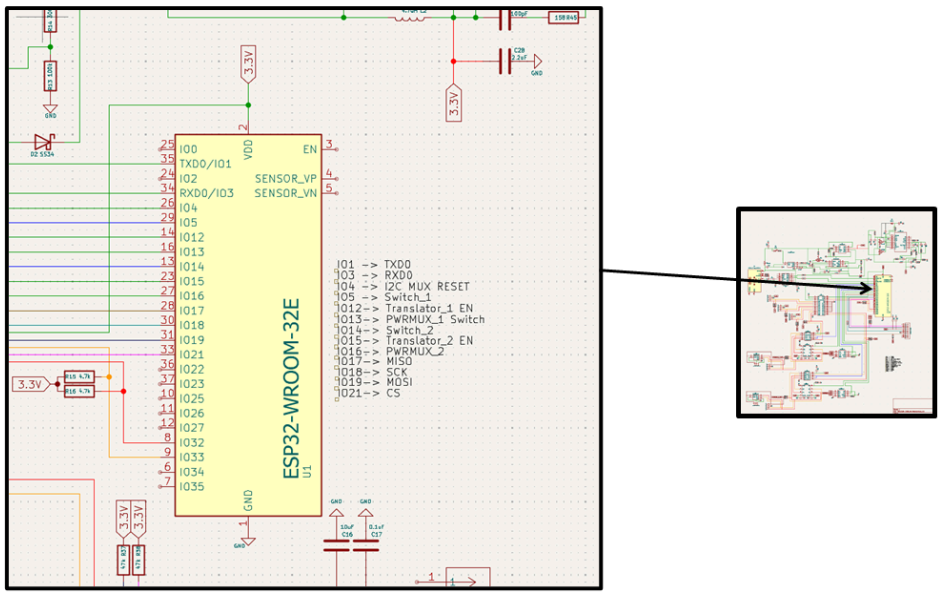


Figure 6.12: MCU Schematic (ESP32-WROOM-32E-N4)

The above diagram illustrates the central role that the ESP32-WROOM-32E microprocessor plays in the OpenSense system. The ESP32 provides both control over I2C communications as well as management of the power selection and voltage translations via its GPIOs. Each of several labeled GPIO lines, which are connected to multiple control points throughout the design (i.e., I2C multiplexer reset line, level shifter enable lines, analog switch select lines, and TPS2110A power multiplexor control lines), have been clearly documented next to the ESP32 footprint to aid in firmware development.

Both the SDA and SCL lines from the ESP32 are also pulled high to 3.3V with 4.7kΩ resistors (R15 and R16), which ensures proper I2C signal levels at the microcontroller's end. In addition, decoupling capacitors (C16 and C17) are placed near the VDD pin to provide stable power to the ESP32 while filtering transient signals and noise. This clean, single point of control allows the ESP32 to fully manage the transitions between different voltage domains as well as manage communications with peripherals, providing a seamless method for users to configure the device through software.

6.4 Sensor Schematics

Our project will make use of many different sensors and so we have included the schematics for two of the ones we will use below.

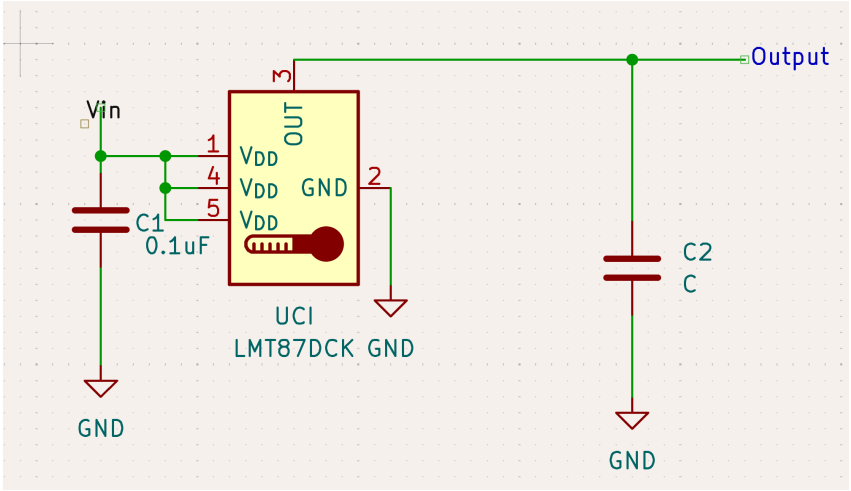


Figure 6.13: Temperature Sensor Schematic

This is the schematic for a temperature sensor we will include that will be easy to use and configure. It makes use of the LMT87 which is a useful and accurate sensor while also being easy to implement. The decoupling and output capacitor are not strictly necessary however according to research they should help keep the readings as accurate as possible as they enable a SAR to ADC draw burst of current without negatively affecting the temperature readings.

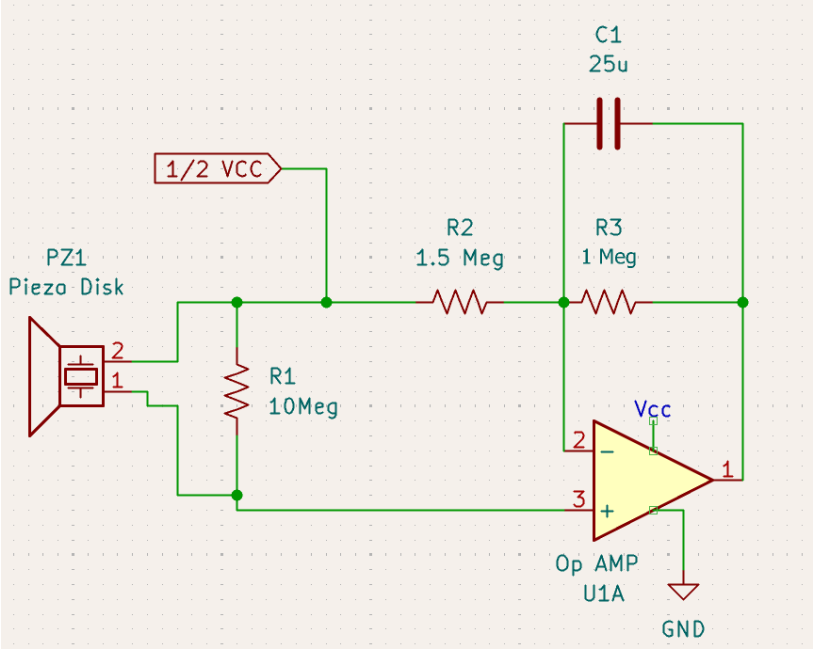


Figure 6.14: Pressure Sensor Schematic

This is the schematic for a pressure sensor that our sensor will also make use of. It uses a piezoelectric transducer which will be what senses the pressure for this sensor. By default the sensor will output ½ VCC but when pressure is applied to it then the output will increase through the op amp which can then be used to to calculate the force and pressure.

6.5 Monochrometer Electronics

6.5.1 Transimpedance Amplifier

To accurately measure the optical power at the output of the spectrometer, the system employs a photodetector which generates a photocurrent proportional to the incident light intensity depending on the detector's responsitivity to the wavelength. Because this output current is typically very small, often in the microampere range, it must be converted into a measurable voltage signal before it can be processed by the OpenSense platform. To achieve this, a transimpedance amplifier (TIA) is used. The TIA converts the photocurrent into a proportional voltage by passing it through a high-gain feedback resistor within an operational amplifier circuit. This configuration not only amplifies the signal but also maintains low noise and high bandwidth, ensuring accurate and stable readings across the full spectral range. The resulting voltage output can then be easily collected by the OpenSense data acquisition system, allowing the intensity of each wavelength to be recorded and analyzed.

The first step in designing the TIA is to determine an appropriate feedback resistor, which sets the relationship between the photocurrent generated by the detector and the resulting output voltage. The resistor value must be chosen based on the maximum expected photocurrent and the desired output voltage range, ensuring the amplifier operates linearly without saturation. The voltage range we will see is from a maximum of 5V to a minimum of 0V. For the photocurrent, we look at the datasheet of our selected photodiode. Referring to the datasheet of the selected TEMD5080X01 photodiode, a reverse light current of approximately 50 μA can be expected under bright ambient illumination of about 500 lux. Using the formula given below, we calculate that the feedback resistor should have a value of around 100 kΩ.

$$R_1 = \frac{V_{o,max} - V_{o,min}}{I_{p,max}}$$

After selecting the feedback resistor, the next step in the transimpedance amplifier design is to determine the value of the feedback capacitor (C_x), which works in parallel with the resistor to ensure circuit stability and noise reduction. The feedback capacitor helps to limit the amplifier's bandwidth, preventing high-frequency oscillations caused by the interaction between the photodiode's junction capacitance and the op-amp's input capacitance. Choosing this value requires balancing response speed and stability—a larger capacitor improves stability and allows for a higher gain but reduces bandwidth, while a smaller one increases bandwidth at the risk of noise or instability. Thus, we determine the capacitor value based on our desired bandwidth using the relationship below:

$$C_1 = \frac{1}{2\pi R_1 f_{-3dB}}$$

Since we plan to sample at a rate of 100 Hz, plugging in the value for our previously chosen resistor, we find that we need a capacitor value of 15.9 nF. While this capacitor greatly reduces our bandwidth, it is still within what we need for our system while allowing us to get greater gain to our photodetector in order to measure the signals we need.

The last part of this process is determining the value of the resistors in the bias network. This is determined based on the voltage provided to the rails of the amplifier and the reference voltage at the noninverting input. The equation given for this is as follows:

$$R_2 = \frac{V_{cc} - V_{ref}}{V_{ref}} * R_3$$

Given that the rails see a voltage V_{cc} of 5V and a V_{ref} of 0.1V, we can solve for the relationship between R_2 and R_3 to be $R_2 = 49R_3$. With this relationship, we select appropriate values of these resistors which are available, resulting in R_2 = 13.7kΩ and R_3 = 280Ω.

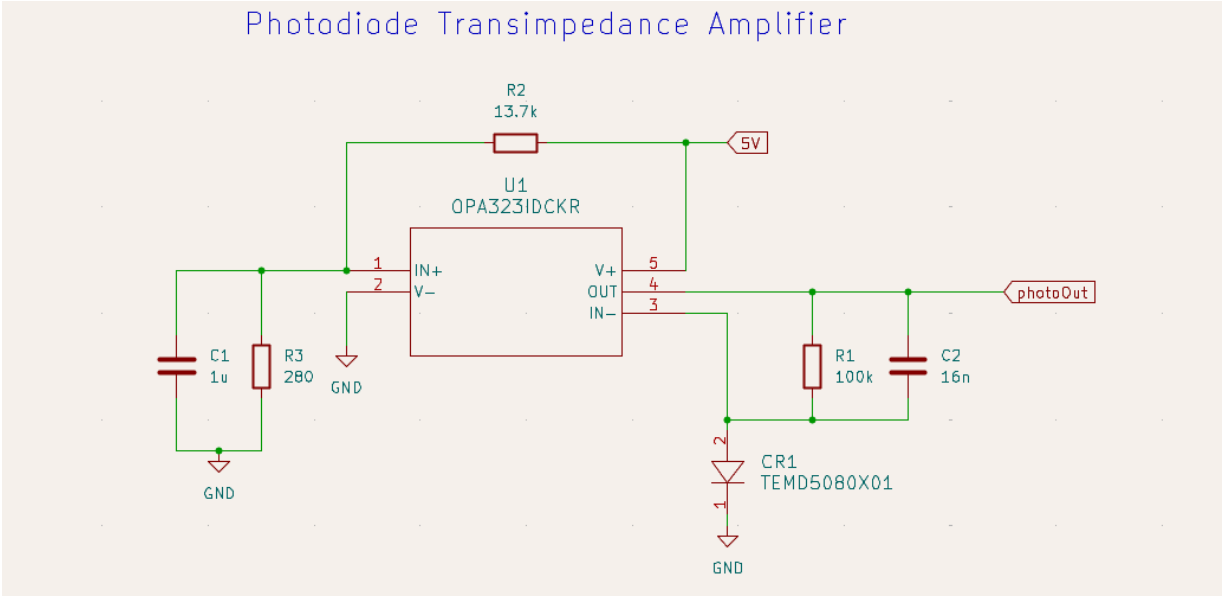


Figure 6.15: Transimpedance Amplifier Schematic

6.5.2 Motor Controller

To control the angle of the diffraction grating within the monochromator, the system uses an Adafruit 1404 servomotor to both provide the rotation and record the current angular position using an external potentiometer feedback line. The servomotor itself is controlled using an ATTiny84 microcontroller. This feedback output produces an analog voltage proportional to the grating's angular position, allowing the system to precisely track the wavelength being sampled at every step of the scan. The servo is driven by an ATtiny84 microcontroller, which generates the appropriate PWM control signal to set the grating angle and coordinates the timing of mechanical movement and data acquisition.

After each incremental rotation, the microcontroller reads two key signals using its built-in analog-to-digital converter: the servo feedback voltage, which is converted into an absolute angular position, and the photodiode output, which represents the optical intensity at that wavelength. These measurements ensure that every data point in the spectrum is tied to a known mechanical state of the grating. Once the ADC conversions are complete, the microcontroller communicates the results to the OpenSense platform using the I²C interface, ensuring a compact and reliable digital communication link. The OpenSense platform can then visualize, log, or analyze the data in real time, in order to deliver user-readable measurements and complete spectrums.

The proper connections between the motor and the microcontroller are shown in the schematic below:

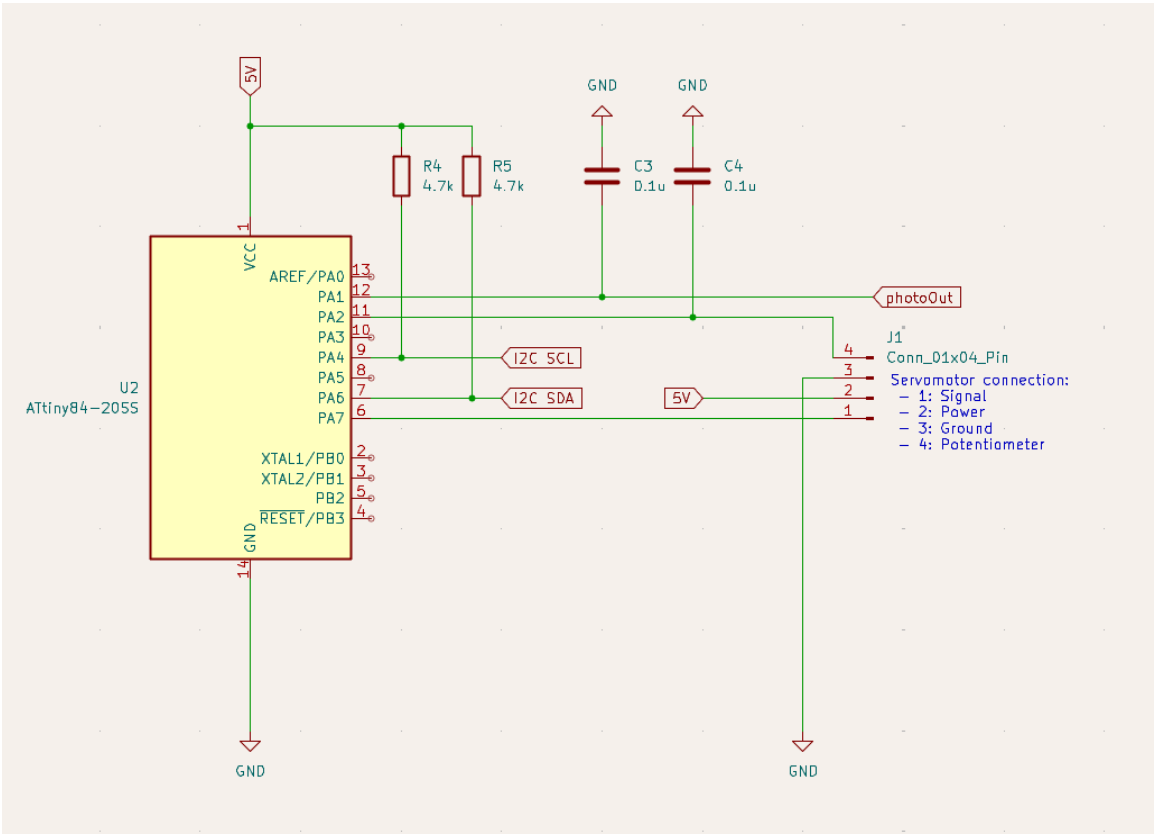


Figure 6.16: Servo-motor schematic

6.6 Optical Spectrometer Design Calculations

We begin our design by establishing the key performance criteria that the spectrometer must meet, with spectral range and spectral resolution serving as the primary parameter. The spectral range defines the span of wavelengths the device can accurately measure while the resolution refers to how finely the device can distinguish the different wavelengths. Because the target range of our system was to be from 300 to 800 nm, we can find that our system has a total bandwidth ($\Delta\lambda$) of 500 nm with its center wavelength (λ_c) at 550 nm. The desired spectral resolution target for our system is set to be less than or equal to five millimeters, ensuring sufficient details in the measurements made.

In addition to the target performance goals, several design parameters are defined by the components selected for the spectrometer. This includes the focal lengths of the

mirrors, chosen to be 25 mm each, and the groove density (G) of the diffraction grating of 1200 lines per millimeter.

With these parameters defined, we can begin the process of designing the spectrometer, by calculating the angles. Our design uses a Czerny-Turner configuration, which usually has a total angle (Φ) between 24 - 30 degrees. Starting from this guideline, we adjust the Φ angle until we find a solution where the incident angle (α) and reflected angle (β) are approximately the same, solving for these two angles using the following equations:

$$\alpha = sin^{-1}\left(\frac{\lambda_c G}{2cos(\frac{\Phi}{2})}\right) - \frac{\Phi}{2}$$

$$\beta = \Phi - \alpha$$

This relationship where $\alpha \simeq \beta$ occurs when Φ = 20 degrees, and solving for the incident and reflected angles, we get that α = 9.58 degrees and β = 10.42 degrees. [76]

The next step in the design process involves analyzing the rotation of the diffraction grating to determine the angular change required to achieve the target spectral resolution. This relationship is governed by the diffraction grating equation for reflective systems, which links the diffracted wavelength to the incident angle and groove spacing as shown below:

$$a[sin(\theta_m) + sin(\theta_i)] = m\lambda$$

By rearranging this equation, the wavelength can be expressed as a function of the incident angle, allowing us to calculate how small changes in grating rotation correspond to measurable shifts in wavelength at the detector. [17] This analysis establishes the mechanical precision needed from the stepper motor to meet the system's resolution target. Using Matlab, we created a plot showcasing the angle vs wavelength relationship.

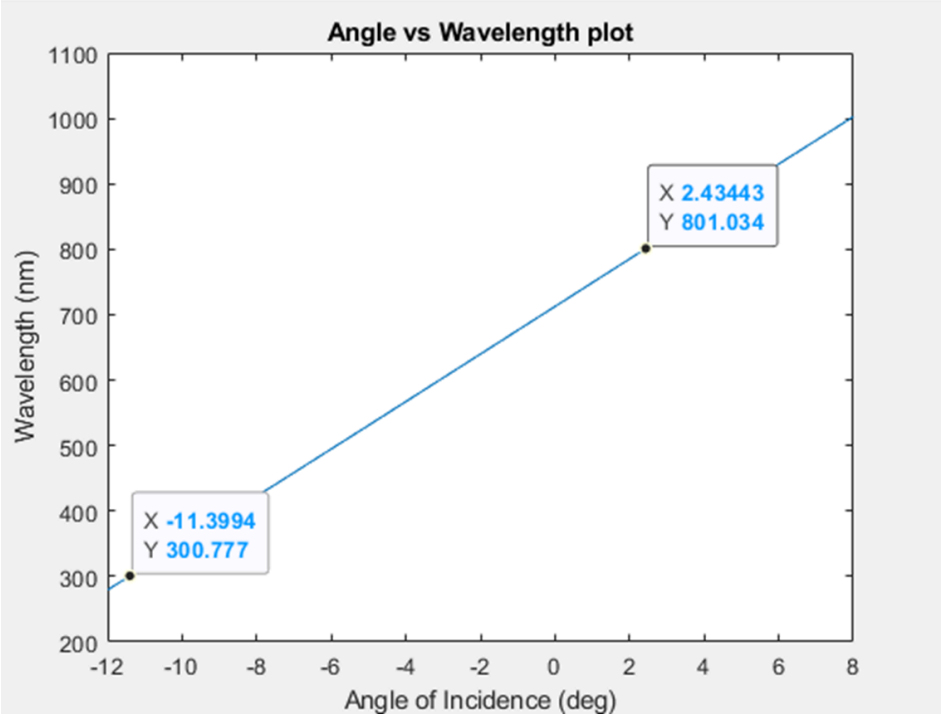


Figure 6.17 - Angle vs Wavelength plot

From the resulting plot, it can be observed that a total grating rotation of approximately 13% of a full revolution is required to span the complete 300–800 nm wavelength range. Assuming the relationship between grating rotation and wavelength is approximately linear—a reasonable simplification based on the plot—this allows us to estimate the angular change needed to resolve a 5 nm interval. Using this approximation, the corresponding rotation angle can be determined through the calculations shown below.

$$\frac{\Delta\theta}{\Delta\lambda} = \frac{2.5 - (-11.4)\ deg}{800 - 300\ nm} = 0.0278\ deg/nm$$

$$\Delta\theta = 0.0278\ deg/nm\ *\ 5\ nm = 0.139\ deg$$

Thus, we know that a rotation of at least 0.139° is required to achieve the target spectral resolution, a level of precision that our selected motor can readily provide. With the Adafruit 858 stepper motor's 0.702° step angle and 1:16 gear reduction, the effective step size is 0.0439° per step, allowing for smooth and accurate wavelength scanning well below the required threshold. This confirms that the motor system not only meets but exceeds the mechanical precision necessary for the spectrometer to resolve 5 nm wavelength differences across the full 300–800 nm range.

Chapter 7 - Software design

This chapter describes the detailed software architectural design for the sensor data acquisition and monitoring system for our OpenSense project, with the ESP32 capabilities in mind. The design supports a modular architecture that utilizes the ESP-IDF software framework and the FreeRTOS kernel to achieve deterministic real-time sampling over five sensor channels at 10 Hz, while logging data to SD card storage and transmitting telemetry to web clients simultaneously. The software architecture is based on a separation of concerns through the implementation of task-oriented concurrency, non-blocking queue-mediated communications, and hardware abstraction layers that allow for flexible sensor configuration and run-time reconfiguration without rebooting the system.

The next sections describe the architectural component level view, control flow patterns, use cases, state behavior, object-oriented abstractions, user interface design, data transfer mechanisms, and internal data structures that implement the system requirements as defined in the earlier chapters. Each diagram is accompanied by a detailed textual description that explains the design rationale, methods of implementation, and integration points between the subsystems.

7.1 Software Architecture - Subsystem/Component Level

The system architecture consists of five primary subsystems operating on the ESP32 platform running FreeRTOS. The Configuration Service manages persistent storage through NVS and provides atomic configuration updates. The Sampling Subsystem interfaces with sensors via I²C through a TCA9548A multiplexer, triggering ADC conversions at 10 Hz intervals. The Logging Subsystem receives samples through FreeRTOS queues and writes sector-aligned blocks to the SD card via the VFS/FatFs stack. The Web Server Subsystem hosts the HTTP server with static file handlers, REST API endpoints, and WebSocket support for real-time telemetry. The Discovery Service performs boot-time sensor identification across multiplexer channels and manages driver binding. Data flows unidirectionally from sensors through queues to both the logger and telemetry broadcaster, ensuring sampling timing remains deterministic regardless of I/O latency.

7.1.1 WebSocket Implementation Architecture

The system uses an ESP-IDF-native HTTP server (esphttpserver) to implement WebSocket communication. This is possible because the esphttpserver is designed to provide an integrated WebSocket capability, allowing the sensor platform to engage in real-time bidirectional communications with web browser clients.

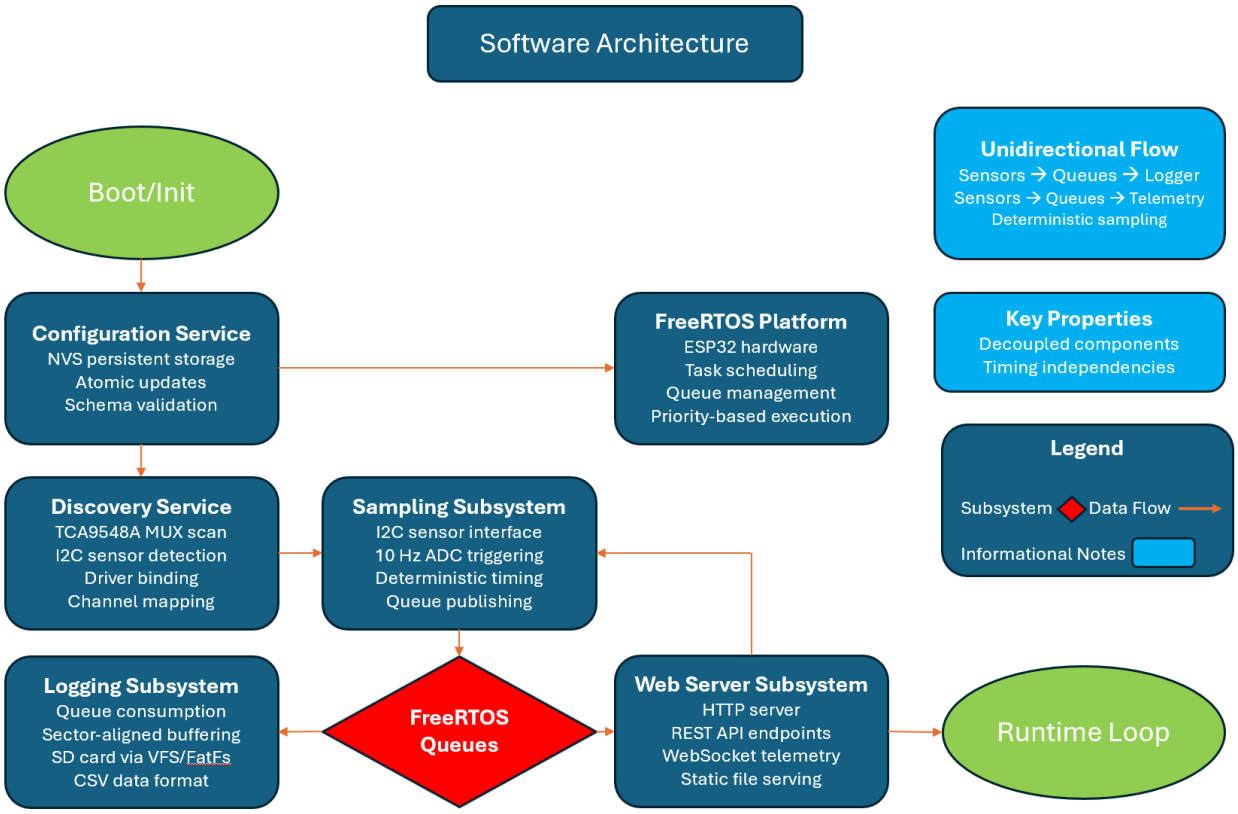


Figure 7.1 Software architecture flowchart

An ESP-IDF-native implementation was used instead of alternatives such as ESPAsyncWebServer. Three main factors influenced this decision. First, since it is part of the ESP-IDF core, there are no additional external dependency requirements; therefore, version compatibility will never be a concern. Second, using the same server instance for both HTTP and WebSocket endpoints reduces the amount of memory being used. Lastly, the synchronous nature of the ESP-IDF implementation fits well with the FreeRTOS-based task architecture used by the system; it maintains the determinism required for use in real-time applications.

A WebSocket endpoint was created and assigned the URI path “/ws” and set with the iswebsocket flag enabled. Once a client connects to this endpoint, the server performs an automatic handshake upgrade as per the specification in RFC 6455. Once the client establishes a connection, the server calls the websockethandler() function to process all incoming frames and pulls out the socket file descriptor(s) associated with each frame. These socket file descriptors are then placed into a global array called wsclients for use in broadcast operations.

When transmitting data via WebSocket frames, the system uses the httpdwssendframeasync() function so that slow clients do not block the operation of the Telemetry Task. The httpdwssendframeasync() function is an asynchronous API that allows a developer to queue a frame for sending and returns immediately whether the

operation was successfully queued (ESPOK) or if the send buffer on the target client is full or the target client's socket has closed (ESPFAIL). Any failed clients are immediately removed from the active client list.

The server configuration also includes an automatic mechanism for implementing PING/PONG heartbeat functionality to allow for health monitoring of each connected client. If a client does not respond to a PING frame sent to them within 10 seconds, they are removed from the active client list and their socket is closed. In addition to this, the system is configured to perform periodic keep-alive functions for each client to clean up stale sockets (TCP keepalive settings of 30 seconds idle, 10 seconds between transmissions, and 3 retries).

7.2 Main Program Flow

The process of initializing the system is initiated by hardware configuration of the GPIO pins, I²C controllers, ADC channels, SPI for SD card access, etc., after which the Configuration Service loads the configuration from NVS in JSON format and validates the schema before loading it, restoring the previous version if corrupted. Next, the Discovery Service queries each of the eight multiplexers to find a valid I²C address and calls the init function of the registered driver. Once the driver has been initialized, the handle of the driver and its configuration mapping are stored back to NVS.

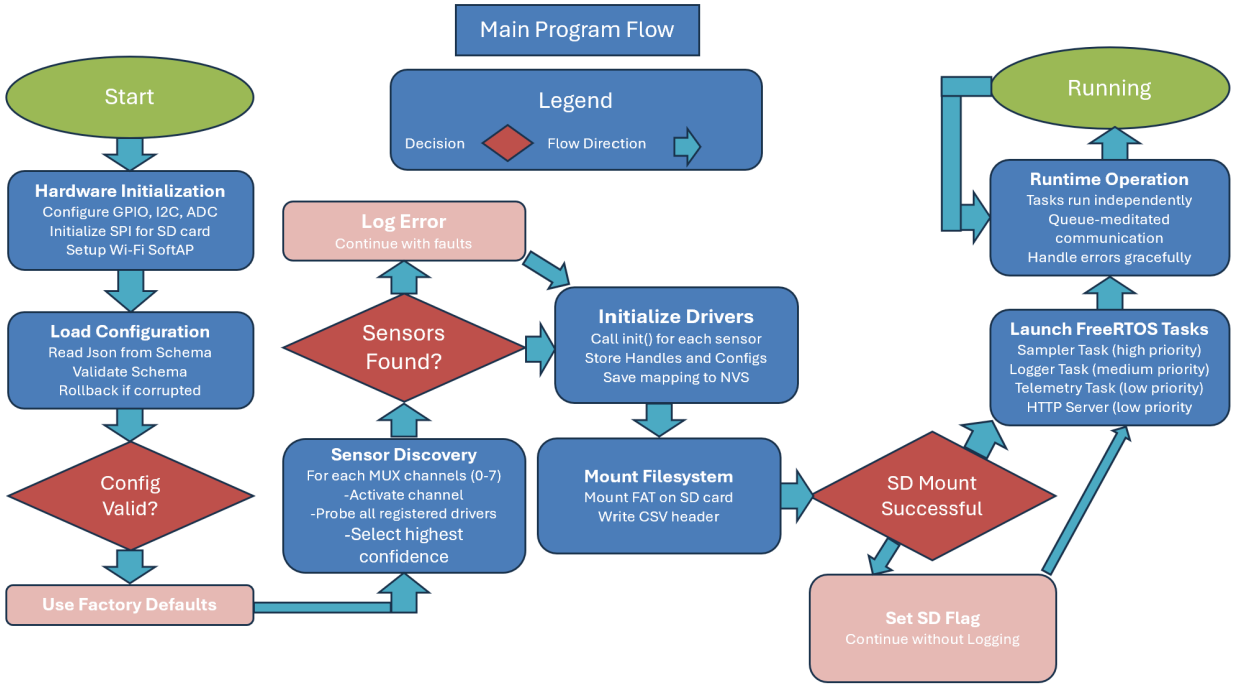


Figure 7.2 - Main program flowchart

The system then mounts the FAT file system on the SD card and creates or rotates the CSV files as necessary. After that, the system will start three tasks: the Sampler Task will run at a high priority on a 100 ms timer, read all of the discovered sensor handles and put their values into the sample queue along with a timestamp; the Logger Task will dequeue the samples from the sample queue, accumulate them into a ring buffer and

write them out to CSV once per second in a sector aligned block; the Telemetry Task will package the most recent values from the logger task into a small JSON frame and send it out over the WebSocket to all of the currently connected clients. The HTTP server will always be running and will service static file requests, configuration get/post requests, and WebSocket upgrades. If an error occurs in the system, there are fallbacks available: if the SD card write fails, it will flag an error condition, but it will not stop the sampling of data; if there is a bus fault on one of the multiplexer channels, it will disable that channel (and continue to operate on the other channels).

7.3 Use Case Diagram

The primary actor is the user, who interacts with the system through various use cases. To interact with the system, the user connects to the SoftAP network broadcast by the ESP32 and accesses the web page using any browser. Once on the web page, the users can see live time series charts of the last ten seconds of measurement of all five sensors. The users can also adjust system configuration via the configuration panel, which allows adjustment of sensor type, I²C address, ADC attenuations, calibration factor, and logging rate for submission as a single atomic action. The users can initiate sensor rescans for individual multiplexer channels or globally when hardware is replaced. The users can monitor system health through status indicators of the SD card write status, faults in the multiplexer channel faults and the overall sample integrity. The users can also download logged CSV files from the system via the web interface or remove the SD card to obtain them. The secondary actor includes the System performing autonomous actions (use cases), including: automatic sensor discovery upon start-up, periodic SD card flushing, persistence of configuration to NVS, WebSockets broadcasting, and graceful degradation if a single channel fails. A secondary optional Maintenance Actor can access additional diagnostic endpoints to view raw probe data, trigger bus recoveries, and perform a factory reset by deleting the NVS namespace.

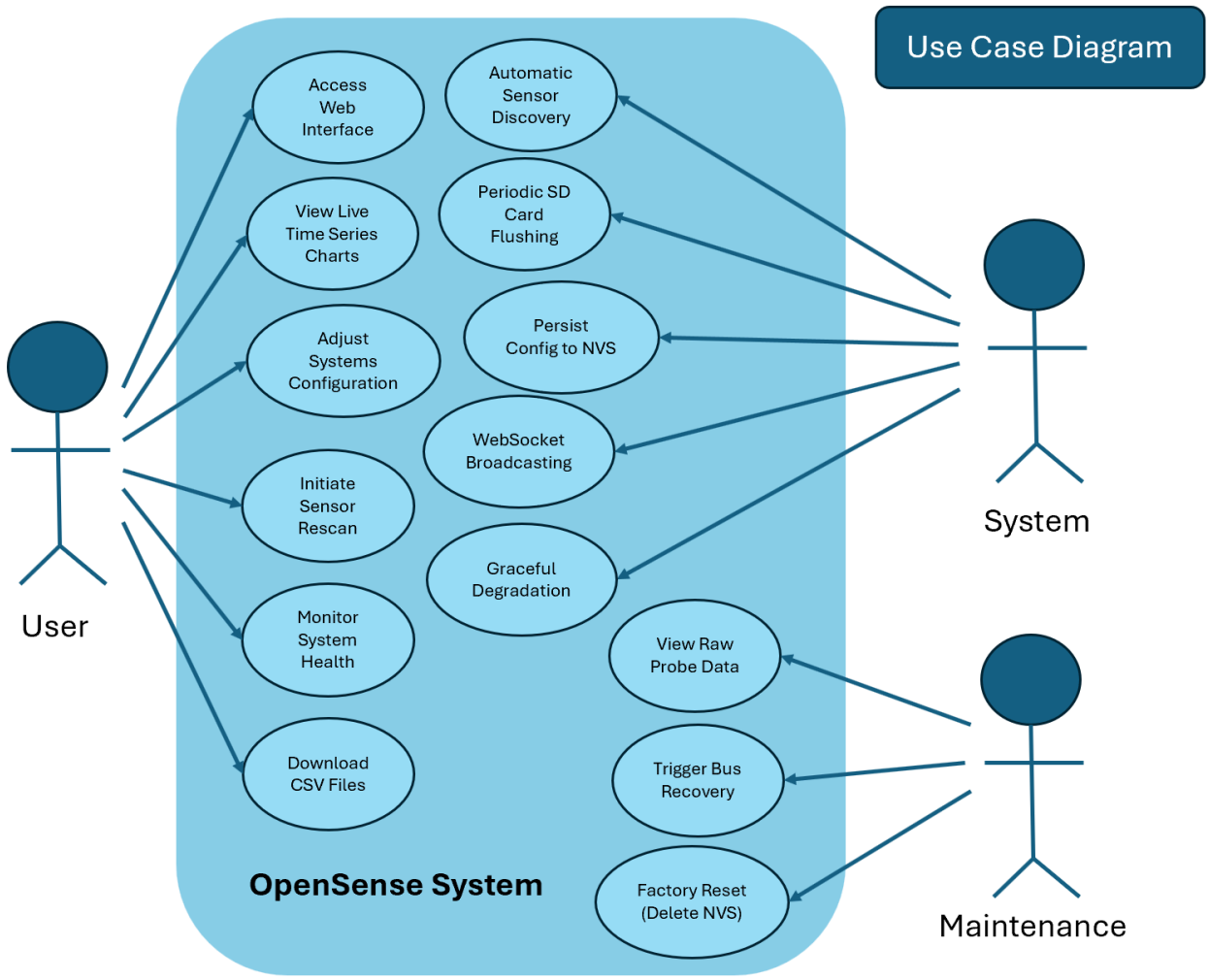
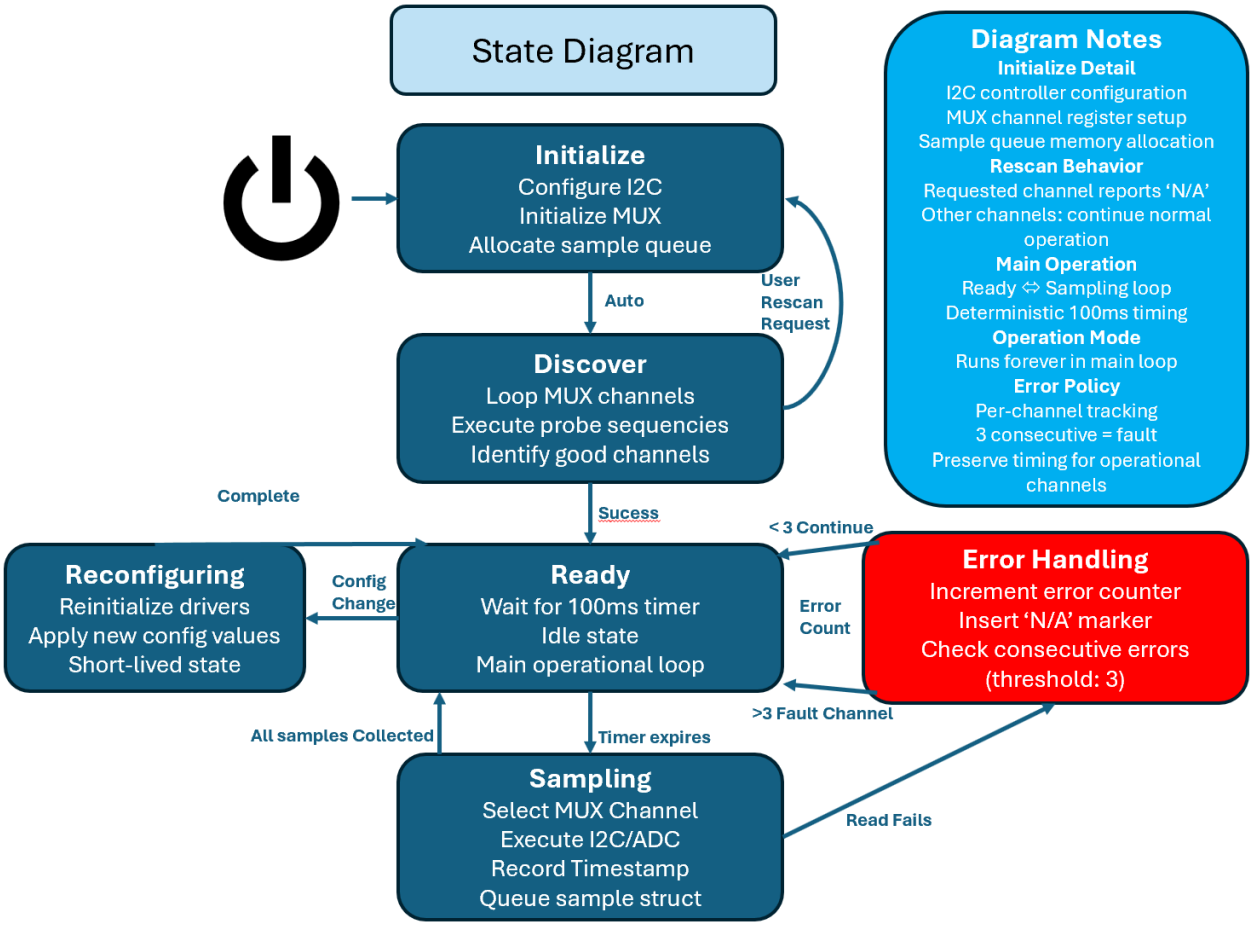


Figure 7.3 - Use case Diagram

7.4 State Diagram - Sampler Task

In the case of the sampler task operating as a finite state machine, there are five primary states. The first of these states is the initial state that only executes one to configure the I²C controllers, initialize the multiplexer control register, and allocate the sample queue. When all this has been done, the task will move to the discover state (automatically) and start a loop where it goes through the multiplexers' channels, executing driver probe sequences. Once the probe sequence has executed successfully and the multiplexer channel is known, the task moves to the ready state and waits for the 100-ms timer interrupt. When the 100 ms timer expires, the task will enter the sampling state. In this state, the task will invoke each of the driver handles in sequence. Each invocation includes the process of selecting the proper multiplexer channel, executing the I²C transaction or ADC conversion, recording a timestamp for the results, and adding the sample struct to the sample queue. The task will remain in the sampling state until all samples have been collected. At this point, the task will return to the ready

state. The task will also move to the error handling state if any of the driver reads fail. The error handling state will cause the task to increment a per-channel error counter and add an "na" marker into the sample stream. The task will stay in the error handling state until there have been three consecutive errors for a particular channel. When this



happens, the system will mark that channel as "faulted", remove its handle from the active list, and return to the ready state to begin collecting samples again from the remaining channels. The user can send a rescan request via the web interface to force a transition from the ready state back to the discovering state for the specified channel. While the user-initiated rescan is in progress, the channel requested by the user will report "na", but the other channels will continue to collect data at their normal rates. Any configuration changes made to the sensors that will require the drivers to be reinitialized with new values will place the task in a short-lived reconfiguring state. The task will then return to the ready state and resume normal operations. The task will run in the ready-sampling loop forever, with error transitions occurring so long as they do not impact the ability of the task to provide deterministic timing for the operational channels.

7.5 Class Diagram - Driver Abstraction

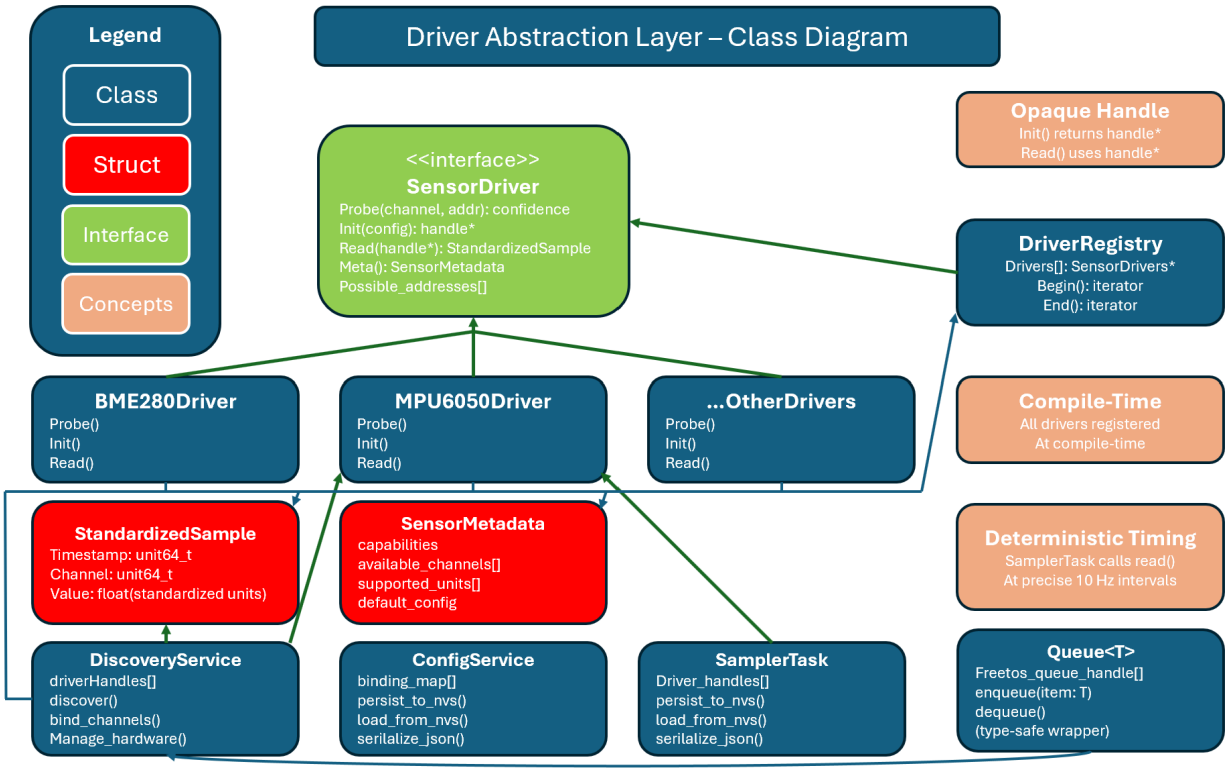


Figure 7.5 - Driver Abstraction Layer - Class Diagram

The driver abstraction layer consists of several interconnected classes and structures. The sensor driver structure serves as an interface contract, having function pointers for four functions: probe, init, read, and meta, and an array of possible I²C addresses. The driver registry class maintains an array at compile-time of all registered sensor driver instances, providing for iterator access for discovery operations. Each concrete driver implementation, such as BME280 driver or MPU6050 driver, implements the sensor driver interface, providing hardware understanding. The probe function provides a multiplexer channel and an I²C address to return a confidence score and optional identifying data. The init function configures the sensor hardware, setting output data rates, filter parameters, and measurement range, returning an opaque handle structure. The read function takes as its parameter this handle structure and fills a standardized sample structure with timestamp, channel identifier, and measurement value in standardized units. The meta function returns a sensor metadata structure that describes capabilities, available channels, supported units, and default configuration. The discovery service class carries the binding process, iterating through multiplexer channels and driver registry, calling probe functions, and managing the driver handle list. The config service class provides persistence to disk by serializing this binding map to NVS as a versioned JSON structure. The sampler task class contains an array of driver handles and calls read functions at 10Hz timing intervals. The queue abstraction wraps FreeRTOS queue primitives, providing type-safe enqueue and dequeue operations for standardized sample structures passed between tasks.

7.6 User Interface Design

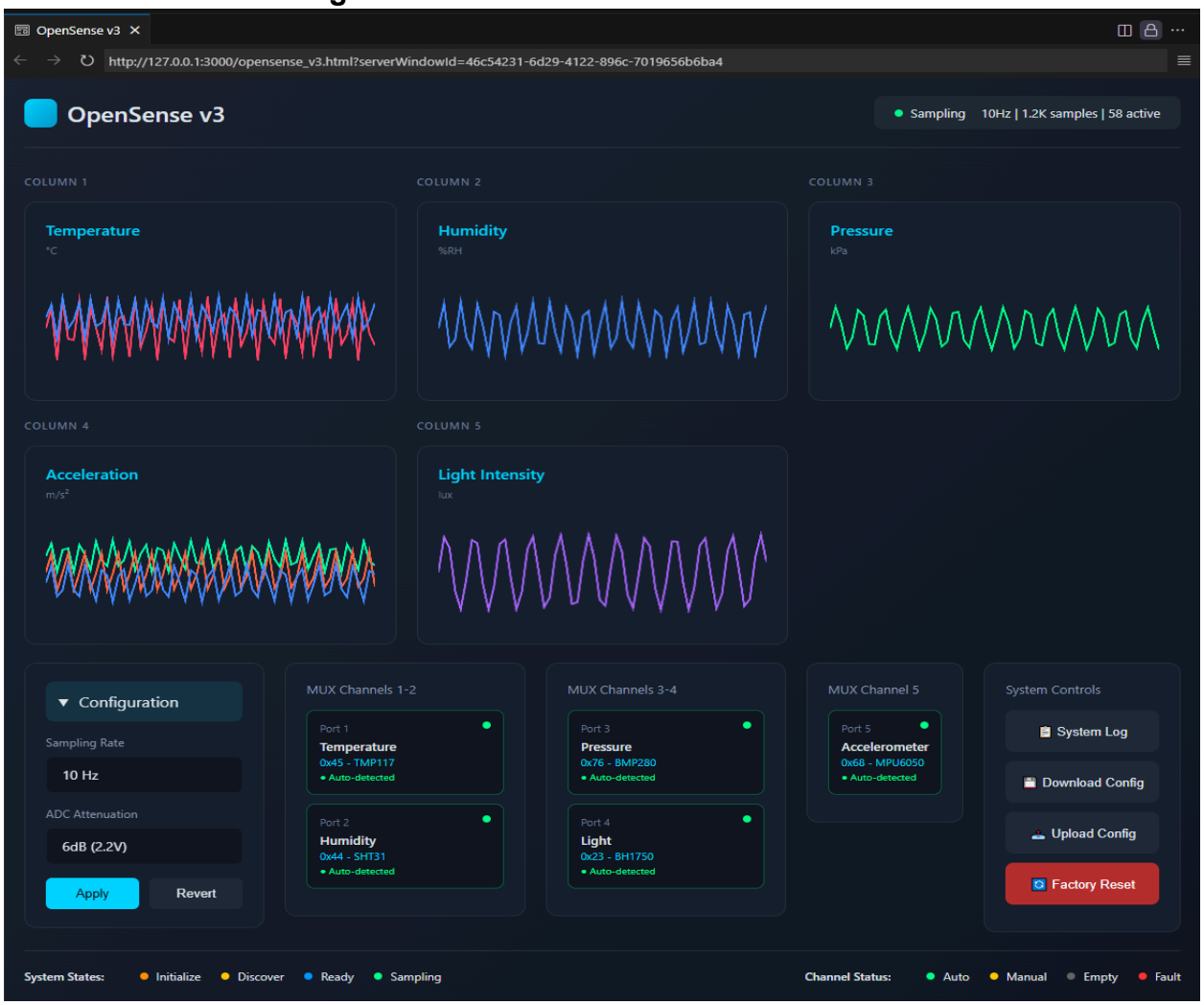


Figure 7.6 - User Interface conceptual

The diagram above is a conceptual image for the user interface of the single page website application.This single-page, fully functional, and simple-to-use web-based application is designed to be viewed and used equally well from desktop computer screens and mobile devices. At the top of the application are five vertically aligned real-time charts, all have the same x-axis that shows the elapsed time since the start of the application (in seconds), each chart has color-coded trace lines representing different sensor channels, y-axis labels show the unit of measurement for each channel and each chart has a ten second rolling window that will be updated at the sample rate that was chosen by the user. These charts are rendered efficiently by Chart.js, and they are implemented using two circular buffers where the oldest buffer samples shift off the beginning of the buffer while the newest samples append to the end of the buffer. Below the charts is a horizontal status bar that indicates some of the most important critical system health metrics: the write status of the SD card is shown as a green/yellow/red color code, the current sampling rate, the total number of samples that have been

logged, and the number of currently active channels. There is a drop-down configuration panel below the status bar that allows users to configure many aspects of the system including: a drop-down menu for each of the eight multiplexer ports that allow users to choose between automatically detected sensors or manual override; text fields for entering the I²C addresses of the sensors in hexadecimal notation; drop-down menus for choosing the attenuation level of the Analog-to-Digital Converter (ADC); numeric fields for entering calibration values with units; and a field for entering the logging rate in Hertz. The configuration panel also includes "Apply" and "Revert" buttons that can be clicked to atomically update the configuration. The "multiplexer channel" view is an 8x8 tile grid where each tile represents a multiplexer port and shows information about that port such as the port number, the name of the current sensor connected to that port, the I²C address of the current sensor connected to that port, and a status icon that indicates whether the port is currently empty, has an auto-detected sensor attached, has a manually assigned driver attached, or if it is faulty. Users can click on a tile to expand its detail view and provide options to re-scan the channel or manually assign a driver to the channel. The navigation menu includes links to the system log, to upload/download the configuration file, and to perform a factory reset on the device. All of the interactive components in this application provide instant visual feedback to the user; all of the components that cannot be interactively modified (i.e., they are disabled) are grayed out to prevent accidental modifications, and all errors are reported back to the user as dismissable toast notifications. The application will only redraw any changed information and, therefore, will continue to run smoothly even when running on relatively modest mobile hardware.

7.7 Data Transfer Diagram

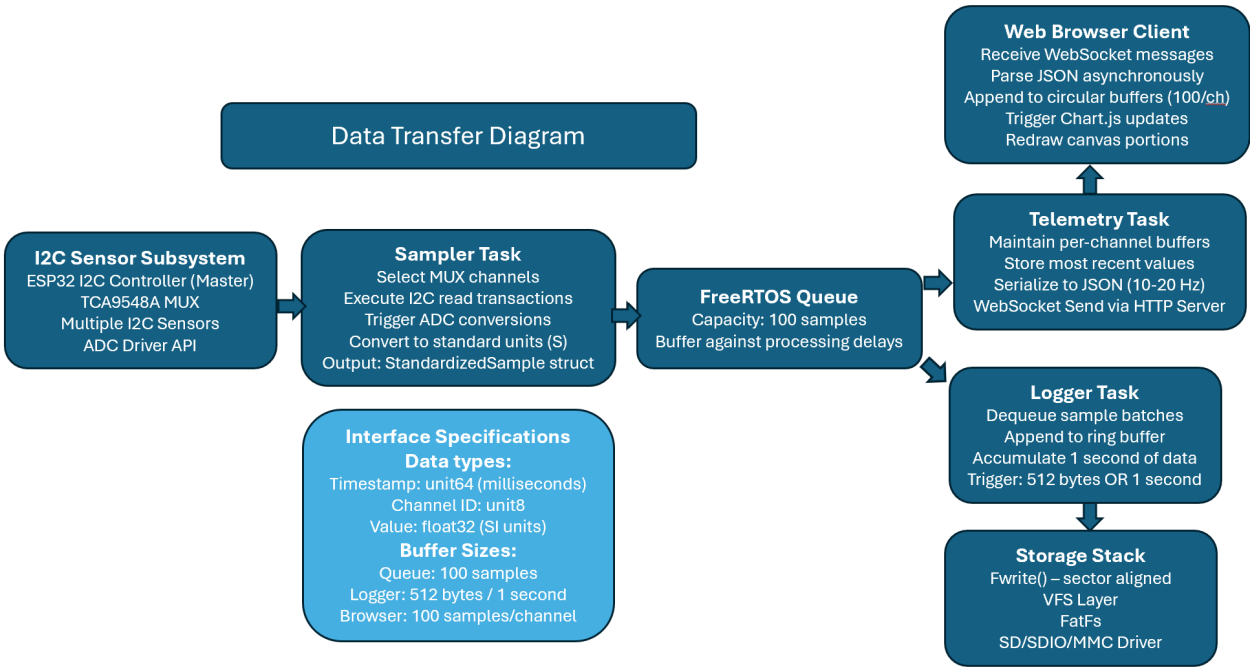


Figure 7.7 - Data Transfer Diagram

The flow of data is a unidirectional flow from each subsystem and has defined interfaces between subsystems. The sensor data originates from I²C devices connected to the TCA9548A multiplexer using the ESP32 I²C controller as master. The sampler task selects multiplexer channels by writing single byte commands to the multiplexer control register at address 0x70 and performs I²C read transactions on sensor registers or triggers ADC conversions using the ADC driver API. The raw sensor readings are converted to standard units within the driver read functions, producing structures of type standardized sample containing timestamps (uint64) in milliseconds-since-boot; channel identifiers (uint8); and values (float32) in SI units. These structures are enqueued into a FreeRTOS queue with a capacity of 100 samples, providing a buffer against temporary processing delays. The logger task dequeues samples in batches, appending them to a ring buffer that accumulates one second of data across all channels. Once the buffer reaches 512 bytes or one second elapses, the task calls fwrite with sector-aligned data passing through the VFS layer to FatFs to the SD/SDIO/MMC driver. In parallel, the telemetry task dequeues the same samples, maintaining separate per-channel circular buffers holding the most recent value. Every 10-20 Hz intervals, the task serializes these latest values into compact JSON frames and calls WebSocket send functions provided by the HTTP server, which fragments and transmits the data over TCP to all connected clients. The web browser receives WebSocket messages, asynchronously parses the JSON payload, appends new values to in-memory circular buffers holding 100 samples per channel, and triggers Chart.js update calls to redraw affected portions of the canvas. Configuration changes flow in reverse: the browser posts JSON to a REST endpoint; the HTTP server handler validates and passes the structure to the config service, which performs atomic writes to NVS flash and notifies the sampler task via a control queue prompting driver re-initialization without disrupting the sampling cadence.

The WebSocket transmission path operates at 10 Hz, with the telemetry task encoding sensor data into compact JSON frames approximately 100 bytes in size. The task maintains a static buffer for JSON encoding, formats data as {"t": timestamp, "c":[values], "s":[status]}, and broadcasts to all connected clients using httpd_ws_send_frame_async. A mark-and-sweep pattern handles send failures: the task first iterates through all clients attempting transmission and marking failures, then removes failed clients from the array in a second pass. This prevents array modification during iteration while ensuring failed clients don't accumulate. The frame drop strategy prioritizes real-time operation over guaranteed delivery, with clients detecting gaps via timestamp discontinuities. Network issues affect only visualization; data logging continues uninterrupted as the logger task operates independently through FreeRTOS queues.

7.8 Data Structures

This application uses several different data structures that are tailored for their unique optimization requirements for embedded real-time systems. The StandardizedSample is a basic data unit (struct) in this application. The StandardizedSample is defined as follows: uint64t time stamp; uint8t channel ID; uint8t sensor type; uint16t bit field to

represent error states; float value representing the measurement made by the sensor in standard units. The total length of the StandardizedSample is 16 bytes, which was determined to optimize for cache efficiency when copying samples. The SampleQueue is defined as a freeRTOS queue handle (sample queue), and is initialized at start-up with a 100-slot capacity. The SampleQueue provides a non-blocking enqueue mechanism from within an interrupt service routine and a blocking dequeue mechanism with a time-out in a task context. The Ring Buffer is utilized to accumulate log buffer data (it is a fixed-size byte array of 4096 bytes with head and tail index counters that are both represented as size_ts). The Ring Buffer provides a mutex to prevent simultaneous access by multiple threads and a fullness indicator that initiates a flush operation when the buffer reaches sector boundaries. The DriverHandle encapsulates the state of a sensor, including its I²C address, MUX channel, and a table of function pointers that provide access to functions to perform various driver actions. Additionally, each DriverHandle contains an opaque void pointer that references the sensor's specific driver state created when the driver's init function is called. In addition to the previously mentioned fields, it also contains fields referencing metadata generated by the driver's meta function. The DriverRegistry is a compile-time array of SensorDriver objects. Each object in this array consists of a set of function pointers that will be called during the driver's initialization process, an optional array of possible I²C addresses that may be used by the driver, and an identifier of the form of a C-style string. This array allows the registry to iterate through the list of registered drivers without any dynamic memory allocation. The ConfigBlob is a representation of the configuration state of the entire system as a single JSON object stored in NVS. The ConfigBlob is composed of the following items: a version number that can be used for atomic rollback; an array of items that maps the MUX channel of a sensor to the name and/or address of its driver; a parameter object containing the calibration coefficients and attenuation levels for the various sensors; and a metadata object containing timestamps and other probing-related information. The telemetry frame is a representation of the data that will be transmitted over the network and is designed to utilize minimal bandwidth. It is composed of a timestamp, a channels array that contains the values of the first five channels that were enabled, and a status array indicating if there were any errors during the previous five seconds. Therefore, the telemetry frame consumes approximately 100 bytes when uncompressed and supports transmitting ten frames every second without exceeding the bandwidth limitations of the 802.11 wireless link.

The WebSocket system is based on a simple array for tracking currently active connections. An array of integers declared as wsclients[MAXWSCLIENTS] contains the socket file descriptor for each of the current active connections. The integer wsclientcount tracks how many of the slots are being used. Ten is the chosen value for MAXWSCLIENTS due to the following reasons. The ESP32's default 12 socket limit reserves 10 sockets for WebSocket connections, and 2 for the transient HTTP requests that may be made during the course of normal operation; each client connection needs about 2 KB of RAM for its TCP buffer, so ten connections need 20 KB of RAM, and it is rare for more than two simultaneous users to connect, i.e., one user's technician's phone and another's laptop. Array access is protected from race conditions by the wsclientsmutex, which is locked prior to accessing the array using the standard

FreeRTOS mechanism: xSemaphoreTake() locks the mutex before the access, and xSemaphoreGive() unlocks it afterward. When the Telemetry Task attempts to acquire the mutex, it will time out after 100 milliseconds and skip sending the telemetry packet. There are 40 bytes required for the array (each element is a four-byte integer, there are ten elements), four bytes required for the counter variable (wsclientcount), approximately 80 bytes for the FreeRTOS mutex structure, and 128 bytes for the JSON encoding buffer, for a total of 252 bytes required statically. As needed for each active client, dynamic memory will be allocated by the lwIP network stack for TCP send and receive buffers, and this will be approximately 2 KB for each client. At full capacity with all ten possible clients connected, the amount of memory consumed will be approximately 21 KB, or less than five percent of the 520 KB of SRAM available on the ESP32. Each of the previously mentioned data structures is designed to avoid the use of dynamic memory allocation once they have been initialized. Therefore, each of them is designed to have a predictable memory footprint and to avoid memory fragmentation when the device is operated in the field for extended periods of time.

7.9 Implementation Considerations

Several key strategies have been included in the design of the sampler to allow the sampler to function appropriately under real-world conditions. The relative priorities of the tasks were defined based on their importance, with the sampler task having the highest priority, thus allowing for a predetermined scheduling. This ensures that the sampler task completes before the next sampling interval starts. The logger task is assigned a middle priority to prevent the logger task from being delayed due to the long time required to write data to an SD card; however, it also ensures that the logger task completes before the next sampling cycle. The lower priority assigned to the HTTP server and telemetry tasks allows them to handle high volumes of bursty network traffic without interfering with the ability to collect data. FreeRTOS queue sizes were determined by performing worst-case latency analysis. The sample queue size is 100 samples. At a sample rate of 10 Hertz, this buffer size would provide 10 seconds of buffer time. Therefore, it would be able to absorb the erase time of SD cards (which is typically multi-hundred milliseconds) and keep the data from being lost.

Static memory allocation is used for all buffers, queues, and driver structures. Static memory allocation prevents memory fragmentation from occurring in memory. Additionally, static memory allocation ensures that memory usage is predictable over long-term deployments. The configuration service uses NVS versioned atomic write operations with read-modify-write method and CRC validation to ensure that if power is lost during data collection, then the last configuration written will be rolled back to the prior stable state instead of producing undefined behavior. The error handling mechanism employs the principle of graceful degradation. Each sensor channel is monitored for failure through a per-channel fault tracking mechanism. Therefore, as long as one or more channels are operating properly, the system will remain operational while indicating which channels failed via both the CSV log and web UI.

The web socket broadcasting methodology minimizes bandwidth usage by utilizing compact JSON encoded messages to transmit payload information and by distributing this payload across all clients, which subsequently reduces the CPU overhead and memory needed to complete the transmission, as opposed to creating custom transmission methodologies for each client. The websocket implementation utilizes several resiliency methods to facilitate the functionality in unreliable network environments typical of field deployments.

A JavaScript client implements an exponential backoff re-connection method, which has a minimum delay of one second, and doubles the delay with a maximum delay of thirty (30) seconds with each successive failed re-connection. The client's "on close" event will automatically retry the re-connection of the web socket until the connection is successfully established, while the client's "on open" event will reset the delay with each successful establishment of the web socket connection. These features enable the field technician to have no manual interaction to recover from network outages, server restarts, or loss of power.

Server-side stale connection detection employs a combination of three different complimentary methods: send failure detection when httpdwssendframeasync returns ESPFAIL, indicating that the send buffer is full or the socket has been closed, the periodic transmission of PING/PONG heartbeat frames every thirty (30) seconds with a ten (10) second response requirement from the client, and a periodic TCP probe to determine whether the connection is still active, initiated every thirty (30) seconds, and repeated every ten (10) seconds. Together, these mechanisms will identify and clean up dead connections within approximately sixty (60) seconds maximum.

The system will exhibit graceful degradation in various failure scenarios; when all clients discontinue communication with the server, the telemetry task will continue to execute; however, it will bypass broadcast operations since wsclientcount equals zero, resulting in reduced CPU utilization from approximately three percent to one percent. Frame drops due to network congestion will result in timestamp discontinuities being visible to the client; however, the data logging will remain completely unaffected, since the logger task is independent and communicates through FreeRTOS queues. If the HTTP server fails or needs to be restarted, all web socket connections will be terminated, but the sampler and logger tasks will continue to operate unabated, and the clients will begin attempting to re-connect in the range of 1 to 30 seconds based upon their current state in the exponential backoff algorithm. The use of snprintf with explicit length checking will provide buffer overflow protection; if the returned length exceeds TELEMETRYBUFFERSIZE of 128 bytes, the frame will be discarded and an error will be logged. The 38-byte safety margin above the worst-case payload size of approximately 90 bytes will prevent memory corruption from maliciously formed sensor data or from value ranges outside of those expected.

SD card logging will utilize sector-aligned writes and batch flushing to reduce the write amplification associated with SD card logging, thereby increasing the longevity of the SD card. Only during the transition between files will there be a need to call fsync explicitly to balance the requirements of data integrity and SD card endurance.

The driver abstraction layer provides a standardized interface for registration of drivers at compile time and dynamic association of the registered drivers with sensors at run time, thus removing the overhead of virtual functions and providing the capability to maintain the advantages of accessing sensors directly on the hardware, along with the advantage of implementing polymorphism for handling sensors.

Chapter 8 - System Fabrication / Prototype Construction

8.1 PCB

For the PCB and schematic we used the KiCad software for creating the design. We included everything all on one board to make it easier to incorporate into the sensor however some of the subsections will get some more depth down below. We took care to follow the guidelines for designing a PCB by making sure the traces are all the right width and the components have the proper distance between them. The PCB design was made by using the previous schematics developed on KiCad and manually converting it to a PCB format and going through it to place the components in their optimal positions and making sure we minimize noise and signal degradation. All components were also clearly labeled as well to make the soldering process smoother by being able to tell which is which. By making sure the PCB is as well designed as possible we hope to improve performance and reliability.

Our MCU board contains the ESP32-Wroom, the SD Card slot, the power system, as well as all the other essential components in order to reduce the need for additional boards other than the sensors that will be plugged in. As our intention is to make our sensor as portable as possible we tried to avoid making the design too bulky since a complicated PCB shape and size will make it harder to secure a case that fits it properly. This is the main reason we decided to include everything built in onto the MCU board rather than creating separate smaller boards that connect to it although after testing it we will see if we need to adjust our approach.

Some other notable things about our PCB is that the connectors on the right edge are placed there to simplify the wiring for the analog inputs off the board. The LEDs are also placed in visible locations near the relevant ICs that control them to make it easier to see if they work correctly. There are also some test pads spread throughout the entire PCB to make it easier to test the key nodes without disturbing any of the actual connectors. The female headers for the external sensor boards are also placed on the bottom of the PCB with each port having various components like the switches and capacitors to make sure each sensor will be independently powered and monitored in order to save energy on unused sensors.

Below are our initial PCB layouts.

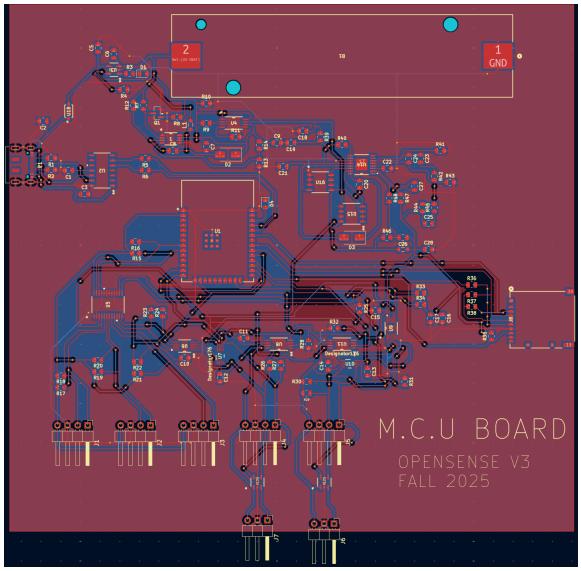


Figure 8.1: 2D MCU Initial Board Layout

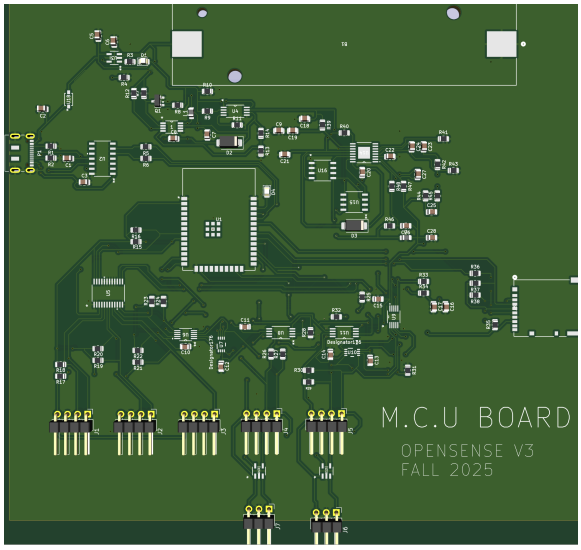


Figure 8.2: 3D MCU Initial Board Layout

Below is a zoomed in focus on the power portion of the PCB.For the power system we have implemented two voltage regulators, one to serve as a boost to keep part of the board at a consistent 5V while the other regulator serves as a standard buck converter. The USB port is placed in the top left corner of the PCB to make it easier to access. Employing these two regulators will help keep our components safe and reduce overheating in the event of switching from charging with the cable to using the batteries as that switch could have a negative effect if not properly accounted for. Each of the regulators is surrounded by its inductor and diode and the capacitors nearby are there to minimize the switching loop area.

Our power system is also set up so that the device can run on just the batteries but when plugged into the USB cable it will automatically switch power sources and begin to use the charger cable to provide more input voltage. The Li-ion charger is placed lower than the USB VBUS with battery connectors nearby so that the high-current charge path is short. The sense resistors and MOSFETS serve as protection for the battery and are located near the charger IC as well as its decoupling capacitors to prevent it from being damaged or overcharged. By placing this area away from the actual MCU we are trying to limit any thermal coupling and overheating.

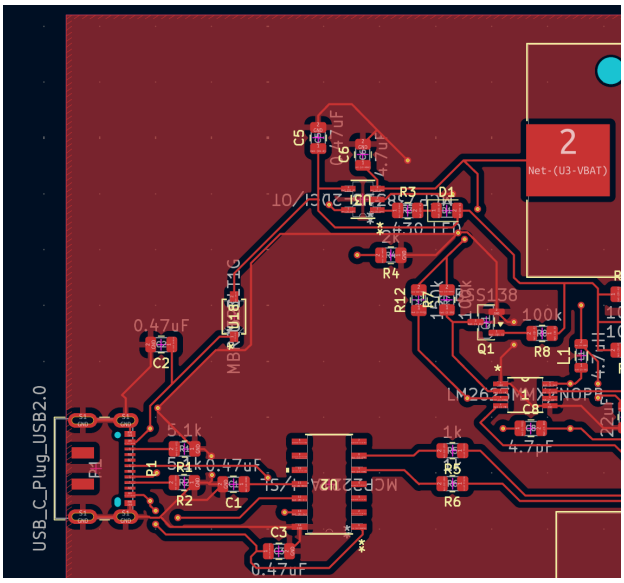


Figure 8.3: 2D Power and Regulator Close-up

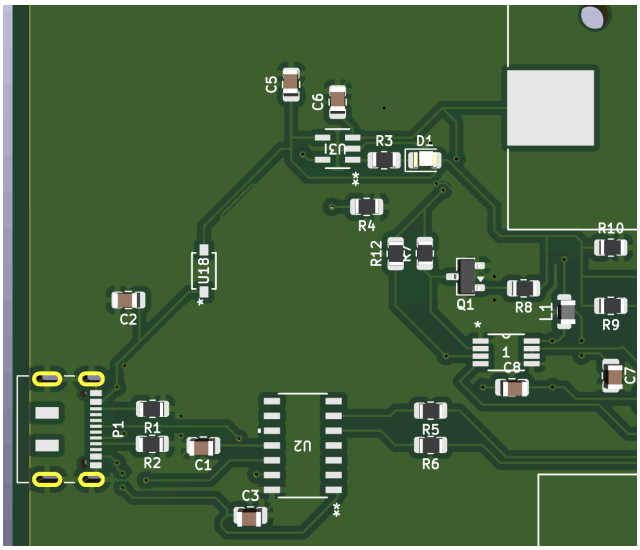


Figure 8.4: 3D Power and Regulators Close-up

We also made full use of the capabilities of the ESP32-Wroom in our PCB design and it was very important for making our entire design work. Ensuring that we connected everything properly to all the right ports was a bit of a challenge but with that we can ensure that it will function properly when tested. The reason the ESP32 was placed in the center is so that the GPIO routes to all six sensor ports without any complicated and long traces. Each module supply pin also has decoupling supply pins nearby with short vias to the ground plane. We also made sure to place the reset and boost buttons, the programming header, and the status LED near the MCU but also close to the edge of the board to make it easy for us to access and program without interfering with other components or risking damaging the board. Below is a brief close up on the connections for the ESP32-Wroom we used.

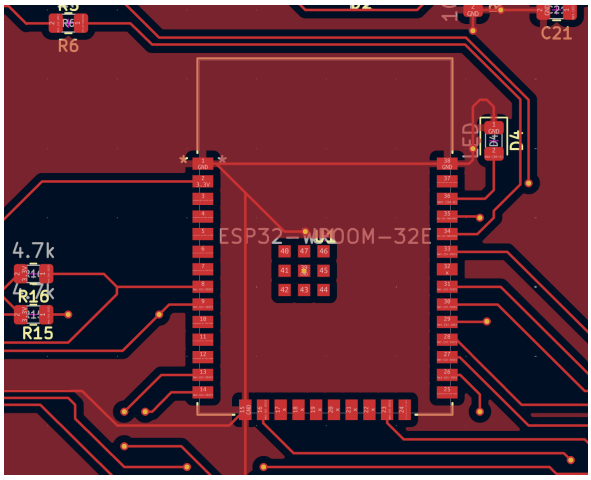


Figure 8.5: 2D Close-up of MCU

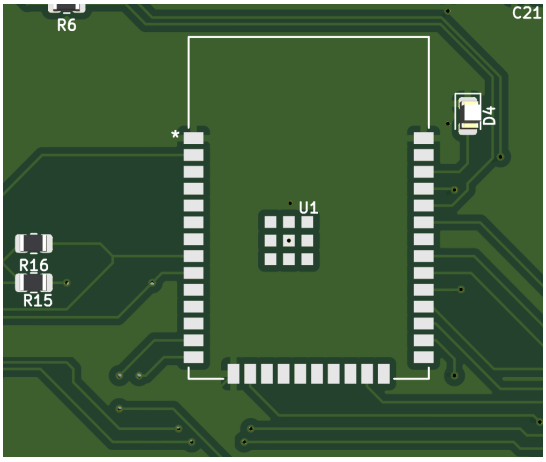


Figure 8.6: 3D Close-up of MCU

8.2 Sensors

We have also created PCBs for some of the sensors we will be using below. These sensors include a temperature sensor as well as a pressure sensor that will be supported by our project by default. They are relatively simple but should work well when attached to our main board. Our goal is for them to be interchangeable so that people can plug in their own sensors to use but it's still important to include a design for the initial ones that we will be using.

The temperature sensor pcb relies on the LMT87 sensor for it to work. The datasheet lists that usually it's better to connect the PCB to a ground and power plane however according to an Altium blog we found that it can sometimes affect the results of temperature tests if the pour is under the sensor but not under any other elements. As a result we chose to use traces to connect it instead to enhance the accuracy. The capacitors are also connected in a way to enable a SAR to ADC to draw bursts of current to further help the sensor get accurate readings without interference as it keeps the output voltage where it should be.

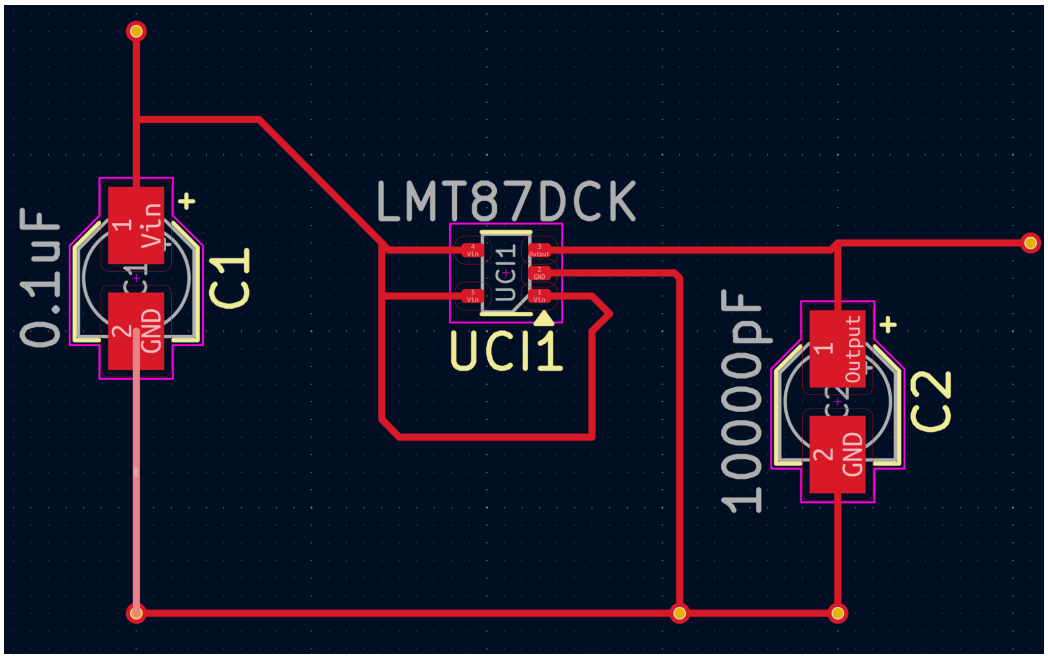


Figure 8.7: 2D Closeup of Temperature Sensor

For the pressure sensor, we plan on using a Piezoelectric Transducer such as the Murata 7BB-27-4L0 which will detect the pressure changes and change the output voltage to calculate the pressure. A high force will make the output go up to the input voltage while a moderate force would only produce about half of that. The high resistances of the resistors are to set the gain and the high-pass and low-pass behavior and R1 will set the DC return. The sensor will also use an op-amp which will take in the small signal from the piezo and then amplify it with the controlled gain and bandwidth so create a low impedance output used by the rest of the circuit.

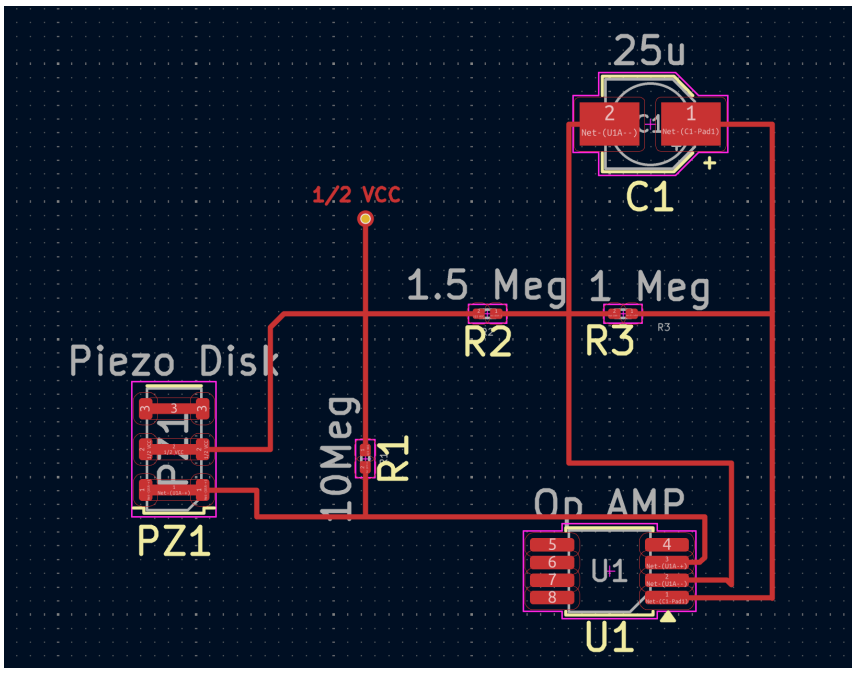


Figure 8.8: 2D Closeup of Pressure Sensor

The monochromator's electronics are integrated onto a single PCB, which supports the two primary functional blocks of the system: the servomotor control circuitry and the transimpedance amplifier stage for the photodiode signal. Consolidating both functions on one board simplifies the design, reduces wiring complexity, and allows a single microcontroller to coordinate all data acquisition and communication tasks.

The PCB includes the ATtiny84 microcontroller, which generates the PWM control signal for the servomotor and performs analog-to-digital conversion on both the servo feedback signal and the photodiode amplifier output. To support external mounting of the mechanical components, the board provides a 4-pin male header for the servomotor, carrying power, ground, the PWM control line, and the potentiometer feedback line from the internal position sensor. Similarly, the photodiode is connected to the TIA input through a 2-pin female header, allowing the detector to be positioned at the output of the optical path rather than directly on the PCB.

This layout ensures that the PCB remains compact and electrically clean while still enabling the optical and mechanical components to be physically located where required in the spectrometer assembly. The complete closeup of the PCB is shown below.

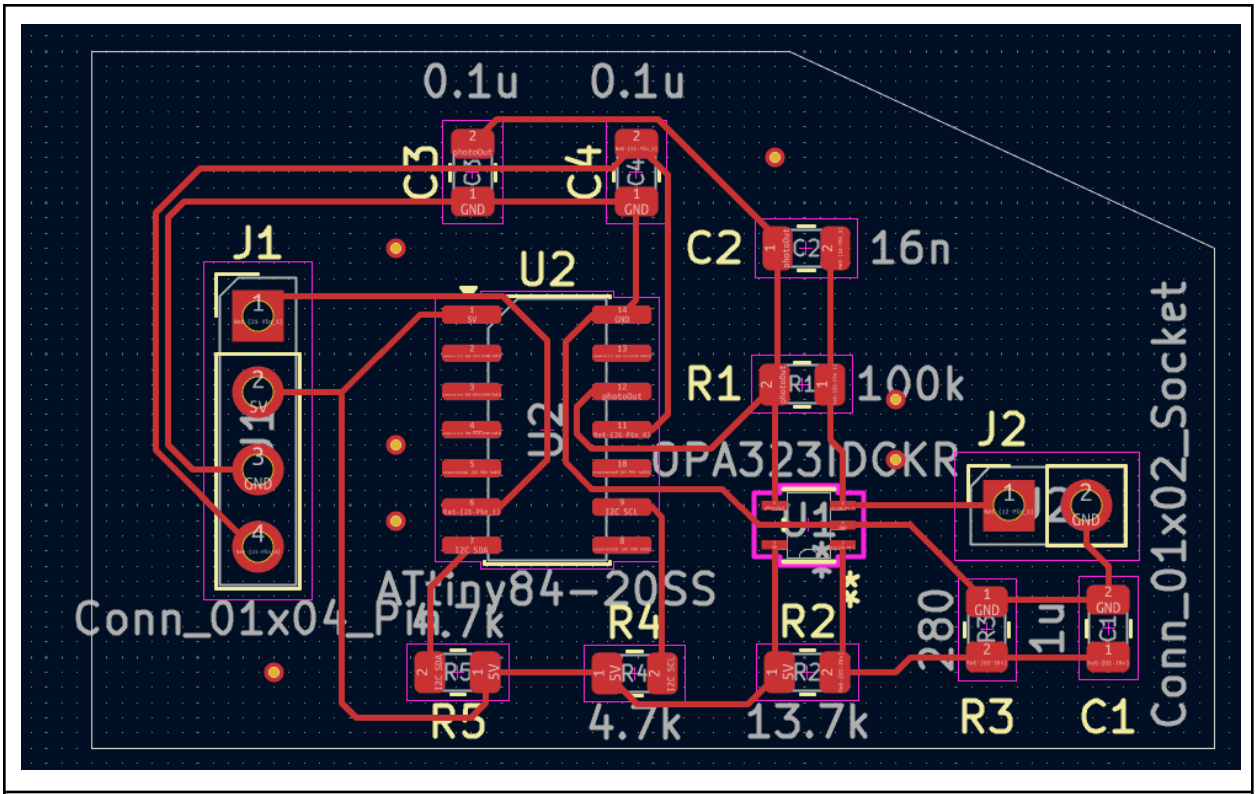


Figure 8.9 2D Close-up of Monochrometer PCB

Chapter 9 - System Testing and Evaluation

9.1 - Optoelectronic Feasibility

To verify the feasibility of the optical design, a physical demonstration setup was constructed, as shown in the image below:

Figure 9.1 - First Spectrometer Prototype

This prototype served as a simplified model of the final spectrometer, using lenses in place of the collimating and focusing mirrors to emulate their optical behavior. The diffraction grating was mounted on a rotational stage, allowing observation of how the diffracted wavelength shifted with changes in grating angle. The experimental results closely matched the theoretical predictions presented in Chapter 6, confirming the validity of the optical design and its calculated parameters.

During assembly and testing, a minor issue was observed: the output beam drifted laterally instead of remaining stationary during rotation. Further analysis revealed that this was caused by a slight misalignment of the rotation axis, which was not positioned directly beneath the diffraction grating surface due to the mounting used in construction of the prototype. This finding highlights the importance of precise mechanical alignment in future builds to ensure consistent beam positioning.

A second prototype was constructed with the parts chosen in chapter 3 to more accurately test the system. The major changes from the first prototype included the introduction of mirrors for focusing and collimation rather than the use of lens, as well as the use of an entry slit instead of a pinhole. A figure showing this prototype is below:

Figure 9.2 - Second Spectrometer Prototype

From this test, a few observations were made. The first is that not enough light was able to enter the system through the chosen entrance slit to make the device practical for observing ambient light or sunlight. In order to correct this, a wider entrance slit or a focusing element before the entrance is necessitated. Furthermore, in the construction of the prototype the angle of the collimating mirror was too steep, producing aberrations. This was mainly due to limitations in the breadboard spacing and will be fixed for the final design.

In addition to the physical demonstration, the team plans to conduct a comprehensive optical simulation of the spectrometer using Zemax ray-tracing software. This simulation will model the full optical path, allowing detailed analysis of system performance, alignment tolerances, and potential aberrations. However, at the time of this report, the Zemax simulation is still in progress and will be completed in a later phase of the project.

9.2 - Power Testing

One of the main focuses for our project is the ability for it to seamlessly transition from being powered by the battery to being powered by the charging cable when plugged in and also when the cable is unplugged. In order to properly test this we constructed the power system on a breadboard including the USB breakout board to be able to plug in the power and use a power supply to simulate the battery to reduce the risk of accidents

from using the lithium batteries in a lab. We had the power supply give an input voltage of 3.7 V and tested to see if the output would be 3.7 V as expected. Then we plugged in the USB charger and we had to see what the output would be and got 4.8 V which was in the range of acceptable values. This gave us a bit of trouble at first as we were unable to get that 4.8 V reading consistently until we realized our TPS2110 was shorting and replaced it with a different one that immediately fixed the problem.

We then tested having both sources active at the same time and the board was able to accurately detect the USB connection and prioritize it over the power supply and give us the 4.8 V output. With this we know that the power detection and switching system works correctly and should continue to work once we properly implement the actual battery rather than the power supply. This should ensure that when our project is done, it should be able to run on battery power when it's actually being used somewhere to collect data and afterwards can be recharged with the USB and still be used while plugged in. The USB-C port that we connected to the breadboard worked fine, however Dr. Weeks suggested to us that we consider alternatives due to how delicate the USB-C can be. As a result it might change in the future once we are able to test different types such as the USB type A that he recommended instead.

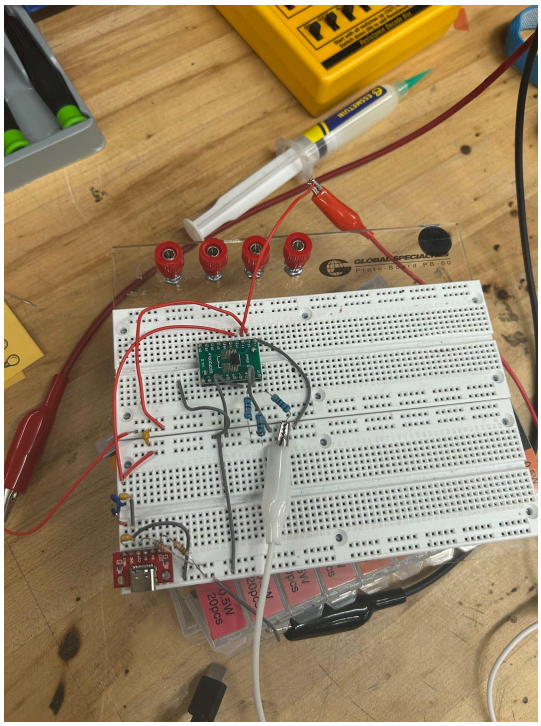


Figure 9.3 - Breadboard connected to power supply without charger

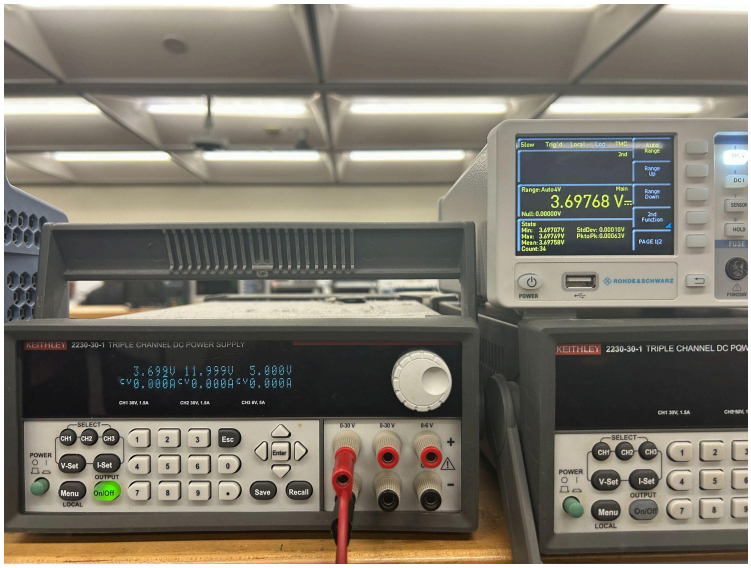


Figure 9.4 - Output without charger

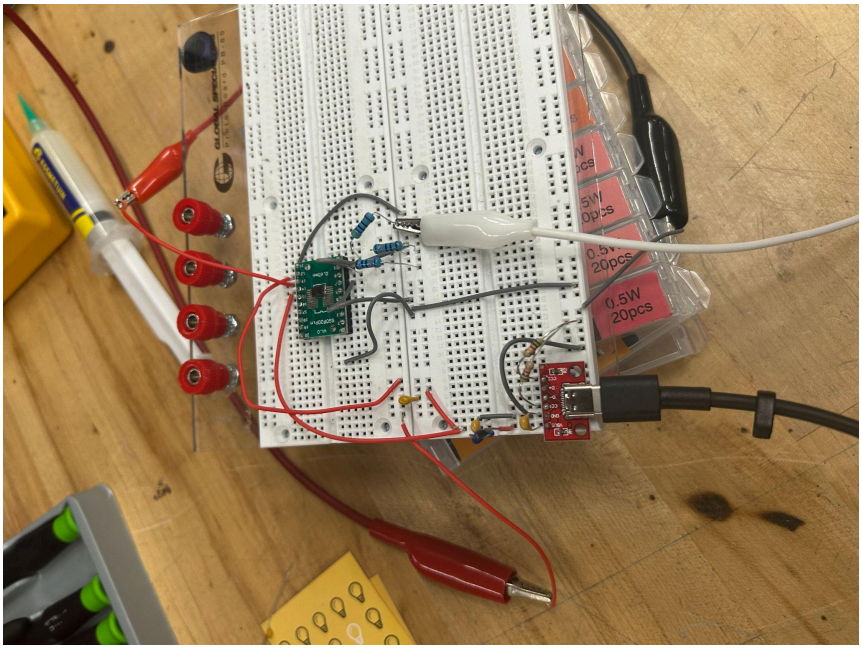


Figure 9.5 - Breadboard connected to power supply with charger plugged in

Figure 9.6 - Output with charger and power supply

When we attempted to test the regulators we purchased, we found that they did not work very well with our breadboards. This is due to the nature of switching regulators — they require a tight layout and high frequency operation. Both the parasitic capacitance and the stray inductance present on breadboards create unstable operation environments, which causes both the LM2623 and the LM4702 regulators to fail in operating at their specified conditions. We therefore have difficulty establishing a reliable working condition to validate the regulators' behavior using a temporary setup; and thus we are now looking at purchasing additional regulators just for testing purposes to verify our design strategy prior to committing to the final PCB.

As an alternative means of validating our regulators, we ran some simulations to help us understand what we could expect from our regulators in a proper layout and to identify some possible alternatives. With simulations, we can determine the expected performance characteristics (including output voltage, switching stability and response to input changes) without the interfering effects caused by the use of a breadboard. In particular, we simulated the LT1767 as a substitute boost regulator in place of the LM4702, and the simulation results indicated that the LT1767 could regulate an input voltage range of 4.7 V to 5.1 V into a stable output voltage of 3.3 V. Thus, the LT1767 would be a viable alternative if the LM4702 fails. Currently, our intention is to test the original regulators selected when the components become available and to move to a more suitable testing environment. The backup regulators we identified via simulation, such as the LT1767, will only be considered if the primary regulators do not operate as intended in the final PCB configuration. Thus, we will ensure that we achieve our desired design while providing a valid contingency option.

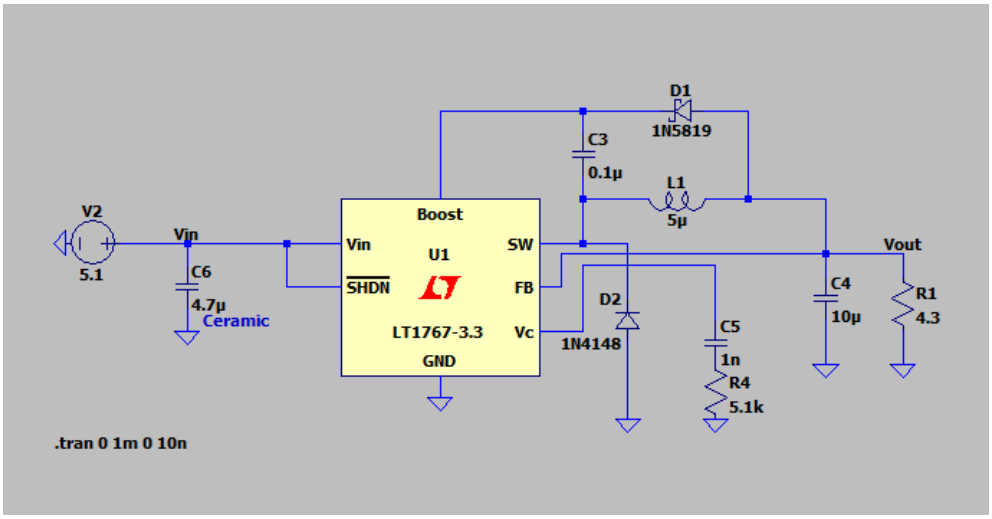


Figure 9.7 - Simulated LT1767 regulator circuit

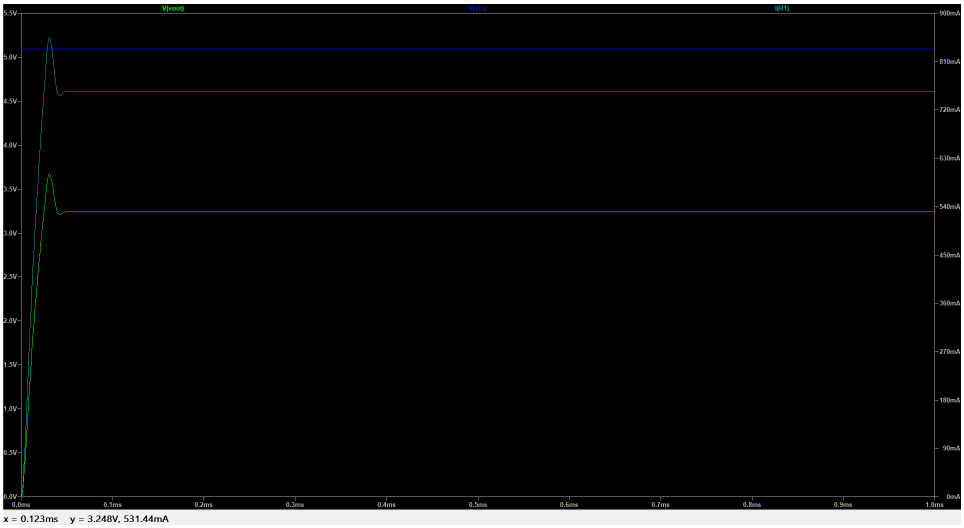


Figure 9.8 - Simulated LT1767 regulator results

We also found an appropriate back-up regulator - the LT1302-5 to support the reliability of both validation and possible substitution of the LM2623. Given that this regulator is connected to the battery path, it serves a vital function within the total power structure of the board. It is very important that this stage is functioning properly at all times. The existence of a second option provides additional confidence in our design, as it ensures that should the LM2623 not perform as expected when built onto the PCB, the LT1302-5 will serve as a fully functional substitute.

In order to determine whether or not this back-up option would work, we created a simulated test circuit and analyzed the data produced by the simulation. We simulated the LT1302-5 with respect to the expected battery input range, paying particular attention to its capability to maintain a stable output voltage from its booster under various loading conditions. Based upon the data provided by the simulation, the LT1302-5 demonstrated consistent performance, successfully converting an input of 3.5V into a regulated output of 5V. Therefore, the results of the simulation demonstrate that the LT1302-5 is able to meet the requirements of our battery powered sub-system, and therefore demonstrates that it would be suitable for use in the event the LM2623 fails to provide the required functionality during the physical prototype phase.

Although our ultimate plan remains to utilize the LM2623 in our final product, the availability of the LT1302-5 provides us with a reliable back-up option, should the primary regulator fail to provide the required functionality during the physical prototype phase. By utilizing the LT1302-5 as a back-up option, we are able to proceed with the knowledge that the battery boost portion of our system has been validated through both simulation and back-up planning.

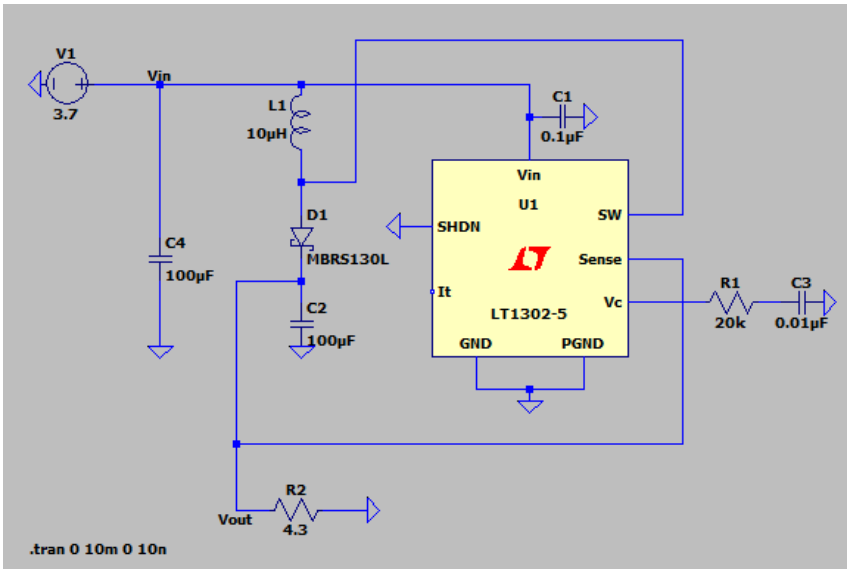


Figure 9.9 - Simulated LT1302 regulator circuit

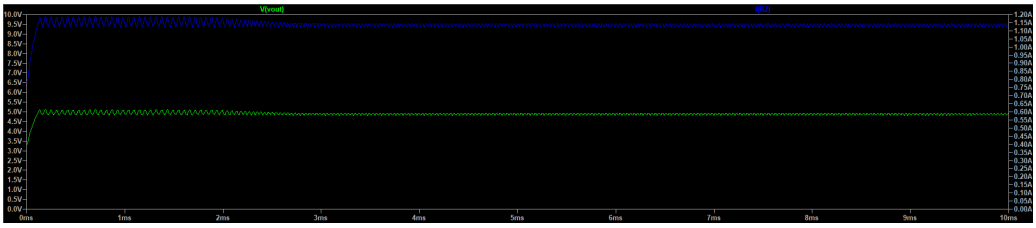


Figure 9.10 - Simulated LT1302 regulator results

9.3 - Software Testing

Checking the software integration with the functionality of the microcontroller unit (MCU) used in this project, the ESP32-WROOM, was essential to ensure proper communication with the sensor array. For this demonstration, the ESP32 development board was connected to a BME280 environmental sensor, which measures temperature, humidity, and barometric pressure. The firmware was developed using the ESP-IDF framework and follows a modular, task-based architecture built upon FreeRTOS. At the core of the system is the sampler task, a high-priority FreeRTOS task operating on a 100-millisecond timer interrupt, allowing it to achieve a consistent 10 Hz sampling rate. The sampler task also implements a state machine that transitions through initialization, ready, sampling, and error-handling states. Each channel includes fault detection, allowing the system to degrade gracefully and continue operation on functioning channels even if one sensor fails.

Upon boot, the discovery service automatically scans for connected sensors. The system utilizes a driver abstraction layer with a compile-time registry of sensor drivers, each implementing standardized probe, initialize, and read functions. This design allows for easy extensibility, as new sensors can be added by simply registering additional drivers. The current implementation includes drivers for the BME280 environmental sensor and the MPU6050 six-axis motion sensor.

The ESP32 hosts an embedded web server using ESP-IDF's native HTTP server component, providing real-time visualization of the sensor data. The device broadcasts its own WiFi access point, and once connected to this network, users can access a single-page application dashboard by navigating to the device's IP address in a browser. The telemetry task broadcasts sensor readings over a WebSocket connection at 10 Hz, transmitting compact JSON frames of approximately 100 bytes each. The web dashboard displays five real-time charts with a ten-second rolling window, a channel status grid, and a configuration panel for adjusting system settings. The WebSocket client implements automatic reconnection with exponential backoff to maintain a stable connection. Additionally, the logger task records all sensor data to an SD card via the SDMMC interface, writing CSV-formatted entries with timestamps, channel identifiers, sensor types, values, and error counts, with periodic flushes occurring every second to ensure data integrity.

Chapter 10 - Administrative Content

10.1 - Budget

This project is sponsored and funded by D.R.A.C.O. Labs. To ensure responsible use of this support, one of our design goals is to keep the system low-cost; with that being said, the group has agreed upon a maximum budget of $600. This amount accounts for the total cost of our project, including prototyping, fabrication, testing, and revisions. The budget allows room for error, however, with careful planning and resource management, we are confident this target can be achieved. A $60 figure is the goal for cost per device in regards to volume production after creating a successful platform. This budget is inspired by our first bill of materials and past iterations of this same OpenSense project. As previously mentioned, one of the main intentions of the platform is to be low cost, so keeping this number by volume around the cost of previous versions, if not less, is an important design constraint to consider.

10.2 - Bill of Materials *(Tentative)*

Table 10.1- Bill of Materials

Component	Part Number	Qty.	Unit Price	Total Price	Vendor
MCU	ESP32-WROOM-32E	1	$10.00	$10.00	Amazon
Power MUX	TPS2110A	3	$2.64	$7.92	DigiKey
I²C MUX	TCP9548APWR	1	$1.32	$1.32	DigiKey
IC Switch	TS3A24157DGSR	2	$.89	$1.78	DigiKey
Analog-To-Digital Converter	MCP3221A5T	2	$1.91	$3.82	DigiKey
Level Translator	TXS0102DCUR	2	$0.56	$1.12	DigiKey
SD Card Reader	410-380	1	$7.55	$7.55	DigiKey
USB-to-UART Bridge	MCP2221A	1	$2.53	$2.53	DigiKey
RED LEDS	SML-D12U1WT86	3	$0.12	$0.36	DigiKey
YELLOW LEDS	LTST-C191KSKT	5	$0.15	$0.75	DigiKey
GREEN LEDS	LTST-C170GKT	5	$0.10	$0.50	DigiKey
Reset Button	PTS645SM43SMTR92	1	$0.32	$0.32	DigiKey

Table 10.1- Bill of Materials					
Micro SD card	2648-SC1629-ND	1	$8.93	$8.93	DigiKey
Regulator chip	LM2623AMM/NOPB	1	$2	$2	DigiKey
2nd Regulator chip	LM274002MH/NOPB	1	$3.88	$3.88	DigiKey
Concave Mirrors	CM127-025-G01	2	$46.40	$92.80	ThorLabs
Diffraction Grating	GH13-12V	1	$106.47	$106.47	ThorLabs
Photodiode	TEMD5080X01	1	$1.9	$1.9	DigiKey
Servo Motor	Adafruit 1404	1	$14.95	$14.95	DigiKey
Op-amp	Opa323	1	$0.74	$0.74	DigiKey
Microcontroller	ATTiny84	1	$1.62	$1.62	DigiKey
Lithium-Ion Battery Holder	1042	1	$5.58	$5.58	DigiKey
Lithium Battery	18650 Rechargeable Battery	1	$15.99	$15.99	Amazon
Battery Charger	MCP73831T-2DCI/OT	1	$2.79	$2.79	DigiKey
Schockty Diode	MBR0520LT1G	1	$0.23	$0.23	DigiKey
USB-C Breakout Board	Adafruit 4090	1	$2.95	$2.95	Adafruit
Transistor	A04468	2	$4.10	$8.20	DigiKey
Transistor	BSS138	1	$0.28	$0.28	DigiKey
Diode	SS14	1	$0.29	$0.29	DigiKey
Diode	SS34	1	$0.67	$0.67	DigiKey

10.3 - Project Milestones

Table 10.2 - Project Milestones

Project Initialization				
No.	**Milestone**	**Description**	**Start Date**	**Anticipated End Date**

1	Finalize Project Idea	Group meets in person and discusses project idea, routes of success, etc., and decides on project path	8/18/25	8/26/25
2	Meet with Mentor	Group meets in person with Dr. Mike Borowczak - *project mentor* - to discuss project path and inquire about additional goals he has set for the project	8/26/25	9/2/25
3	Divide and Conquer Report	Complete the first 10 pages of the final document (DNC Report)	8/26/25	9/5/25
4	Divide and Conquer Meeting	The group will meet with the Advisors to discuss the project path through the DNC report	9/8/25	9/8/25
5	Divide and Conquer Revision	Group takes feedback from Advisors and makes necessary adjustments to the project path	9/8/25	9/12/25
6	60-page draft	A 60-page draft is generated fo it r the final paper	9/12/25	10/31/25
7	60-page draft revision	The group takes feedback and makes necessary adjustments.	10/31/25	11/7/25
8	120-page report & demo	Final Paper and Demonstration of the project idea and the project path	11/7/25	11/25/25

Project Fabrication				
No.	**Milestone**	**Description**	**Start Date**	**Anticipated End Date**
1	Major Component Selection	The group decides on what major components will be the best fit for the project path	1/5/26	TBD
2	Design Schematic	Group develops a design that will best work for the project path	TBD	TBD
3	Design the front end of the application	The group will begin working on the application that will send and receive the sensor values	TBD	TBD
4	Order Prototype	Group orders the designed PCB and test to see if it works as expected and what needs to be improved	TBD	TBD
5	Refine Design	The group will redesign the schematic based on how	TBD	TBD
6	Meet with advisor	The group takes feedback from Advisors and	TBD	TBD

	and sponsor	makes necessary adjustments to the project path after showing them the prototype		
7	Order the refined version and test	The group will take all the feedback and improvements to order the final PCB and test it, and make sure the application also works	TBD	TBD
8	Finish the paper	The group will finish writing about everything we have experienced with the project	TBD	TBD
9	Final paper and presentation	The group will submit the final paper and present our finished sensor to the review board	TBD	TBD

Chapter 11 - Conclusion

The OpenSense v3 platform is being developed to address the need for an affordable, transparent, and highly customizable sensor system, especially within educational environments where accessibility and adaptability are essential. Existing solutions are often costly and closed-source, limiting student engagement and preventing users from understanding or modifying the underlying technology. Our work expands on previous iterations of the OpenSense project by focusing on a platform that is customizable to accept user-defined sensors while still keeping the requirements of being low-cost, easy to use, and capable of supporting both I²C and analog sensors without requiring users to alter firmware. This ensures that students, teachers, and hobbyists can incorporate a wide range of sensors while maintaining a simple and reliable user experience

The aim of our platform centers on creating a low-cost, user-friendly, and highly customizable sensor system that can support a broad range of educational and technical applications. The project aims to enable transparent data collection from multiple sensors simultaneously, provide onboard storage through SD-card logging, and present measurement results through a browser-based graphical interface hosted directly on the device. Basic objectives focus on reliable data acquisition and logging, open-source documentation, and ease of configuration, while more advanced goals expand the system with modular power management, improved portability, extended battery operation, and support for user-defined sensor models. Stretch goals include wireless data transfer, deeper automation, and additional enhancements that improve usability and versatility while keeping cost and accessibility at the forefront of the design.

During the design and development, our team managed several constraints involving balancing performance, usability, and affordability while working within practical engineering and time limitations. Cost is a major consideration, as the platform is intended to remain accessible for educational use, leading the team to emphasize affordable components and efficient design choices. Time constraints also influence development, with PCB fabrication, testing cycles, and team scheduling requiring

careful planning to avoid delays. Usability is another key constraint, since the system must be operable by students and educators without advanced technical training, requiring clear documentation, intuitive interfaces, and straightforward setup. Finally, the platform must maintain portability and reliable operation, meaning size, power consumption, and system robustness must be managed throughout design and implementation.

Each component in our design was considered with both our goals and constraints in mind. The ESP32-WROOM microcontroller was selected as the system's core due to its processing capability, onboard wireless connectivity, and compatibility with the ESP-IDF framework. To support multiple I²C devices simultaneously, the design incorporates a TCA9548A I²C multiplexer, allowing each sensor channel to operate independently without bus conflicts. Since the platform supports both 3.3V and 5V sensors, level shifters and the TPS2110A power multiplexer were included to safely handle mixed-voltage conditions. For analog sensors, the MCP3221 ADC converts analog outputs into digital measurements for the MCU. A micro-SD card provides local data logging, allowing operation without constant network access. Additional components such as USB-C power and charging circuitry, battery systems, switching ICs, and communication hardware ensure reliable power distribution, data integrity, and expandability.

The software for OpenSense v3 is designed to provide a reliable, deterministic, and fully autonomous sensor acquisition system, and it is built around the ESP32 using the ESP-IDF framework with FreeRTOS to coordinate all system operations. Its purpose is to simultaneously collect real-time data from up to five sensors at 10 Hz, log that data to an SD card, and broadcast live telemetry to any connected web clients—all while maintaining strict timing guarantees and preventing any single subsystem from blocking or slowing the others. To achieve this, the system is organized into specialized subsystems: the Configuration Service stores and validates system settings; the Discovery Service scans all multiplexer channels and binds the correct drivers; the Sampling Subsystem triggers I²C reads and ADC conversions on a fixed 100-ms schedule; the Logging Subsystem receives standardized samples via FreeRTOS queues and writes them to the SD card in sector-aligned batches; and the Web Server and Telemetry Subsystem host the entire user interface on the ESP32 and stream JSON-encoded sensor data through WebSockets. Non-blocking queues and strict task prioritization ensure that SD card latency, network traffic, or configuration changes never interfere with the timing of data collection. The software was built using task-oriented concurrency, static memory allocation, driver abstraction layers, and clearly defined state machines, all of which allow the platform to support modular sensor drivers, recover gracefully from errors, and run indefinitely in field conditions without rebooting.

Testing to ensure the key elements of the design were correct was initially performed on a breadboard using SMD-to-DIP adapters so that surface-mount components could be safely and repeatedly evaluated in a temporary setup. This approach allowed the team to prototype each subsystem before committing to a full PCB layout. In total, several systems were tested including but not limited to power regulation, USB/battery

switching, and sensor interfacing. Beyond simply verifying that the circuits powered on, extensive power-path testing was performed by simulating battery input with a bench supply, confirming that the system correctly switched between battery and USB power and that the TPS2110A prioritization behaved as expected. During regulator testing, the LM2623 and LM4702 switching regulators showed poor performance on the breadboard due to parasitic effects, leading the team to run SPICE simulations to confirm expected behavior and assess alternatives like the LT1767 and LT1302-5. These tests, combined with breadboard validation and power-path experiments, provided the confidence needed to finalize key design decisions before ordering the PCB.

Moving forward, our main priorities are assembling the PCB and beginning with building out the full OpenSense platform. Our focus will be on achieving the core project goals required for our presentation and ensuring that the system performs reliably in real-world use. To reduce potential delays caused by part availability or manufacturing timelines, we ordered components early and completed preliminary hardware prototyping in advance. At the same time, working in parallel on both the MCU firmware and the website will help maintain steady progress and ensure that each subsystem remains aligned throughout development.

Appendix A

[1] “Science Probeware & Experiment Software for Teachers.,” Vernier. [Online]. Available: https://www.vernier.com/product-category/

[2] “GlobiSens – Next generation science data logging,” GlobiSens. [Online]. Available: https://globisens.net/products/

[3] “Spectrometers,” Ocean Optics. [Online]. Available: https://www.oceanoptics.com/spectrometers/

[4] “Making a Cell Phone Spectrophotometer | Science Project.” [Online]. Available: https://www.sciencebuddies.org/science-fair-projects/project-ideas/Chem_p100/chemistry/make-a-cell-phone-spectrophotometer

[5] “Senior Design Group 21.” [Online]. Available: https://www.ece.ucf.edu/seniordesign/fa2023sp2024/g21/index.html

[6] *erebus-labs/sense_platform*. (Aug. 21, 2025). C. Erebus Labs. [Online]. Available: https://github.com/erebus-labs/sense_platform

[7] “Spectrometer Configurations: Littrow, Czerny-Turner and More,” Ossila. [Online]. Available: https://www.ossila.com/pages/spectrometer-optics

[8] GCT, “USB4105 – USB Type-C Receptacle, USB 2.0.” USB4105, Feb. 27, 2023. [Online]. Available: https://gct.co/files/specs/usb4105-spec.pdf

[9] Molex, “105450-0101 – USB Type-C Receptacle, 16-Pin.” 1054500101. [Online]. Available: https://www.molex.com/en-us/products/part-detail/1054500101

[10] GP Batteries, “GP Batteries Data Sheet.” GP210AAH. [Online]. Available: https://ind.gpbatteries.com/pub/media/uploads/pdf/rechargeable_NiMH/by_series/standard_series_cylindrical/GP210AAH.pdf

[11] “18650 Lithium-Ion Batteries,” Keystone Electronics Corp/. [Online]. Available: https://www.keyelco.com/category.cfm/Holders-Plastic-PCB/18650-Lithium-Ion-Holders-and-Contacts/id/1198

[12] Panasonic, “Panasonic NCR18650B 3400mAh (Green).” NCR18650B. [Online]. Available: https://www.shoptronica.com/files/Panasonic-NCR18650.pdf

[13] Texas Instruments, “BQ2407x Standalone 1-Cell 1.5-A Linear Battery Chargers with Power Path.” BQ24072, BQ24073, BQ24074, BQ24075, BQ24079, Sept. 2008. [Online]. Available: https://www.ti.com/lit/ds/symlink/bq24074.pdf

[14] “MP2315,” Monolithic Power. [Online]. Available: https://www.monolithicpower.com/en/mp2315.html

[15] Texas Instruments, “TPS56x200 4.5-V to 17-V Input, 2-A, 3-A Synchronous Step-down Voltage Regulator in 6-Pin SOT-23.” TPS562200, TPS563200, May 2023. [Online]. Available: https://www.ti.com/lit/ds/symlink/tps563200.pdf

[16] “Visible Ruled Reflective Diffraction Gratings,” Thorlabs. [Online]. Available: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=8626

[17] “Diffraction Gratings Tutorial,” Thorlabs. [Online]. Available: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=9026.

[18] “Diffraction Gratings Ruled and Holographic,” Horiba. [Online]. Available: https://www.horiba.com/usa/scientific/technologies/diffraction-gratings/diffraction-gratings-ruled-and-holographic/

[19] “Selection of Dispersing Systems,” mks | Newport. [Online]. Available: https://www.newport.com/g/diffraction-grating-selection-guide

[20] “Reflective Holographic Gratings,” Thorlabs. [Online]. Available: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=25

[21] “Reflective Holographic Gratings,” Edmund Optics. [Online]. Available: https://www.edmundoptics.com/f/reflective-holographic-gratings/11855/
[22] “33025FL01-210H Holographic Diffraction Grating,” mks | Newport. [Online]. Available: https://www.newport.com/p/33025FL01-210H
[23] “Optical Mirror Selection Guide,” mks | Newport. [Online]. Available: https://www.newport.com/g/optical-mirror-selection-guide
[24] “Dielectric or Metal: Which Mirror Coating is Better?,” Blueridge Optics. [Online]. Available: https://blueridgeoptics.com/guide-to-optical-mirror-coatings/
[25] “Concave Mirrors: Protected Aluminum (450 nm - 20 µm),” Thorlabs. [Online]. Available: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=1161
[26] “Concave Mirrors: Protected Silver (450 nm - 20 µm),” Thorlabs. [Online]. Available: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=6623
[27] “Concave Spherical Mirrors,” Edmund Optics. [Online]. Available: https://www.edmundoptics.com/f/concave-spherical-mirrors/12242/
[28] “Photodiodes,” Thorlabs. [Online]. Available: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=285
[29] Vishay Semiconductors, “Silicon PIN Photodiode.” TEMD5080X01. [Online]. Available: https://www.vishay.com/docs/81643/temd5080x01.pdf
[30] ams-OSRAM, “Silicon PIN Photodiode with Vλ Characteristics.” SFH 2716 A01. [Online]. Available: https://look.ams-osram.com/m/7b1e3a693e76b757/original/SFH-2716-A01.pdf
[31] Vishay Semiconductors, “Silicon PIN Photodiode.” T1670P6. [Online]. Available: https://www.electroverge.com/images/com_sellacious/products/attachments/5296/562_635a86aad8463-t1670p6.pdf
[32] “Stepper vs Servo,” AMCI. [Online]. Available: https://www.amci.com/industrial-automation-resources/plc-automation-tutorials/stepper-vs-servo/
[33] “Servo Motor Fundamentals,” islproducts. [Online]. Available: https://islproducts.com/design-note/servo-motor-fundamentals/?srsltid=AfmBOorCT_vhphaSAhxJjgDwK44iru9Az5kt_qI1iR2kquQjZGaUU-Pb
[34] “Stepper Motors Basics: Types, Uses, and Working Principles,” monolithic power. [Online]. Available: https://www.monolithicpower.com/en/learning/resources/stepper-motors-basics-types-uses?srsltid=AfmBOoopH7U95V6BD8kCS-IMnimrWSjBzF2TiVNj7RmK7Q32wSuHTcrF
[35] MikroElektronika, “28BYJ-48 – 5V Stepper Motor.” MIKROE-1530. [Online]. Available: https://download.mikroe.com/documents/datasheets/step-motor-5v-28byj48-datasheet.pdf
[36] Analog Devices, “QSH4218-x-10k Hardware Manual.” QSH4218-51-10-049. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/QSH4218-x-10k_datasheet_rev1.50.pdf
[37] Adafruit, “Small Reduction Stepper Motor - 5VDC 32-Step 1/16 Gearing.” 858. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/1155/858_Web.pdf
[38] Espressif, “HTTP Server ESP-IDF Programming Guide.” [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/protocols/esp_http_server.html
[39] me-no-dev, “ESPAsyncWebServer Async Web Server for ESP8266/ESP32.” [Online]. Available: https://github.com/me-no-dev/ESPAsyncWebServer
[40] I. Plauska, A. Liutkevičius, and A. Janavičiūtė, “Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller,” *Electronics*, vol. 12, no. 1, 2023, doi: 10.3390/electronics12010143.

[41] F. Guan, L. Peng, L. Perneel, and M. Timmerman, “Open source FreeRTOS as a case study in real-time operating system evolution,” *J. Syst. Softw.*, vol. 118, pp. 19–35, 2016, doi: https://doi.org/10.1016/j.jss.2016.04.063.
[42] S. Balakrishna, M. Thirumaran, and V. Solanki, “A Framework for IoT Sensor Data Acquisition and Analysis,” *EAI Endorsed Trans. Internet Things*, pp. 1–13, Oct. 2018, doi: 10.4108/eai.21-12-2018.159410.
[43] Espressif, “Non-Volatile Storage (NVS) Library ESP-IDF Programming Guide.” [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/nvs_flash.html
[44] Espressif, “FAT Filesystem Support (FatFs) ESP-IDF Programming Guide.” [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/fatfs.html
[45] Espressif, “Inter-Integrated Circuit (I2C) Driver ESP-IDF Programming Guide.” [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/I2C.html
[46] Texas Instruments, “TCA9548A Low-Voltage 8-Channel I2C Switch with Reset.” TCA9548A, 2022. [Online]. Available: https://www.ti.com/lit/ds/symlink/tca9548a.pdf
[47] Espressif, “Only one STA can join the SoftAP otherwise ‘max ...’” [Online]. Available: https://github.com/espressif/esp-idf/issues/10511
[48] Espressif, “Analog-to-Digital Converter (ADC) ESP-IDF Programming Guide.” [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/adc.html
[49] M. J. Espinosa-Gavira, A. Agüera-Pérez, J. C. Palomares-Salas, J. M. Sierra-Fernandez, P. Remigio-Carmona, and J. J. G. de-la-Rosa, “Characterization and Performance Evaluation of ESP32 for Real-time Synchronized Sensor Networks,” *Procedia Comput. Sci.*, vol. 237, pp. 261–268, 2024, doi: https://doi.org/10.1016/j.procs.2024.05.104.
[50] Espressif, “SD/SDIO/MMC Driver ESP-IDF Programming Guide.” [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/sdmmc.html
[51] I. Allafi and T. Iqbal, “Design and implementation of a low cost web server using ESP32 for real-time photovoltaic system monitoring,” in *2017 IEEE Electrical Power and Energy Conference (EPEC)*, 2017, pp. 1–5. doi: 10.1109/EPEC.2017.8286184.
[52] C. Project, “Official Documentation.” [Online]. Available: https://www.chartjs.org/docs/
[53] M. I. Labib, M. ElGazzar, A. Ghalwash, and S. N. AbdulKader, “An efficient networking solution for extending and controlling wireless sensor networks using low-energy technologies.,” *PeerJ Comput. Sci.*, vol. 7, p. e780, 2021, doi: 10.7717/peerj-cs.780.
[54] “Wi-Fi Driver - ESP32 - — ESP-IDF Programming Guide latest documentation,” Espressif.
[55] Espressif, “Virtual Filesystem (VFS) ESP-IDF Programming Guide.” [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/vfs.html
[56] Texas Instruments, “TCA9548A Low-Voltage 8-Channel I2C Switch with Reset.” TCA9548A. [Online]. Available: https://www.ti.com/lit/ds/symlink/tca9548a.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-wwe&ts=1758024396288&ref_url=https%253A%252F%252Fwww.ti.com%252Fgeneral%252Fdocs%252Fsuppproductinfo.tsp%253FdistId%253D10%2526gotoUrl%253Dhttps%253A%252F%252Fwww.ti.com%252Flit%252Fgpn%252Ftca9548a
[57] Espressif, “Analog-to-Digital Converter (ADC) Calibration Driver ESP-IDF Programming Guide.” [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/adc_calibration.html

[58] Espressif, “ESP32-S3-WROOM-1 ESP32-S3-WROOM-1U Datasheet Version 1.6.” July 25, 2025. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf
[59] T. Instruments, “I2C Bus Pull-up Resistor Calculation.” [Online]. Available: https://www.ti.com/lit/pdf/slva689
[60] NXP Semiconductors, “I2C-bus Specification and User Manual,” Oct. [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf
[61] Texas Instruments, “I2C Stuck Bus: Prevention and Workarounds.” [Online]. Available: https://www.ti.com/lit/pdf/scpa069
[62] I. Project, “I2Cdetect detect I2C chips.” [Online]. Available: https://linux.die.net/man/8/i2cdetect
[63] B. Sensortec, “BME280 Combined Humidity and Pressure Sensor – Datasheet.” [Online]. Available: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf
[64] T. D. K. InvenSense, “MPU-6000/MPU-6050 Register Map and Descriptions,” Rev. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf
[65] SMBus Implementers, “System Management Bus (SMBus) Specification, Version 3.2.” [Online]. Available: https://www.smbus.org/specs/SMBus_3_2_20220112.pdf
[66] NXP Semiconductors, “I2C-bus specification and user manual.” UM10204. [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf
[67] J. Valdez and J. Becker, “Understanding the I2C Bus.” Texas Instruments, June 2015. [Online]. Available: https://www.ti.com/lit/an/slva704/slva704.pdf
[68] P. Ghosh, “IPC-2221 Standards in PCB Design,” Sierra Circuits. [Online]. Available: https://www.protoexpress.com/blog/ipc-2221-circuit-board-design/
[69] S. Fawley, “9 Benefits of Artificial Intelligence (AI) in 2025,” University of Cincinnati. [Online]. Available: https://online.uc.edu/blog/artificial-intelligence-ai-benefits/
[70] B. Schneier and N. Sanders, “AI Mistakes Are Way Weirder Than Human Mistakes,” IEEE Spectrum. [Online]. Available: https://spectrum.ieee.org/ai-mistakes-schneier
[71] “AI Demystified,” Stanford | University IT. [Online]. Available: https://uit.stanford.edu/service/techtraining/ai-demystified
[72] A. Bick, A. Blandin, and D. Deming, “The Impact of Generative AI on Work Productivity,” Federal Reserve Bank of ST. Louis. [Online]. Available: https://www.stlouisfed.org/on-the-economy/2025/feb/impact-generative-ai-work-productivity?utm_source=chatgpt.com
[73] T. Babina and A. Fedyk, “The effects of AI on firms and workers,” Brookings. [Online]. Available: https://www.brookings.edu/articles/the-effects-of-ai-on-firms-and-workers/
[74] A. Zewe, “Explained: Generative AI’s environmental impact,” MIT News. [Online]. Available: https://news.mit.edu/2025/explained-generative-ai-environmental-impact-0117
[75] M. Cunningham, “The AI revolution is likely to drive up your electricity bill. Here’s why. - CBS News,” CBS News. [Online]. Available: https://www.cbsnews.com/news/artificial-intelligene-ai-data-centers-electricity-bill-energy-costs/?utm_source=chatgpt.com
[76] Ibsen Photonics, “Spectrometer Design Guide,” Ibsen Photonics A/S, Farum, Denmark, White Paper, 2023. [Online]. Available: https://ibsen.com/wp-content/uploads/ebook/spectrometer-resources/Spectrometer-Design-Guide.pdf

Appendix D - Software Code

//main.c

```
#include <stdio.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_log.h"
#include "esp_wifi.h"
#include "nvs_flash.h"
#include "driver/gpio.h"
#include "opensense_types.h"
#include "config_service.h"
#include "discovery_service.h"
#include "sampler_task.h"
#include "logger_task.h"
#include "telemetry_task.h"
#include "web_server.h"

static const char *TAG = "MAIN";

// Global queue and driver handles
QueueHandle_t g_sample_queue = NULL;
driver_handle_t g_active_drivers[MAX_SENSORS];
uint8_t g_active_driver_count = 0;

static ws_client_manager_t g_ws_manager = {
    .ws_client_count = 0
};

/* Initialize WiFi in SoftAP mode - Chapter 3.3.5 */
static void init_wifi_softap(void) {
    esp_netif_init();
    esp_event_loop_create_default();
    esp_netif_create_default_wifi_ap();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    wifi_config_t wifi_config = {
        .ap = {
            .ssid = "OpenSense",
            .ssid_len = strlen("OpenSense"),
            .channel = 1,
            .password = "opensense2024",
            .max_connection = MAX_WS_CLIENTS,
            .authmode = WIFI_AUTH_WPA2_PSK
        },
    };

    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &wifi_config));
    ESP_ERROR_CHECK(esp_wifi_start());
```

```
    ESP_LOGI(TAG, "WiFi SoftAP started. SSID: OpenSense");
}

void app_main(void) {
    ESP_LOGI(TAG, "OpenSense v3 Starting...");

    // Initialize NVS for configuration storage
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    // Initialize configuration service (Chapter 7.1)
    ESP_ERROR_CHECK(config_init());
    ESP_LOGI(TAG, "Configuration loaded from NVS");

    // Initialize I²C and multiplexer hardware
    ESP_ERROR_CHECK(discovery_init_hardware());
    ESP_LOGI(TAG, "I2C and TCA9548A initialized");

    // Discover sensors on all channels (Chapter 3.3.17)
    g_active_driver_count = discovery_scan_all_channels(g_active_drivers, MAX_SENSORS);
    ESP_LOGI(TAG, "Discovered %d active sensors", g_active_driver_count);

    // Create sample queue (Chapter 7.8)
    g_sample_queue = xQueueCreate(SAMPLE_QUEUE_SIZE, sizeof(standardized_sample_t));
    if (g_sample_queue == NULL) {
        ESP_LOGE(TAG, "Failed to create sample queue");
        return;
    }

    // Initialize and mount SD card (Chapter 7.2)
    if (logger_init() != ESP_OK) {
        ESP_LOGE(TAG, "SD card initialization failed, continuing without logging");
    } else {
        ESP_LOGI(TAG, "SD card mounted and CSV ready");
    }

    // Initialize WiFi in SoftAP mode
    init_wifi_softap();

    // Create WebSocket client mutex
    g_ws_manager.ws_clients_mutex = xSemaphoreCreateMutex();

    // Start web server with WebSocket support (Chapter 7.1.1)
    ESP_ERROR_CHECK(web_server_init(&g_ws_manager));
    ESP_LOGI(TAG, "Web server started");

    // Start FreeRTOS tasks (Chapter 7.2)
    sampler_task_init();      // High priority - 10 Hz sampling
    logger_task_start();      // Medium priority - SD card writes
    telemetry_task_start(&g_ws_manager);  // Lower priority - WebSocket broadcasts

    ESP_LOGI(TAG, "All tasks started. System operational.");
```

```
    ESP_LOGI(TAG, "Connect to WiFi: OpenSense, Password: opensense2024");
    ESP_LOGI(TAG, "Access UI at: http://192.168.4.1");
}
```

//logger_task.c

```
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"
#include "esp_vfs_fat.h"
#include "sdmmc_cmd.h"
#include "driver/sdmmc_host.h"
#include "logger_task.h"
#include "opensense_types.h"

static const char *TAG = "LOGGER";

#define MOUNT_POINT "/sdcard"
#define CSV_FILE_PATH "/sdcard/opensense_data.csv"

static FILE *g_csv_file = NULL;
static ring_buffer_t g_ring_buffer;
static uint32_t g_sample_count = 0;
static bool g_sd_status = false;

/* Initialize ring buffer */
static void ring_buffer_init(void) {
    memset(&g_ring_buffer, 0, sizeof(ring_buffer_t));
    g_ring_buffer.mutex = xSemaphoreCreateMutex();
}

/* Flush ring buffer to SD card in sector-aligned blocks */
static void ring_buffer_flush(void) {
    if (g_csv_file == NULL || g_ring_buffer.head == 0) {
        return;
    }

    xSemaphoreTake(g_ring_buffer.mutex, portMAX_DELAY);

    // Write accumulated data
    size_t written = fwrite(g_ring_buffer.buffer, 1, g_ring_buffer.head, g_csv_file);
    if (written != g_ring_buffer.head) {
        ESP_LOGW(TAG, "Partial write to SD: %d/%d bytes", written, g_ring_buffer.head);
        g_sd_status = false;
    } else {
        g_sd_status = true;
    }

    // Reset buffer
    g_ring_buffer.head = 0;
    g_ring_buffer.needs_flush = false;

    xSemaphoreGive(g_ring_buffer.mutex);
}
```

```
/* Append sample to ring buffer */
static void ring_buffer_append(const char *data, size_t len) {
    xSemaphoreTake(g_ring_buffer.mutex, portMAX_DELAY);

    // Check if buffer has space
    if (g_ring_buffer.head + len < RING_BUFFER_SIZE) {
        memcpy(&g_ring_buffer.buffer[g_ring_buffer.head], data, len);
        g_ring_buffer.head += len;

        // Mark for flush at 512-byte boundary or 1 second worth
        if (g_ring_buffer.head >= 512) {
            g_ring_buffer.needs_flush = true;
        }
    }

    xSemaphoreGive(g_ring_buffer.mutex);
}

/* Logger task - dequeues and writes to SD */
static void logger_task(void *pvParameters) {
    ESP_LOGI(TAG, "Logger task started");

    TickType_t last_flush = xTaskGetTickCount();
    standardized_sample_t sample;
    char line_buffer[128];

    while (1) {
        // Dequeue sample with timeout
        if (xQueueReceive(g_sample_queue, &sample, pdMS_TO_TICKS(100)) == pdTRUE) {
            // Format CSV line: timestamp,channel,type,value,error_flags
            int len = snprintf(line_buffer, sizeof(line_buffer),
                "%llu,%u,%u,%.3f,%u\n",
                sample.timestamp_ms,
                sample.channel_id,
                sample.sensor_type,
                sample.value,
                sample.error_flags
            );

            if (len > 0 && len < sizeof(line_buffer)) {
                ring_buffer_append(line_buffer, len);
                g_sample_count++;
            }
        }

        // Flush every 1 second or when buffer is full
        TickType_t now = xTaskGetTickCount();
        if ((now - last_flush) >= pdMS_TO_TICKS(1000) || g_ring_buffer.needs_flush) {
            ring_buffer_flush();
            last_flush = now;
        }
    }
}

esp_err_t logger_init(void) {
```

```
    ESP_LOGI(TAG, "Initializing SD card");

    // Initialize ring buffer
    ring_buffer_init();

    // Mount SD card using SDMMC (Chapter 3.3.4)
    esp_vfs_fat_sdmmc_mount_config_t mount_config = {
        .format_if_mount_failed = false,
        .max_files = 5,
        .allocation_unit_size = 512  // Sector-aligned
    };

    sdmmc_host_t host = SDMMC_HOST_DEFAULT();
    sdmmc_slot_config_t slot_config = SDMMC_SLOT_CONFIG_DEFAULT();

    sdmmc_card_t *card;
    esp_err_t ret = esp_vfs_fat_sdmmc_mount(MOUNT_POINT, &host, &slot_config, &mount_config,
&card);

    if (ret != ESP_OK) {
        ESP_LOGE(TAG, "Failed to mount SD card: %s", esp_err_to_name(ret));
        return ret;
    }

    ESP_LOGI(TAG, "SD card mounted: %s", card->cid.name);

    // Open CSV file (append mode)
    g_csv_file = fopen(CSV_FILE_PATH, "a");
    if (g_csv_file == NULL) {
        ESP_LOGE(TAG, "Failed to open CSV file");
        return ESP_FAIL;
    }

    // Write CSV header if new file
    struct stat st;
    if (stat(CSV_FILE_PATH, &st) != 0 || st.st_size == 0) {
        fprintf(g_csv_file, "timestamp_ms,channel_id,sensor_type,value,error_flags\n");
        fflush(g_csv_file);
    }

    g_sd_status = true;
    return ESP_OK;
}

void logger_task_start(void) {
    xTaskCreate(logger_task, "logger", 4096, NULL, 15, NULL);
}

bool logger_get_sd_status(void) {
    return g_sd_status;
}

uint32_t logger_get_sample_count(void) {
    return g_sample_count;
}
```

//web_server.c

```
#include <string.h>
#include "esp_http_server.h"
#include "esp_log.h"
#include "web_server.h"
#include "config_service.h"
#include "sampler_task.h"
#include "opensense_types.h"
#include "cJSON.h"

static const char *TAG = "WEB_SERVER";

static httpd_handle_t g_server = NULL;
static ws_client_manager_t *g_ws_manager = NULL;

extern const uint8_t ucf_logo_png_start[] asm("_binary_UCF_Logo_png_start");
extern const uint8_t ucf_logo_png_end[] asm("_binary_UCF_Logo_png_end");
extern const uint8_t draco_lab_png_start[] asm("_binary_Draco_Lab_png_start");
extern const uint8_t draco_lab_png_end[] asm("_binary_Draco_Lab_png_end");

extern const uint8_t index_html_start[] asm("_binary_index_html_start");
extern const uint8_t index_html_end[] asm("_binary_index_html_end");
extern const uint8_t app_js_start[] asm("_binary_app_js_start");
extern const uint8_t app_js_end[] asm("_binary_app_js_end");
extern const uint8_t style_css_start[] asm("_binary_style_css_start");
extern const uint8_t style_css_end[] asm("_binary_style_css_end");


static esp_err_t ws_handler(httpd_req_t *req) {
    int fd = httpd_req_to_sockfd(req);

    // initial connection
    if (req->method == HTTP_GET) {
        ESP_LOGI(TAG, "WebSocket handshake from fd %d", fd);

        // add client
        if (g_ws_manager != NULL) {
            if (xSemaphoreTake(g_ws_manager->ws_clients_mutex, pdMS_TO_TICKS(1000)) == pdTRUE) {
                // Check if client already exists
                bool already_exists = false;
                for (int i = 0; i < g_ws_manager->ws_client_count; i++) {
                    if (g_ws_manager->ws_clients[i] == fd) {
                        already_exists = true;
                        break;
                    }
                }

                // Add new client
                if (!already_exists && g_ws_manager->ws_client_count < MAX_WS_CLIENTS) {
                    g_ws_manager->ws_clients[g_ws_manager->ws_client_count++] = fd;
                    ESP_LOGI(TAG, "WebSocket client connected! Total clients: %d",
g_ws_manager->ws_client_count);
                } else if (already_exists) {
                    ESP_LOGW(TAG, "Client fd %d already registered", fd);
                } else {
                    ESP_LOGW(TAG, "Max WebSocket clients reached (%d)", MAX_WS_CLIENTS);
```

```
            }

            xSemaphoreGive(g_ws_manager->ws_clients_mutex);
        } else {
            ESP_LOGE(TAG, "Failed to acquire mutex for client registration");
        }
    }

    return ESP_OK;
}

// Handle incoming WebSocket frames
httpd_ws_frame_t ws_pkt;
memset(&ws_pkt, 0, sizeof(httpd_ws_frame_t));
ws_pkt.type = HTTPD_WS_TYPE_TEXT;

// Get frame length first
esp_err_t ret = httpd_ws_recv_frame(req, &ws_pkt, 0);
if (ret != ESP_OK) {
    ESP_LOGE(TAG, "httpd_ws_recv_frame failed to get frame len: %s", esp_err_to_name(ret));
    return ret;
}

// If there's payload, allocate buffer and receive it
if (ws_pkt.len > 0) {
    uint8_t *buf = malloc(ws_pkt.len + 1);
    if (buf == NULL) {
        ESP_LOGE(TAG, "Failed to allocate memory for WS frame");
        return ESP_ERR_NO_MEM;
    }

    ws_pkt.payload = buf;
    ret = httpd_ws_recv_frame(req, &ws_pkt, ws_pkt.len);
    if (ret != ESP_OK) {
        ESP_LOGE(TAG, "httpd_ws_recv_frame failed: %s", esp_err_to_name(ret));
        free(buf);
        return ret;
    }

    buf[ws_pkt.len] = '\0';
    ESP_LOGI(TAG, "Received WS message: %s", buf);

    free(buf);
}

return ESP_OK;
}

/* Serve index.html */
static esp_err_t index_handler(httpd_req_t *req) {
    httpd_resp_set_type(req, "text/html");
    httpd_resp_sendstr(req, (const char*)index_html_start);
    return ESP_OK;
}

/* Serve app.js */
```

```
static esp_err_t app_js_handler(httpd_req_t *req) {
    httpd_resp_set_type(req, "application/javascript");
    httpd_resp_sendstr(req, (const char*)app_js_start);
    return ESP_OK;
}

/* Serve style.css */
static esp_err_t style_css_handler(httpd_req_t *req) {
    httpd_resp_set_type(req, "text/css");
    httpd_resp_sendstr(req, (const char*)style_css_start);
    return ESP_OK;
}

/* Serve UCF Logo */
static esp_err_t ucf_logo_handler(httpd_req_t *req) {
    httpd_resp_set_type(req, "image/png");
    httpd_resp_send(req, (const char*)ucf_logo_png_start, ucf_logo_png_end - ucf_logo_png_start);
    return ESP_OK;
}

/* Serve Draco Lab Logo */
static esp_err_t draco_lab_logo_handler(httpd_req_t *req) {
    httpd_resp_set_type(req, "image/png");
    httpd_resp_send(req, (const char*)draco_lab_png_start, draco_lab_png_end - draco_lab_png_start);
    return ESP_OK;
}

/* GET /config - Return current configuration */
static esp_err_t config_get_handler(httpd_req_t *req) {
    const config_blob_t *config = config_get();

    // Build JSON response
    cJSON *root = cJSON_CreateObject();
    cJSON_AddNumberToObject(root, "version", config->version);
    cJSON_AddNumberToObject(root, "sample_rate_hz", config->sample_rate_hz);

    // Add channel map array
    cJSON *channels = cJSON_CreateArray();
    for (int i = 0; i < MAX_MUX_CHANNELS; i++) {
        cJSON *ch = cJSON_CreateObject();
        cJSON_AddNumberToObject(ch, "mux_channel", config->channel_map[i].mux_channel);
        cJSON_AddNumberToObject(ch, "i2c_addr", config->channel_map[i].i2c_addr);
        cJSON_AddStringToObject(ch, "driver_name", config->channel_map[i].driver_name);
        cJSON_AddBoolToObject(ch, "auto_detected", config->channel_map[i].auto_detected);
        cJSON_AddItemToArray(channels, ch);
    }
    cJSON_AddItemToObject(root, "channels", channels);

    char *json_str = cJSON_Print(root);
    httpd_resp_set_type(req, "application/json");
    httpd_resp_sendstr(req, json_str);

    free(json_str);
    cJSON_Delete(root);
    return ESP_OK;
}
```

```
/* POST /config - Update configuration */
static esp_err_t config_post_handler(httpd_req_t *req) {
    char buf[1024];
    int ret = httpd_req_recv(req, buf, sizeof(buf) - 1);
    if (ret <= 0) {
        httpd_resp_send_500(req);
        return ESP_FAIL;
    }
    buf[ret] = '\0';

    // Parse JSON and update config
    cJSON *root = cJSON_Parse(buf);
    if (root == NULL) {
        httpd_resp_send_err(req, HTTPD_400_BAD_REQUEST, "Invalid JSON");
        return ESP_FAIL;
    }

    config_blob_t new_config;
    memcpy(&new_config, config_get(), sizeof(config_blob_t));

    // Update fields from JSON
    cJSON *sample_rate = cJSON_GetObjectItem(root, "sample_rate_hz");
    if (sample_rate) {
        new_config.sample_rate_hz = sample_rate->valueint;
    }

    // Atomically update config
    esp_err_t err = config_update(&new_config);
    cJSON_Delete(root);

    if (err == ESP_OK) {
        httpd_resp_sendstr(req, "{\"status\":\"ok\"}");
    } else {
        httpd_resp_send_500(req);
    }

    return err;
}

/* POST /rescan - Rescan specific channel */
static esp_err_t rescan_handler(httpd_req_t *req) {
    char buf[32];
    int ret = httpd_req_recv(req, buf, sizeof(buf) - 1);
    if (ret <= 0) {
        httpd_resp_send_500(req);
        return ESP_FAIL;
    }
    buf[ret] = '\0';

    cJSON *root = cJSON_Parse(buf);
    cJSON *channel = cJSON_GetObjectItem(root, "channel");

    if (channel) {
        sampler_rescan_channel(channel->valueint);
        httpd_resp_sendstr(req, "{\"status\":\"ok\"}");
```

```
    } else {
        httpd_resp_send_err(req, HTTPD_400_BAD_REQUEST, "Invalid request");
    }

    cJSON_Delete(root);
    return ESP_OK;
}

esp_err_t web_server_init(ws_client_manager_t *ws_manager) {
    g_ws_manager = ws_manager;

    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.max_open_sockets = 4;
    config.max_uri_handlers = 12;
    config.lru_purge_enable = true;

    ESP_LOGI(TAG, "Starting HTTP server on port %d", config.server_port);

    if (httpd_start(&g_server, &config) != ESP_OK) {
        ESP_LOGE(TAG, "Failed to start server");
        return ESP_FAIL;
    }

    // Register URI handlers
    httpd_uri_t index_uri = {.uri = "/", .method = HTTP_GET, .handler = index_handler};
    httpd_uri_t app_js_uri = {.uri = "/app.js", .method = HTTP_GET, .handler = app_js_handler};
    httpd_uri_t style_css_uri = {.uri = "/style.css", .method = HTTP_GET, .handler = style_css_handler};
    httpd_uri_t config_get_uri = {.uri = "/config", .method = HTTP_GET, .handler = config_get_handler};
    httpd_uri_t config_post_uri = {.uri = "/config", .method = HTTP_POST, .handler = config_post_handler};
    httpd_uri_t rescan_uri = {.uri = "/rescan", .method = HTTP_POST, .handler = rescan_handler};
    httpd_uri_t ucf_logo_uri = {.uri = "/images/UCF_Logo.png", .method = HTTP_GET, .handler =
ucf_logo_handler};
    httpd_uri_t draco_lab_uri = {.uri = "/images/Draco_Lab.png", .method = HTTP_GET, .handler =
draco_lab_logo_handler};

    // WebSocket endpoint
    httpd_uri_t ws_uri = {
        .uri = "/ws",
        .method = HTTP_GET,
        .handler = ws_handler,
        .is_websocket = true
    };

    httpd_register_uri_handler(g_server, &index_uri);
    httpd_register_uri_handler(g_server, &app_js_uri);
    httpd_register_uri_handler(g_server, &style_css_uri);
    httpd_register_uri_handler(g_server, &config_get_uri);
    httpd_register_uri_handler(g_server, &config_post_uri);
    httpd_register_uri_handler(g_server, &rescan_uri);
    httpd_register_uri_handler(g_server, &ws_uri);
    httpd_register_uri_handler(g_server, &ucf_logo_uri);
    httpd_register_uri_handler(g_server, &draco_lab_uri);

    ESP_LOGI(TAG, "Web server started successfully");

    return ESP_OK;
```

```
}

void web_server_stop(void) {
    if (g_server) {
        httpd_stop(g_server);
        g_server = NULL;
    }
}

httpd_handle_t web_server_get_handle(void) {
    return g_server;
}
```

//telemetry_task.c

```
#include <stdio.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"
#include "esp_http_server.h"
#include "telemetry_task.h"
#include "web_server.h"
#include "logger_task.h"
#include "opensense_types.h"
#include "esp_timer.h"
#include <inttypes.h>

static const char *TAG = "TELEMETRY";

#define TELEMETRY_RATE_HZ 10
#define TELEMETRY_PERIOD_MS (1000 / TELEMETRY_RATE_HZ)

static ws_client_manager_t *g_ws_manager = NULL;

// Latest values storage - updated by copy from queue
static struct {
    float latest_value;
    uint64_t latest_timestamp;
    bool has_data;
} g_channel_data[MAX_SENSORS];

// Global latest values that sampler updates directly
// Declare as extern so sampler can write to them
float g_telemetry_values[MAX_SENSORS] = {0};
bool g_telemetry_has_data[MAX_SENSORS] = {false};

/* Build compact JSON telemetry frame */
static int build_telemetry_frame(char *buffer, size_t buffer_size) {
    int len = snprintf(buffer, buffer_size, "{\"t\":%" PRIu64 ",\"c\":[",
                       (uint64_t)(esp_timer_get_time() / 1000));

    // Add channel values - use the global values updated by sampler
    for (int i = 0; i < 5; i++) {
        if (g_telemetry_has_data[i]) {
            len += snprintf(buffer + len, buffer_size - len, "%.2f", g_telemetry_values[i]);
        } else {
            len += snprintf(buffer + len, buffer_size - len, "null");
```

```
        }

        if (i < 4) {
            len += snprintf(buffer + len, buffer_size - len, ",");
        }
    }

    // Add status info
    len += snprintf(buffer + len, buffer_size - len, "],\"sd\":%s,\"cnt\":%" PRIu32 "}",
                    logger_get_sd_status() ? "true" : "false",
                    logger_get_sample_count());

    return len;
}

/* Broadcast frame to all connected WebSocket clients */
static void broadcast_frame(const char *frame, size_t len) {
    if (g_ws_manager == NULL || g_ws_manager->ws_client_count == 0) {
        return;
    }

    httpd_handle_t server = web_server_get_handle();
    if (server == NULL) {
        return;
    }

    if (xSemaphoreTake(g_ws_manager->ws_clients_mutex, pdMS_TO_TICKS(100)) != pdTRUE) {
        return;
    }

    httpd_ws_frame_t ws_pkt = {
        .type = HTTPD_WS_TYPE_TEXT,
        .payload = (uint8_t *)frame,
        .len = len
    };

    int failed_clients[MAX_WS_CLIENTS];
    int failed_count = 0;

    for (int i = 0; i < g_ws_manager->ws_client_count; i++) {
        int fd = g_ws_manager->ws_clients[i];
        esp_err_t ret = httpd_ws_send_frame_async(server, fd, &ws_pkt);
        if (ret != ESP_OK) {
            failed_clients[failed_count++] = i;
        }
    }

    // Remove failed clients
    for (int i = failed_count - 1; i >= 0; i--) {
        int idx = failed_clients[i];
        for (int j = idx; j < g_ws_manager->ws_client_count - 1; j++) {
            g_ws_manager->ws_clients[j] = g_ws_manager->ws_clients[j + 1];
        }
        g_ws_manager->ws_client_count--;
    }
```

```
    xSemaphoreGive(g_ws_manager->ws_clients_mutex);
}

/* Main telemetry task */
static void telemetry_task(void *pvParameters) {
    ESP_LOGI(TAG, "Telemetry task started");

    char telemetry_buffer[TELEMETRY_BUFFER_SIZE];
    uint32_t frame_count = 0;

    while (1) {
        vTaskDelay(pdMS_TO_TICKS(TELEMETRY_PERIOD_MS));

        // Build and broadcast frame
        int len = build_telemetry_frame(telemetry_buffer, sizeof(telemetry_buffer));

        if (len > 0 && len < sizeof(telemetry_buffer)) {
            broadcast_frame(telemetry_buffer, len);
            frame_count++;

            // Debug log every 5 seconds
            if (frame_count % 50 == 0) {
                ESP_LOGI(TAG, "Sent frame #%lu: %s", frame_count, telemetry_buffer);
            }
        }
    }
}

void telemetry_task_start(ws_client_manager_t *ws_manager) {
    g_ws_manager = ws_manager;

    // Initialize telemetry values
    for (int i = 0; i < MAX_SENSORS; i++) {
        g_telemetry_values[i] = 0;
        g_telemetry_has_data[i] = false;
    }

    xTaskCreate(telemetry_task, "telemetry", 4096, NULL, 10, NULL);
}

void telemetry_send_now(void) {
    char buffer[TELEMETRY_BUFFER_SIZE];
    int len = build_telemetry_frame(buffer, sizeof(buffer));
    if (len > 0) {
        broadcast_frame(buffer, len);
    }
}

/* Called by sampler to update telemetry values */
void telemetry_update_value(uint8_t channel, float value) {
    if (channel < MAX_SENSORS) {
        g_telemetry_values[channel] = value;
        g_telemetry_has_data[channel] = true;
    }
}
```

// sampler_task.c

```
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"
#include "esp_timer.h"
#include "sampler_task.h"
#include "telemetry_task.h"
#include "opensense_types.h"

static const char *TAG = "SAMPLER";

#define DIRECT_MODE 1

static uint8_t g_channel_error_count[MAX_MUX_CHANNELS] = {0};
static bool g_channel_faulted[MAX_MUX_CHANNELS] = {false};

static void sampler_task(void *pvParameters) {
    ESP_LOGI(TAG, "========================================");
    ESP_LOGI(TAG, "SAMPLER TASK STARTED (DIRECT MODE)");
    ESP_LOGI(TAG, "g_active_driver_count = %d", g_active_driver_count);
    ESP_LOGI(TAG, "========================================");

    if (g_active_driver_count == 0) {
        ESP_LOGE(TAG, "NO ACTIVE DRIVERS!");
        while(1) { vTaskDelay(pdMS_TO_TICKS(10000)); }
    }

    for (int i = 0; i < g_active_driver_count; i++) {
        driver_handle_t *d = &g_active_drivers[i];
        ESP_LOGI(TAG, "Driver[%d]: ch=%d addr=0x%02X", i, d->mux_channel, d->i2c_addr);
    }

    uint32_t sample_count = 0;

    while (1) {
        vTaskDelay(pdMS_TO_TICKS(SAMPLE_PERIOD_MS));

        for (int i = 0; i < g_active_driver_count; i++) {
            driver_handle_t *driver = &g_active_drivers[i];

            if (g_channel_faulted[driver->mux_channel]) {
                continue;
            }

            if (driver->driver == NULL || driver->driver->read == NULL) {
                continue;
            }

            // Prepare sample
            standardized_sample_t sample = {
                .timestamp_ms = esp_timer_get_time() / 1000,
                .channel_id = driver->mux_channel,
                .sensor_type = driver->sensor_type,
                .error_flags = ERROR_FLAG_NONE,
                .value = 0.0f
            };
```

```
            // Read sensor
            int result = driver->driver->read(driver->driver_state, &sample);

            if (result != 0) {
                g_channel_error_count[driver->mux_channel]++;
                sample.error_flags = ERROR_FLAG_READ_FAIL;

                if (g_channel_error_count[driver->mux_channel] >= 10) {
                    g_channel_faulted[driver->mux_channel] = true;
                    ESP_LOGE(TAG, "Channel %d FAULTED", driver->mux_channel);
                }
            } else {
                g_channel_error_count[driver->mux_channel] = 0;
                sample_count++;

                // UPDATE TELEMETRY DIRECTLY!
                telemetry_update_value(driver->mux_channel, sample.value);

                // Log every second
                if (sample_count % 10 == 0) {
                    ESP_LOGI(TAG, "Sample #%lu: ch=%d val=%.2f",
                             sample_count, driver->mux_channel, sample.value);
                }
            }

            // Send to queue for logger
            if (g_sample_queue != NULL) {
                xQueueSend(g_sample_queue, &sample, 0);
            }
        }
    }
}

void sampler_task_init(void) {
    ESP_LOGI(TAG, "Creating sampler task...");
    xTaskCreate(sampler_task, "sampler", 4096, NULL, 20, NULL);
}

void sampler_rescan_channel(uint8_t mux_channel) {
    g_channel_faulted[mux_channel] = false;
    g_channel_error_count[mux_channel] = 0;
}

bool sampler_is_channel_faulted(uint8_t mux_channel) {
    return g_channel_faulted[mux_channel];
}
```

//discovery_service.c

```
#include <string.h>
#include "driver/i2c.h"
#include "esp_log.h"
#include "discovery_service.h"
#include "driver_registry.h"
#include "opensense_types.h"

static const char *TAG = "DISCOVERY";
```

```
// Protoype declarations
// TOGGLE THIS TO SWITCH MODES:
// 1 = Direct mode esp to sensor
// 0 = MUX mode esp to TCA9548A to sensor
#define DIRECT_MODE 1

// I²C Configuration (Chapter 3.3.3)
#define I2C_MASTER_NUM I2C_NUM_0
#define I2C_MASTER_SDA_IO 21
#define I2C_MASTER_SCL_IO 22

#if DIRECT_MODE
   #define I2C_MASTER_FREQ_HZ 100000  // 100 kHz for direct mode (more reliable)
#else
   #define I2C_MASTER_FREQ_HZ 400000  // 400 kHz Fast Mode for MUX
#endif

#define TCA9548A_ADDR 0x70

static bool g_i2c_initialized = false;

/* I2C Scanner for debugging - helps identify connected devices */
static void i2c_scanner(void) {
   ESP_LOGI(TAG, "=== I2C Bus Scanner ===");
   int found = 0;

   for (uint8_t addr = 1; addr < 127; addr++) {
      i2c_cmd_handle_t cmd = i2c_cmd_link_create();
      i2c_master_start(cmd);
      i2c_master_write_byte(cmd, (addr << 1) | I2C_MASTER_WRITE, true);
      i2c_master_stop(cmd);

      esp_err_t ret = i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, pdMS_TO_TICKS(50));
      i2c_cmd_link_delete(cmd);

      if (ret == ESP_OK) {
         ESP_LOGI(TAG, "  Found device at address: 0x%02X", addr);
         if (addr == 0x76 || addr == 0x77) {
            ESP_LOGI(TAG, "    ^ This is BME280!");
         } else if (addr == 0x70) {
            ESP_LOGI(TAG, "    ^ This is TCA9548A MUX!");
         }
         found++;
      }
   }

   if (found == 0) {
      ESP_LOGW(TAG, "  No I2C devices found!");
      ESP_LOGW(TAG, "  Check: 1) Wiring  2) Power  3) Pull-up resistors");
   } else {
      ESP_LOGI(TAG, "  Scan complete: %d device(s) found", found);
   }
   ESP_LOGI(TAG, "========================");
}
```

```
/* Check if device exists at address */
static bool probe_address(uint8_t addr) {
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (addr << 1) | I2C_MASTER_WRITE, true);
    i2c_master_stop(cmd);

    esp_err_t ret = i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, pdMS_TO_TICKS(100));
    i2c_cmd_link_delete(cmd);

    return (ret == ESP_OK);
}

#if !DIRECT_MODE
/* Select TCA9548A multiplexer channel (MUX MODE ONLY) */
static esp_err_t mux_select_channel(uint8_t channel) {
    if (channel >= MAX_MUX_CHANNELS) {
        return ESP_ERR_INVALID_ARG;
    }

    uint8_t channel_byte = (1 << channel);

    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (TCA9548A_ADDR << 1) | I2C_MASTER_WRITE, true);
    i2c_master_write_byte(cmd, channel_byte, true);
    i2c_master_stop(cmd);

    esp_err_t ret = i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, pdMS_TO_TICKS(1000));
    i2c_cmd_link_delete(cmd);

    return ret;
}

/* Deselect all TCA9548A channels (MUX MODE ONLY) */
static esp_err_t mux_deselect_all(void) {
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (TCA9548A_ADDR << 1) | I2C_MASTER_WRITE, true);
    i2c_master_write_byte(cmd, 0x00, true);  // Disable all channels
    i2c_master_stop(cmd);

    esp_err_t ret = i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, pdMS_TO_TICKS(1000));
    i2c_cmd_link_delete(cmd);

    return ret;
}
#endif

esp_err_t discovery_init_hardware(void) {
    if (g_i2c_initialized) {
        return ESP_OK;
    }

#if DIRECT_MODE
    ESP_LOGI(TAG, "=======================================");
```

```
    ESP_LOGI(TAG, "*** DIRECT MODE ENABLED (No MUX) ***");
    ESP_LOGI(TAG, "Wire BME280 directly to ESP32:");
    ESP_LOGI(TAG, "  BME280 VCC → ESP32 3.3V");
    ESP_LOGI(TAG, "  BME280 GND → ESP32 GND");
    ESP_LOGI(TAG, "  BME280 SDA → ESP32 GPIO 21");
    ESP_LOGI(TAG, "  BME280 SCL → ESP32 GPIO 22");
    ESP_LOGI(TAG, "========================================");
#else
    ESP_LOGI(TAG, "*** MUX MODE ENABLED ***");
#endif

    // Configure I²C master
    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ
    };

    esp_err_t err = i2c_param_config(I2C_MASTER_NUM, &conf);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "I2C param config failed");
        return err;
    }

    err = i2c_driver_install(I2C_MASTER_NUM, conf.mode, 0, 0, 0);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "I2C driver install failed");
        return err;
    }

    g_i2c_initialized = true;
    ESP_LOGI(TAG, "I2C initialized at %d kHz", I2C_MASTER_FREQ_HZ / 1000);

    // Run I2C scanner for debugging
    i2c_scanner();

#if !DIRECT_MODE
    // Initialize TCA9548A - disable all channels initially (MUX MODE ONLY)
    err = mux_deselect_all();
    if (err != ESP_OK) {
        ESP_LOGW(TAG, "TCA9548A init failed, multiplexer may not be present");
    } else {
        ESP_LOGI(TAG, "TCA9548A multiplexer initialized successfully");
    }
#endif

    return ESP_OK;
}

bool discovery_scan_channel(uint8_t mux_channel, driver_handle_t *handle) {
    if (handle == NULL) {
        ESP_LOGE(TAG, "Handle is NULL!");
        return false;
```

```
    }

#if DIRECT_MODE
    // DIRECT MODE: Only scan once (ignore mux_channel after 0)
    if (mux_channel != 0) {
        return false;
    }

    ESP_LOGI(TAG, "Scanning for sensors (Direct Mode)...");
#else
    // MUX MODE: Select the MUX channel first
    if (mux_select_channel(mux_channel) != ESP_OK) {
        ESP_LOGW(TAG, "Failed to select MUX channel %d", mux_channel);
        return false;
    }
#endif

    // Get all registered drivers
    int driver_count = 0;
    const sensor_driver_t **drivers = driver_registry_get_all(&driver_count);

    // NULL CHECK: Make sure we have drivers
    if (drivers == NULL || driver_count == 0) {
        ESP_LOGE(TAG, "No drivers registered! driver_count=%d", driver_count);
        return false;
    }

    ESP_LOGI(TAG, "Found %d registered drivers", driver_count);

    int best_confidence = 0;
    const sensor_driver_t *best_driver = NULL;
    uint8_t best_addr = 0;

    // Try each driver's probe function
    for (int i = 0; i < driver_count; i++) {
        const sensor_driver_t *driver = drivers[i];

        // NULL CHECK: Skip NULL drivers
        if (driver == NULL) {
            ESP_LOGW(TAG, "Driver %d is NULL, skipping", i);
            continue;
        }

        ESP_LOGI(TAG, "Trying driver: %s (has %d addresses)",
                 driver->name ? driver->name : "unknown", driver->num_addrs);

        // Try each possible address for this driver
        for (int j = 0; j < driver->num_addrs; j++) {
            uint8_t addr = driver->possible_addrs[j];

            ESP_LOGI(TAG, "  Checking address 0x%02X...", addr);

#if DIRECT_MODE
            // In direct mode, first check if device exists at address
            if (!probe_address(addr)) {
                ESP_LOGI(TAG, "    No device at 0x%02X", addr);
```

```
            continue;
        }
        ESP_LOGI(TAG, "    Device found at 0x%02X!", addr);
#endif

        // NULL CHECK: Make sure probe function exists
        if (driver->probe == NULL) {
            ESP_LOGW(TAG, "    Driver has no probe function!");
            continue;
        }

        uint8_t probe_data[8] = {0};
        int confidence = driver->probe(mux_channel, addr, probe_data);

        ESP_LOGI(TAG, "    %s probe confidence: %d", driver->name, confidence);

        if (confidence > best_confidence) {
            best_confidence = confidence;
            best_driver = driver;
            best_addr = addr;
        }
    }
  }

  // Found a sensor
  if (best_driver != NULL && best_confidence > 0) {
    ESP_LOGI(TAG, "*** Found %s at 0x%02X (confidence %d) ***",
             best_driver->name, best_addr, best_confidence);

    // NULL CHECK: Make sure init function exists
    if (best_driver->init == NULL) {
        ESP_LOGE(TAG, "Driver has no init function!");
        return false;
    }

    // Initialize driver
    void *driver_state = best_driver->init(mux_channel, best_addr);
    if (driver_state != NULL) {
        handle->i2c_addr = best_addr;
        handle->mux_channel = mux_channel;
        handle->sensor_type = SENSOR_TYPE_BME280;  // Set based on driver
        handle->driver_state = driver_state;
        handle->driver = best_driver;

        ESP_LOGI(TAG, "*** %s initialized successfully! ***", best_driver->name);
        return true;
    } else {
        ESP_LOGE(TAG, "Failed to initialize %s driver", best_driver->name);
    }
  } else {
    ESP_LOGW(TAG, "No sensor found (best_confidence=%d)", best_confidence);
  }

  return false;
}
```

```
int discovery_scan_all_channels(driver_handle_t *handles, int max_handles) {
    int found_count = 0;

    if (handles == NULL) {
        ESP_LOGE(TAG, "Handles array is NULL!");
        return 0;
    }

#if DIRECT_MODE
    ESP_LOGI(TAG, "=== Direct Mode Sensor Discovery ===");

    // In direct mode, just scan once
    if (discovery_scan_channel(0, &handles[found_count])) {
        found_count++;
    }

    if (found_count == 0) {
        ESP_LOGW(TAG, "No sensors found in Direct Mode!");
        ESP_LOGW(TAG, "Check wiring:");
        ESP_LOGW(TAG, "  BME280 VCC → 3.3V");
        ESP_LOGW(TAG, "  BME280 GND → GND");
        ESP_LOGW(TAG, "  BME280 SDA → GPIO 21");
        ESP_LOGW(TAG, "  BME280 SCL → GPIO 22");
    }
#else
    ESP_LOGI(TAG, "Scanning all %d MUX channels...", MAX_MUX_CHANNELS);

    for (uint8_t ch = 0; ch < MAX_MUX_CHANNELS && found_count < max_handles; ch++) {
        if (discovery_scan_channel(ch, &handles[found_count])) {
            found_count++;
        }
    }

    // Deselect all channels after scan
    mux_deselect_all();
#endif

    ESP_LOGI(TAG, "Discovery complete: %d sensors found", found_count);
    return found_count;
}

esp_err_t discovery_bus_recovery(uint8_t mux_channel) {
    ESP_LOGI(TAG, "Performing bus recovery");

#if !DIRECT_MODE
    // Select channel (MUX MODE ONLY)
    if (mux_select_channel(mux_channel) != ESP_OK) {
        return ESP_FAIL;
    }
#endif

    // Send 9 clock pulses for bus recovery
    gpio_set_direction(I2C_MASTER_SCL_IO, GPIO_MODE_OUTPUT);
    gpio_set_direction(I2C_MASTER_SDA_IO, GPIO_MODE_OUTPUT);

    for (int i = 0; i < 9; i++) {
```

```
        gpio_set_level(I2C_MASTER_SCL_IO, 1);
        vTaskDelay(pdMS_TO_TICKS(1));
        gpio_set_level(I2C_MASTER_SCL_IO, 0);
        vTaskDelay(pdMS_TO_TICKS(1));
    }

    // Send STOP condition
    gpio_set_level(I2C_MASTER_SDA_IO, 0);
    vTaskDelay(pdMS_TO_TICKS(1));
    gpio_set_level(I2C_MASTER_SCL_IO, 1);
    vTaskDelay(pdMS_TO_TICKS(1));
    gpio_set_level(I2C_MASTER_SDA_IO, 1);

    ESP_LOGI(TAG, "Bus recovery complete");
    return ESP_OK;
}
```

// config_service.c

```
#include <string.h>
#include "nvs_flash.h"
#include "nvs.h"
#include "esp_log.h"
#include "config_service.h"
#include "opensense_types.h"
#include "cJSON.h"

static const char *TAG = "CONFIG";
static const char *NVS_NAMESPACE = "opensense";
static const char *CONFIG_KEY = "config_blob";

static config_blob_t g_current_config;
static SemaphoreHandle_t g_config_mutex = NULL;

/* Load configuration from NVS */
static esp_err_t config_load_from_nvs(void) {
    nvs_handle_t nvs_handle;
    esp_err_t err = nvs_open(NVS_NAMESPACE, NVS_READONLY, &nvs_handle);

    if (err != ESP_OK) {
        ESP_LOGW(TAG, "NVS namespace not found, using defaults");
        return ESP_ERR_NOT_FOUND;
    }

    size_t required_size = sizeof(config_blob_t);
    err = nvs_get_blob(nvs_handle, CONFIG_KEY, &g_current_config, &required_size);

    nvs_close(nvs_handle);

    if (err != ESP_OK) {
        ESP_LOGW(TAG, "Config blob not found in NVS");
        return err;
    }

    ESP_LOGI(TAG, "Config loaded, version: %u", g_current_config.version);
    return ESP_OK;
}
```

```
/* Save configuration to NVS atomically */
static esp_err_t config_save_to_nvs(const config_blob_t *config) {
    nvs_handle_t nvs_handle;
    esp_err_t err = nvs_open(NVS_NAMESPACE, NVS_READWRITE, &nvs_handle);

    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to open NVS for write");
        return err;
    }

    // Atomic write with version increment
    err = nvs_set_blob(nvs_handle, CONFIG_KEY, config, sizeof(config_blob_t));
    if (err == ESP_OK) {
        err = nvs_commit(nvs_handle);
    }

    nvs_close(nvs_handle);

    if (err == ESP_OK) {
        ESP_LOGI(TAG, "Config saved, version: %u", config->version);
    } else {
        ESP_LOGE(TAG, "Failed to save config: %s", esp_err_to_name(err));
    }

    return err;
}

/* Initialize with default configuration */
static void config_init_defaults(void) {
    memset(&g_current_config, 0, sizeof(config_blob_t));
    g_current_config.version = 1;
    g_current_config.sample_rate_hz = SAMPLE_RATE_HZ;

    // Initialize channel map with defaults
    for (int i = 0; i < MAX_MUX_CHANNELS; i++) {
        g_current_config.channel_map[i].mux_channel = i;
        g_current_config.channel_map[i].i2c_addr = 0x00;
        g_current_config.channel_map[i].auto_detected = false;
        strncpy(g_current_config.channel_map[i].driver_name, "none", 32);
    }

    // Initialize sensor parameters
    for (int i = 0; i < MAX_SENSORS; i++) {
        g_current_config.params[i].calibration_coeff = 1.0f;
        g_current_config.params[i].adc_atten = 0;
    }

    ESP_LOGI(TAG, "Initialized with default configuration");
}

esp_err_t config_init(void) {
    g_config_mutex = xSemaphoreCreateMutex();
    if (g_config_mutex == NULL) {
        return ESP_FAIL;
    }
```

```
    // Try to load from NVS, use defaults if not found
    if (config_load_from_nvs() != ESP_OK) {
        config_init_defaults();
        config_save_to_nvs(&g_current_config);
    }

    return ESP_OK;
}

const config_blob_t* config_get(void) {
    return &g_current_config;
}

esp_err_t config_update(const config_blob_t *new_config) {
    if (new_config == NULL) {
        return ESP_ERR_INVALID_ARG;
    }

    xSemaphoreTake(g_config_mutex, portMAX_DELAY);

    // Increment version for atomic rollback
    config_blob_t updated_config = *new_config;
    updated_config.version = g_current_config.version + 1;

    // Save to NVS first
    esp_err_t err = config_save_to_nvs(&updated_config);

    if (err == ESP_OK) {
        // Update in-memory copy only if save succeeded
        memcpy(&g_current_config, &updated_config, sizeof(config_blob_t));
        ESP_LOGI(TAG, "Configuration updated to version %u", g_current_config.version);
    }

    xSemaphoreGive(g_config_mutex);
    return err;
}

esp_err_t config_factory_reset(void) {
    nvs_handle_t nvs_handle;
    esp_err_t err = nvs_open(NVS_NAMESPACE, NVS_READWRITE, &nvs_handle);

    if (err == ESP_OK) {
        err = nvs_erase_all(nvs_handle);
        if (err == ESP_OK) {
            err = nvs_commit(nvs_handle);
        }
        nvs_close(nvs_handle);
    }

    // Reinitialize with defaults
    if (err == ESP_OK) {
        config_init_defaults();
        ESP_LOGI(TAG, "Factory reset complete");
    }

    return err;
```

}

/main/CMakeLists.txt

```
idf_component_register(
    SRCS
        "main.c"
        "sampler_task.c"
        "logger_task.c"
        "telemetry_task.c"
        "web_server.c"
        "config_service.c"
        "discovery_service.c"
        "drivers/bme280_driver.c"
        "drivers/mpu6050_driver.c"
        "drivers/driver_registry.c"
    INCLUDE_DIRS
        "include"
    EMBED_TXTFILES
        "web/index.html"
        "web/app.js"
        "web/style.css"
    EMBED_FILES
        "web/images/UCF_Logo.png"
        "web/images/Draco_Lab.png"
)
/CMakeLists
# OpenSense CMakeLists.txt
cmake_minimum_required(VERSION 3.16)

set(EXTRA_COMPONENT_DIRS "main")
set(SDKCONFIG_DEFAULTS "sdkconfig.defaults")

# Add this line:
set(PARTITION_CSV_PATH "${CMAKE_SOURCE_DIR}/partitions.csv")

include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(opensense)
# OpenSense Flash Partition Table
# Name,   Type, SubType, Offset,  Size,    Flags
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 0x180000,
storage,  data, spiffs,  0x190000,0x70000,
```

/main/web/style.css

```
/* OpenSense Dashboard - UCF/DRACO Lab Theme */
*{box-sizing:border-box;margin:0;padding:0}
html{scroll-behavior:smooth}
body{font-family:system-ui,-apple-system,Segoe
UI,Roboto,Helvetica,Arial,sans-serif;color:#0b0b0b;background:var(--white);line-height:1.6;min-height:100
vh}

/* Color Tokens */
:root{--black:#000;--navy:#0A2342;--light-blue:#63B3ED;--white:#f0f8ff;--muted:#f6f8fb;--border:#e5eaf1;--
focus:#2563eb;--success:#10b981;--warning:#f59e0b;--error:#ef4444}
```

```
/* Focus */
:focus{outline:3px solid var(--focus);outline-offset:2px}
:focus:not(:focus-visible){outline:none}

/* Header */
.site-header{position:sticky;top:0;z-index:50;background:linear-gradient(180deg,var(--black),var(--navy));c
olor:var(--white);display:flex;align-items:center;justify-content:space-between;padding:.75rem
1.5rem;border-bottom:4px solid var(--light-blue);box-shadow:0 2px 8px rgba(0,0,0,.15)}
.brand{display:flex;align-items:center;gap:.5rem}
.brand-text{font-weight:700;font-size:1.1rem;letter-spacing:.3px}
.top-nav ul{list-style:none;display:flex;gap:1rem;align-items:center}
.status-item{display:flex;align-items:center;gap:.4rem;padding:.4rem
.8rem;background:rgba(99,179,237,.15);border-radius:999px;border:1px solid
rgba(99,179,237,.3);transition:all 120ms ease}
.status-item:hover{background:rgba(99,179,237,.25);transform:translateY(-1px)}
.status-label{font-size:.85rem;font-weight:500;opacity:.9}
.status-value{font-weight:700;color:var(--light-blue)}
.status-indicator{font-size:1.2em;animation:pulse 2s ease-in-out infinite}
@keyframes pulse{0%,100%{opacity:1}50%{opacity:.6}}

/* Hero */
.hero{padding:3rem 1rem 2rem;text-align:center;background:linear-gradient(180deg,var(--navy),#17375e
60%,#1b1b1b 100%);color:var(--white);border-bottom:4px solid var(--light-blue)}
.title-row{max-width:1100px;margin:0 auto .75rem;padding:0 1rem;display:grid;grid-template-columns:1fr
auto 1fr;align-items:center;gap:1.5rem}
.title-row h1{font-size:clamp(1.75rem,4vw,2.5rem);margin:0;font-weight:700}
.title-img{width:120px;height:120px;object-fit:contain;display:block}
.title-img.left{justify-self:end}
.title-img.right{justify-self:start}
.subtitle{margin:.5rem 0
0;font-size:clamp(1rem,2.2vw,1.15rem);opacity:.95;color:var(--light-blue);font-weight:500}

/* Main Content */
.main-content{max-width:1400px;margin:0 auto}
.content-block{padding:2rem 1.5rem;max-width:1200px;margin:0 auto}
.section-title{font-size:1.75rem;font-weight:700;color:var(--navy);margin-bottom:1.5rem;text-align:center}

/* Charts */
.charts-block{background:var(--white)}
.charts-grid{display:grid;grid-template-columns:repeat(auto-fit,minmax(320px,1fr));gap:1.25rem}
.chart-card{background:var(--muted);border:1px solid
var(--border);border-radius:14px;overflow:hidden;box-shadow:0 2px 8px
rgba(0,0,0,.06);transition:transform 120ms ease,box-shadow 120ms ease}
.chart-card:hover{transform:translateY(-2px);box-shadow:0 6px 14px rgba(0,0,0,.1)}
.chart-header{background:linear-gradient(135deg,var(--navy),#17375e);color:var(--white);padding:.75rem
1rem;display:flex;justify-content:space-between;align-items:center;border-bottom:2px solid
var(--light-blue)}
.chart-header h3{font-size:1rem;font-weight:600;margin:0}
.live-indicator{width:10px;height:10px;background:var(--success);border-radius:50%;animation:blink 2s
ease-in-out infinite;box-shadow:0 0 8px var(--success)}
@keyframes blink{0%,100%{opacity:1}50%{opacity:.3}}
.chart-container{padding:1rem;height:250px;background:white}

/* Config Panel */
```

```
.config-block{background:var(--muted);border-top:1px solid var(--border);border-bottom:1px solid
var(--border)}
.config-panel{background:white;border:1px solid
var(--border);border-radius:14px;padding:1.5rem;box-shadow:0 2px 8px rgba(0,0,0,.06)}
.config-grid{display:flex;gap:1.5rem;align-items:flex-end;flex-wrap:wrap}
.config-item{display:flex;flex-direction:column;gap:.5rem;flex:1;min-width:200px}
.config-item label{font-weight:600;color:var(--navy);font-size:.95rem}
.config-item input{padding:.65rem 1rem;border:2px solid
var(--border);border-radius:8px;font-size:1rem;font-weight:600;transition:all 120ms
ease;background:white;font-family:inherit}
.config-item input:hover{border-color:var(--light-blue)}
.config-item input:focus{outline:none;border-color:var(--light-blue);box-shadow:0 0 0 3px
rgba(99,179,237,.2)}
.config-actions{display:flex;gap:.75rem}

/* Buttons */
.button{display:inline-block;padding:.65rem
1.25rem;border-radius:999px;text-decoration:none;font-weight:600;font-size:.95rem;border:2px solid
var(--light-blue);cursor:pointer;transition:all 120ms ease;font-family:inherit;white-space:nowrap}
.button.solid{background:var(--light-blue);color:var(--navy)}
.button.solid:hover{background:#5aa5dc;border-color:#5aa5dc;transform:translateY(-1px);box-shadow:0
4px 12px rgba(99,179,237,.3)}
.button.ghost{background:transparent;color:var(--light-blue)}
.button.ghost:hover{background:rgba(99,179,237,.1);transform:translateY(-1px)}

/* Multiplexer Grid */
.mux-block{background:var(--white)}
.mux-grid{display:grid;grid-template-columns:repeat(auto-fill,minmax(180px,1fr));gap:1.25rem}
.mux-tile{background:var(--muted);border:1px solid
var(--border);border-radius:12px;padding:1.25rem;text-align:center;box-shadow:0 2px 6px
rgba(0,0,0,.05);transition:transform 120ms ease,box-shadow 120ms
ease;position:relative;overflow:hidden}
.mux-tile::before{content:'';position:absolute;top:0;left:0;right:0;height:3px;background:var(--light-blue);tran
sform:scaleX(0);transition:transform 120ms ease}
.mux-tile:hover{transform:translateY(-2px);box-shadow:0 6px 12px rgba(0,0,0,.08)}
.mux-tile:hover::before{transform:scaleX(1)}
.mux-header{font-weight:700;color:var(--navy);margin-bottom:.75rem;font-size:.9rem;text-transform:upper
case;letter-spacing:.5px}
.mux-sensor{font-size:1.15rem;font-weight:700;margin:.5rem 0;color:var(--navy)}
.mux-addr{font-family:'Courier
New',monospace;color:#666;font-size:.85rem;background:white;padding:.35rem
.65rem;border-radius:6px;display:inline-block;margin-top:.5rem;border:1px solid var(--border)}
.mux-empty{color:#9ca3af;font-style:italic;padding:1rem 0;font-size:1rem}
.mux-status{margin-top:.75rem;padding:.4rem
.8rem;border-radius:999px;font-size:.8rem;font-weight:600;display:inline-flex;align-items:center;gap:.35re
m}
.mux-status.auto{background:#d1fae5;color:#065f46;border:1px solid #a7f3d0}
.mux-status.manual{background:#fef3c7;color:#92400e;border:1px solid #fde68a}

/* Footer */
.site-footer{background:linear-gradient(180deg,var(--navy),var(--black));color:var(--white);text-align:center
;padding:1.5rem 1rem;border-top:4px solid var(--light-blue);margin-top:2rem}
.site-footer p{margin:0;font-size:.95rem;opacity:.95}

/* Responsive */
@media(max-width:1024px){.charts-grid{grid-template-columns:repeat(auto-fit,minmax(280px,1fr))}}
```

```
@media(max-width:768px){html,body{overflow-x:hidden}.site-header{flex-wrap:wrap;row-gap:.75rem;padding:.65rem 1rem}.brand-text{font-size:.95rem}.top-nav ul{flex-wrap:wrap;gap:.5rem;width:100%;justify-content:center}.status-item{padding:.35rem .65rem;font-size:.85rem}.hero{padding:2.5rem 1rem 1.75rem}.title-row{grid-template-columns:1fr;text-align:center;gap:1rem}.title-img{width:90px;height:90px;justify-self:center}.content-block{padding:1.5rem 1rem}.section-title{font-size:1.5rem}.charts-grid{grid-template-columns:1fr;gap:1rem}.config-grid{flex-direction:column;align-items:stretch;gap:1rem}.config-item{width:100%}.config-actions{width:100%;flex-direction:column}.button{width:100%;text-align:center}.mux-grid{grid-template-columns:repeat(2,1fr);gap:1rem}}
@media(max-width:480px){.title-img{width:70px;height:70px}.mux-grid{grid-template-columns:1fr}.status-item{flex:1;justify-content:center}}

/* Utilities */
.hidden{display:none!important}

/* Scrollbar */
::-webkit-scrollbar{width:10px;height:10px}
::-webkit-scrollbar-track{background:var(--muted)}
::-webkit-scrollbar-thumb{background:var(--light-blue);border-radius:5px}
::-webkit-scrollbar-thumb:hover{background:#5aa5dc}
```

/main/web/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>OpenSense v3 Dashboard</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header class="site-header" role="banner">
    <div class="brand">
      <span class="brand-text">OpenSense v3 Dashboard</span>
    </div>
    <nav class="top-nav" aria-label="Status">
      <ul>
        <li class="status-item">
          <span class="status-label">SD Card:</span>
          <span id="sd-status" class="status-indicator">⚪</span>
        </li>
        <li class="status-item">
          <span class="status-label">Rate:</span>
          <span id="sample-rate" class="status-value">10 Hz</span>
        </li>
        <li class="status-item">
          <span class="status-label">Samples:</span>
          <span id="sample-count" class="status-value">0</span>
        </li>
        <li class="status-item">
          <span class="status-label">Active:</span>
          <span id="active-channels" class="status-value">0</span>
        </li>
      </ul>
    </nav>
  </header>
```

```
<main id="main" class="main-content">
  <section class="hero" aria-labelledby="page-title">
    <div class="title-row">
      <img class="title-img left" src="images/UCF_Logo.png" alt="UCF Logo" />
      <h1 id="page-title">Real-time Sensor Monitoring</h1>
      <img class="title-img right" src="images/Draco_Lab.png" alt="Draco Lab Logo" />
    </div>
    <p class="subtitle">Open Source Multi-Sensor Data Acquisition Platform</p>
  </section>

  <section class="content-block charts-block" aria-labelledby="charts-title">
    <h2 id="charts-title" class="section-title">Real-time Data Streams</h2>

    <div class="charts-grid">
      <div class="chart-card">
        <div class="chart-header">
          <h3>Channel 0: <span id="value-0">--</span></h3>
          <div class="live-indicator"></div>
        </div>
        <div class="chart-container">
          <canvas id="chart-0" width="400" height="150"></canvas>
        </div>
      </div>

      <div class="chart-card">
        <div class="chart-header">
          <h3>Channel 1: <span id="value-1">--</span></h3>
          <div class="live-indicator"></div>
        </div>
        <div class="chart-container">
          <canvas id="chart-1" width="400" height="150"></canvas>
        </div>
      </div>

      <div class="chart-card">
        <div class="chart-header">
          <h3>Channel 2: <span id="value-2">--</span></h3>
          <div class="live-indicator"></div>
        </div>
        <div class="chart-container">
          <canvas id="chart-2" width="400" height="150"></canvas>
        </div>
      </div>

      <div class="chart-card">
        <div class="chart-header">
          <h3>Channel 3: <span id="value-3">--</span></h3>
          <div class="live-indicator"></div>
        </div>
        <div class="chart-container">
          <canvas id="chart-3" width="400" height="150"></canvas>
        </div>
      </div>

      <div class="chart-card">
```

```
                <div class="chart-header">
                    <h3>Channel 4: <span id="value-4">--</span></h3>
                    <div class="live-indicator"></div>
                </div>
                <div class="chart-container">
                    <canvas id="chart-4" width="400" height="150"></canvas>
                </div>
            </div>
        </div>
    </section>

    <section class="content-block config-block" aria-labelledby="config-title">
        <h2 id="config-title" class="section-title">Configuration</h2>
        <div class="config-panel">
            <div class="config-grid">
                <div class="config-item">
                    <label for="config-rate">Sample Rate (Hz)</label>
                    <input type="number" id="config-rate" min="1" max="100" value="10">
                </div>
                <div class="config-actions">
                    <button id="btn-apply" class="button solid">Apply Changes</button>
                    <button id="btn-revert" class="button ghost">Revert</button>
                </div>
            </div>
        </div>
    </section>

    <section class="content-block mux-block" aria-labelledby="mux-title">
        <h2 id="mux-title" class="section-title">Multiplexer Channels</h2>
        <div class="mux-grid" id="mux-channels"></div>
    </section>
  </main>

  <footer class="site-footer" role="contentinfo">
    <p>OpenSense v3 — Senior Design Group 1 | Sponsored by Dr. Mike Borowczak and the DRACO
Lab</p>
  </footer>

  <script src="app.js"></script>
</body>
</html>
```

/web/app.js

```
console.log('OpenSense Dashboard Loading...');

var ws = null;
var chartData = [];
var canvases = [];
var colors = ['#3498db', '#e74c3c', '#2ecc71', '#9b59b6', '#f39c12'];

function initCharts() {
  console.log('Initializing charts...');
  for (var i = 0; i < 5; i++) {
    var canvas = document.getElementById('chart-' + i);
    if (!canvas) {
      console.log('Canvas chart-' + i + ' not found');
      continue;
```

```
        }
        canvases[i] = canvas;
        chartData[i] = new Array(100).fill(null);
        drawChart(i);
    }
    console.log('Charts initialized');
}

function drawChart(channelId) {
    var canvas = canvases[channelId];
    if (!canvas) return;

    var ctx = canvas.getContext('2d');
    var width = canvas.width;
    var height = canvas.height;
    var data = chartData[channelId];
    var color = colors[channelId % colors.length];

    // Clear canvas
    ctx.fillStyle = '#1a1a2e';
    ctx.fillRect(0, 0, width, height);

    // Draw grid
    ctx.strokeStyle = '#333355';
    ctx.lineWidth = 1;
    for (var i = 0; i < 5; i++) {
        var y = (height / 5) * i;
        ctx.beginPath();
        ctx.moveTo(0, y);
        ctx.lineTo(width, y);
        ctx.stroke();
    }

    // Find min/max for scaling
    var validData = data.filter(function(v) { return v !== null; });
    if (validData.length < 2) return;

    var min = Math.min.apply(null, validData);
    var max = Math.max.apply(null, validData);
    var range = max - min;
    if (range < 0.1) range = 1;
    var padding = range * 0.1;
    min -= padding;
    max += padding;

    // Draw line
    ctx.strokeStyle = color;
    ctx.lineWidth = 2;
    ctx.beginPath();

    var firstPoint = true;
    for (var i = 0; i < data.length; i++) {
        if (data[i] === null) continue;

        var x = (i / (data.length - 1)) * width;
        var y = height - ((data[i] - min) / (max - min)) * height;
```

```
            if (firstPoint) {
                ctx.moveTo(x, y);
                firstPoint = false;
            } else {
                ctx.lineTo(x, y);
            }
        }
        ctx.stroke();

        // Draw scale labels
        ctx.fillStyle = '#888';
        ctx.font = '10px Arial';
        ctx.fillText(max.toFixed(1), 5, 12);
        ctx.fillText(min.toFixed(1), 5, height - 5);
    }

    function updateChart(channelId, value) {
        if (channelId >= 5) return;

        chartData[channelId].shift();
        chartData[channelId].push(value);
        drawChart(channelId);

        // Update value display
        var valueEl = document.getElementById('value-' + channelId);
        if (valueEl) {
            valueEl.textContent = value.toFixed(2);
        }
    }

    function connectWebSocket() {
        var wsUrl = 'ws://' + location.host + '/ws';
        console.log('Connecting to:', wsUrl);

        ws = new WebSocket(wsUrl);

        ws.onopen = function() {
            console.log('WebSocket connected!');
            var sdStatus = document.getElementById('sd-status');
            if (sdStatus) sdStatus.textContent = '🟢';
        };

        ws.onmessage = function(event) {
            try {
                var data = JSON.parse(event.data);

                if (data.c) {
                    var activeCount = 0;
                    for (var i = 0; i < data.c.length && i < 5; i++) {
                        if (data.c[i] !== null) {
                            updateChart(i, data.c[i]);
                            activeCount++;
                        }
                    }
                    var activeEl = document.getElementById('active-channels');
```

```
            if (activeEl) activeEl.textContent = activeCount;
        }

        if (data.sd !== undefined) {
            var sdStatus = document.getElementById('sd-status');
            if (sdStatus) sdStatus.textContent = data.sd ? '🟢' : '🔴';
        }

        if (data.cnt !== undefined) {
            var cntEl = document.getElementById('sample-count');
            if (cntEl) cntEl.textContent = data.cnt;
        }
    } catch (e) {
        console.error('Parse error:', e);
    }
};

ws.onclose = function() {
    console.log('WebSocket closed, reconnecting...');
    var sdStatus = document.getElementById('sd-status');
    if (sdStatus) sdStatus.textContent = '🟡';
    setTimeout(connectWebSocket, 2000);
};

ws.onerror = function(e) {
    console.error('WebSocket error:', e);
};
}

function loadConfig() {
    fetch('/config')
        .then(function(r) { return r.json(); })
        .then(function(config) {
            console.log('Config:', config);
            var rateEl = document.getElementById('config-rate');
            var sampleRateEl = document.getElementById('sample-rate');
            if (rateEl) rateEl.value = config.sample_rate_hz || 10;
            if (sampleRateEl) sampleRateEl.textContent = (config.sample_rate_hz || 10) + ' Hz';
        })
        .catch(function(e) {
            console.error('Config error:', e);
        });
}

function applyConfig() {
    var rateEl = document.getElementById('config-rate');
    var rate = rateEl ? parseInt(rateEl.value) : 10;

    fetch('/config', {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify({sample_rate_hz: rate})
    })
    .then(function(r) {
        if (r.ok) {
            console.log('Config updated');
```

```
            loadConfig();
        }
    })
    .catch(function(e) {
        console.error('Config update error:', e);
    });
}

window.onload = function() {
    console.log('Window loaded');
    initCharts();
    loadConfig();
    connectWebSocket();

    var applyBtn = document.getElementById('btn-apply');
    var revertBtn = document.getElementById('btn-revert');
    if (applyBtn) applyBtn.onclick = applyConfig;
    if (revertBtn) revertBtn.onclick = loadConfig;
};
```

Appendix E - ChatGPT Prompts and Outcomes

[1a] **Case study 1 prompt:** What properties would make a good battery for a sensor

[1a] **Case study 1 ChatGpt Answer:** A good sensor battery must match load, lifetime, and safety constraints. Use these criteria.

Electrical

- Nominal voltage fits regulators and peripherals.
- Sufficient energy: (E \approx V_{\text{nom}}\times \text{Ah}). Runtime (\approx \dfrac{E \cdot \eta}{P_{\text{avg}}}).
- Low internal resistance for wake-up/pulse loads; verify voltage droop at peak current.
- Stable voltage vs. state of charge and temperature.
- Low self-discharge for long sleep duty cycles.

Power profile fit

- Supports duty-cycled peaks (TX/RX, sensing) without brownouts.
- Good pulse performance at end-of-life.
- If rechargeable: acceptable charge time and charge-current limits.

Environment

- Temperature range matches deployment. Check capacity and ESR at min/max °C.
- Humidity and altitude tolerance; venting not required by enclosure.

Form factor & mechanics

- Size, mass, and mounting align with enclosure and CG.
- Robust leads/connector; reverse-polarity and short protection.
- Replaceability or sealed-for-life as required.

Safety & compliance

- Chemistry aligned with project rules and transport limits.
- Built-in protection (OVP/UVP/OCP/OTP) or external BMS.
- Certifications where needed (UL, UN38.3, IEC).

Longevity

- Cycle life (rechargeable) or shelf life (primary).
- Calendar aging at storage temperature.
- Predictable capacity fade for maintenance planning.

Monitoring

- Accurate fuel-gauge curve for the chemistry/regulator.
- Simple SOC telemetry or low-battery threshold that avoids false resets.

Cost & logistics

- Total cost of ownership: cell + BMS + charger + regulatory + replacement labor.
- Availability over product life; multi-source if possible.

Chemistry quick picks

- Primary lithium (Li-$SOCl_2$): very low self-discharge, long life, poor high-pulse without buffer. Best for ultra-low-power, multi-year.

- Alkaline: cheap, wide availability, high self-discharge under heat, weak at high pulse and cold. OK for prototypes.
- Coin cell (CR2032/2450): tiny, low pulse, good for µW–mW beacons. Add storage capacitor for radios.
- NiMH: safer than Li-ion, moderate energy density, higher self-discharge, decent pulse. Works when Li-ion is disallowed.
- $LiFePO_4$: safer than Li-ion, flat discharge, good pulse, lower Wh/L than NMC. Strong choice if safety prioritized and recharge allowed.
- Supercap + primary: supercap handles bursts, primary supplies average. Useful for narrow pulse radios.

Design levers

- Add bulk capacitance near radio/MCU to handle 10–100 ms bursts.
- Choose regulators with low IQ and good transient response.
- Tune firmware duty cycle: shorten radio on-time, batch sensor reads, deep sleep aggressively.

Basic sizing workflow

1. Measure average power $P_{\text{avg}} = \sum (I_i \cdot V \cdot t_i)/T$.
2. Add 20–30% margin for temperature, aging, and regulator losses (η).
3. Check pulse current vs. cell ESR: $\Delta V = I_{\text{peak}}\cdot \text{ESR}$; ensure V_{min} stays above brownout.
4. Verify worst-case cold capacity and EOL voltage.
5. Validate with a week-long duty-cycle test log.

If Li-ion is banned

- Prefer $LiFePO_4$ for rechargeable needs.
- Prefer Li-$SOCl_2$ or CR coin for multi-year primary nodes.
- NiMH is acceptable where recharge and safety matter and volume allows; use low-self-discharge cells.

This set covers the properties and the selection path.

[2a] **Case Study 2 prompt:** What is I2C and its standards
[2b] **Case Study 2 ChatGpt Answer:** I²C is a 2-wire, synchronous serial bus for short-range, on-board communication between a controller and peripherals.

Core facts

- Lines: SDA (data) and SCL (clock), open-drain/open-collector with pull-ups.
- Topology: multi-drop, multi-master capable, arbitration on SDA.
- Framing: START, 7-bit or 10-bit address + R/W, ACK/NACK, optional repeated START, STOP.
- Flow control: clock stretching by targets allowed.
- Logic levels: referenced to bus VDD; pull-ups define HIGH.
- Typical bus capacitance limit: ≈400 pF (application dependent).

Official I²C speed grades (per NXP I²C-bus spec)

- Standard-mode (Sm): 100 kHz
- Fast-mode (Fm): 400 kHz
- Fast-mode Plus (Fm+): 1 MHz
- High-speed mode (Hs): 3.4 MHz
- Ultra-Fast-mode (UFm): 5 MHz, write-only, no clock stretching

Addressing

- 7-bit: 112 usable addresses after reserveds.
- 10-bit: extended form with 2-byte header.
- Reserved addresses include: general call (0x00), Hs controller codes, 10-bit prefix range, and others.

Electrical design notes

- Pull-ups: choose via rise-time target and bus capacitance. Approximate: (t_r \approx 0.8473,R_{PU}C_{BUS}).
- Common ranges: 10 kΩ–1 kΩ depending on speed and load; Fm+ often needs ≤1 kΩ.

- Keep traces short and balanced; add series resistors or buffers if needed for reflections/EMI.

Compliance and related standards

- I²C-bus Specification and User Manual (NXP, UM10204): canonical definition of protocol, timing, electrical limits, speed grades, and reserved addresses.
- SMBus (System Management Bus): I²C-derived. Adds timeouts, stricter voltage/timing, default 100 kHz, Packet Error Code option, alert line (SMBALERT#). Interoperable with care.
- PMBus: command set layered on SMBus for digital power management.
- F-I²C/I²C buffers and level shifters: vendor app notes specify how to extend bus length, isolate capacitance, or bridge voltages.
- MIPI I3C: successor bus with higher speed and in-band interrupts; not backward-compatible electrically, though some controllers support I²C fallback.

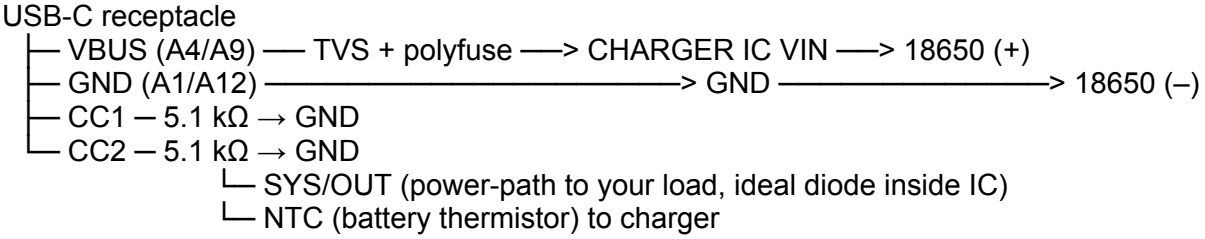
When choosing I²C for a sensor

- Check peak current for pull-up sizing and rise times at your speed grade.
- Verify target supports required features (clock stretching, 3.4 MHz, 10-bit address) and VDD.
- Budget bus capacitance across all devices, traces, cables, and ESD parts.
- Reserve addresses early and document them; avoid conflicts with reserved ranges.
- Consider SMBus rules if using laptop/management ecosystems or needing timeouts/PEC.

[3a] **Case Study 3 Prompt:** What is the best way to connect a USB-C to make it so that a USB-C can charge an 18650 battery

[3b] **Case Study 3 ChatGpt Answer:** Use a USB-C 5V sink + single-cell Li-ion charger with power-path. This is the simplest, safe, standards-compliant way to charge an 18650.

Topology



Why this works

- USB-C 5V only: Two 5.1 kΩ Rd pull-downs on CC1/CC2 announce a sink. Any compliant source provides 5 V. No PD is required.
- Charger IC handles the CC/CV profile (4.20 V, constant current then constant voltage, auto-termination ~0.05–0.1 C), safety timers, and input current limiting so you don't overload the USB source.
- Power-path lets the system run while charging and prevents the battery from back-feeding VBUS.

Recommended charger ICs (single-cell, 5V input)

- TI BQ24072/BQ24074: 1 A class, USB/adapter input limit select, integrated power-path and ideal diode.
- TI BQ25601D: Higher current, full power-path, good telemetry.
- Microchip MCP73871: 1 A class, power-path, simple BOM.
- TP4056: Very common, but no power-path and minimal protections; add a protection IC and don't draw system power while charging.

If you need >5V or >3A from the source, add a USB-C PD sink controller (e.g., STUSB4500, FUSB302 + MCU) and negotiate a PD contract. For a single 18650 this is usually unnecessary.

USB-C wiring specifics

- Tie all VBUS pins together, all GND pins together.
- CC1 and CC2: each 5.1 kΩ to GND (Rd). Leave SBU and D+/D– unconnected unless you need data.
- Add TVS diode on VBUS to GND and a resettable fuse. Optional input OVP/ideal diode controller for extra protection.

Charger setup checklist

- Charge current (I_CHG): set per datasheet resistor. Choose ≤ the advertised Type-C current. If you cannot read the source's Rp, cap I_CHG ≤ 500–900 mA to be safe with legacy ports.
- Termination: 0.05–0.1 C typical.
- Battery safety: use cells with built-in protection or add DW01 + dual MOSFET-style protection board.

- Thermal: connect an NTC on the cell to the charger's TS pin (JEITA temp limits).
- Power-path: connect your system to the IC's SYS/OUT rail, not directly to the battery.
- Layout: short, wide traces for battery and VBUS; solid ground; good thermal vias under the IC.

If Li-ion is restricted in your project

- Swap for $LiFePO_4$ and use a $LiFePO_4$-specific charger (3.60V CV). USB-C front end stays the same.
- For NiMH, use a NiMH charger IC; do not reuse Li-ion chargers.

Bring-up test plan

1. Verify CC pull-downs and 5V on VBUS with no battery attached.
2. Limit I_CHG low (e.g., 200–300 mA) for first tests. Confirm CC→CV transition near 4.20 V.
3. Load the SYS rail and check the IC maintains regulation without brownout.
4. Confirm termination current, thermistor response, and no reverse current to VBUS.

[4a] **Case Study 4 Prompt:** What's the expected price range to custom order a PCB for a sensor and what are the best options

[4b] **Case Study 4 ChatGpt Answer:** Short answer: $5–$200 for bare PCBs and $80–$600+ per build for turnkey prototypes, depending on size, layers, speed, and assembly scope.

Typical prototype price bands (USD)

- Bare PCB, low-cost offshore (2-layer, ≤100×100 mm, 5–10 pcs): $2–$20 + shipping. Example: JLCPCB advertises "$2 / 5 pcs" entry pricing. (cart.jlcpcb.com)
- US pooled fab (per-sq-in pricing, 3 boards): $5/in² (2-layer), $10/in² (4-layer); 4–12+ day ship. OSH Park publishes these rates. (docs.oshpark.com)
- US quick-turn specials (per board): ~$33 each for simple 2-layer, $66 each for simple 4-layer, with limits on area and features. Advanced Circuits advertises these programs. (AdvancedPCB)
- Laser-cut paste stencil (frameless): $12–$40 typical from offshore vendors; US framed stencils are much higher. Example anecdote: $12 from China; US framed $199–$269. (Reddit)

Turnkey PCB assembly (PCBA)

- China fab+assembly (JLCPCB, PCBWay): low NRE; assembly quote excludes parts and bare-PCB cost; dynamic by BOM and placements. Good for small, common parts. (pcbway.com)

- US platforms (MacroFab, Sierra Circuits): higher unit cost but faster logistics, domestic support, and easier compliance. MacroFab advertises real-time pricing; Sierra markets 24-hour fab and 5-day turnkey. (MacroFab, Inc.)

Ballpark for a sensor board (example: 50×50 mm, 2-layer, 10 pcs, ~150 SMD placements, common parts):

- Offshore fab + assembly: $15–$60 bare PCBs + $25–$150 stencil + $150–$400 assembly labor + parts at market price + $20–$60 shipping. Vendor quotes vary; see program pages above. (cart.jlcpcb.com)

- US pooled fab + domestic assembly: $75–$150 bare PCBs (per-in² math) + stencil if needed + higher assembly labor. Faster turns possible. (docs.oshpark.com)

Cost drivers to control

- Area and layers dominate bare-PCB cost. Vendors state price scales with dimensions, stackup, materials, and rules. (Sierra Circuits)

- Lead time premiums: “super-swift” or 24-hour turns cost more. (docs.oshpark.com)

- Assembly complexity: placements, fine-pitch, double-sided, through-hole, and specialty parts raise labor and yield risk. Vendor pages note assembly quotes exclude parts and depend on files. (pcbway.com)

Best options by need

- Cheapest prototypes, no assembly: JLCPCB / PCBWay deals. Add a low-cost stencil if hand-assembling. (cart.jlcpcb.com)

- US-made small runs, predictable per-in² pricing: OSH Park. (docs.oshpark.com)

- Domestic quick-turn or compliance-sensitive: Advanced Circuits “$33/$66 Each,” Sierra Circuits quick-turn. (AdvancedPCB)

- Turnkey assembly with flexible sourcing: MacroFab (US network). (MacroFab, Inc.)

Actionable next steps

1. Quote your board at two vendors: one pooled US and one offshore. Compare landed cost and lead time. Use their instant quote tools. (cart.jlcpcb.com)

2. If assembling, generate centroid + BOM with MPNs; upload for dynamic PCBA quotes. (MacroFab, Inc.)

3. Order a frameless stencil for hand assembly or specify vendor-supplied stencil for turnkey. (pcbway.com)

If you share size, layer count, finish, and a rough BOM, I can compute a tighter budget.

[5a] **Case Study 5 Prompt:** What sort of MCU would be best suited for a sensor versus it's competition with pros and cons

[5b] **Case Study 5 ChatGpt Answer:** Pick the MCU by radio, energy budget, and sensing/analog needs. Use this short map.

Fast picks by use-case

- Coin-cell, multi-year, BLE beacon: *Nordic nRF52/53*, *TI CC26xx*, *Silabs EFR32BG*.
  Pros: integrated BLE, <1–3 µA sleep with RTC, mature stacks. Cons: BLE brings bigger flash/RAM and stack complexity.

- Ultra-low-energy logger, no radio or external radio (LoRa/433): *Ambiq Apollo3/4*, *STM32L0/L4/L5/U5*, *MSP430*.
  Pros: sub-µA sleep, low IQ peripherals, good ADCs. Cons: modest CPU; Ambiq tooling quirks.

- Wi-Fi sensor (HTTP/MQTT), wall or large battery: *Espressif ESP32-C3/C6/S3*.
  Pros: cheapest Wi-Fi, strong SDK, BT on some SKUs. Cons: idle power is high; careful power-path and sleep design required.

- Long-range sub-GHz (meters→kilometers) with integrated MCU: *TI CC1312R/CC1352* (sub-GHz or dual-band).
  Pros: very low RX/TX energy, mature 802.15.4/Simplicity. Cons: RF tuning and regional band rules.

- General sensor hub with rich analog and low power: *STM32L4/L5/U5*, *Renesas RA4/RA6*, *NXP LPC55S*.
  Pros: broad peripherals, low active µA/MHz, strong ecosystem. Cons: more expensive than ESP-class parts.

- Cost-first, no radio, flexible I/O timing: *Raspberry Pi RP2040*.
  Pros: very cheap, PIO offloads sensors. Cons: no RF, higher sleep than ULP leaders.

Family snapshots (pros ▾ / cons ▴)

- Nordic nRF52/53 (BLE/Thread/Zigbee)
  - ▾ Excellent low-power BLE, robust SDK/examples, DC-DC, good ADC/SAADC
  - ▴ BLE stack learning curve; RF layout care; 53 is pricier

- Silabs EFR32BG (BLE) / EFR32FG (sub-GHz)
  - ▾ Strong RF, hardware crypto, tools for low-power profiling
  - ▴ Licensing/tool friction for some stacks

- TI CC26xx/CC13xx
  - ▾ Very low sleep, dual-band options (CC1352), Sensor Controller (ULP co-proc)
  - ▴ TI-RTOS specifics; supply chain varies

- Ambiq Apollo3/4 (ARM M4F)
  - ▾ Best-in-class sleep currents, ULP sensor/RTC domains
  - ▴ Smaller ecosystem; HAL differs from STM32/LL

- STM32L0/L4/L5/U5
  - ▾ Broad parts, low IQ peripherals, op-amps/comp on some SKUs, TrustZone (L5/U5)
  - ▴ More costly than bare-bones options

- MSP430
  - ▾ Deterministic ULP, great for simple metering/loggers
  - ▴ Aging ecosystem, limited performance

- ESP32-C3 (RISC-V), C6 (adds 2.4 GHz + 802.15.4), S3 (dual-core)
  - ▾ Cheapest Wi-Fi/BT, rich peripherals, active community
  - ▴ Deep-sleep is decent but not coin-cell friendly with frequent Wi-Fi; RF spikes need bulk caps

- RP2040
  - ▾ Dual-core, PIO for sensor buses, very low cost
  - ▴ No RF, external QSPI flash, sleep not ULP-tier

- Renesas RA / NXP LPC55S
  - ▾ Modern M33 with security, good low-power, industrial temp options
  - ▴ Tooling less ubiquitous than STM32

Selection rules of thumb

1. Radio first:
   BLE → nRF52/CC26xx/EFR32BG.
   Sub-GHz → CC13xx or external SX126x + low-power MCU.
   Wi-Fi → ESP32-C3/C6; budget battery accordingly.

2. Energy math:
   Sleep dominates. Target <2 µA sleep with RTC for coin-cell. Aim <50–100 µA/MHz active and short wake time.

3. Analog needs:
   Prefer MCUs with 12–16-bit ADC, low-noise refs, PGA/comp if sensors are small-signal.

STM32L4/U5 and MSP430 excel.

4. Security/compliance:
   If deploying at scale, prefer M33 + TrustZone + crypto (STM32L5/U5, RA6, LPC55S).

5. Dev speed:
   If schedule rules, pick the stack you or your team already know and can debug quickly.

Minimal decision matrix

- Coin-cell BLE tag: nRF52832/833 or CC2640R2.
- BLE + richer DSP: nRF52840 or EFR32BG22.
- LoRa/FSK node: CC1312R (or STM32L4 + SX1262).
- Wi-Fi sensor cam/mic: ESP32-S3.
- Wi-Fi basic telemetry: ESP32-C3.
- No radio, months on AA: STM32L452, MSP430FR series, Ambiq Apollo3.
- Cost-down with quirky I/O timing: RP2040.

If you share radio, battery, and sensor interface details, I can map one concrete part number and a power budget.

[6a] **Case Study 6 Prompt:** What sorts of design constraints and standards might be expected when incorporating optics such as a spectrometer into a sensor

[6b] **Case Study 6 ChatGpt Answer:** Use these constraints and standards when adding optics (e.g., a spectrometer) to a sensor.

Safety and EMC

- Laser safety if you use laser excitation: classify and label per IEC 60825-1; align with FDA Laser Notice 56 in the U.S. (IEC Webstore)
- Photobiological safety for broadband/LED sources: IEC 62471 and updates (risk groups, exposure limits). (IEC Webstore)
- Electrical safety for lab/measurement equipment: IEC 61010-1. (IEC Webstore)
- EMC immunity/emissions for measurement and lab gear: IEC 61326-1. (IEC Webstore)

Optical design constraints

- Stray light and dynamic range: baffles, low-scatter surfaces, slit/aperture control, order-sorting. Verify SNR and linearity across the band.

- Spectral calibration & resolution: use known line/feature standards and track drift over temp. Raman/IR examples: ASTM E1840 (Raman shift calibration), ASTM E2529 (Raman resolution checks), ASTM E1252 and ASTM E1790/E1655 for IR/NIR qualitative and quantitative method practices. (ASTM International | ASTM)

- Throughput vs. resolution trade: slit width, grating choice, f/# match to fiber or objective.

- Detector management: dark current, cooling or TEC control, integration time limits, pixel non-uniformity correction.

- Illumination stability: regulate source intensity and warm-up; log reference spectra.

Mechanical/thermal constraints

- Alignment stability: rigid opto-mechanical mounts, low CTE mismatch, vibration tolerance.

- Thermal control: temperature hold or compensation to stabilize center wavelength and detector noise.

- Contamination control: dust seals, purge or desiccant, clean materials near the beam to limit outgassing/fluorescence.

- Ingress/robustness: pick enclosure and window materials suited to the spectrum; avoid stress-birefringence if polarization matters.

Electrical/firmware constraints

- Low-noise analog front end: grounding, shielding, and separated returns for TEC/laser drivers versus detector readout.

- Clock and EMI hygiene: spread-spectrum or filtering so digital clocks do not imprint on spectra; meet IEC 61326-1 limits. (IEC Webstore)

- Power budget: handle source and TEC peaks without droop; isolate from the rest of the sensor platform.

- Data quality pipeline: bad-pixel maps, dark subtraction, nonlinearity correction, wavelength calibration, and radiometric/relative intensity calibration.

Method validation and QA

- Calibration artifacts and SRMs: use line lamps or Raman/IR materials per ASTM guides; schedule periodic verification and document pass/fail limits. (ASTM International |

ASTM)

- Chemometric methods: if doing NIR quantitation, build and validate models per ASTM E1655; for qualitative NIR follow ASTM E1790; for general IR, ASTM E1252. (metrohm.com)

Integration checklist

1. Choose source class (LED/halogen/laser) and confirm IEC 62471 or IEC 60825-1 path. (IEC Webstore)
2. Define spectral range, target resolution, and required SNR; size slit/grating/detector accordingly.
3. Control stray light and temperature; design for field alignment retention.
4. Meet IEC 61010-1 safety and IEC 61326-1 EMC in your target market. (IEC Webstore)
5. Implement calibration, verification intervals, and acceptance criteria per applicable ASTM methods. (ASTM International | ASTM)

If you share your wavelength band, source type, and enclosure limits, I can map exact standards scope and a test plan.