Personal Data Acquisition Prototype

Alex Gibson
Julian Henry
Jaxon Young

Table of Contents

Executive Summary

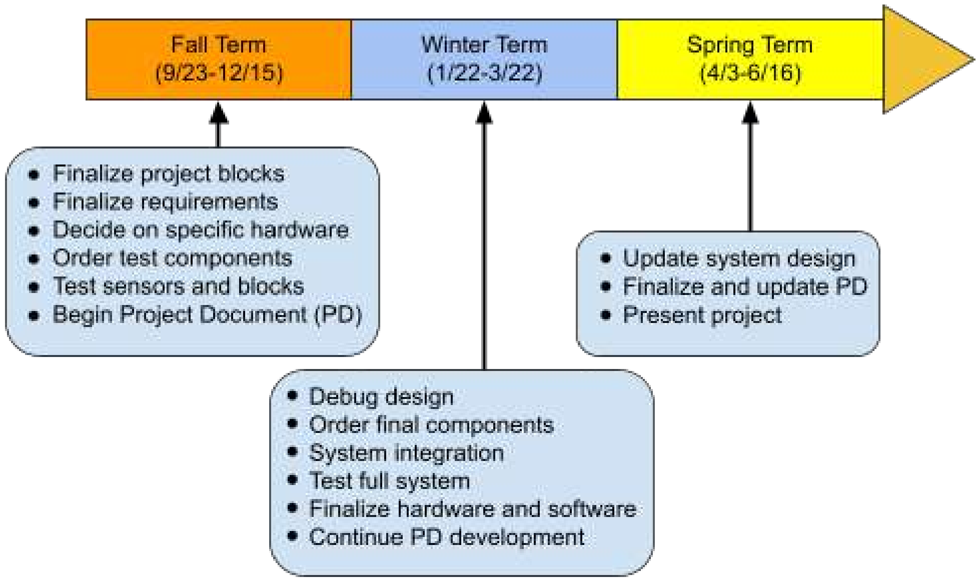
Our project aimed to bridge the gap between professional data collection and everyday users by providing a user-friendly yet precise data acquisition system. While existing solutions serve professionals well, they often lack user-friendliness for the average consumer. Conversely, consumer-friendly options are scarce, and tend to be lower-end devices that do not return specific, accurate data. Our goal was to provide a professional-grade data collection device that the common user can easily set up and use. Some data that could be desired include position, speed, acceleration, displacement, and rotation.

Our device utilized a modular design, with the ability to interchange sensors based on the application. We used STM32 microcontrollers that communicated to a Raspberry Pi via Ethernet cables using the CAN Bus communication protocol. The Raspberry Pi served dual roles, both hosting a database for sensor data storage and serving a user interface for sensor configuration and data visualization. Key sensors integrated into the system include GPS, an ADC for strain gauges, and a block with a mikroBUS™ interface that allowed any Mikroe Click Sensor to be used with the system with minimal reconfiguration.

The project was developed throughout the 2023/24 school year. Fall term was used for planning and prototyping, winter term was used for developing the system, and spring term was used to fix the last-minute issues that came up and present the final design.

As of now, we have successfully developed a functional prototype. Achievements include implementing a modular design, integrating STM32 microcontrollers, and developing a user-friendly UI. Challenges included mastering Rust and EGUI for UI development and custom PCB design complexities.

The top next steps for future development include reworking the custom PCB designs for the sensor modules to reduce reliance on external microcontroller boards, and to enhance UI functionality and aesthetics. Additionally, ensuring backup components and prioritizing environmental sustainability and user safety remain critical considerations for future development.



Section 1: Project Overview

The purpose of this document is to give a description of the project by outlining the key features of the personal data acquisition prototype, describe the protocols used by the team for communication, explain where this project fits into the current product market, and to give a timeline for the development of the project. This section of the document outlines the project by giving an overview of the different features of the system and by describing how the team will go about designing the system.

1.1 Description

The purpose of the personal data acquisition prototype project is to design a data recorder prototype for a general consumer target market. The prototype will be designed to be used with a go-kart. Sensors will be connected to the data recorder in a modular fashion. These sensors will be developed on custom-designed PCBs that interface with STM32 microcontrollers. Each microcontroller will send data to a Raspberry Pi that transmits data to a database. This data can then be viewed visually by the user through a UI that supports data recording and configuration of the sensors. An STM32 microcontroller will be chosen for interfacing with the sensors, the firmware for the microcontrollers will be developed using the RUST programming language, the database will use SQLite, and the GUI will be made using the EGUI framework in RUST.

1.2 Team Contacts and Protocols

This section includes contact information for all team members. This section also contains a table showing our team protocols for communicating with each other and our project partner.

TABLE I
Team Contacts

Name	Email	Role
Alex Gibson	gibsona2@oregonstate.edu	Microcontroller Design
Julian Henry	henryju@oregonstate.edu	Treasurer, PCB Design
Jaxon Young	youngjax@oregonstate.edu	Database, Circuit Design

TABLE II
Protocols

Area	Protocol	Standard
Meeting	Due to busy scheduling, our team at this time does not have a standard meeting time. However, we will make an effort to meet each week when time permits and meet with our project partner on Tuesdays at 3:30pm via Zoom/ Discord	Google Doc notes to discuss topics on the table. Use of the website When2meet in order to find a time when our group can collectively meet.

Contact/ Communication	Team uses SMS and Discord to keep in contact.	Daily updates on progress.
Record/ Data Keeping	Team records meeting outcome and plans for next meeting on shared Google Doc	After each meeting, record the summary. Write down plans for the next meeting and what to be done beforehand.

1.2.1 Diversity, Equity, and Inclusion Statement
Our team will value everyone's ideas and will treat each other with respect. In our design, we will do our best to use language and visuals that are inclusive and understandable to everyone, creating a useful system for people from diverse backgrounds. We will collaborate closely and communicate openly to achieve these goals.

1.2.2 Communication Analysis
Communication with the project partner will happen primarily through Discord. Weekly check-in meetings will be conducted through Zoom. The project partner would like us to communicate what we will be working on, in enough detail that they understand what we are attempting to do and how we plan to do it. This level of communication will be especially important around our work relating to Rust, Embassy, and connection to their existing system of wireless communication to be used for the project, as they have more experiences with these topics and can offer assistance. There will also be a project GitHub that will contain all project information for the project partner to see at their convenience.

1.3 Gap Analysis
The purpose of this project is to fill the need for a cheap, accessible data collection tool for sport vehicles such as mountain bikes and go-karts. This project will provide a method for non-technical people to collect and understand the performance data of their activities. We will design this project to work with a go-kart, but leave open the option of customization so it could be redesigned for use with other vehicles with similar characteristics in the future.
The end users of this project will be individuals from various age groups and skill levels with an interest in sports vehicles. These users may not have extensive technical knowledge but are interested in either improving their performance or gaining insights into their activities.
Currently, we are aiming for the cost of the system itself to be around $300. There are currently two main brands that provide data acquisition systems for go-karting: Alfano and AiM. Alfano data acquisition systems cost from 455€ to 855€ (approximately $487 to $915) [1]. AiM Kart Data Logger Systems cost from £470 to £520 (approximately $577 to $639) [2]. Thus, our system would open the market of high-functioning data acquisition devices to people of lower socioeconomic status.
By providing a simple and customizable solution, this project aims to cater to the needs of our end users and enable them to collect and analyze performance data without the complexity and cost barriers associated with existing tools in the market.

1.4 Timeline and Task List

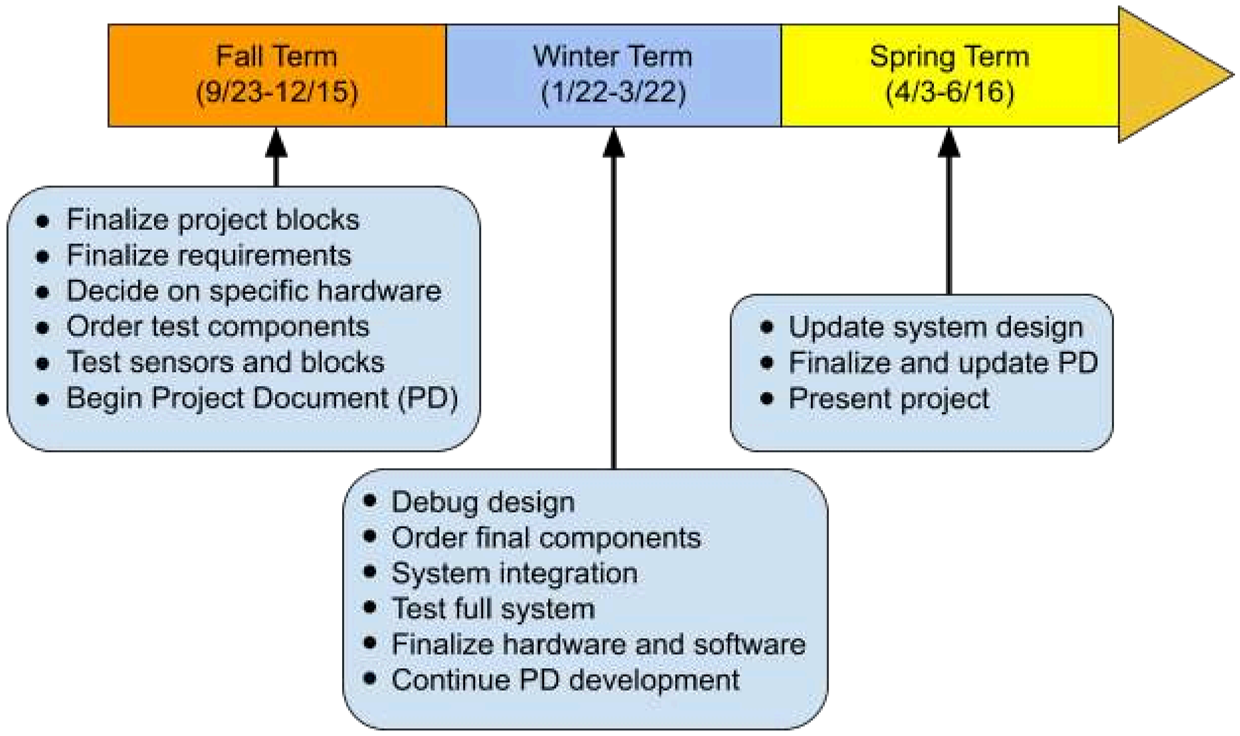


Fig. 1. Project Timeline

TABLE III
Project Task List

Task List	Impact Risk	Expected Hours	Week Due	Champion	Actual Hours
Draft Project Document section 1	3	9	3	Alex	9
Finalize system requirements	10	12	4	Jaxon	15
Decide what sensors to use	10	5	5	Julian	3
Decide on a microcontroller	8	4	5	Alex	5
Determine power requirements	8	3	5	Jaxon	3
Finalize project blocks	5	4	5	Julian	10
Draft Project Document section 2	3	9	6	Alex	9
Finalize Project Document section 1 & 2	2	4	8	Jaxon	4
Draft Project Document section 3 & 5	5	10	9	Julian	13
Learn and practice Rust	9	5	10	Alex	8

Research components	8	6	11	Jaxon	6
Finalize block Interfaces	9	9	12	Julian	9
Prototype first block	9	15	12	Alex	4
Draft ADC and GPS block validations	7	8	13	Jaxon	10
Draft Mikroe Sensor and Power block validations	7	8	13	Julian	10
Draft UI and Logger block validations	7	8	13	Alex	10
Prototype sensor, logger, and database communication	7	6	13	Jaxon	20
Develop UI	9	10	14	Julian	20
Verify first blocks	8	12	15	Alex	15
Revise system architecture	10	15	16	Jaxon	12
Research communication protocols	8	8	17	Julian	12
Block Verification: Block 1 & 2	7	10	18	Alex	20
Update Project document	4	4	19	Jaxon	23
Finalize PCBs and order	9	10	20	Julian	22
Order final components	8	5	20	Alex	22
Provide evidence for system verification	7	4	20	Jaxon	10
Assemble blocks	10	15	20	Julian	10
Test entire system	10	20	20	Alex	10
Finalize executive summary	1	2	30	Jaxon	1
Finalize project documentation	2	12	30	Julian	4

1.5 References

[1] “Alfano Shop,” alfano.com, https://www.alfano.com/en/shop/ (accessed Jan. 21, 2024).

[2] “Kart - AIM Technologies,” aimtechnologies.com, https://www.aimtechnologies.com/kart/ (accessed Jan. 21, 2024).

1.6 Revision Table

TABLE IV
Section 1 Revision Table

5/10/2024	Julian Henry: Updated task list for final submission
3/8/2024	Team: Updated task list
1/21/2024	Julian Henry: Revised section 1 using instructor feedback
12/4/2023	Julian Henry: Revised section 1 using feedback from the WK8 Project Document assignment
11/05/2023	Julian Henry, Alex Gibson: Revised section 1 using feedback from the first submission
10/22/2023	Julian Henry: Proofread document
10/20/2023	Jaxon Young: Drafted executive summary and section 1.2 Alex Gibson: Drafted project overview
10/19/2023	Julian Henry: Drafted sections 1.2.1, 1.2.2, 1.3, and 1.4 Alex Gibson: Drafted section 1.1
10/16/2023	Jaxon Young: Initial Document Creation. Section 1 outline created

2. Impacts and Risks

2.1 Design Impact Statement

2.1.1 Introduction

The personal data acquisition prototype is a modular data recorder used to take measurements such as GPS position and output them to a user through a user interface (UI). The UI will read the data from a database that will also be used to allow the user to record and store data. The goal of the design is to create a user-friendly modular data recorder for the general public. Although the device is not intended to have one specific application, the prototype will be designed so that it can be tested using a go-kart.

The purpose of this section is to report on the impacts that the personal data acquisition prototype could have on the public and the environment. Along with this, since engineers hold an important responsibility to consider the impacts of their designs, this section presents possible solutions for some of these negative impacts. This section does this by describing the impacts on public health, safety, and welfare, the cultural and social impacts related to the people creating the components of the device and the data privacy of those who are using it, the environmental impacts of materials used in electronics when they are not properly disposed of, and the economic impacts that the device could have on the data acquisition market for general applications as well as for go-kart specific applications.

2.1.2 Public Health, Safety, and Welfare Impacts

2.1.2.1 Risks of Battery-Powered Electronics

This device poses a risk to public safety by using batteries in the design. This is an issue because lithium batteries can fail, resulting in chemical and combustion reactions [1]. The main causes for such failures are damage to the batteries and improper charging methods. Some examples of how these batteries can become damaged are through physical damage such as dropping or puncturing the batteries and through environmental damage such as using them in high-temperature environments or charging them in below-freezing temperatures [1]. Along with this, the batteries shall be charged by following the manufacturer’s instructions on charging to avoid issues such as overcharging [1]. To avoid these issues, the product will be designed to avoid damage to the batteries by placing them inside an enclosure to reduce the risk of physical damage. Along with this, a charger in compliance with the manufacturer's instructions shall be included as well as information for the user on how to properly charge the device and what environments it can be used in.

2.1.2.2 Risk of Injury Due to Improper Application Installation

This device will be used to collect data in outdoor applications such as on a go-kart, mountain bike, or snowboard. Since this device will feature multiple modular sensor modules connected by cables to a main module, cables could possibly become tangled in mechanical components, resulting in the user crashing and injuring themselves. Thus, it is important that these cables do not get in the way of the moving components of a go-kart or mountain bike. One way to reduce this risk is to make sure all cable connections are resistant to vibrations. Vibrations can cause connectors to loosen or become damaged [2]. Along with this, instructions will be provided to the user explaining how to properly secure the device for their chosen application.

2.1.3 Cultural and Social Impacts

2.1.3.1 Material Sourcing Considerations

Our design is composed almost entirely of semiconductors and electronic components. Unfortunately, the conditions for tech factory workers are often not the best. Workers may be exposed to toxic chemicals, sustain injuries from machines, work long hours for insufficient pay, be forced to live in substandard housing, and more [3]. To mitigate the negative social and cultural impact of our design, we aim to source components from companies that prioritize their workers' well-being. While identifying the best companies for this can be challenging, a review of global working conditions in 2019 revealed that “the shortest average hours of work per week are found in North America and Europe and Central Asia, particularly in Northern, Southern and Western Europe” [4].
Thus, the project will prioritize the utilization of components sourced from factories in North America, Europe, or Central Asia.

2.1.3.2 Data Privacy and Security Considerations

Our design relies on Wi-Fi transmission to transmit data throughout the system. Inadequate safeguards could result in breaches of data, potentially causing harm to users and reducing trust in the technology. Data transmitted over Wi-Fi without any encryption or password protection is completely open and can be read by third parties as long as they are within range of the signal. According to data from the NTIA Internet Use Survey, “73 percent of Internet-using households in 2019 had significant concerns about online privacy and security risks, and 35 percent said such worries led them to hold back from some online activities” [5]. Thus, to limit the negative social and cultural impact of our design, we will implement a Wi-Fi security protocol. Wi-Fi security protocols such as Wired Equivalent Privacy (WEP), Wi-Fi Protected Access (WPA), Wi-Fi Protected Access 2 (WPA 2), or Wi-Fi Protected Access 3 (WPA 3) would keep the transmitted data hidden and block hackers from our network [6]. This step would help to create a system with which users can confidently engage, knowing that their data is secure and their privacy is protected.

2.1.4 Environmental Impacts

2.1.4.1 E-waste

Our design will incorporate various electronics to ensure proper operation. As the system ages, its electronics will fail and eventually need to be thrown out. This is known as electronic waste (e-waste). E-waste contains many toxic components such as lead, mercury, flame retardants, and lithium [7]. If not properly disposed of, these toxic components can contaminate surrounding groundwater and soil [7]. This can damage the surrounding ecosystem and wildlife. In addition to toxic components, e-waste also contains valuable materials such as gold, copper, silver, iron, and zinc [8]. Salvaging these precious materials can affect the environment both positively and negatively. If these materials are properly recycled, materials may have the chance to be reused in the next design, reducing the amount of e-waste produced overall. However, improper recycling of e-waste can negatively impact the environment. Many developed and developing countries employ improper standards for recycling e-waste. Open-air burning and acid baths are sometimes used to extract valuable materials, creating toxic substances that can leach into the environment [9]. To help combat these issues, our design will focus on increasing the lifespan of our electronic devices in various ways such as ensuring proper storage and use conditions according to manufacturers, implementing robust enclosures when in use, and including extensive documentation so users know how to care for the components. In addition, we will provide information on the proper disposal of e-waste.

2.1.5 Economic Factors

2.1.5.1 Affordability Compared to Current Solutions

The primary economic factor for our system will be the setup required to ensure the proper operation of sensors and other system components. After the initial setup and testing, the system will primarily be composed of readily available components and small custom PCBs. Currently, we are aiming for the cost of the system itself to be around $300. There are currently two main brands that provide data acquisition systems for go-karting: Alfano and AiM. Alfano data acquisition systems cost from 455€ to 855€ (approximately $487 to $915) [10]. AiM Kart Data Logger Systems cost from £470 to £520 (approximately $577 to $639) [11]. Thus, our system would open the market of high-functioning data acquisition devices to people of lower socioeconomic status.

5.2 Impact on Economic Equality in the Go-Kart Racing Community

Our system will provide a more affordable data logger system for go-kart enthusiasts of lower socioeconomic status or more casual / newer go-karters who may not want to burn several hundred dollars on an auxiliary piece of equipment. Additionally, it may also help make go-kart racing more equitable for competitors from different economic backgrounds. The purpose of a go-kart data acquisition system is to provide real-time kart monitoring information to the driver which can make them faster and more consistent [12]. If all other differences are ignored, a racer with a data acquisition system will likely do better in a race than a racer without, due to the quality of information they are able to receive. The high price point of current data acquisition systems means that racers with more capital are more able to reap the benefits of a data acquisition system than racers with a tighter budget. Our lower-range price point would then hold the potential to make go-kart racing more equitable in this way.

2.1.6 Conclusion

Engineers hold a pivotal role in addressing societal needs and desires. As professionals in this field, it is imperative that we carefully consider the impact of our products across various domains, ensuring a comprehensive evaluation while aligning with the ultimate objectives of the project. By acknowledging the potential impacts of our design, we can make the proper adjustments in our design and build process in order to reduce the negative impacts and risk factors. Our areas of impact and possible solutions include

- Lithium Battery Fire/Explosion
    - Follow manufacturers' charging instructions
    - Enclose battery to protect from physical damage or extreme temperature
- Privacy/Security Risk
    - Wi-Fi Security Protocols
- Material Sourcing
    - Sourcing materials from companies that focus on factory worker well-being
    - Readily Available, Affordable
- Electronic Waste
    - Increase lifespan of electronics by properly enclosing to avoid condensation and adverse conditions
    - Provide instructions on proper disposal

2.2 Risks

This section will contain a risk assessment table, TABLE V, listing several risks the team has identified that could impact the project's success. Each risk has a risk ID, a description, category, and probability rating (low, med, high), as well as an impact score (low, med, high) showing how detrimental to our project's success the risk would be if it happened, and an action plan containing steps to reduce the risks' impact.

TABLE V
Risk Assessment Table

Risk ID	**Risk Description**	**Risk Category**	**Risk Probability**	**Risk Impact**	**Action Plan**
R1	Incompatible interface	Technical	Low	High	- Plan system interfaces exhaustively
R2	Team member needs to take time for personal matters unexpectedly	Organizational	Med	Med	- Gather time-sensitive materials team member had - List work team member had to complete and make a plan to delay/adjust deadlines - Redistribute work for expected time of absence. If it is unreasonable, contact course instructor
R3	Battery catches fire	Safety	Low	High	- Make sure to have a fire extinguisher nearby when working with batteries - Use extinguisher to prevent further ignition
R4	Component delays (from Chinese New Year, etc.)	Schedule	Low	Med	- Note estimated shipping time when ordering components - Order components two weeks before they are needed, accounting for the estimated shipping time
R5	Component goes out of stock	Technical	High	Med	- Plan backup components
R6	Go-kart breaks the system	Safety	Med	High	- Carefully plan out how the system will be attached - Maintain up-to-date documentation and notes so system can be rebuilt quickly - Order duplicate components if economically viable
R7	Weather prevents testing	Environmental	Med	Med	- Look at the weather forecast before testing
R8	Issues with third-party APIs (changes in functionality, downtime, etc.)	Technical	Low	High	- Use a well-known API - Keep a backup in mind in case a switch is necessary - Be prepared to implement code at the register level if an API becomes unusable.

R9	Poor time estimation for tasks	Schedule	High	Med	- Double all time estimates - Consult with partner to create realistic time estimates - Create personal deadlines ahead of actual deadline

2.3 File Links

2.3.1 References (IEEE)

[1] OSHA, “Preventing Fire and/or Explosion Injury from Small and Wearable Lithium Battery Powered Devices,” osha.gov. https://www.osha.gov/sites/default/files/publications/shib011819.pdf (accessed Nov. 8, 2023).

[2] “The Importance of Anti-Vibration Features in Industrial Electrical Connectors,” energy5.com. https://energy5.com/the-importance-of-anti-vibration-features-in-industrial-electrical-connectors (accessed Nov. 8, 2023).

[3] M. Blanding, “How China is screwing over its poisoned factory workers,” Wired, https://www.wired.com/2015/04/inside-chinese-factories/ (accessed Nov. 9, 2023).

[4] Eurofound and International Labour Organization, “Working conditions in a global perspective,” Publications Office of the European Union, and International Labour Organization, https://www.ilo.org/wcmsp5/groups/public/---dgreports/---dcomm/---publ/documents/publication/wcms_696174.pdf (accessed Nov. 8, 2023).

[5] M. Cao, “Nearly three-fourths of online households continue to have digital privacy and security concerns,” National Telecommunications and Information Administration, https://www.ntia.gov/blog/2021/nearly-three-fourths-online-households-continue-have-digital-privacy-and-security-concerns (accessed Nov. 9, 2023).

[6] D. Ghimiray, “Wi-Fi Security: WEP vs WPA or WPA2,” Avast https://www.avast.com/c-wep-vs-wpa-or-wpa2 (accessed Nov. 9, 2023).

[7] “E-Waste & its negative effects on the environment,” elytus, https://elytus.com/blog/e-waste-and-its-negative-effects-on-the-environment.html (accessed Nov. 9, 2023).

[8] “Cleaning up electronic waste (e-waste),” epa.gov, https://www.epa.gov/international-cooperation/cleaning-electronic-waste-e-waste (accessed Nov. 9, 2023).

[9] S. Manikandan, D. Inbakandan, C. Valli Nachiyar, S. Karthick Raja Namasivayam, "Towards sustainable metal recovery from e-waste: A mini review," Sustainable Chemistry for the Environment, https://doi.org/10.1016/j.scenv.2023.100001 (accessed Nov. 9, 2023).

[10] “Alfano Shop,” alfano.com, https://www.alfano.com/en/shop/ (accessed Nov. 9, 2023).

[11] “Kart - AIM Technologies,” aimtechnologies.com, https://www.aimtechnologies.com/kart/ (accessed Nov. 9, 2023).

[12] “Karting Data & Timing,” competitionmotorsport.com, https://competitionmotorsport.com/collections/karting-data-timing (accessed Nov. 9, 2023).

2.3.2 File Links

[13] https://drive.google.com/file/d/1RbJMouxU02F9vibak4BKAWVEbUblvC5h/view?usp=sharing
The content for section 2.2 was mostly created during lecture time. This link shows an image of the document created at that point.

2.4 Revision Table

TABLE VI
Section 2 Revision Table

4/21/24	Julian Henry: Updated section 2 with Design Impact Assessment content
11/19/23	Revised the introduction according to feedback from Rachel Cate, Lane Hartless, and Neha Suryadevara Revised section 3.1 according to feedback from Neha Suryadevara Revised section 4.1 according to feedback from Cameron Hicks and Neha Suryadevara Changed instances of “could” and “should” to “shall” or “will” according to feedback from Cameron Hicks Revised the conclusion according to feedback from Neha Suryadevara
11/19/2023	Julian Henry, Alex Gibson: Revised section 2 using feedback from the draft submission
11/12/23	Reformatted the references section according to feedback from Rachel Cate and Frank Wong
11/09/2023	Julian Henry: Section 2 updated with file links section and minor updates
11/05/2023	Julian Henry: Section 2 outline created, section 2.2 drafted

3. Top-Level Architecture

3.1 Block Diagram

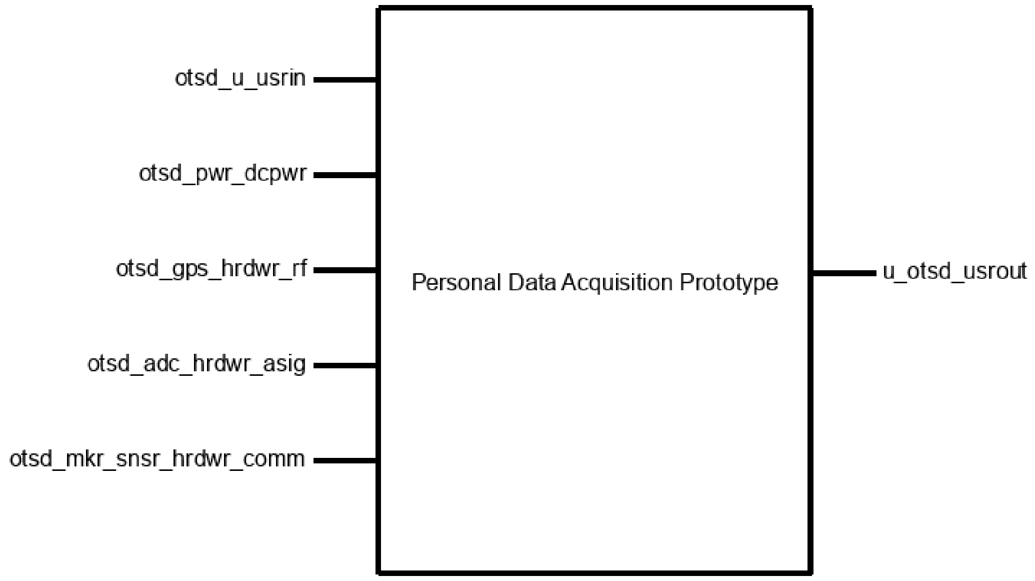


Fig. 2. Black Box Image

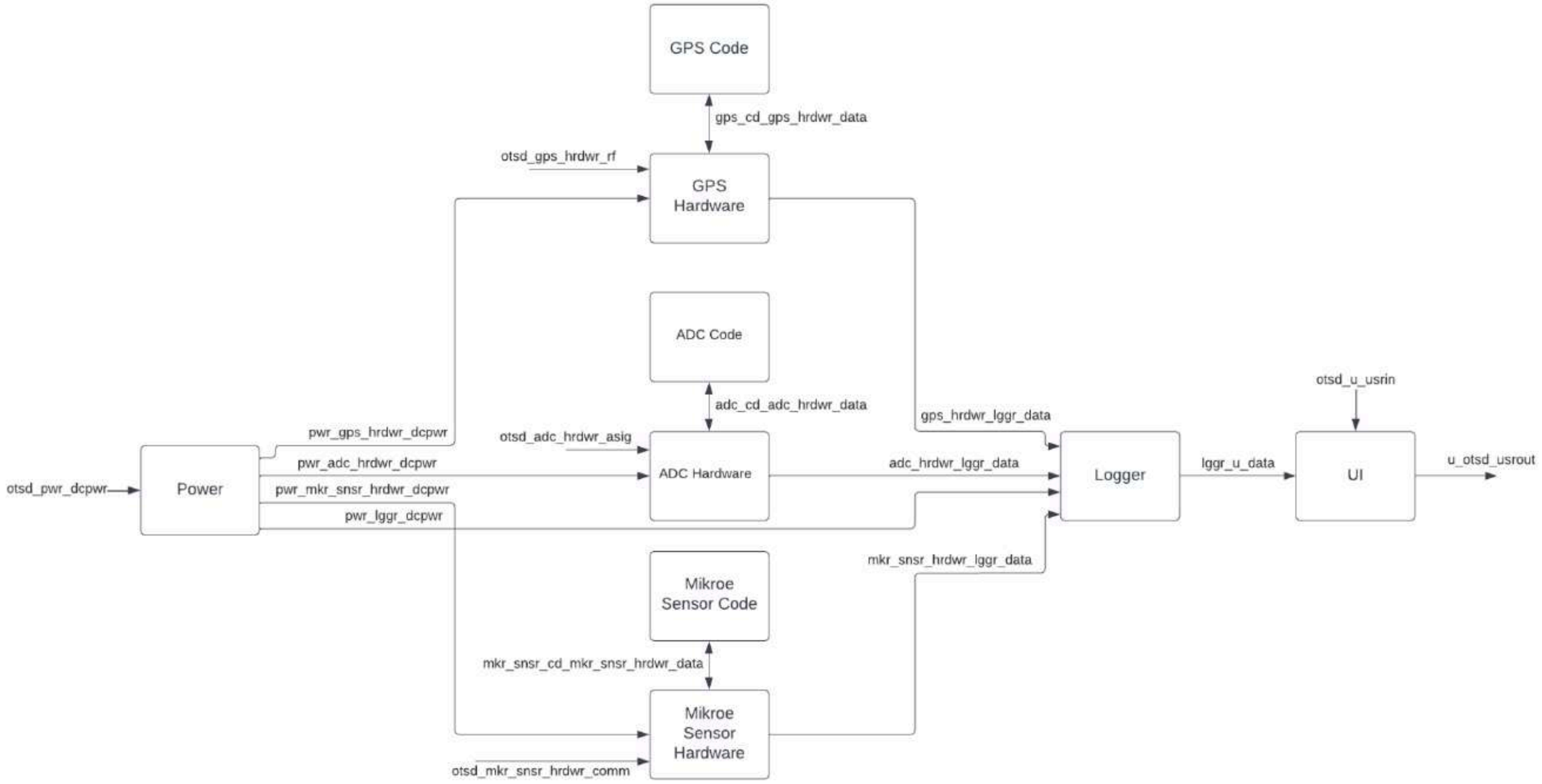


Fig. 3. System Block Diagram [9]

3.2 Block Descriptions

This section gives a short description of what each block from figure 3 in section 3.1 accomplishes for the system and what it contains.

3.2.1 ADC Hardware

The ADC Hardware Block is used by the system to measure analog voltage values. It satisfies the system's modularity requirement due to it connecting to the logger module through an RS45 cable, the precision requirement by taking 24-bit data samples, and the sample rate requirement by including an ADC that can sample at a rate significantly higher than 100Hz. The block features an ADC, a microcontroller, a programmer connection, a linear regulator, a CAN bus transceiver, and interface connectors. The ADC will be able to read analog voltage values from a strain gauge that connects to the block. These voltage values can then be sent to the microcontroller through SPI communication protocol. The microcontroller can then send these values to the logger block using CAN bus communication.

3.2.2 ADC Code

The ADC Code block is used by the system to provide the microcontroller code necessary for the ADC hardware block to function correctly. The code works by first configuring the ADC with the correct settings before taking analog voltage measurements at a rate of 100Hz. After each measurement, it sends the data over the CAN bus interface which is then received by the logger module. The code is uploaded to the microcontroller using a Serial Wire Debug interface (SWD) provided by an ST-Link programmer device.

3.2.3 GPS Hardware

The GPS Block is used by the system to measure position values. It satisfies the system's modularity requirement due to it connecting to the logger module through an RS45 cable, the precision requirement by taking 8-bit position measurements, and the sample rate requirement by including a GPS that can sample at a rate of at least 5Hz. The block features a NEO-M9N GPS, a microcontroller, a programmer connection, a linear regulator, a CAN bus transceiver, and interface connectors. The GPS will be able to read RF signals from satellites that give the position and transmission time of the satellite. This information is then used to calculate the position of the GPS which is then sent to the microcontroller through UART communication protocol. The microcontroller can then send these values to the logger block using CAN bus communication.

3.2.4 GPS Code

The GPS Code block is used by the system to provide the microcontroller code necessary for the GPS hardware block to function correctly. The code works by first configuring the GPS with the correct settings before taking position values at a rate of 5Hz. After each measurement, it sends the data over the CAN bus interface which is then received by the logger module. The code is uploaded to the microcontroller using a Serial Wire Debug interface (SWD) provided by an ST-Link programmer device.

3.2.5 Mikroe Sensor Hardware

The Mikroe Sensor Hardware Block features a MikroBUS(TM) connection interface, a microcontroller, a programming interface, a voltage regulator, and a CAN transceiver. This block will allow the system to use any Mikroe Click sensor board with minimal configuration. This will increase the functionality of the system while not requiring the design and assembly of additional custom sensor boards.

The MikroBUS(TM) interface will be used to connect any Mikroe Click sensor board to our system. This will greatly increase the potential functionality of our system by enabling it to use any of the 525 current Mikroe click sensor boards available on Mikroe's website [3]. The firmware for the connected Click Sensor board can be sent to the microcontroller using the programming port. The connected Mikroe Click sensor board can then send data to the microcontroller, which works to update the logger block with the sensor data using CAN bus communication through the CAN transceiver.

3.2.6 Mikroe Sensor Code

The Mikroe Sensor Code block is used by the system to provide the microcontroller code necessary for the Mikroe Sensor hardware block to function correctly. The code works by first configuring the Mikroe Sensor with the correct settings before taking measurements at a rate of 100Hz. After each measurement, it sends the data over the CAN bus interface which is then received by the logger module. The code is uploaded to the microcontroller using a Serial Wire Debug interface (SWD) provided by an ST-Link programmer device.

3.2.7 Logger

The logger will collect data from the sensors and store them into a database. In this application, a Raspberry PI 4B will be used as the logger device. The Raspberry PI 4B will be powered ideally at 5V however, can be operated at a minimum voltage of 4.8V and a maximum of 5.2V. An RS485 CAN Hat will interface with the desired sensor modules using CAN communication. Using low-speed CAN mode, the logger will record incoming sensor data at a maximum rate of 100Kbit/sec. The logger will take in incoming sensor messages ranging from 8 bits to 24 bits depending on the sensor module. Incoming GPS and Mikroe sensor messages will be in the size of 8-bits while the ADC sensor will send 24-bit messages. A local database located on the pi will store the incoming sensor data using SQLite, making a query after every new message sent. The logger will continue to read and store sensor data until reaching a specified time. Although most of the system will be programmed in rust, the PI will be programmed in python.

3.2.8 UI

The UI module will help the user interface with the logger and sensors of the system. Programmed in rust using the EGUI library, the UI will fetch data from the logger module's database and manipulate data based on the user’s preference. The user will be able to select which sensors they would like to view/manipulate the data of and specify the amount of time they would like the logger to record data for. During data collection, the user will have two options on how they would like to view the incoming recorded data. An option to output data into a table or graph will be given. The user can also specify the period of time they would like to view of the data.

3.2.9 Power

This block will provide power to the entire system including the GPS block, the ADC block, the Mikroe sensor block, and the logger block. The power block will be rechargeable and able to support the system with all blocks using their peak current for an hour on one charge. This means that it must have a charge capacity greater than the sum of all the peak currents required by every block multiplied by one hour. According to the block interfaces, each of the three sensor blocks has a peak input current of 170mA, and the logger has a peak of 1.75A. This gives: (170mA + 170mA + 170mA + 1.75A)*(1 hour) = 2.26A*1h = 2.26 Ah, or 2260 mAh. To accomplish this, the power block features a commercially available portable charger.

3.3 Interface Definitions

TABLE VII
Block interface table

Name	Properties
otsd_u_usrin	• **Type:** three interactable checkboxes lined up 1x3 to record member variable • **Type:** interactable "slider" bar to record integer • **Usability:** understandable by 9/10 users
otsd_pwr_dcpwr	• **Inominal:** 1A • **Ipeak:** 1.75A • **Vmax:** 5.25V • **Vmin:** 4.75V
otsd_gps_hrdwr_rf	• **Datarate:** 50bits/sec • **Messages:** messages contain the position and time of the GPS satellites • **Protocol:** NMEA-0183
otsd_adc_hrdwr_asig	• **Other:** 24 bit signal measurement resolution • **Vmax:** 3.3V • **Vrange:** 0-3.3V
otsd_mkr_snsr_hrdwr_comm	• **Datarate:** 100kbps or more • **Messages:** Data from the Mikroe Click sensor board that is connected • **Other:** MikroBUS-style Socket
u_otsd_usrout	• **Other:** UI updates at least every 30 seconds • **Type:** Graph of data vs time • **Usability:** Human-readable numeric output used to give specific values
pwr_gps_hrdwr_dcpwr	• **Inominal:** 33mA • **Ipeak:** 170mA • **Vmax:** 5.2V • **Vmin:** 4.8V

pwr_adc_hrdwr_dcpwr	● **Inominal:** 33mA ● **Ipeak:** 170mA ● **Vmax:** 5.2V ● **Vmin:** 4.8V
pwr_lggr_dcpwr	● **Inominal:** 1A ● **Ipeak:** 1.25A ● **Vmax:** 5.2V ● **Vmin:** 4.8V
pwr_mkr_snsr_hrdwr_dcpwr	● **Inominal:** 33mA ● **Ipeak:** 170mA ● **Vmax:** 5.2V ● **Vmin:** 4.8V
gps_hrdwr_lggr_data	● **Datarate:** 100Kbit/sec CAN Bus speed ● **Messages:** 8 bits ● **Protocol:** CAN bus
adc_hrdwr_lggr_data	● **Datarate:** 100Kbit/sec CAN bus speed ● **Messages:** 24 bits ● **Protocol:** CAN bus
lggr_u_data	● **Messages:** data from all sensor modules ● **Messages:** float type ● **Other:** SQLite Query
mkr_snsr_cd_mkr_snsr_hrdwr_data	● **Messages:** Binary code file ● **Other:** Pin header connection ● **Protocol:** Serial Wire Debug
mkr_snsr_hrdwr_lggr_data	● **Datarate:** 100Kbit/sec CAN Bus speed ● **Messages:** 8 bits ● **Protocol:** CAN bus
adc_cd_adc_hrdwr_data	● **Messages:** Binary code file ● **Other:** Pin header connection ● **Protocol:** Serial Wire Debug
gps_cd_gps_hrdwr_data	● **Messages:** Binary code file ● **Other:** Pin header connection

	● **Protocol:** Serial Wire Debug

3.4 References and File Links

3.4.1 References (IEEE)

[1] Analog Devices, “AD7124-8 (Rev. F),” analog.com, https://www.analog.com/media/en/technical-documentation/data-sheets/ad7124-8.pdf, (accessed Nov. 28, 2023).

[2] u-blox, “NEO-M9N Integration Manual,” u-bloxcom, https://content.u-blox.com/sites/default/files/NEO-M9N_Integrationmanual_UBX-19014286.pdf, (accessed Nov. 28, 2023).

[3] “Sensors: Click boards,” mikroe.com, https://www.mikroe.com/click/sensors (accessed Dec. 3, 2023).

[4] “Portable Charger Info Page,” Amazon, https://www.amazon.com/gp/product/B07PG6C5C7 (accessed Dec. 4, 2023).

[5] “Standard specifications - MIKROE,” MikroBus Standard specifications, https://download.mikroe.com/documents/standards/mikrobus/mikrobus-standard-specification-v200.pdf (accessed Dec. 5, 2023).

[6] “DATASHEET - Raspberry Pi,” Raspberry Pi 4 Model B Datasheet, https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf (accessed Dec. 5, 2023).

[7] T. Thurium, “Three easy ways to brick a Raspberry Pi,” Twilio Blog, https://www.twilio.com/blog/3-ways-brick-raspberry-pi (accessed Dec. 4, 2023).

[8] KVASER, “The CAN Bus Protocol Tutorial,” kvaser.com, https://www.kvaser.com/can-protocol-tutorial/, (accessed Nov. 28, 2023).

3.4.2 File Links

[9] https://drive.google.com/file/d/1paBB9K00yNNS22RuKPmCooDzpVnGpm-5/view?usp=sharing
This links to a copy of the original document used to create figure 3.

3.5 Revision Table

TABLE VI
Section 3 Revision Table

4/21/2024	Julian Henry: Updated section with descriptions from portal
3/10/2024	Team: updated interface definitions
3/8/2024	Team: Updated block diagrams and descriptions

12/4/2023	Julian Henry: Filled out section 3.3 from project portal, updated several references Alex Gibson: Added references to section 3.3
12/3/2023	Alex Gibson: Added sources for 3.2.1 and 3.2.2
11/30/2023	Julian Henry: Section 3.1 filled in
11/29/2023	Julian Henry: Section 3 outline created

4. Block Validations

This section starts with the Block Validations that your team members have completed. There needs to be a section for each block in your project.

4.1 ADC Hardware

4.1.1 Description

The ADC Hardware Block is used by the system to measure analog voltage values. It satisfies the system's modularity requirement due to it connecting to the logger module through an RS45 cable, the precision requirement by taking 24-bit data samples, and the sample rate requirement by including an ADC that can sample at a rate significantly higher than 100Hz. The block features an ADC, a microcontroller, a programmer connection, a linear regulator, a CAN bus transceiver, and interface connectors. The ADC will be able to read analog voltage values from a strain gauge that connects to the block. These voltage values can then be sent to the microcontroller through SPI communication protocol. The microcontroller can then send these values to the logger block using CAN bus communication.

4.1.2 Design

This section describes the design of the ADC sensor hardware block. It includes a black box diagram showing the interfaces of the block. These interfaces are the voltages connected to the ADC, the DC power used to power the module, the programmer connection used to upload code to the microcontroller, and the CAN bus communication line used to communicate with the logger module. This section also includes schematic diagrams for the hardware design. These diagrams cover the microcontroller used to interface with the ADC and CAN bus, linear regulator used to convert the 4.8-5.2V power input to a 3.3V output, CAN transceiver used to convert the digital data signal to an analog differential pair signal, the ADC used to convert external analog voltages to 24-bit digital values, and the connectors used for all of the interfaces.

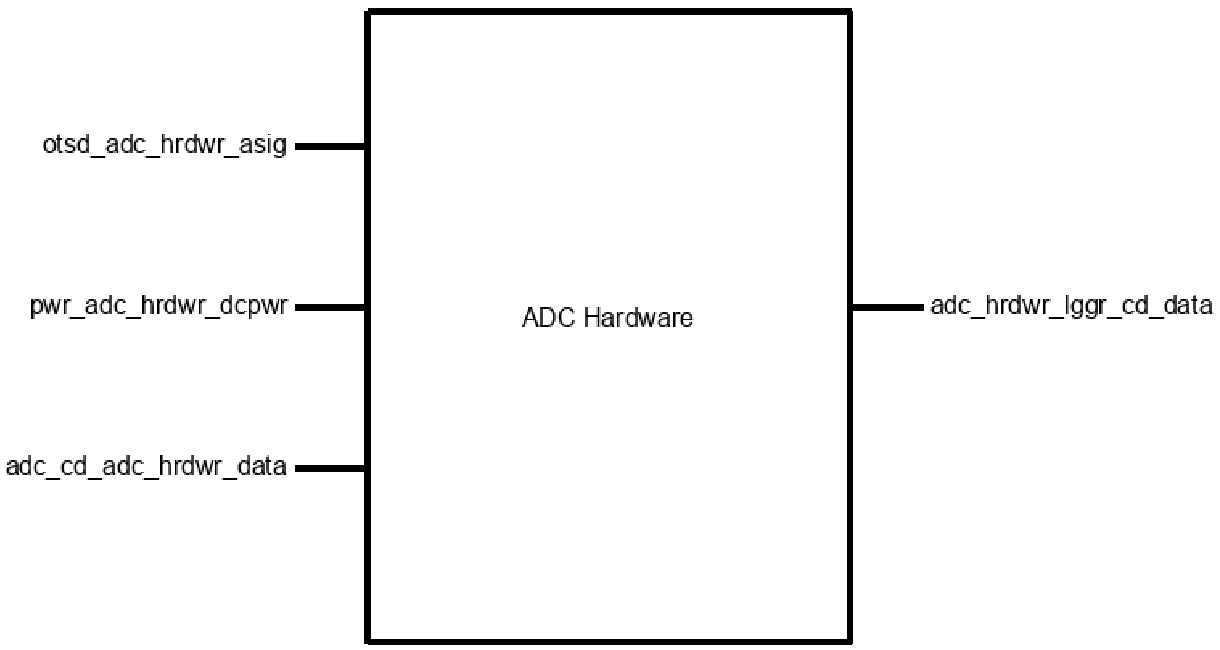


Fig. 1. Black box image of the ADC Hardware block

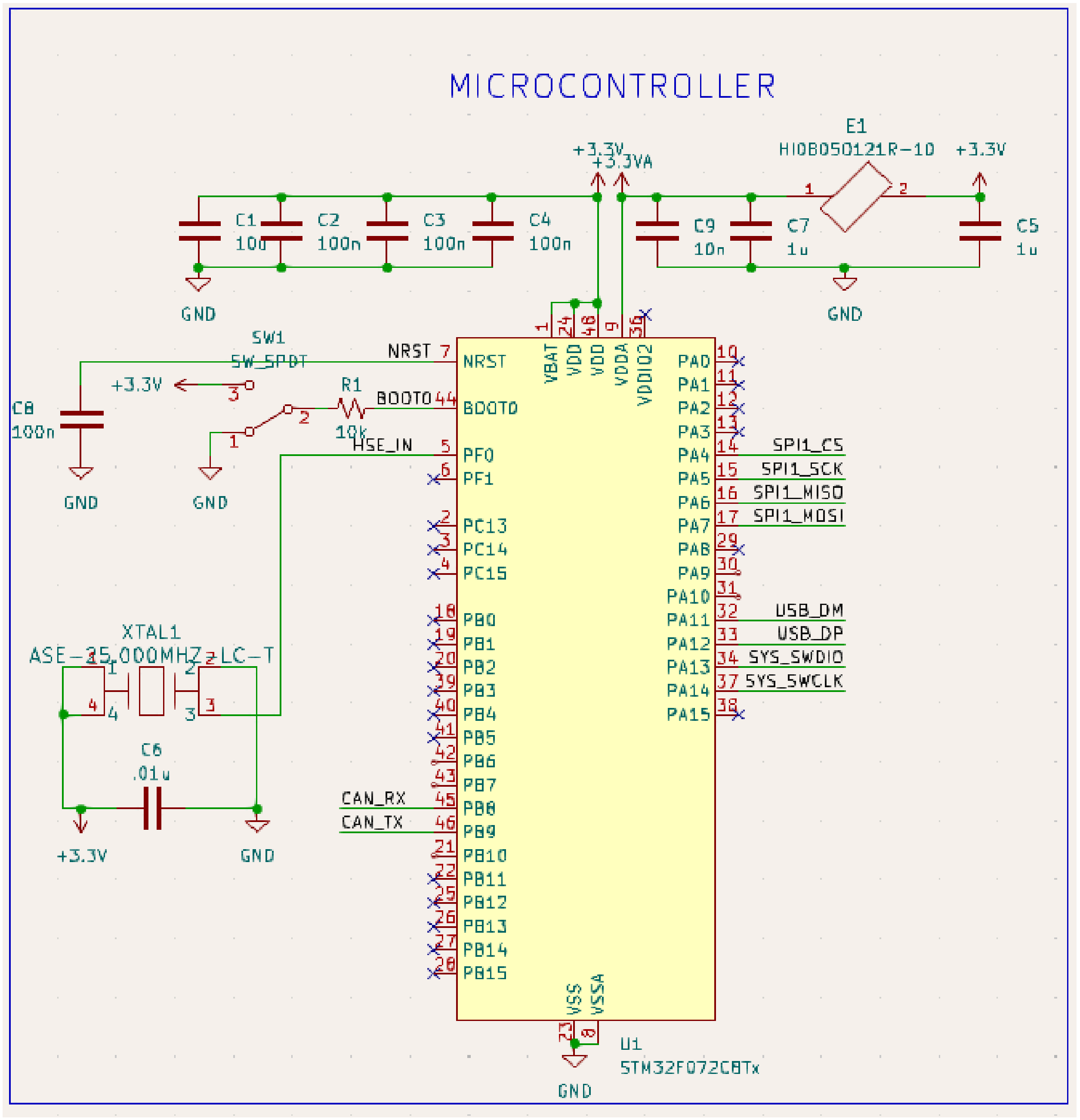


Fig. 2. ADC Hardware Block Microcontroller Schematic

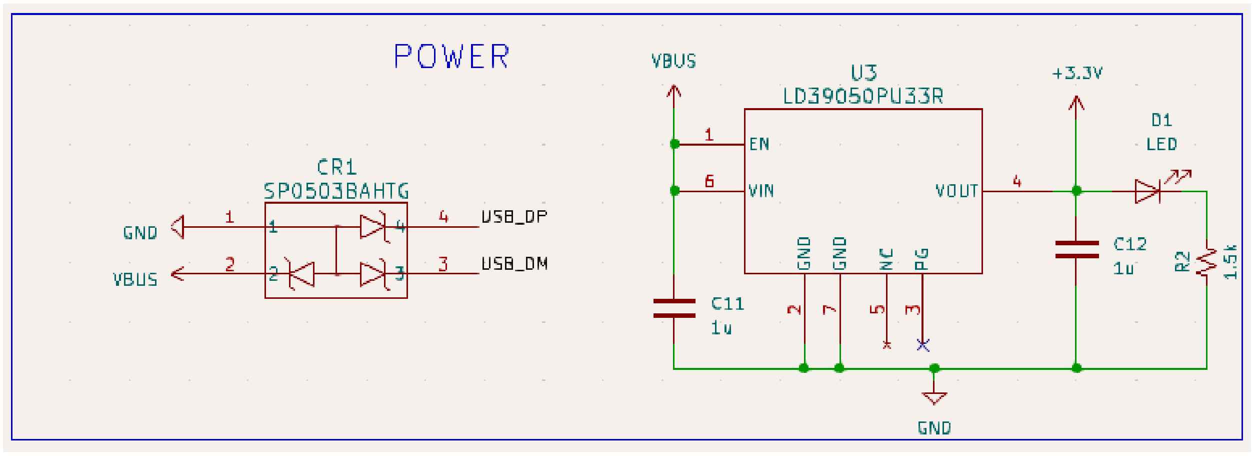


Fig. 3. ADC Hardware Block Power Schematic

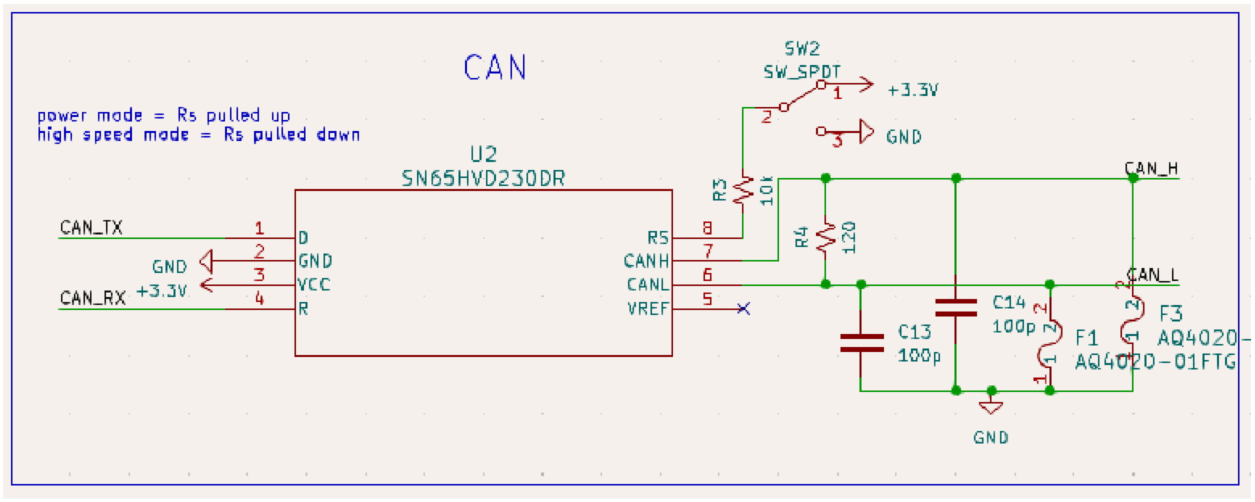


Fig. 4. ADC Hardware Block CAN Transceiver Schematic

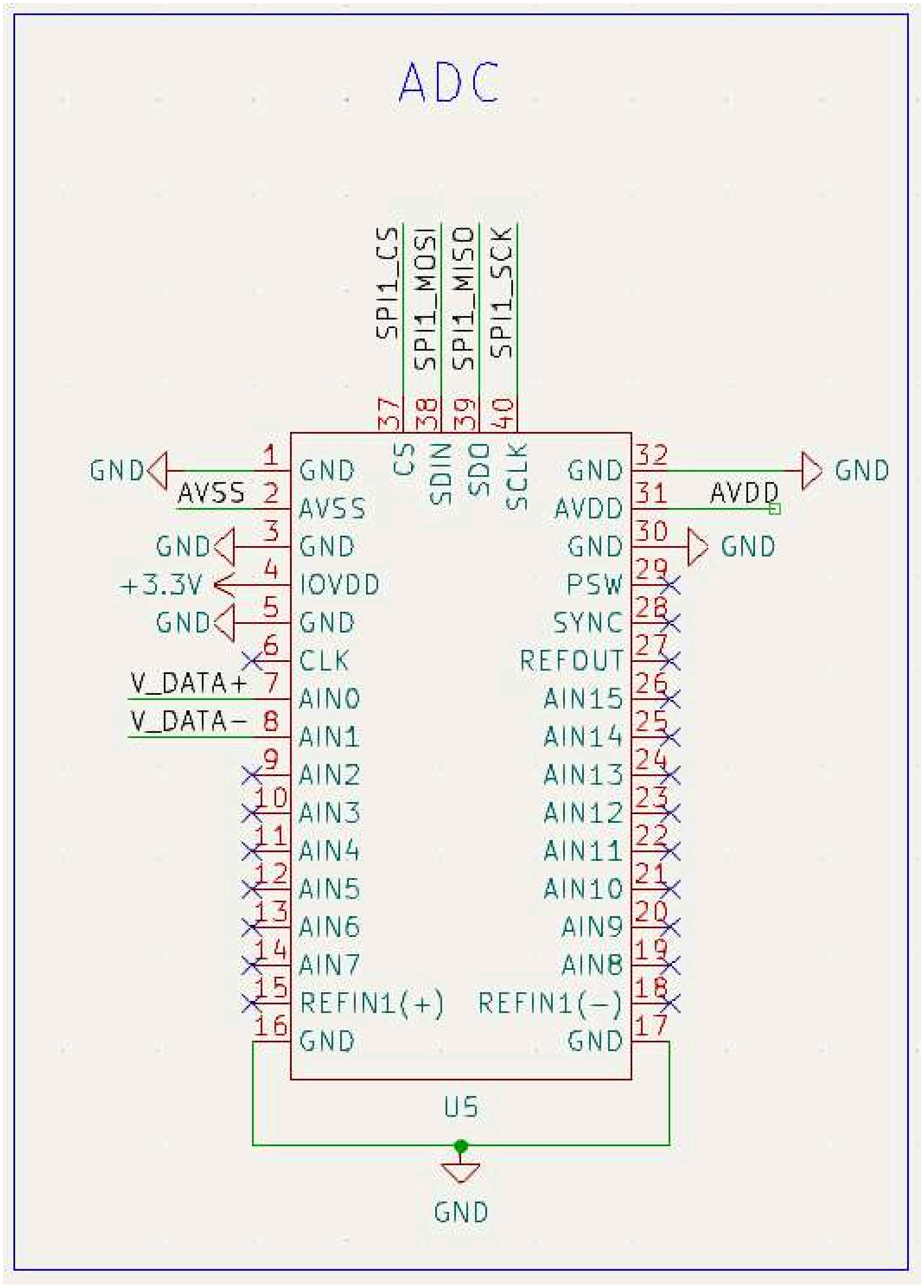


Fig. 5. ADC Hardware Block ADC Daughter Board

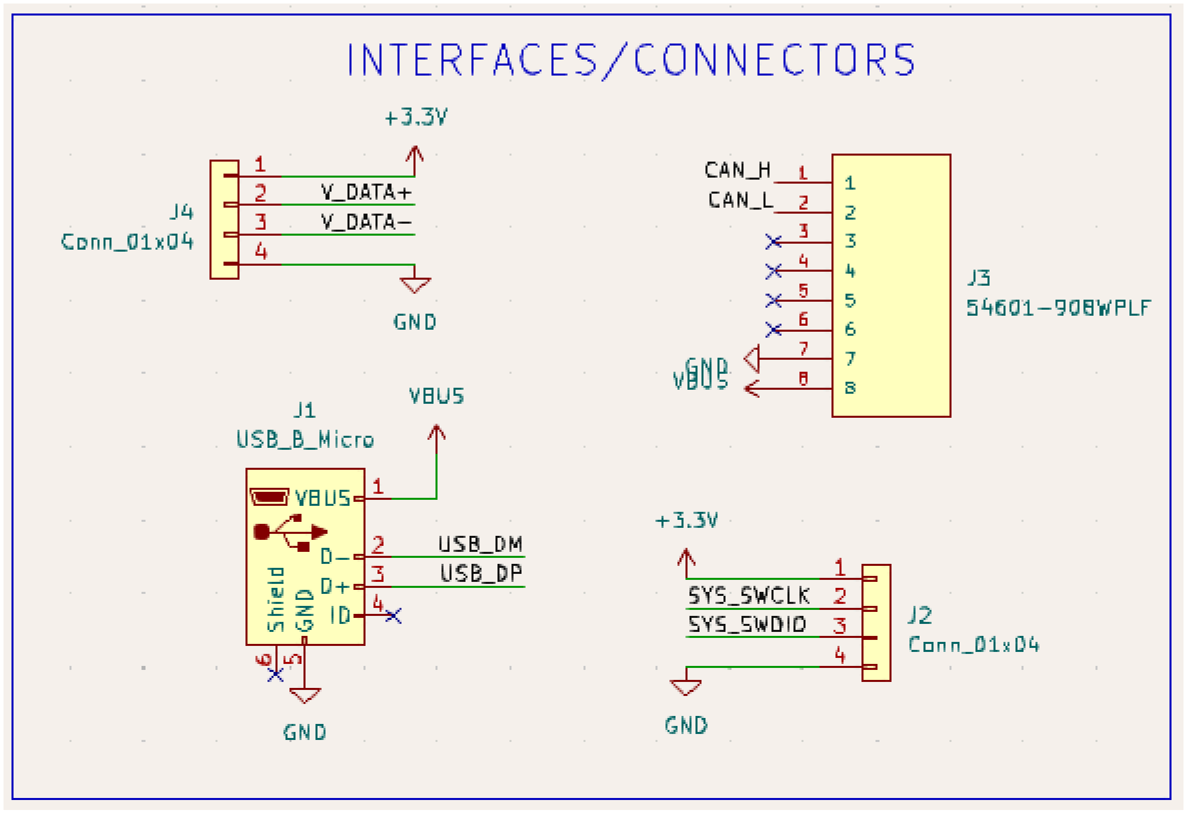


Fig. 6. ADC Hardware Block Interfaces / Connectors Schematic

4.1.3 General Validation

The ADC Hardware Block needs to be able to sample analog voltage values between 0 and 3.3 volts with a minimum resolution of 24-bits per sample and output that data through CAN bus protocol to the logger block. Along with this, due to the sampling rate requirement for the system, the ADC must take these measurements at a minimum sampling rate of 100Hz. Lastly, the ADC block must also be able to do these operations with a voltage input between 4.8 and 5.2 volts.

The Microcontroller schematic shown in figure 2 features an STM32F072C8T6TR microcontroller. This specific microcontroller was chosen due to it having the necessary peripheral support for all of the devices used in all of the sensor blocks for the entire system. These peripherals are I2C, SPI, UART, CAN, USB, and pulse width modulation. This means that the same microcontroller can be integrated into each of the 3 sensor hardware designs. Along with this, the microcontroller was also chosen due to it being low cost and featuring an IC package with a low amount of pins that are easily solderable. The microcontroller circuit design features a 3.3 volt connection provided by the LD39050PU33R which is used to power the microcontroller. Each power input pin is connected to a 100nF decoupling capacitor to filter out noise as recommended by the hardware development document for the microcontroller [1]. A ferrite bead is used to isolate noise from the 3.3V source for the VDDA pin. This design for isolating VDDA from the other power input pins was found in a STM32 board design video guide [2]. The reset pin (NRST) is connected to a 100n capacitor to prevent the microcontroller from resetting due to noise. A 25 MHz crystal oscillator is used as the clock source for the microcontroller. According to the hardware development document, a 4 to 32 MHz external oscillator should be used for the design [1]. A decoupling capacitor is connected between the 3.3V and GND pin on the oscillator to filter noise. The tri-state pin on the oscillator is connected to 3.3V in order to enable the oscillator’s output [3]. The boot mode pin was connected to a switch so that it could be pulled high or low for either boot option. Lastly, pins PB8 and PB9 are used for the can peripheral, pins PA4 - PA7 are used for the SPI peripheral, PA11 and PA12 are used for the USB peripheral, and PA13 and PA14 are used for the programmer interface [4].

The power schematic shown in figure 3 features an LD39050PU33R linear regulator and an SP0503BAHTG TVS diode block used to protect the USB connection against electrostatic discharge. The power for the system works where a 5.0V input called VBUS is connected to the input pin of the linear regulator which then outputs 3.3V. VBUS is provided by either a micro-USB connection or a RJ45 connection. The application information for the linear regulator recommends connecting 1uF capacitors to the Vin and Vout inputs [5]. Along with this, Vin is also connected to the enable pin in order to enable the output of the IC. Lastly, an LED is attached to Vout to give a visual indication that power is being output by the linear regulator.

The CAN transceiver schematic shown in figure 4 features an SN65HVD230DR transceiver used to convert digital CAN data from the microcontroller to a differential pair data output. As specified in the datasheet for the transceiver, data being sent from the microcontroller is connected to the D pin and data being received is connected to the R pin. The differential pair data is output on pins CANH and CANL which are connected by a 120 ohm termination resistor to improve signal quality [6] [7]. According to the device datasheet, the RS pin should be pulled high for power mode or pulled low for high speed mode [7]. In the schematic, a switch is used so that either mode can be used. Lastly, capacitors and TVS diodes are used on the CANH and CANL connections to reduce noise and protect against large surges in voltage.

The ADC schematic shown in figure 5 features an EVAL-AD7124-8-PMDZ board which is a breakout board for the AD7124-8 ADC. This board will be used as a daughter board to the ADC hardware by connecting it to the PCB for the ADC hardware circuit. The SPI connection from the microcontroller is used to communicate with the ADC [8]. The ADC is powered by the same 3.3V source as the microcontroller, and this voltage is used to power the analog output pins AVDD and AVSS which are used as the reference voltage for the device [9]. Lastly, the ADC uses pins AIN0 and AIN1 as the positive and negative inputs for the measured voltage.

The interface/connectors schematic shown in figure 6 features a micro-USB connector used for data transfer and powering the linear regulator, a RJ45 connector used to transfer the CAN data and provide power when connected to the logger module, a strain gauge connector for the analog voltage input measured by the ADC, and pin headers used for a JTAG programming connection.

While this circuit is designed with the intention of using CAN bus protocol in power mode, an alternate solution using high speed mode is possible by using the switch to connect the RS pin on the transceiver to GND and desoldering the 120 ohm resistor. Another alternate solution for this design would be to connect LEDs to some of the extra GPIO pins for debugging purposes. Lastly, an alternative solution for the ADC would be to use a different reference voltage from the 3.3V powering the IC in order to increase the voltage range that the ADC can measure. Along with this, more AIN pins on the ADC could be used to measure multiple analog voltage inputs at once.

4.1.4 Interface Validation

Table I
ADC Hardware Block interface validation table

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**
otsd_adc_hrdwr_asig : Input		
Other: 24 bit signal measurement resolution	The AD7124-8 was chosen as the ADC to be used on the board. This ADC has a 24-bit resolution on its measurements [8].	The general description section on page 4 of the datasheet for the AD7124-8 states that “the device contains a low noise, 24-bit Σ-Δ analog-to-digital converter (ADC)” [8].
Vmax: 3.3V	The ADC will have its analog reference voltage (AVDD) set to the 3.3V output from the linear regulator.	Linear regulator will output 3.3V when powered by 5.0V [5].
Vrange: 0-3.3V	The ADC will have AVDD set to the 3.3V output from the microcontroller.	Analog voltage range is 0V to AVDD for bipolar mode as described on page 48 of the datsheet [8].
pwr_adc_hrdwr_dcpwr : Input		
Inominal: 33mA	Nominal current of microcontroller with a system clock of 60MHz.	The microcontroller will typically draw 26 mA at 25 ℃ and 39 mA at 85 ℃ [4]. Since the ADC has a significantly lower current draw, it can be estimated that the nominal current will be between 25 and 39 mA [8].
Ipeak: 170mA	Peak power input current of the microcontroller and ADC chip being used [4] [8].	Microcontroller and ADC will draw at most 170mA [4] [8]. Since the microcontroller and ADC are not being fully utilized, the current draw will be less than this maximum value.

Vmax: 5.2V	Ideal input voltage is 5.0V so this value allows for some variance without going too far out of range.	Max operating input voltage for the linear regulator is 5.5V as described on page 6 of the datasheet, meaning 5.2V is in range [5].
Vmin: 4.8V	Ideal input voltage is 5.0V so this value allows for some variance without going too far out of range.	Since the linear regulator is described on page 6 of the datasheet as outputting a value of Vin - 0.3 with a minimum Vin value of 1.5V, a value of 4.8 will satisfy both conditions above to output 3.3V [5].
adc_hrdwr_lggr_cd_data : Output		
Datarate: 100Kbit/sec CAN bus speed	This speed was chosen because the design will be using a CAN bus connection in low-speed mode to improve signal quality. CAN bus configured for low-speed mode has a maximum data rate of 125 Kbit/sec [6].	According to page 5 of the SN65HVD23 transceiver datasheet, low-speed mode is configured by pulling RS to 3.3V [7].
Messages: 24 bits of data	A CAN bus transmission will occur after each ADC sample.	ADC samples are 24-bits of data [8].
Protocol: CAN bus	CAN bus was chosen as the protocol to communicate between the logger block and the ADC block because of its application in automotive systems and because our project partner is familiar with the protocol.	The chosen microcontroller supports CAN bus peripherals on the PB8 and PB9 pins [4]. The CAN_RX and CAN_TX signals from these two pins are connected to the CAN transceiver which converts the digital values sent and received by these pins to differential pair signals [7].
adc_cd_adc_hrdwr_data : Bi-directional		
Messages: Binary code file	Code must be compiled down to machine code to be run on the microcontroller.	Binary instructions stored in flash memory are used to control the microcontroller [10].
Other: Pin header connection	Pin headers were chosen for an easy connection method using an ST-Link programmer.	St-Link connector uses pin connections to program the microcontroller [11].
Protocol: Serial Wire Debug	SWD is used by the ST-Link programmer.	SWD sends a data stream over two wires to program ARM CPUs such as STM32 microcontrollers [12].

4.1.5 Verification Process

1. Connect jumper wires between the adc_cd_adc_hrdwr_data interface and the ST-Link programmer connected to a computer, then program the microcontroller with the ADC block testbench firmware.
2. Disconnect the programmer and connect the ADC block's adc_hrdwr_lggr_cd_data CAN bus connection to the adc_hrdwr_lggr_cd_data CAN bus connection on the raspberry-pi.
3. Connect the ADC block's pwr_adc_hrdwr_dcpwr input to a 4.8V power source.
4. Run the ADC block testbench python script on the raspberry pi.
5. Connect the otsd_adc_hrdwr_asig input to GND and confirm that the raspberry-pi is receiving a 3 byte sequence of 0x00 0x00 0x00 on the adc_hrdwr_lggr_cd_data interface.
6. Connect the otsd_adc_hrdwr_asig input to 3.3V and confirm that the raspberry-pi is receiving a 3 byte sequence of approximately 0xFF 0xFF 0xFF on the adc_hrdwr_lggr_cd_data interface.
7. Confirm in the code of both the ADC microcontroller and the raspberry-pi that both are configuring the CAN bus protocol to run at 100Kb/sec.
8. Change the pwr_adc_dcpwr input to 5.2V and repeat steps 5-7.

4.1.6 References and File Links

[1] STMicroelectronics, "Getting started with STM32F0x1/x2/x8 hardware development," st.com, https://www.st.com/resource/en/application_note/an4080-getting-started-with-stm32f0x1x2x8-hardware-development-stmicroelectronics.pdf, (accessed Mar 3, 2024).

[2] Phil's Lab, "KiCad 6 STM32 PCB Design Full Tutorial - Phil's Lab #65," youtube.com, https://www.youtube.com/watch?v=aVUqaB0IMh4&ab_channel=Phil%E2%80%99sLab, (accessed Mar. 6, 2024).

[3] Abracon, "3.3V CMOS Compatible 3.2 x 2.5mm SMD Crystal Oscillator," abracon.com, https://abracon.com/Oscillators/ASEseries.pdf, (accessed Mar. 6, 2024).

[4] STMicroelectronics, "STM32F072x8 STM32F072xB," st.com, https://www.st.com/content/ccc/resource/technical/document/datasheet/cd/46/43/83/22/d3/40/c8/DM00090510.pdf/files/DM00090510.pdf/jcr:content/translations/en.DM00090510.pdf, (accessed Dec. 6, 2023).

[5] STMicroelectronics, "LD39050," st.com, https://www.st.com/content/ccc/resource/technical/document/datasheet/12/43/cc/93/21/f6/41/66/CD00227527.pdf/files/CD00227527.pdf/jcr:content/translations/en.CD00227527.pdf, (accessed Jan. 27, 2024).

[6] KVASER, “The CAN Bus Protocol Tutorial,” kvaser.com, https://www.kvaser.com/can-protocol-tutorial/, (accessed Nov. 28, 2023).

[7] Texas Instruments, “SN65HVD230DR,” ti.com, https://www.ti.com/lit/ds/symlink/sn65hvd230.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-wwe&ts=1709546681296&ref_url=https%253A%252F%252Fwww.google.com%252F, (accessed Mar 4, 2024).

[8] Analog Devices, “AD7124-8 (Rev. F),” analog.com, https://www.analog.com/media/en/technical-documentation/data-sheets/ad7124-8.pdf, (accessed Nov. 28, 2023).

[9] Analog Devices, “EVAL-AD7124-8-PMDZ Overview,” wiki.analog.com, https://wiki.analog.com/resources/eval/user-guides/circuits-from-the-lab/eval-ad7124-8-pmdz, (accessed Mar. 6, 2024).

[10] STMicroelectronics, “STM32F0 series Cortex-M0 programming manual,” st.com, https://www.st.com/resource/en/programming_manual/pm0215-stm32f0-series-cortexm0-programming-manual-stmicroelectronics.pdf, (accessed Mar 4, 2024).

[11] Adafruit, “ST-Link STM8/STM32 v2 Compatible Programmer & Emulator,” adafruit.com, https://www.adafruit.com/product/2548, (accessed Mar 4, 2024).

[12] Silicon Labs, “Serial Wire Debug (SWD), https://community.silabs.com/s/article/serial-wire-debug-swd-x?language=en_US, community.silabs.com, (accessed Mar 4, 2024).

4.1.7 Revision Table

TABLE II
Subsection 4.1 Revision Table

3/6/2024	Alexander Gibson: Updated section 4.1.1 with a statement about how the block contributes to the overall system, provided schematic diagrams in section 4.1.2, rewrote section 4.1.3 to discuss the new schematics, updated section 4.1.4 by adding a table label, including new interfaces and their properties, and fixing Vmin and Vmax values due to them being incorrectly switched, updated section 4.1.5 to prove all interfaces of the block, and updated section 4.1.6 with more datasheet links.
1/24/2023	Alexander Gibson: Added intro paragraph to design section, updated interface names and values to match student portal, and modified interface validation to match changes in design choices.
12/6/2023	Alexander Gibson: Filled out sections 4.1.1 through 4.1.7

4.2 ADC Code

4.2.1 Description

The ADC Code block is used by the system to provide the microcontroller code necessary for the ADC hardware block to function correctly. The code works by first configuring the ADC with the correct settings before taking analog voltage measurements at a rate of 100Hz. After each measurement, it sends the data over the CAN bus interface which is then received by the logger module. The code is uploaded to the microcontroller using a Serial Wire Debug interface (SWD) provided by an ST-Link programmer device.

4.2.2 Design

This section provides a black box diagram and pseudocode to describe the functionality of the ADC Code block.

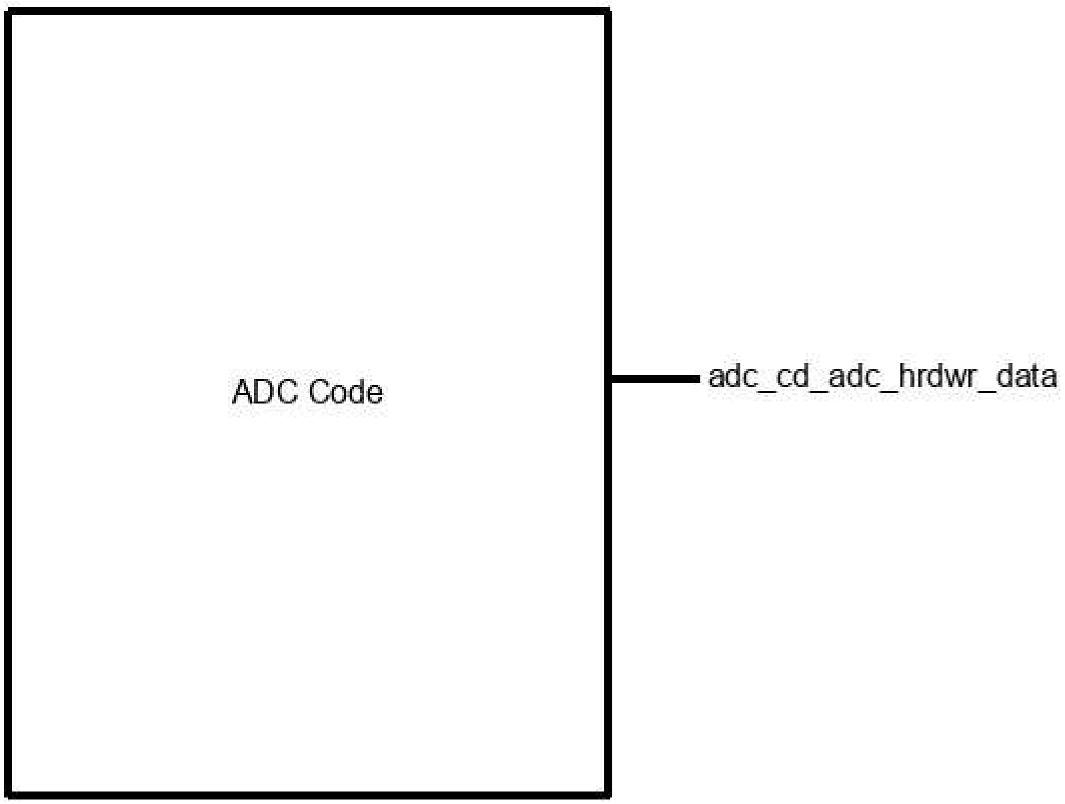


Fig. 1. Black box image of the ADC Code block

```
Import libraries.
Bind interrupts for CAN bus peripheral.
Create objects for CAN bus and SPI connections.
Configure CAN object to use a bitrate of 100Kbps.
Configure SPI object to use a bitrate of 100Kbps, have an idle
CLK level of high, and to capture data on second CLK transition.
Initialize a data buffer to hold 3 bytes of data.
```

```
Reset the ADC using SPI communication
Configure the ADC through SPI to use its AVDD pin as the
reference voltage to data measurements.
Loop
     Read ADC voltage measurement.
     Create a transfer frame for CAN communication protocol.
     Send the transfer frame to the CAN data bus.
     Wait 10 ms.
end
```

Fig. 2. Pseudocode for the ADC Code block

4.2.3 General Validation

The ADC Code block needs to be able to take 24-bit analog voltage measurements at a rate of 100Hz and send them out over CAN bus. This code also needs to be able to be uploaded to an STM32F072C8T6TR microcontroller through a pin header connection using SWD.

Figure 2 from section 4.2.2 shows how this code will function. By using the embassy framework library, objects can be created for the SPI and CAN bus peripherals. These objects are first created by passing the correct pin values for their respective I/O pins, and any necessary interrupts used for their functionality. They are then configured to communicate at a rate of 100Kbps. According to the datasheet for the ADC, the SPI protocol used to communicate with it requires the CLK signal to be held high while idle and for data to be clocked on the second CLK transition [2]. In order to configure the ADC to use the AVDD pin as the reference voltage for measurements, register 0x19 needs to be set to 0x78 in order to set the correct reference. After this, the code infinitely loops by taking ADC measurements and sending them over can. It does this by using a 10 ms timer so that measurements can be taken at exactly 100Hz. The loop waits for the timer to complete before sending the read request value of 0x42 to the ADC. After receiving the data, it then outputs the data in a CAN transmission frame from the CAN_tx pin on the microcontroller. The timer is then reset and the loop repeats.

4.2.4 Interface Validation

Table I
ADC Code Block interface validation table

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**
adc_cd_adc_hrdwr_data : Bi-directional		

Messages: Binary code file	Code must be compiled down to machine code to be run on the microcontroller.	Binary instructions stored in flash memory are used to control the microcontroller [3].
Other: Pin header connection	Pin headers were chosen for an easy connection method using an ST-Link programmer.	St-Link connector uses pin connections to program the microcontroller [4].
Protocol: Serial Wire Debug	SWD is used by the ST-Link programmer.	SDW sends a data stream over two wires to program ARM CPUs such as STM32 microcontrollers [5].

4.2.5 Verification Process

1. Connect the ST-link programmer between the computer and the pin headers of the adc_cd_adc_hrdwr_data interface.
2. Flash the code onto the microcontroller.
3. Connect the ADC hardware block to the logger block and confirm that the logger is receiving data.

4.2.6 References and File Links

[1] Embassy Project Contributors, “embassy_stm::can,” docs.embassy.dev, https://docs.embassy.dev/embassy-stm32/git/stm32f413rh/can/index.html, (accessed Dec. 6, 2023).

[2] Analog Devices, “AD7124-8 (Rev. F),” analog.com, https://www.analog.com/media/en/technical-documentation/data-sheets/ad7124-8.pdf, (accessed Nov. 28, 2023).

[3] STMicroelectronics, “STM32F0 series Cortex-M0 programming manual,” st.com, https://www.st.com/resource/en/programming_manual/pm0215-stm32f0-series-cortexm0-programming-manual-stmicroelectronics.pdf, (accessed Mar 4, 2024).

[4] Adafruit, “ST-Link STM8/STM32 v2 Compatible Programmer & Emulator,” adafruit.com, https://www.adafruit.com/product/2548, (accessed Mar 4, 2024).

[5] Silicon Labs, “Serial Wire Debug (SWD), https://community.silabs.com/s/article/serial-wire-debug-swd-x?language=en_US, community.silabs.com, (accessed Mar 4, 2024).

4.2.7 Revision Table

TABLE II
Subsection 4.2 Revision Table

3/8/2024	Alexander Gibson: Created and filled out sections 4.2.1 - 4.2.7

4.3 GPS Block

4.3.1 Description

The GPS Block is used by the system to measure position values. It satisfies the system's modularity requirement due to it connecting to the logger module through an RS45 cable, the precision requirement by taking 8-bit position measurements, and the sample rate requirement by including a GPS that can sample at a rate of at least 5Hz. The block features a NEO-M9N GPS, a microcontroller, a programmer connection, a linear regulator, a CAN bus transceiver, and interface connectors. The GPS will be able to read RF signals from satellites that give the position and transmission time of the satellite. This information is then used to calculate the position of the GPS which is then sent to the microcontroller through UART communication protocol. The microcontroller can then send these values to the logger block using CAN bus communication.

4.3.2 Design

This section describes the design of the GPS sensor hardware block. It includes a black box diagram showing the interfaces of the block. These interfaces are the RF satellite signals received by the GPS, the DC power used to power the module, the programmer connection used to upload code to the microcontroller, and the CAN bus communication line used to communicate with the logger module. This section also includes schematic diagrams for the hardware design. These diagrams cover the microcontroller used to interface with the GPS and CAN bus, linear regulator used to convert the 4.8-5.2V power input to a 3.3V output, CAN transceiver used to convert the digital data signal to an analog differential pair signal, the GPS used to receive position values, and the connectors used for all of the interfaces.

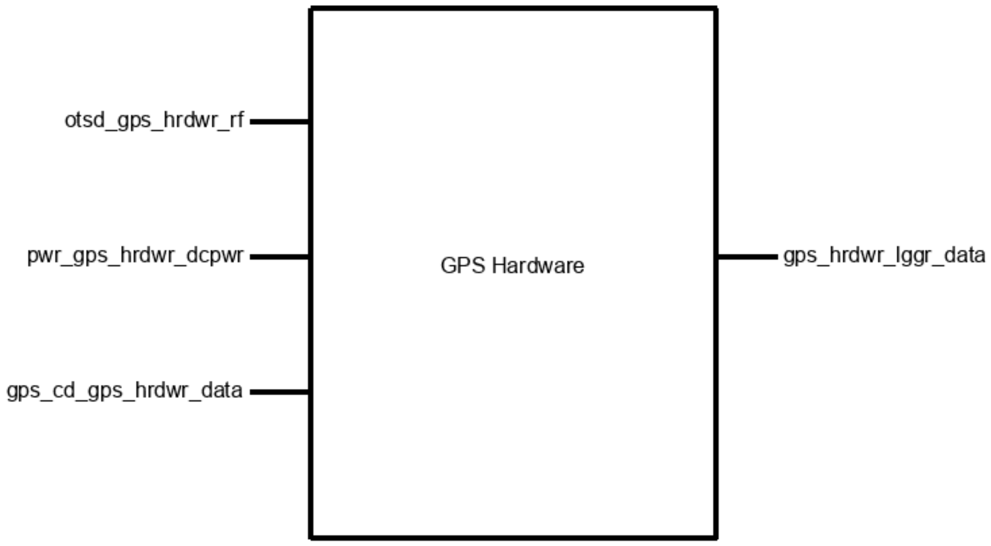


Fig. 1. Black box image of the block

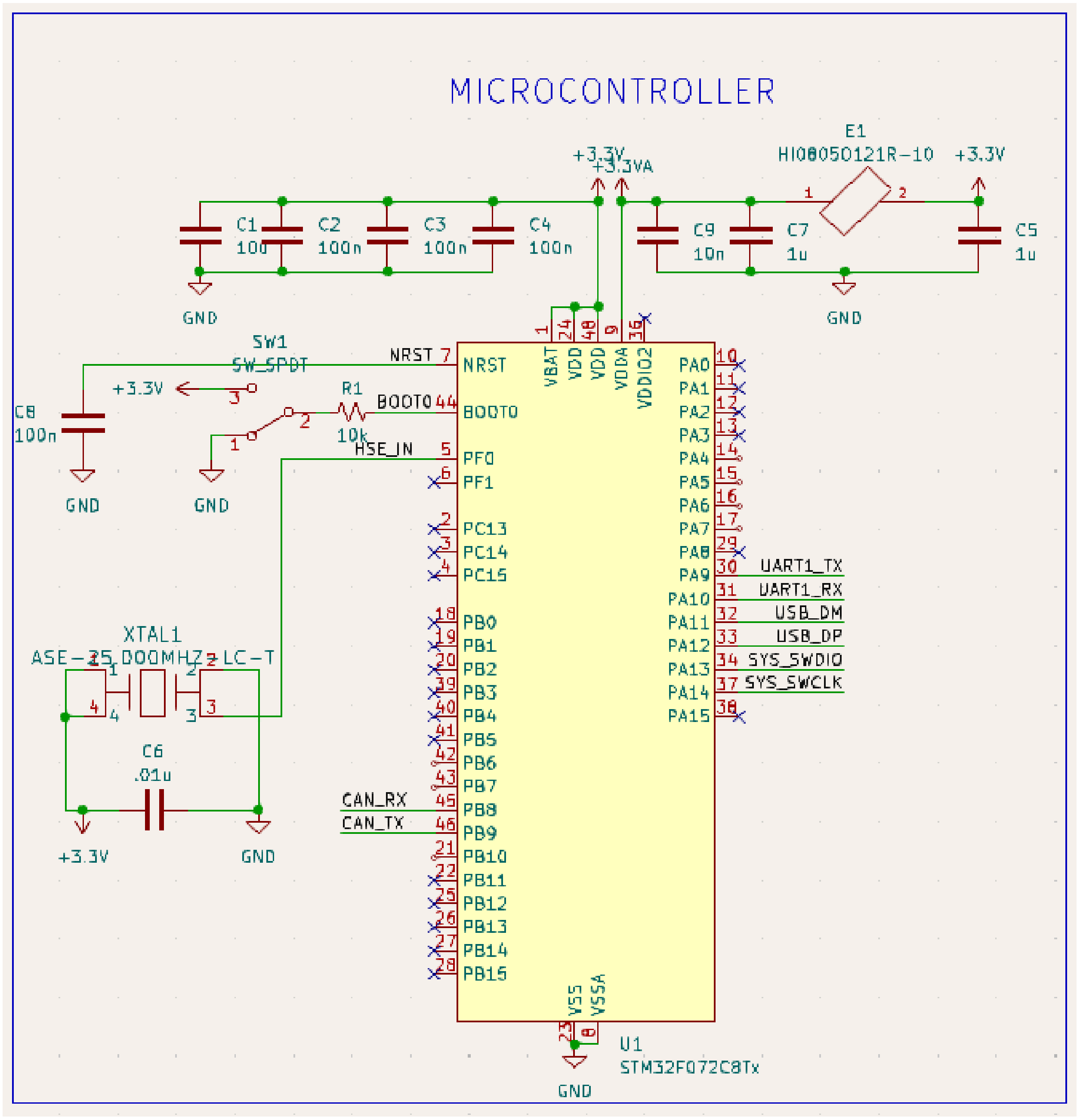


Fig. 2. GPS Hardware Block Microcontroller Schematic

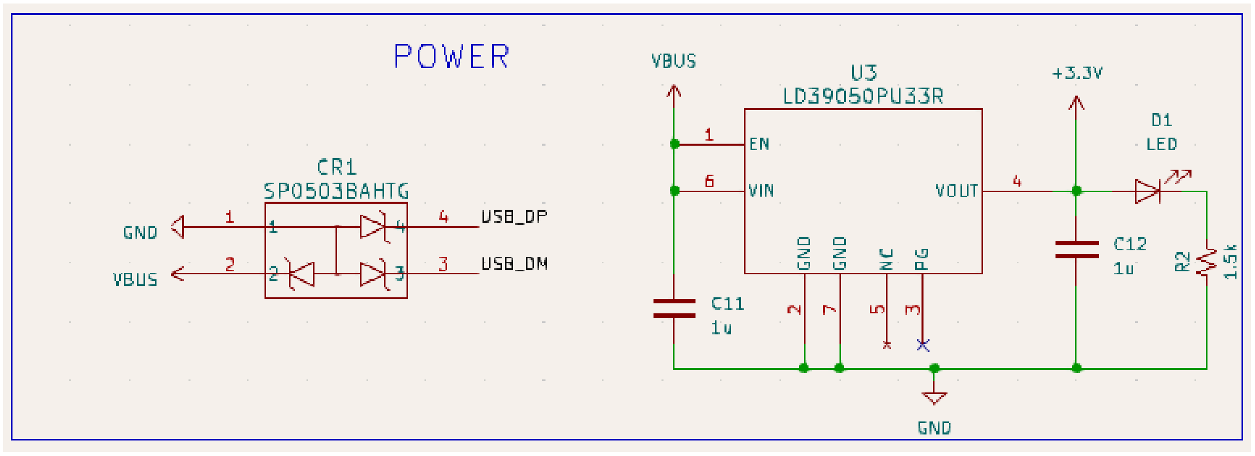


Fig. 3. GPS Hardware Block Power Schematic

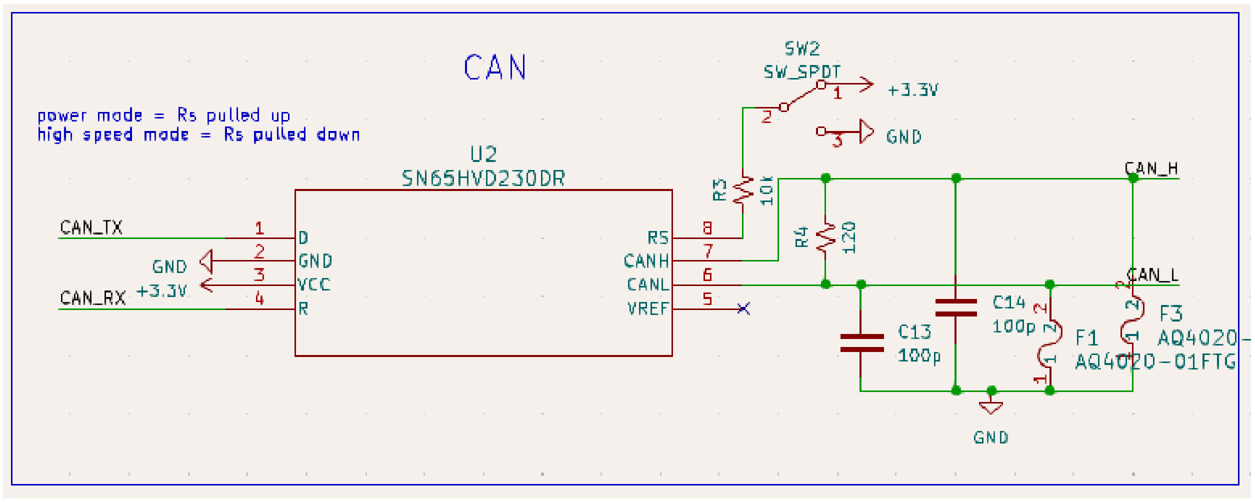


Fig. 4. GPS Hardware Block CAN Transceiver Schematic

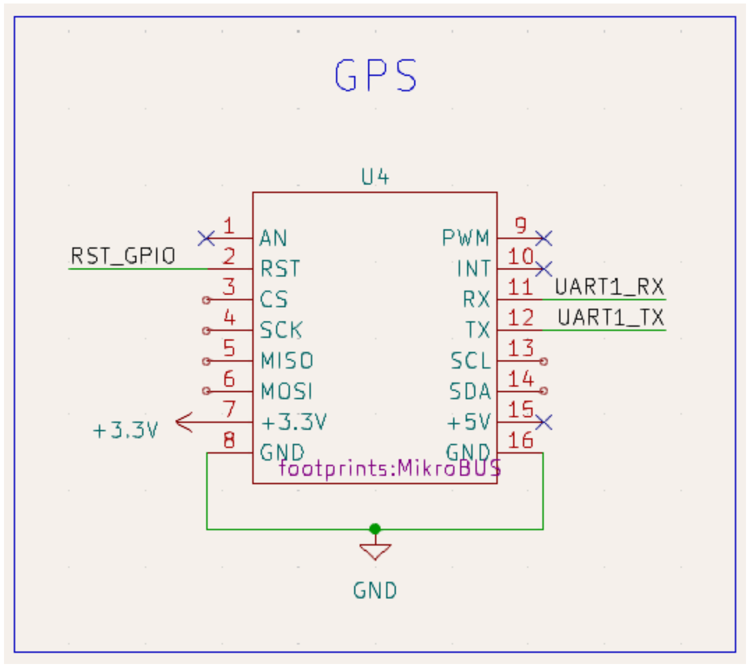


Fig. 5. GPS Hardware Block GPS Daughter Board

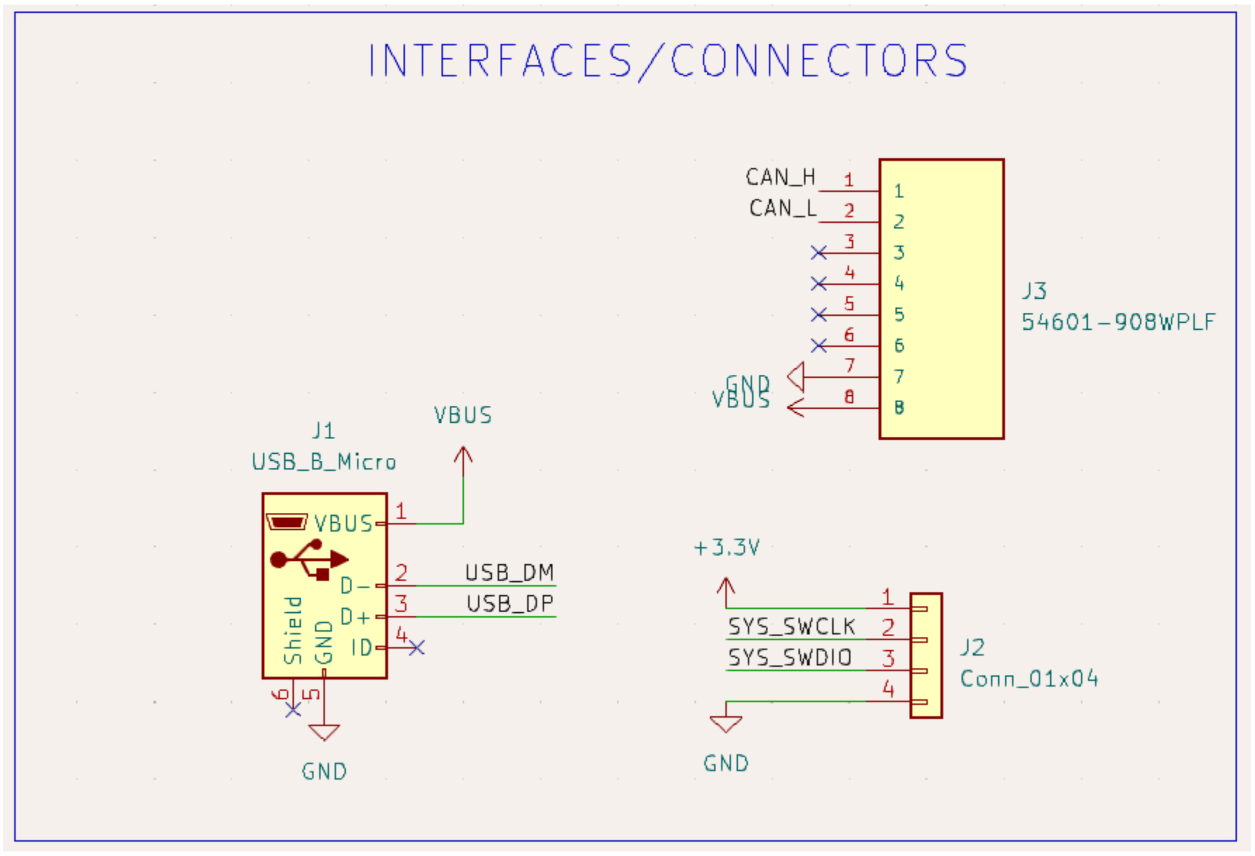


Fig. 6. GPS Hardware Block Interfaces / Connectors Schematic

4.3.3 General Validation

The GPS Hardware Block needs to be able to take in RF signals from GPS satellites and calculate position measurements of at least 8-bits and output that data through CAN bus protocol to the logger block. Along with this, due to the sampling rate requirement for the system, the GPS must take these measurements at a minimum sampling rate of 5Hz. Lastly, the GPS block must also be able to do these operations with a voltage input between 4.8 and 5.2 volts.

The Microcontroller schematic shown in figure 2 features an STM32F072C8T6TR microcontroller. This specific microcontroller was chosen due to it having the necessary peripheral support for all of the devices used in all of the sensor blocks for the entire system. These peripherals are I2C, SPI, UART, CAN, USB, and pulse width modulation. This means that the same microcontroller can be integrated into each of the 3 sensor hardware designs. Along with this, the microcontroller was also chosen due to it being low cost and featuring an IC package with a low amount of pins that are easily solderable. The microcontroller circuit design features a 3.3 volt connection provided by the LD39050PU33R which is used to power the microcontroller. Each power input pin is connected to a 100nF decoupling capacitor to filter out noise as recommended by the hardware development document for the microcontroller [1]. A ferrite bead is used to isolate noise from the 3.3V source for the VDDA pin. This design for isolating VDDA from the other power input pins was found in a STM32 board design video guide [2]. The reset pin (NRST) is connected to a 100n capacitor to prevent the microcontroller from resetting due to noise. A 25 MHz crystal oscillator is used as the clock source for the microcontroller. According to the hardware development document, a 4 to 32 MHz external oscillator should be used for the design [1]. A decoupling capacitor is connected between the 3.3V and GND pin on the oscillator to filter noise. The tri-state pin on the oscillator is connected to 3.3V in order to enable the oscillator’s output [3]. The boot mode pin was connected to a switch so that it could be pulled high or low for either boot option. Lastly, pins PB8 and PB9 are used for the can peripheral, pins PA4 - PA7 are used for the SPI peripheral, PA11 and PA12 are used for the USB peripheral, and PA13 and PA14 are used for the programmer interface [4].

The power schematic shown in figure 3 features an LD39050PU33R linear regulator and an SP0503BAHTG TVS diode block used to protect the USB connection against electrostatic discharge. The power for the system works where a 5.0V input called VBUS is connected to the input pin of the linear regulator which then outputs 3.3V. VBUS is provided by either a micro-USB connection or a RJ45 connection. The application information for the linear regulator recommends connecting 1uF capacitors to the Vin and Vout inputs [5]. Along with this, Vin is also connected to the enable pin in order to enable the output of the IC. Lastly, an LED is attached to Vout to give a visual indication that power is being output by the linear regulator.

The CAN transceiver schematic shown in figure 4 features an SN65HVD230DR transceiver used to convert digital CAN data from the microcontroller to a differential pair data output. As specified in the datasheet for the transceiver, data being sent from the microcontroller is connected to the D pin and data being received is connected to the R pin. The differential pair data is output on pins CANH and CANL which are connected by a 120 ohm termination resistor to improve signal quality [6] [7]. According to the device datasheet, the RS pin should be pulled high for power mode or pulled low for high speed mode [7]. In the schematic, a switch is used so that either mode can be used. Lastly, capacitors and TVS diodes are used on the CANH and CANL connections to reduce noise and protect against large surges in voltage.

The GPS schematic shown in figure 5 features a GNSS 7 CLICK board which acts as a breakout board for the NEO-F9P GPS. This board will be used as a daughter board to the GPS hardware by connecting it to the PCB for the GPS hardware circuit. The UART connection from the microcontroller is used to communicate with the GPS [8]. The GPS is powered by the same 3.3V source as the microcontroller.

The interface/connectors schematic shown in figure 6 features a micro-USB connector used for data transfer and powering the linear regulator, a RJ45 connector used to transfer the CAN data and provide power when connected to the logger module, and pin headers used for a JTAG programming connection.

While this circuit is designed with the intention of using CAN bus protocol in power mode, an alternate solution using high speed mode is possible by using the switch to connect the RS pin on the transceiver to GND and desoldering the 120 ohm resistor. Another alternate solution for this design would be to connect LEDs to some of the extra GPIO pins for debugging purposes.

4.3.4 Interface Validation

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**
otsd_gps_hrdwr_rf : Input		
Datarate: 50bits/sec	A 1500 bit data frame is sent from the satellite to the GPS chip over the course of 30 seconds [9].	GPS chip receives data from GPS satellites [8].
Messages: messages contain the position and time of the GPS satellites	The GPS chip needs to know the position of the satellite and the time it initially sent the signal to use the amount of time that the signal traveled to find the chip's position [10].	NEO-M9N GPS chip is used in the design which calculates position based on the RF signals it receives from satellites.
Protocol: NMEA-0183	This is the protocol used by GPS satellites to communicate with GPS chips [8].	According to its datasheet, the NEO-M9N uses NMEA-0183 to communicate with satellites [8].
pwr_gps_hrdwr_dcpwr : Input		
Inominal: 33mA	Nominal current of microcontroller with a system clock of 60MHz.	The microcontroller will typically draw 26 mA at 25 ℃ and 39 mA at 85 ℃ [4]. Since the GPS has a significantly lower current draw, it

		can be estimated that the nominal current will be between 25 and 39 mA [8].
Ipeak: 170mA	Peak power input current of the microcontroller and ADC chip being used [4] [8].	Microcontroller and GPS will draw at most 170mA [4] [8]. Since the microcontroller and GPS are not being fully utilized, the current draw will be less than this maximum value.
Vmax: 5.2V	Ideal input voltage is 5.0V so this value allows for some variance without going too far out of range.	Max operating input voltage for the linear regulator is 5.5V as described on page 6 of the datasheet, meaning 5.2V is in range [5].
Vmin: 4.8V	Ideal input voltage is 5.0V so this value allows for some variance without going too far out of range.	Since the linear regulator is described on page 6 of the datasheet as outputting a value of Vin - 0.3 with a minimum Vin value of 1.5V, a value of 4.8 will satisfy both conditions above to output 3.3V [5].
gps_hrdwr_lggr_data : Output		
Datarate: 100Kbit/sec CAN bus speed	This speed was chosen because the design will be using a CAN bus connection in low-speed mode to improve signal quality. CAN bus configured for low-speed mode has a maximum data rate of 125 Kbit/sec [6].	According to page 5 of the SN65HVD23 transceiver datasheet, low-speed mode is configured by pulling RS to 3.3V [7].
Messages: 8 bits of data	A CAN bus transmission will occur after each GPS sample.	GPS samples are 8-bits of data [8].
Protocol: CAN bus	CAN bus was chosen as the protocol to communicate between the logger block and the GPS block because of its application in automotive systems and because our project partner is familiar with the protocol.	The chosen microcontroller supports CAN bus peripherals on the PB8 and PB9 pins [4]. The CAN_RX and CAN_TX signals from these two pins are connected to the CAN transceiver which converts the digital values sent and received by these pins to differential pair signals [7].
gps_cd_gps_hrdwr_data : Bi-directional		

Messages: Binary code file	Code must be compiled down to machine code to be run on the microcontroller.	Binary instructions stored in flash memory are used to control the microcontroller [11].
Other: Pin header connection	Pin headers were chosen for an easy connection method using an ST-Link programmer.	St-Link connector uses pin connections to program the microcontroller [12].
Protocol: Serial Wire Debug	SWD is used by the ST-Link programmer.	SDW sends a data stream over two wires to program ARM CPUs such as STM32 microcontrollers [13].

4.3.5 Verification Process

1. Connect jumper wires between the gps_cd_gps_hrdwr_data interface and the ST-Link programmer connected to a computer, then program the microcontroller with the GPS block testbench firmware.
2. Disconnect the programmer and connect the GPS block's gps_hrdwr_lggr_cd_data CAN bus connection to the gps_hrdwr_lggr_cd_data CAN bus connection on the raspberry-pi.
3. Connect the GPS block's pwr_gps_hrdwr_dcpwr input to a 4.8V power source.
4. Run the GPS block testbench python script on the raspberry pi.
5. Confirm that the raspberry pi is receiving data values of at least 8-bits.
6. Confirm in the code of both the GPS microcontroller and the raspberry-pi that both are configuring the CAN bus protocol to run at 100Kb/sec.
7. Change the pwr_gps_dcpwr input to 5.2V and repeat steps 5-7.

4.3.6 References and File Links

[1] STMicroelectronics, "Getting started with STM32F0x1/x2/x8 hardware development," st.com, https://www.st.com/resource/en/application_note/an4080-getting-started-with-stm32f0x1x2x8-hardware-development-stmicroelectronics.pdf, (accessed Mar 3, 2024).

[2] Phil's Lab, "KiCad 6 STM32 PCB Design Full Tutorial - Phil's Lab #65," youtube.com, https://www.youtube.com/watch?v=aVUqaB0IMh4&ab_channel=Phil%E2%80%99sLab, (accessed Mar. 6, 2024).

[3] Abracon, "3.3V CMOS Compatible 3.2 x 2.5mm SMD Crystal Oscillator," abracon.com, https://abracon.com/Oscillators/ASEseries.pdf, (accessed Mar. 6, 2024).

[4] STMicroelectronics, "STM32F072x8 STM32F072xB," st.com, https://www.st.com/content/ccc/resource/technical/document/datasheet/cd/46/43/83/22/d3/40/c8/DM00090510.pdf/files/DM00090510.pdf/jcr:content/translations/en.DM00090510.pdf, (accessed Dec. 6, 2023).

[5] STMicroelectronics, “LD39050,” st.com, https://www.st.com/content/ccc/resource/technical/document/datasheet/12/43/cc/93/21/f6/41/66/CD00227527.pdf/files/CD00227527.pdf/jcr:content/translations/en.CD00227527.pdf, (accessed Jan. 27, 2024).

[6] KVASER, “The CAN Bus Protocol Tutorial,” kvaser.com, https://www.kvaser.com/can-protocol-tutorial/, (accessed Nov. 28, 2023).

[7] Texas Instruments, “SN65HVD230DR,” ti.com, https://www.ti.com/lit/ds/symlink/sn65hvd230.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-wwe&ts=1709546681296&ref_url=https%253A%252F%252Fwww.google.com%252F, (accessed Mar 4, 2024).

[8] u-blox, “NEO-M9N Integration Manual,” u-bloxcom, https://content.u-blox.com/sites/default/files/NEO-M9N_Integrationmanual_UBX-19014286.pdf, (accessed Nov. 28, 2023).

[9] European Space Agency, “GPS Navigation Message,” gssc.esa.int, https://gssc.esa.int/navipedia/index.php/GPS_Navigation_Message, (accessed 12/6/2023).

[10] Federal Aviation Administration, “Satellite Navigation - Global Positioning System (GPS),” faa.gov, https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps, (accessed 12/6/2023).

[11] STMicroelectronics, “STM32F0 series Cortex-M0 programming manual,” st.com, https://www.st.com/resource/en/programming_manual/pm0215-stm32f0-series-cortexm0-programming-manual-stmicroelectronics.pdf, (accessed Mar 4, 2024).

[12] Adafruit, “ST-Link STM8/STM32 v2 Compatible Programmer & Emulator,” adafruit.com, https://www.adafruit.com/product/2548, (accessed Mar 4, 2024).

[13] Silicon Labs, “Serial Wire Debug (SWD), https://community.silabs.com/s/article/serial-wire-debug-swd-x?language=en_US, community.silabs.com, (accessed Mar 4, 2024).

4.3.7 Revision Table

TABLE VIII
Subsection 4.2 Revision Table

3/8/2024	Alexander Gibson: Updated section 4.3.1 with a statement about how the block contributes to the overall system, provided schematic diagrams in section 4.3.2, rewrote section 4.3.3 to discuss the new schematics, updated section 4.3.4 by adding a table label, including new interfaces and their properties, and fixing Vmin and Vmax values due to them being incorrectly switched, updated section 4.3.5 to prove all interfaces of the block, and updated section 4.3.6 with more datasheet links.

1/24/2024	Alexander Gibson: Added intro paragraph to design section, updated interface names and values to match student portal, and modified interface validation to match changes in design choices.
12/6/2023	Alexander Gibson: Filled out sections 4.2.1 through 4.2.7

4.4 GPS Code

4.4.1 Description

The GPS Code block is used by the system to provide the microcontroller code necessary for the GPS hardware block to function correctly. The code works by first configuring the GPS with the correct settings before taking position values at a rate of 5Hz. After each measurement, it sends the data over the CAN bus interface which is then received by the logger module. The code is uploaded to the microcontroller using a Serial Wire Debug interface (SWD) provided by an ST-Link programmer device.

4.4.2 Design

This section provides a black box diagram and pseudocode to describe the functionality of the ADC Code block.

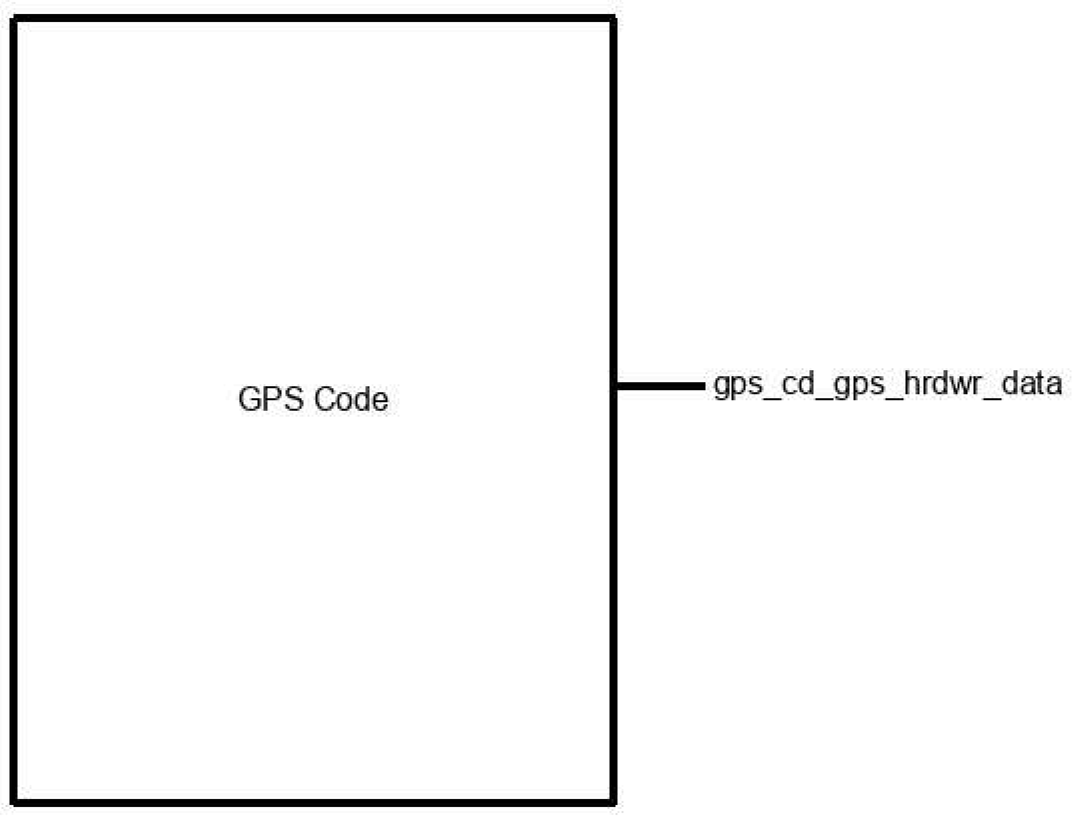


Fig. 1. Black box image of the GPS Code block

```
Import libraries.
Bind interrupts for CAN bus peripheral.
Create objects for CAN bus and UART connections.
Configure CAN object to use a bitrate of 100Kbps.
Reset the GPS using UART communication.
Configure the GPS through UART.
Loop
     Read GPS position measurement.
     Create a transfer frame for CAN communication protocol.
     Send the transfer frame to the CAN data bus.
     Wait 200 ms.
end
```

Fig. 2. Pseudocode for the GPS Code block

4.4.3 General Validation

The GPS Code block needs to be able to take 8-bit or larger position measurements at a rate of 5Hz and send them out over CAN bus. This code also needs to be able to be uploaded to an STM32F072C8T6TR microcontroller through a pin header connection using SWD.

Figure 2 from section 4.4.2 shows how this code will function. By using the embassy framework library, objects can be created for the UART and CAN bus peripherals. These objects are first created by passing the correct pin values for their respective I/O pins, and any necessary interrupts used for their functionality. The CAN object is then configured to communicate at a rate of 100Kbps. According to the datasheet for the GPS, UART protocol is used to read measurements from it [2]. After initializing settings on the GPS, the code infinitely loops by taking GPS measurements and sending them over CAN. It does this by using a 200 ms timer so that measurements can be taken at exactly 5Hz. The loop waits for the timer to complete before sending the read request to the GPS. After receiving the data, it then outputs the data in a CAN transmission frame from the CAN_tx pin on the microcontroller. The timer is then reset and the loop repeats.

4.4.4 Interface Validation

Table I
GPS Code Block interface validation table

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**
gps_cd_gps_hrdwr_data : Bi-directional		

Messages: Binary code file	Code must be compiled down to machine code to be run on the microcontroller.	Binary instructions stored in flash memory are used to control the microcontroller [3].
Other: Pin header connection	Pin headers were chosen for an easy connection method using an ST-Link programmer.	St-Link connector uses pin connections to program the microcontroller [4].
Protocol: Serial Wire Debug	SWD is used by the ST-Link programmer.	SDW sends a data stream over two wires to program ARM CPUs such as STM32 microcontrollers [5].

4.4.5 Verification Process

4. Connect the ST-link programmer between the computer and the pin headers of the gps_cd_gps_hrdwr_data interface.
5. Flash the code onto the microcontroller.
6. Connect the GPS hardware block to the logger block and confirm that the logger is receiving data.

4.4.6 References and File Links

[1] Embassy Project Contributors, “embassy_stm::can,” docs.embassy.dev, https://docs.embassy.dev/embassy-stm32/git/stm32f413rh/can/index.html, (accessed Dec. 6, 2023).

[2] u-blox, “NEO-M9N Integration Manual,” u-bloxcom, https://content.u-blox.com/sites/default/files/NEO-M9N_Integrationmanual_UBX-19014286.pdf, (accessed Nov. 28, 2023).

[3] STMicroelectronics, “STM32F0 series Cortex-M0 programming manual,” st.com, https://www.st.com/resource/en/programming_manual/pm0215-stm32f0-series-cortexm0-programming-manual-stmicroelectronics.pdf, (accessed Mar 4, 2024).

[4] Adafruit, “ST-Link STM8/STM32 v2 Compatible Programmer & Emulator,” adafruit.com, https://www.adafruit.com/product/2548, (accessed Mar 4, 2024).

[5] Silicon Labs, “Serial Wire Debug (SWD), https://community.silabs.com/s/article/serial-wire-debug-swd-x?language=en_US, community.silabs.com, (accessed Mar 4, 2024).

4.4.7 Revision Table

TABLE II
Subsection 4.4 Revision Table

3/8/2024	Alexander Gibson: Created and filled out sections 4.4.1 - 4.4.7

4.5 Mikroe Sensor Hardware Block

4.5.1 Description

The Mikroe Sensor Hardware Block features a MikroBUS(TM) connection interface, a microcontroller, a programming interface, a voltage regulator, and a CAN transceiver. This block will allow the system to use any Mikroe Click sensor board with minimal configuration. This will increase the functionality of the system while not requiring the design and assembly of additional custom sensor boards.

The MikroBUS(TM) interface will be used to connect any Mikroe Click sensor board to our system. This will greatly increase the potential functionality of our system by enabling it to use any of the 525 current Mikroe click sensor boards available on Mikroe's website [1]. The firmware for the connected Click Sensor board can be sent to the microcontroller using the programming port. The connected Mikroe Click sensor board can then send data to the microcontroller, which works to update the logger block with the sensor data using CAN bus communication through the CAN transceiver.

4.5.2 Design

This block must include a MikroBUS(TM) connection interface for the Mikroe Click Sensor Boards to connect to, an STM32 microcontroller to control the logic, a programming connection so the code from the Mikroe Sensor Code block can be uploaded, a power input port with a voltage regulator to make sure it is powered at the correct level, and a CAN bus transceiver and connector to send data to the logger block. See Figure 1 below for the black box image of this block.

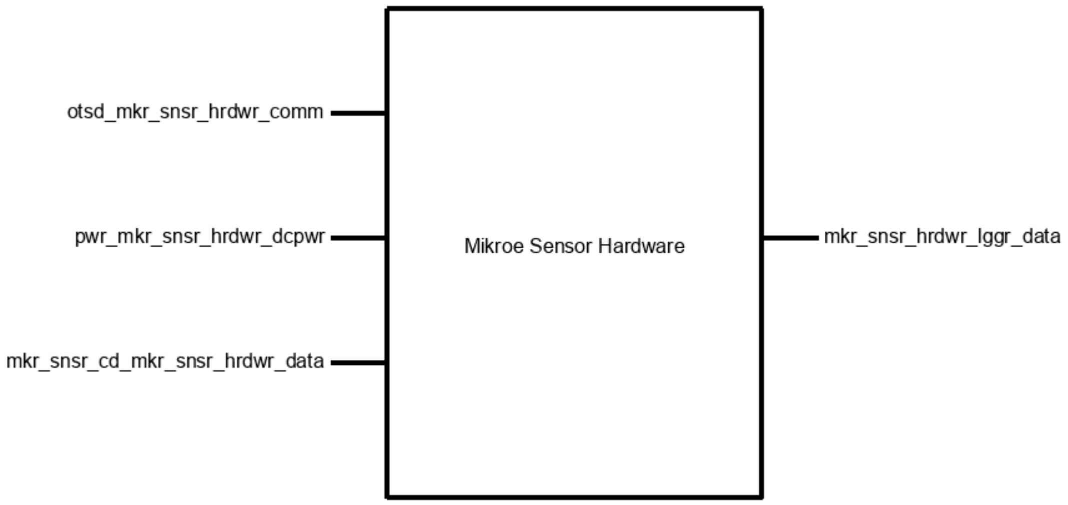


Fig. 1. Mikroe Sensor Hardware Block black box image

The microcontroller is the heart of this block. The microcontroller communicates with the connected Mikroe Click sensor board via the MikroBUS(TM) connection. Then, it relays the sensor data to the logger block using CAN bus communication.

The microcontroller chosen for our design is an STM32F072C8T6TR. Figure 2 below shows the setup for the microcontroller. The capacitors at the power pins were chosen according to STM32 design guidelines [2], as well as an in-depth YouTube tutorial [3]. The boot and reset pins were also set up according to the mentioned resources. The high-speed external oscillator, ASE-25.000MHZ-L-C-T, on the left of the figure, was set up according to suggestions in its datasheet [4].

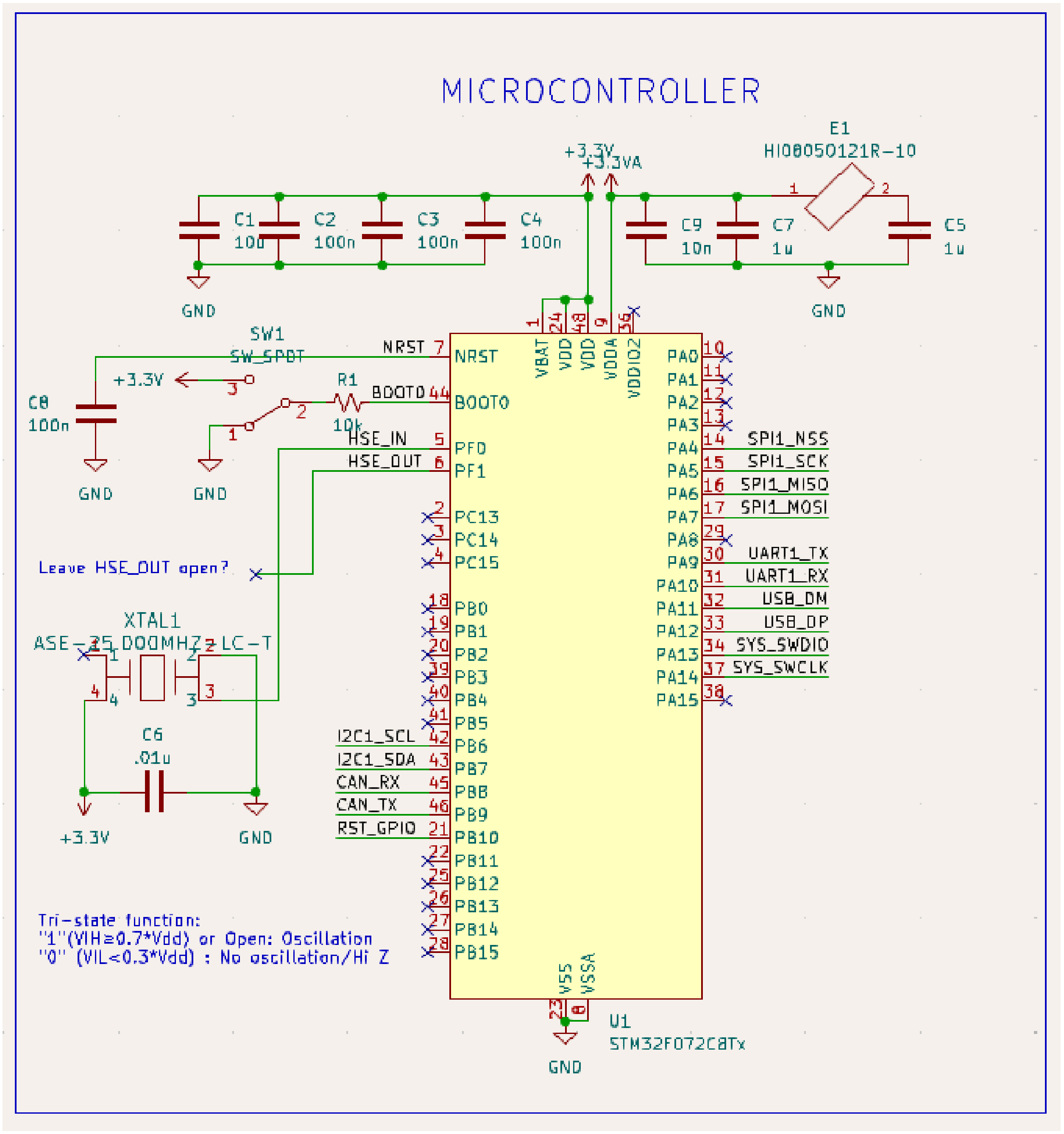


Fig. 2. Mikroe Sensor Hardware Block Microcontroller Schematic

This block will include a voltage regulator to step the 5V input voltage down to a workable 3.3V for the microcontroller. This voltage regulator will be an LD39050PU33R. Figure 3 below shows the “power” section of the block schematic. The voltage regulator portion of the circuit was created using the application information in the LD39050PU33R datasheet [5]. We are also using a SP0503BAHTG for TVS protection as seen in figure 3.

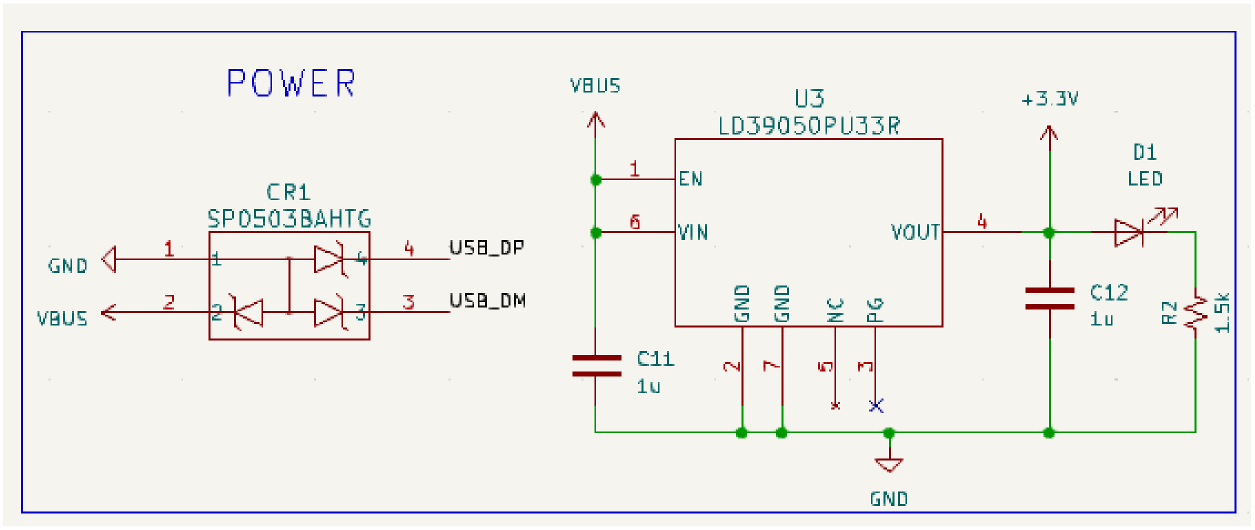


Fig. 3. Mikroe Sensor Hardware Block Power Schematic

The block will also include a CAN transceiver to transmit and receive data between this block and the logger block through the CAN bus protocol. This transceiver will be an SN65HVD230DR. The CAN transceiver was set up according to the application section of its datasheet [6]. Figure 4 below shows the “CAN” section of the block schematic.

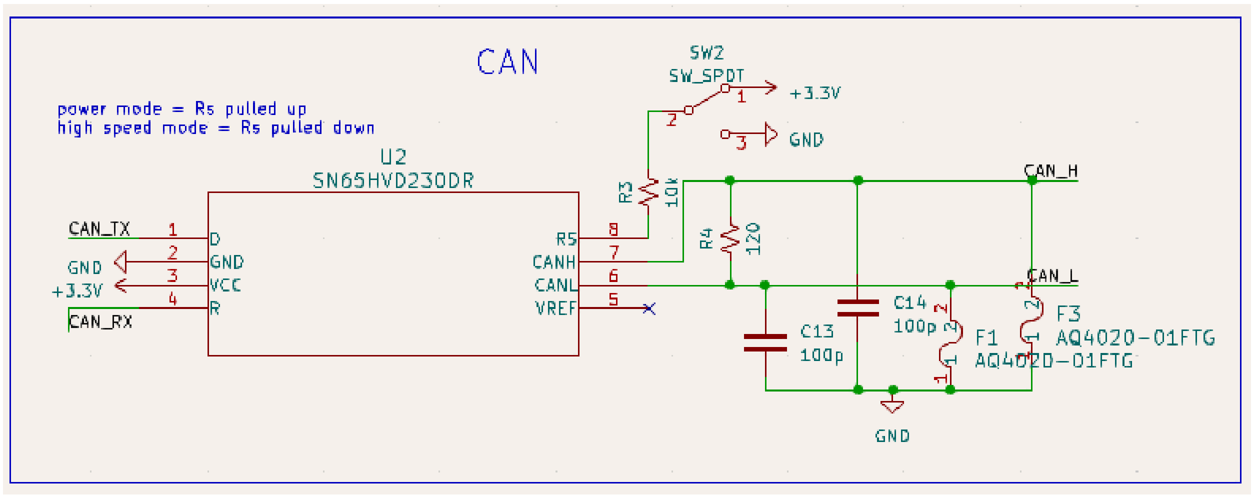


Fig. 4. Mikroe Sensor Hardware Block CAN Schematic

Finally, figure 5 below shows the “Interfaces/Connectors” section of the schematic. Here can be seen the MikroBUS connector for the Mikroe Click Sensors (otsd_mkr_snsr_hrdwr_comm interface), the Micro-USB connector for power (pwr_mkr_snsr_hrdwr_dcpwr interface), the pin header connection for programming (mkr_snsr_cd_mkr_snsr_hrdwr_data interface), and an RJ45 connector for CAN communication (mkr_snsr_hrdwr_lggr_data interface). Note that power can be supplied through the pin headers or the Micro-USB connector.

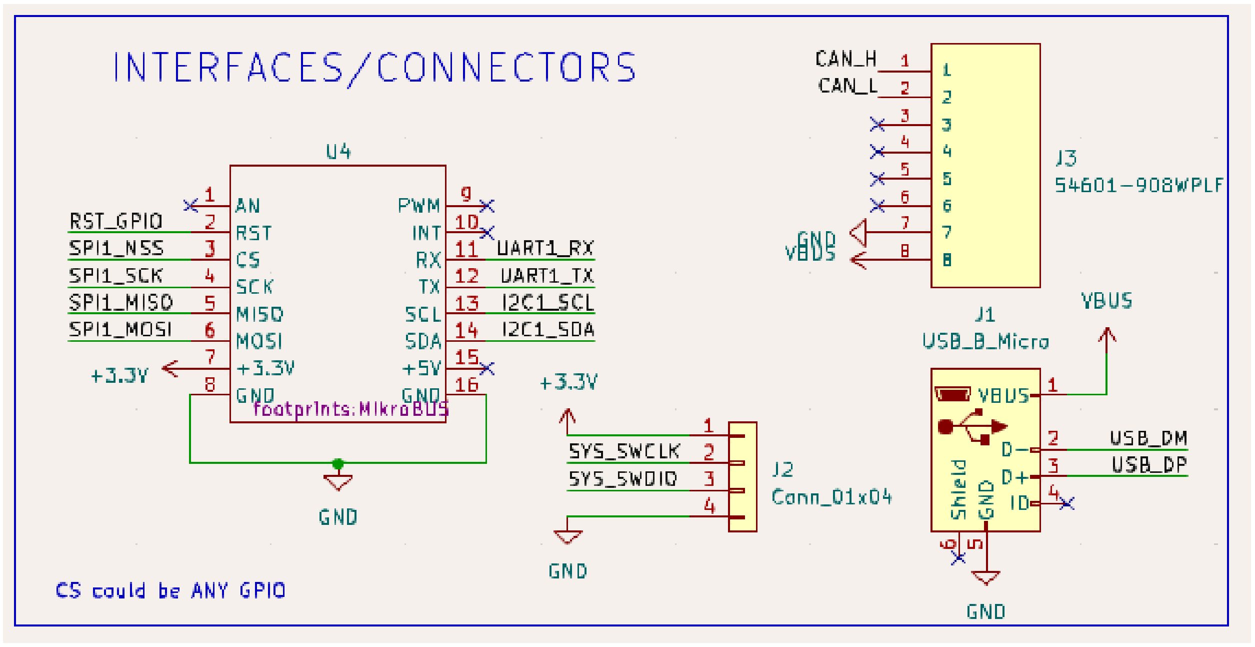


Fig. 5. Mikroe Sensor Hardware Block Interfaces / Connectors Schematic

4.5.3 General Validation

This section will describe why our chosen design details fit the needs of the system as a whole, especially with regard to the block's function, cost, and engineering time. This section will also include an alternate solution for this block's design. The overall function of this block is to take sensor data from a Mikroe Click sensor and output that data to the logger block through the CAN bus protocol.

MikroBUS(TM) was chosen as the connection type for external sensors because of the vast amount of possible sensors to choose from, including biometrics, gas, magnetic, motion, optical, pressure, proximity, temperature & humidity, current, force, and inductance sensors [1]. This will allow the project partner and any future users of the system to have access to near-limitless functionality and be able to adapt the system to their specific needs. The project partner also has previous experience with the company and their products and can attest to their support and the ease of use of their products. Finally, the sensors are well documented and most cost under $50, ensuring the system stays operable and affordable.

The specific STM32 microcontroller that will be used for this block, STM32F072C8T6TR, was chosen for its compatibility with the Embassy IDE, which will be used in development, as well as for its favorable input and output capabilities. Some of the capabilities include pins for USB 2.0, SPI, I2C, UART, and CAN communication [7]. Pins for these purposes can also be seen in Figure 2 above. Additionally, the team will be conducting tests using a Mikroe MCU card with a microcontroller very similar to our final choice microcontroller; using such a similar microcontroller for the final design will help ensure a smooth transition from testing to implementation.

The programming interface for the block will utilize a pin header connection. This was chosen because an ST-Link programmer will be used to program the block and the ST-Link programmer uses pin connections [8]. Additionally, this type of interface will be cost-effective, simple to physically implement, and won't take up much room.

The power interface will use a Micro-USB connector. Using such a common port will help ensure cost-effectiveness and part-availability. We will use Amphenol's 10118194-0001LF Micro-USB connector for our design [9]. The voltage regulator will be an LD39050PU33R. This was chosen for its favorable characteristics including Input voltage from 1.5V to 5.5V, output voltage from 0.8 V to 4.5 V with 100 mV step, and 500mA guaranteed output current [5]. This regulator is also less than two dollars and comes in a 3mm x 3mm package, ensuring it will keep costs down through both the initial purchase price as well as only requiring minimal space on our PCB. Additionally, it has been successfully utilized on boards that include a microcontroller similar to the one we will use such as the STM32 Nucleo-144 boards [10], which ensures it is a commercially viable option. Our implementation of this regulator can be seen in Figure 3 above.

The CAN transceiver chosen for our design is the SN65HVD230DR. This selection was made due to its advantageous features and performance characteristics including that it operates on a single 3.3V supply, is able to be used for Data Rates up to 1 Mbps, and is low power [6]. Furthermore, this transceiver costs less than three dollars. Finally, we have had a chance to use this specific transceiver while testing our blocks, utilizing a breakout board sourced from Amazon [11]. This experience will allow for seamless integration into our final system, making it an ideal choice for our project's requirements. Our implementation of this transceiver can be seen in Figure 4 above.

One alternate solution for this block would be to use a different, single connection protocol to gather data from a sensor such as SPI, UART, or I2C instead of a MikroBUS(TM) interface. This may allow for a more compact and simpler design that can use sensors beyond those offered by Mikroe. However, this would leave the responsibility of ensuring proper sensor communication and operation with the end-user, complicating the task of connecting external sensors. This would hurt the usability of our system and limit the potential user base to technically savvy individuals. These are unacceptable tradeoffs for our project partner, thus the MikroBUS(TM) connection was chosen.

4.5.4 Interface Validation

This section includes a table formatted to show each interface and each interface property relating to the Mikroe Sensor Block, as well as an explanation of why the interface property values were chosen and why the block will work with that value.

TABLE I
Mikroe Sensor Block interface validation table

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**

otsd_mkr_snsr_hrdwr_comm : Input		
Datarate: 100kbps or more	According to the MikroBUS(TM) standard specifications, MikroBUS(TM) has three groups of communications pins: SPI, UART, and I2C [12]. 100kbps was chosen as the data rate based on the minimum capabilities of MikroBUS(TM) communication. The exact data rate will depend on the protocol used by the connected Click Sensor Board.	An STM32F072C8T6TR microcontroller will be used for this block. This line of microcontrollers has been proven to work with MikroBUS(TM) interfaces, as shown by Mikroe's several STM32-based MCU boards designed to work with MikroBUS(TM)-connected Mikroe Click sensors, such as the STM32f413RH MCU board which will be used for testing [13].
Messages: Data from the Mikroe Click sensor board that is connected	This interface must allow communication between a Mikroe Click Sensor board and the Mikroe Sensor Hardware block.	This value will be met by our block because we designed this block to read data from the MikroBUS interface using official STM32 design guidelines [2].
Other: MikroBUS-style Socket	According to the MikroBUS(TM) standard specifications, the standard was created specifically to interface "microcontrollers or microprocessors (mainboards) with integrated circuits and modules (add-on boards)" [12]. Thus, a MikroBUS(TM) interface was chosen.	This value will be met by our block because we will use the official MikroBUS(TM) Standard specifications to implement the socket [12].
mkr_snsr_cd_mkr_snsr_hrdwr_data : Bi-directional		
Messages: Binary code file	Code must be compiled down to machine code to be run on the microcontroller.	Binary instructions stored in flash memory are used to control the microcontroller [14].
Other: Pin header connection	Pin headers were chosen for an easy connection method between the block and an ST-Link programmer.	This value will be met because we will use standard pin header spacing and layout in our physical design.
Protocol: Serial Wire Debug	We will use an ST-Link programmer for programming this block. SWD is supported by the ST-Link programmer [8].	This protocol was chosen because the STM32F072C8T6TR microcontroller can be programmed and debugged using SWD [8].
pwr_mkr_snsr_hrdwr_dcpwr : Input		

Inominal: 33mA	This value was chosen based on the nominal current of the microcontroller being used [7] and an estimate of the nominal current that should be needed by the connected Mikroe Sensor board.	This value will be met by our block because we came up with it by using the STM32F072C8T6TR microcontroller's datasheet [7] and liberal estimates of potential nominal currents required by common Mikroe Sensor boards [1].
Ipeak: 170mA	This value was chosen based on the peak current of the microcontroller being used [7] and an estimate of the peak current that should be needed by the connected Mikroe Sensor board.	This value will be met by our block because we came up with it by using the microcontroller's datasheet [7] and liberal estimates of potential peak currents required by common Mikroe Sensor boards [1].
Vmax: 5.2V	Ideal input voltage is 5.0V. This value was chosen to allow for some variance without going too far out of range.	This value will be met by our block because the maximum input voltage for the LD39050PU33R voltage regulator is 5.5V [5], exceeding this value.
Vmin: 4.8V		This value will be met by our block because the LD39050PU33R voltage regulator can output Vin - 0.3V, for Vin values down to 1.5V. So even with 4.8V, the regulator will be able to output 3.3V [5].
mkr_snsr_hrdwr_lggr_data : Output		
Datarate: 1Kbit/sec CAN bus speed	CAN bus communication protocol allows for a data rate of 1 Kbit/sec [15].	This value will be met by our block because the SN65HVD230DR CAN transceiver allows for data rates from 0 to 1Mbps, according to Figure 1 of the datasheet [6]. And the STM
Messages: 8 bits wide	CAN bus communication protocol can use 8-bit messages [15].	This value will be met by our block because the CAN protocol allows for zero to eight bytes of data [15].
Protocol: CAN bus	CAN bus was chosen as the protocol to communicate between the logger block and the Mikroe Sensor Hardware Block because it is well known with good documentation and support, and	The STM32F072C8T6TR microcontroller allows for CAN communication. According to the datasheet, "The CAN is compliant with specifications 2.0A and B (active) with a bit rate up to 1 Mbit/s" [7].

	our project partner is familiar with the protocol.	

4.5.5 Verification Process

1. Connect the power input of the block to a power supply.
2. Connect a Mikroe Click sensor board to the block using the block's MikroBUS(TM) socket. This proves the otsd_mkr_snsr_hrdwr_comm Other property.
3. Program the microcontroller with the code from the Mikroe Sensor Code block using the block's pin header connection. This proves the mkr_snsr_cd_mkr_snsr_hrdwr_data Other property.
4. Confirm the protocol of the last operation was USB 2.0 by looking at the code or by seeing that the microcontroller was reading data through its USB 2.0 interface, verifying the mkr_snsr_cd_mkr_snsr_hrdwr_data Protocol property.
5. The correct operation of the block will also confirm the mkr_snsr_cd_mkr_snsr_hrdwr_data Messages property because the only way it would work is if the code was successfully sent to the microcontroller as a binary code file.
6. Use an oscilloscope to view the datarate between the Mikroe Click Sensor and the Mikroe Sensor Hardware block, verifying the otsd_mkr_snsr_hrdwr_comm Datarate property.
7. Attach the block to a Raspberry Pi using a CAN bus connection setup.
8. Confirm communication between the Mikroe Click sensor board and the block, as well as the block and the Pi by looking at the data transferred to the Pi.
9. This covers the mkr_snsr_hrdwr_lggr_data Protocol property and the otsd_mkr_snsr_hrdwr_comm Messages property.
10. Note the data rate between the Mikroe Sensor Hardware block and the logger using an oscilloscope, proving the mkr_snsr_hrdwr_lggr_data Datarate property.
11. Note the message width by viewing transmitted data on the Pi, proving the mkr_snsr_hrdwr_lggr_data Messages property.
12. Configure the power supply to provide voltage and current meeting the Vmax and Vmin properties of pwr_mkr_snsr_hrdwr_dcpwr. The Mikroe Sensor Hardware block will continue to function as confirmed by the data transferred between the block and the logger. Additionally, viewing the current drawn from the power supply will prove the pwr_mkr_snsr_hrdwr_dcpwr Inominal and Ipeak properties.

4.5.6 References and File Links

4.5.6.1 References (IEEE)

[1] Mikroe, "Sensors: Click boards," mikroe.com, https://www.mikroe.com/click/sensors (accessed Dec. 7, 2023).

[2] STMicroelectronics, "Getting started with STM32F0x1/x2/x8 hardware development," st.com, https://www.st.com/resource/en/application_note/an4080-getting-started-with-stm32f0x1x2x8-hardware-development-stmicroelectronics.pdf, (accessed Feb. 28, 2024).

[3] Phil's Lab, "KiCad 6 STM32 PCB Design Full Tutorial - Phil's Lab #65," youtube.com,

https://www.youtube.com/watch?v=aVUqaB0IMh4&ab_channel=Phil%E2%80%99sLab, (accessed Mar. 6, 2024).

[4] Abracon, “3.3V CMOS Compatible 3.2 x 2.5mm SMD Crystal Oscillator,” abracon.com, https://abracon.com/Oscillators/ASEseries.pdf (accessed Mar. 6, 2024).

[5] STMicroelectronics, “LD39050,” st.com, https://www.st.com/content/ccc/resource/technical/document/datasheet/12/43/cc/93/21/f6/41/66/CD00227527.pdf/files/CD00227527.pdf/jcr:content/translations/en.CD00227527.pdf (accessed Jan. 28, 2024).

[6] Texas Instruments, “SN65HVD23x 3.3-V CAN Bus Transceivers,” ti.com, https://www.ti.com/lit/ds/symlink/sn65hvd230.pdf, (accessed Feb. 28, 2024).

[7] STMicroelectronics, “STM32F072x8 STM32F072xB,” st.com, https://www.st.com/resource/en/datasheet/stm32f072c8.pdf, (accessed Jan. 23, 2024).

[8] STMicroelectronics, “ST-LINK/V2,” mouser.com, https://www.mouser.com/datasheet/2/389/DM00027105-224707.pdf, (accessed Mar. 10, 2024).

[9] Amphenol Communications Systems, “10118194,” amphenol-cs.com, https://www.amphenol-cs.com/media/wysiwyg/files/drawing/10118194.pdf, (accessed Mar. 6, 2024).

[10] STMicroelectronics, “STM32 Nucleo-144 boards,” st.com, https://www.st.com/resource/en/user_manual/dm00244518.pdf (accessed Jan. 28, 2024).

[11] Waveshare, “Waveshare SN65HVD230 CAN Board Info Page,” Amazon.com, https://www.amazon.com/SN65HVD230-CAN-Board-Communication-Development/dp/B00KM6XMXO (accessed Feb. 28, 2024).

[12] Mikroe, “MikroBUS Standard Specifications,” mikroe.com, https://download.mikroe.com/documents/standards/mikrobus/mikrobus-standard-specification-v200.pdf (accessed Dec. 7, 2023).

[13] Mikroe, “MCU card 29 for STM32 STM32F413RH,” mikroe.com, https://www.mikroe.com/mcu-card-29-for-stm32-stm32f413rh (accessed Dec. 7, 2023).

[14] STMicroelectronics, “STM32F0 series Cortex-M0 programming manual,” st.com, https://www.st.com/resource/en/programming_manual/pm0215-stm32f0-series-cortexm0-programming-manual-stmicroelectronics.pdf, (accessed Mar 4, 2024).

[15] KVASER, “The CAN Bus Protocol Tutorial,” kvaser.com, https://www.kvaser.com/can-protocol-tutorial/, (accessed Dec. 7, 2023).

4.5.7 Revision Table

TABLE II
Subsection 4.3 Revision Table

3/10/2024	Julian Henry: Updated wording and specific details after team meeting
3/6/2024	Julian Henry: Finalized updates for final draft, including section 4.3.8
3/2/2024	Julian Henry: Began updates suggested after draft assignment, including consulting the instructor and two TAs in office hours
1/28/2024	Julian Henry: Updated section with new information based on recent design changes
1/23/2024	Julian Henry: Added final touches
1/21/2024	Julian Henry: Updated the section with current information & general edits
12/7/2023	Julian Henry: Drafted section 4.3

4.6 Mikroe Sensor Code

4.6.1 Description

The Mikroe Sensor Code block is used by the system to provide the microcontroller code necessary for the Mikroe Sensor hardware block to function correctly. The code works by first configuring the Mikroe Sensor with the correct settings before taking measurements at a rate of 100Hz. After each measurement, it sends the data over the CAN bus interface which is then received by the logger module. The code is uploaded to the microcontroller using a Serial Wire Debug interface (SWD) provided by an ST-Link programmer device.

4.6.2 Design

This section provides a black box diagram and pseudocode to describe the functionality of the Mikroe Sensor Code block.

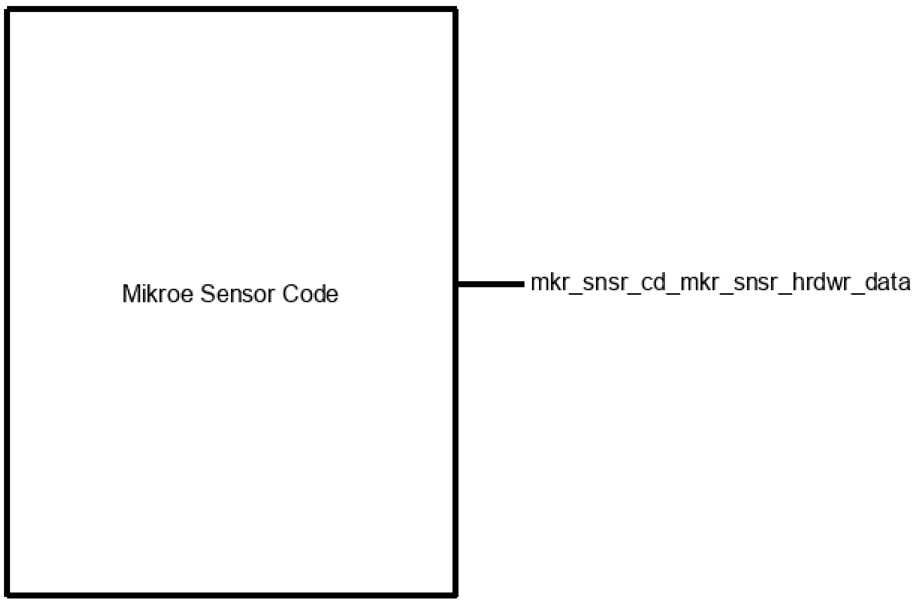


Fig. 1. Black box image of the Mikroe Sensor Code block

```
Import libraries.
Bind interrupts for CAN bus peripheral.
Create objects for CAN bus and SPI connections.
Configure CAN object to use a bitrate of 100Kbps.
Configure SPI object to use a bitrate of 100Kbps, have an idle
CLK level of high, and to capture data on second CLK transition.
Initialize a data buffer.
Configure the sensor through SPI.
Loop
    Read sensor measurement.
    Create a transfer frame for CAN communication protocol.
    Send the transfer frame to the CAN data bus.
    Wait 10 ms.
end
```

Fig. 2. Pseudocode for the Mikroe Sensor Code block

4.6.3 General Validation

The Mikroe Sensor Code block needs to be able to take 8-bit measurements at a rate of 100Hz and send them out over CAN bus. This code also needs to be able to be uploaded to an STM32F072C8T6TR microcontroller through a pin header connection using SWD.

Figure 2 from section 4.6.2 shows how this code will function. By using the embassy framework library, objects can be created for the SPI and CAN bus peripherals. These objects are first created by passing the correct pin values for their respective I/O pins, and any necessary interrupts used for their functionality. They are then configured to communicate at a rate of 100Kbps. After configuring the sensor this, the code infinitely loops by taking sensor measurements and sending them over CAN. It does this by using a 10 ms timer so that measurements can be taken at exactly 100Hz. The loop waits for the timer to complete before sending the read request to the sensor. After receiving the data, it then outputs the data in a CAN transmission frame from the CAN_tx pin on the microcontroller. The timer is then reset and the loop repeats.

4.6.4 Interface Validation

Table I
Mikroe Sensor Code Block interface validation table

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**
mkr_snsr_cd_mkr_snsr_hrdwr_data : Bi-directional		
Messages: Binary code file	Code must be compiled down to machine code to be run on the microcontroller.	Binary instructions stored in flash memory are used to control the microcontroller [2].
Other: Pin header connection	Pin headers were chosen for an easy connection method using an ST-Link programmer.	St-Link connector uses pin connections to program the microcontroller [3].
Protocol: Serial Wire Debug	SWD is used by the ST-Link programmer.	SDW sends a data stream over two wires to program ARM CPUs such as STM32 microcontrollers [4].

4.6.5 Verification Process

7. Connect the ST-link programmer between the computer and the pin headers of the mkr_snsr_cd_mkr_snsr_hrdwr_data interface.
8. Flash the code onto the microcontroller.
9. Connect the ADC hardware block to the logger block and confirm that the logger is receiving data.

4.6.6 References and File Links

[1] Embassy Project Contributors, “embassy_stm::can,” docs.embassy.dev, https://docs.embassy.dev/embassy-stm32/git/stm32f413rh/can/index.html, (accessed Dec. 6, 2023).

[2] STMicroelectronics, “STM32F0 series Cortex-M0 programming manual,” st.com, https://www.st.com/resource/en/programming_manual/pm0215-stm32f0-series-cortexm0-programming-manual-stmicroelectronics.pdf, (accessed Mar 4, 2024).

[3] Adafruit, "ST-Link STM8/STM32 v2 Compatible Programmer & Emulator," adafruit.com, https://www.adafruit.com/product/2548, (accessed Mar 4, 2024).

[4] Silicon Labs, "Serial Wire Debug (SWD), https://community.silabs.com/s/article/serial-wire-debug-swd-x?language=en_US, community.silabs.com, (accessed Mar 4, 2024).

4.6.7 Revision Table

TABLE II
Subsection 4.6 Revision Table

3/8/2024	Alexander Gibson: Created and filled out sections 4.6.1 - 4.6.7

4.7 Logger

4.7.1 Description

The logger sub-module will collect data from the sensors and store them into a database. In this application, a Raspberry PI 4B will be used as the logger device. The Raspberry PI 4B will be powered ideally at 5V however, can be operated at a minimum voltage of 4.8V and a maximum of 5.2V. An RS485 CAN Hat will interface with the desired sensor modules using CAN communication. Using low-speed CAN mode, the logger will record incoming sensor data at a maximum rate of 100Kbi t/sec. The logger will take in incoming sensor messages ranging from 8 bits to 24 bits depending on the sensor module. Incoming GPS and MIKROE sensor messages will be in the size of 8 bits while the ADC sensor will send 24-bit messages. A local database located on the pi will store the incoming sensor data using SQLite, making a query after every new message sent. The logger will continue to read and store sensor data until reaching a specified time. Although most of the system will be programmed in rust, the PI will be programmed in python.

4.7.2 Design

This section provides the black box diagram of the logger sub-module and pseudocode to describe the block's function.

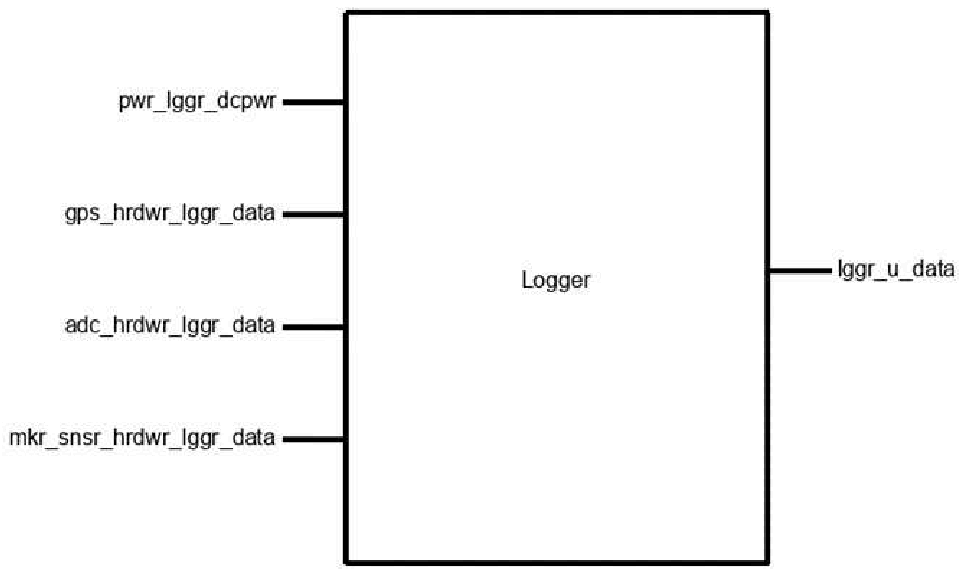


Fig 1. Black box image of logger sub-module

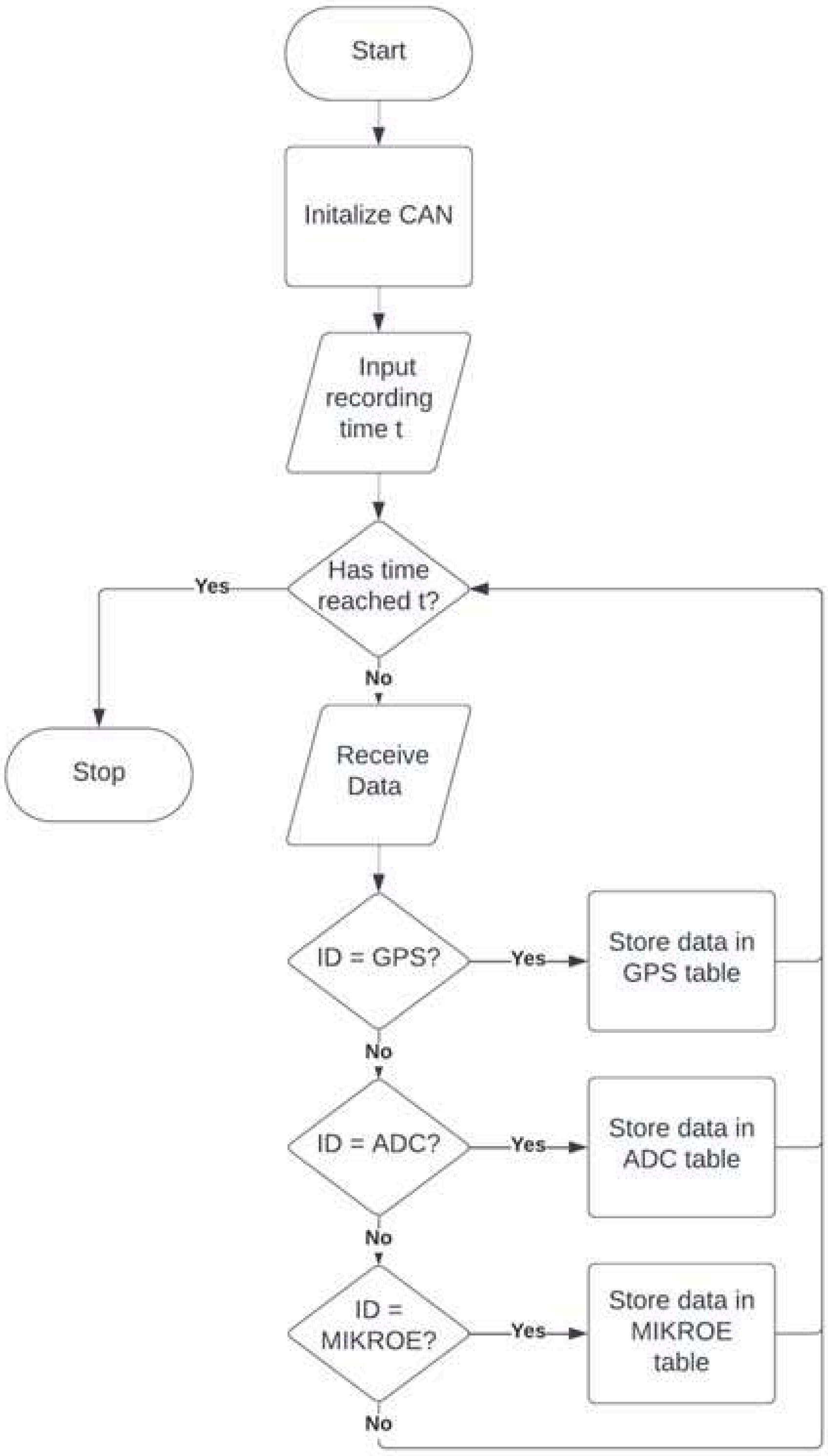


Fig 2. Pseudocode of logger sub-module

4.7.3 General Validation

The logger sub-module needs to be able to take values from each sensor module and output that data into a database. A Raspberry Pi 4b will be used as the logger. The Pi provides a cheap and cost-effective logger that allows for quick and easy setup. The Pi must be supplied with 5V of power to operate correctly [1]. However, the Pi must be able to operate at a voltage input between 4.8V and 5.2V. Each sensor module will interface to the pi through CAN bus protocol. Each sensor module will communicate with the pi at a rate of 100Kbit/sec using low speed CAN mode. The Pi must take incoming messages ranging from the size of 8 bits to 24 bits. This range is due to the GPS and MIKROE modules outputting 8-bit messages, and the ADC module outputting 24-bit messages.

Figure 2 above shows the pseudocode that the logger will generally be running. The logger will be programmed with Python for simplicity and to focus on other areas of the project. At program start, the logger will initialize the CAN bus at a bit rate of 100Kbits/sec. The program will then wait for a float input ranging from 0-60 that specifies recording time in minutes. After a time t has been specified, the program will check if time t has passed, if not the logger will now start to receive incoming data through CAN. All incoming CAN frames will contain an arbitrary ID corresponding to different tables in the database. If there is a matching ID, the incoming data will be stored in that specified table via SQLite. Each stored data member is specified as a float (or REAL in SQL) for accuracy and easy integration with incoming data. After each incoming message has been checked and optionally stored, the program will loop back to receive the next message or quit if time has reached t.

The database platform used to manage and store sensor module data is SQLite. This database is lightweight and stored locally instead of depending upon a separate server. Using a local database allows for zero latency which removes any need for efficient queries [2]. Since the UI block will be using the database to display incoming sensor data in real-time, there will never be an occasion where the UI is unable to load fresh data than is being recorded.

An RS485 CAN HAT will be used to enable CAN communication on the Pi. Our project partner selected the CAN HAT due to simplicity, requiring little to no setup to operate with the Raspberry Pi. The connected CAN bus will operate in low-speed mode for three reasons: improved signal quality, fault tolerance, and a maximum data rate of 125Kbit/sec [3]. In this application, the system will only achieve a maximum data rate of 100Kbit/sec. In low-speed mode, each CAN node carries its own termination whereas in high-speed, there are only terminations at the beginning and end of the bus [3]. During high-speed operation however, if a connected CAN node fails, the entire bus fails. Low speed does not have this same issue due to terminations at each node. Since our design aims to be modular and “plug and play” various sensors, the fault tolerance of low-speed mode works perfectly for this application.

An alternative to this block would be running CAN in high-speed mode for the bus. Instead of having a termination at each CAN node on the bus, a pair of 120 Ohm resistors would be put on both ends of the bus [3]. Since our design is modular, there would need to be an additional module that is strictly for termination. This module would consist of a PCB with a 120 Ohm resistor attached that would follow the modular design of the sensor modules. This would be inserted at the end of the bus after all sensors have been plugged in. Another alternative would be to supply the database of the logger on a server instead of locally. This would allow for access to data from any internet connected device without the need for the logger to be nearby. The database could also be accessed by more than one user allowing for easier sharing of data and storage.

4.7.4 Interface Validation

Interface Property	Why is this interface this value?	Why do you know that your design details <u>for this block</u> above meet or exceed each property?
Pwr_lggr_dcpwr: Input		
Inominal: 1A	No peripherals other than CAN Hat are connected meaning only the standard Pi and Hat are drawing current.	Standard USB Type-C connections supply a maximum of 3A, exceeding this property [4].
Ipeak: 1.25A	Peak current drawn during operation, usually during startup processes.	Standard USB Type-C connections supply a maximum of 3A, exceeding this property [4].
Vmax: 5.2V	Ideal input is 5.0V. This allows for variance and proper operation without damaging the device [1].	Standard USB Type-C connections supply a voltage of 5V [4].
Vmin: 4.8V	Ideal input is 5.0V. This allows for variance and proper operation without damaging the device [1].	Standard USB Type-C connections supply a voltage of 5V [4].
gps_hrdwr_lggr_data: Input		
Datarate: 100Kbit/sec CAN bus speed	System designed with CAN bus in low-speed mode. In low speed, CAN has a maximum data rate of 125Kbit/sec [3]. This value allows for good signal quality while keeping under maximum.	The RS485 has a switch to drop the 120 Ohm termination resistor to enable low speed CAN [5].
Messages: 8 bits	Output data size of GPS sensor	A standard CAN frame can transmit data with a length up to 64 bits [3]. This exceeds the required amount.
Protocol: CAN bus	Was chosen to communicate between sensor blocks and logger. Selected by our project partner as they are familiar with this protocol.	The RS485 CAN Hat enables CAN communication on the Pi [5].
mkr_snsr_hrdwr_lggr_data: Input		
Datarate: 100Kbit/sec CAN bus speed	System designed with CAN bus in low-speed mode. In low speed, CAN has a maximum data rate of 125Kbit/sec [3]. This value allows	The RS485 has a switch to drop the 120 Ohm termination resistor to enable low speed CAN [5].

	for good signal quality while keeping under maximum.	
Messages: 8 bits wide	Output size of expected MIKROE sensor data.	A standard CAN frame can transmit data with a length up to 64 bits [3]. This exceeds the required amount.
Protocol: CAN bus	Was chosen to communicate between sensor blocks and logger. Selected by our project partner as they are familiar with this protocol.	The RS485 CAN Hat enables CAN communication on the Pi [5].
adc_hrdwr_lggr_data: Input		
Datarate: 100Kbit/sec CAN bus speed	System designed with CAN bus in low-speed mode. In low speed, CAN has a maximum data rate of 125Kbit/sec [3]. This value allows for good signal quality while keeping under maximum.	The RS485 has a switch to drop the 120 Ohm termination resistor to enable low speed CAN [5].
Messages: 24 bits wide	Output size of expected ADC sensor data.	A standard CAN frame can transmit data with a length up to 64 bits [3]. This exceeds the required amount.
Protocol: CAN bus	Was chosen to communicate between sensor blocks and logger. Selected by our project partner as they are familiar with this protocol.	The RS485 CAN Hat enables CAN communication on the Pi [5].
Lggr_u_data: Output		
Messages: data from all sensors	Data being outputted is coming from the incoming data of sensors.	Incoming CAN data will be stored in the database.
Messages: float type	Data being stored is as a float value for accuracy and ease of integration for all sensor blocks.	In the program code, when a data message is stored, it will be specified as a float type.
Other: SQLite Query	Database used is SQLite3. SQLite allows for simple database management locally. A query from the database to the UI is used to present information on screen.	SQLite3 is being used to manage the database on the Pi. A SELECT query fetches the data from the database and returns it [6].

4.7.5 Verification Process

1. Connect test leads from power supply bench to USB-C breakout board.

2. Connect USB-C cord from breakout board to the pwr_lggr_dcpwr interface.

3. Set power supply voltage to 4.8V and output power.

4. Connect jumper wires from a dev board capable of CAN communication to the respective CAN_H and CAN_L, this CAN Bus will make up the gps_hrdwr_lggr_data, adc_hrdwr_lggr_data, and mkr_snsr_hrdwr_lggr_data interfaces.

5. Flash dev board with testbench script and confirm bitrate of 100Kbits/sec.

6. Run testbench python script on logger and confirm bitrate of 100Kbits/sec.

7. Confirm on the logger terminal that gps_hrdwr_lggr_data and mkr_snsr_hrdwr_lggr_data interfaces have incoming data that is of 8 bits each.

8. Confirm on the logger terminal that adc_hrdwr_lggr_data interface has incoming data that is of 24 bit length.

9. Open a separate terminal on the logger and run command “SELECT * from adc_data;”, confirm that lggr_u_data has performed a successful SQLite query and is being updated with incoming data.

10. Run command “SELECT typeof (*) from adc_data;”, confirm that lggr_u_data has returned type REAL (a.k.a float type in SQL).

11. Repeat steps 4-10, setting the power supply voltage to 5.2V.

4.7.6 References and File Links

[1] Raspberry Pi (Trading) Ltd, "Datasheet Raspberry Pi 4 Model B," 21 June 2019. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf.

[2] K. Dodds C, "Why you should probably be using SQLite," Epic Web Dev, 25 October 2023. [Online]. Available: https://www.epicweb.dev/why-you-should-probably-be-using-sqlite.

[3] M. Falch, "CAN Bus Explained - A Simple Intro [2023]," CSS Electronics, 2022 April. [Online]. Available: https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial#:~:text=Low%20speed%20CAN%20bus%3A%20This,has%20it's%20own%20CAN%20termination.

[4] N. Enos and G. Brian, "A Primer on USB Type-C® and USB," March 2022. [Online]. Available: https://www.ti.com/lit/wp/slyy109b/slyy109b.pdf?ts=1710092743274&ref_url=https%253A%252F%252Fwww.google.com%252F.

[5] Waveshare, "RS485 CAN HAT," [Online]. Available: https://files.waveshare.com/upload/2/29/RS485-CAN-HAT-user-manuakl-en.pdf.

[6] Python, "sqlite3 — DB-API 2.0 interface for SQLite databases," 10 March 2024. [Online]. Available: https://docs.python.org/3/library/sqlite3.html.

4.7.7 Revision Table

3/10/2024	Jaxon Young: Added references and file links, revised for formatting.
3/5/2024	Jaxon Young: Replaced fig. 2 diagram with pseudocode, updated general validation, refined interface validation, added more detail to verification process.
12/7/2023	Jaxon Young: Section updated, added block draft information
12/6/2023	Jaxon Young: Initial Section Creation

4.8 UI

4.8.1 Description
The UI module will help the user interface with the logger and sensors of the system. Programmed in rust using the EGUI library, the UI will fetch data from the logger modules database and manipulate data based on the user’s preference. The user will be able to select which sensors they would like to view/manipulate the data of and specify the amount of time they would like the logger to record data for. During data collection, the user will have two options on how they would like to view the incoming recorded data. An option to output data into a table or graph vs time will be given. If the option of data vs time is selection, the user can specify a period they would like to view on the graph.

4.8.2 Design
This section provides the black box diagram of the UI module and the associated flowchart with it.

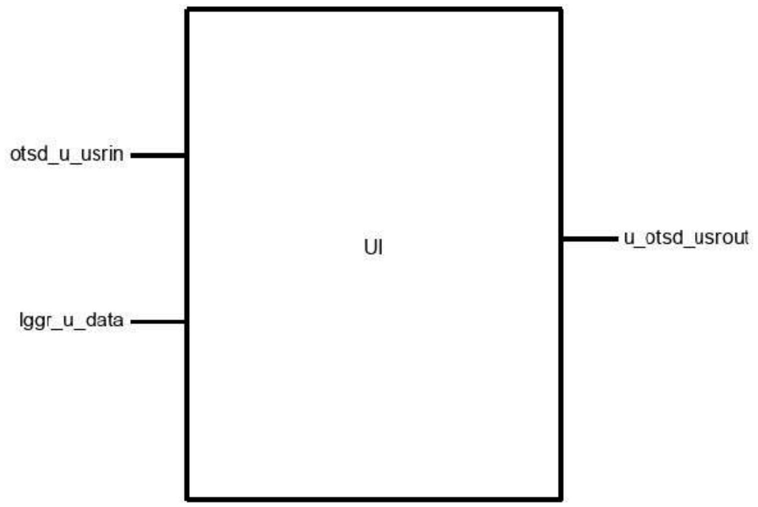


Fig. 1. Black box diagram of UI

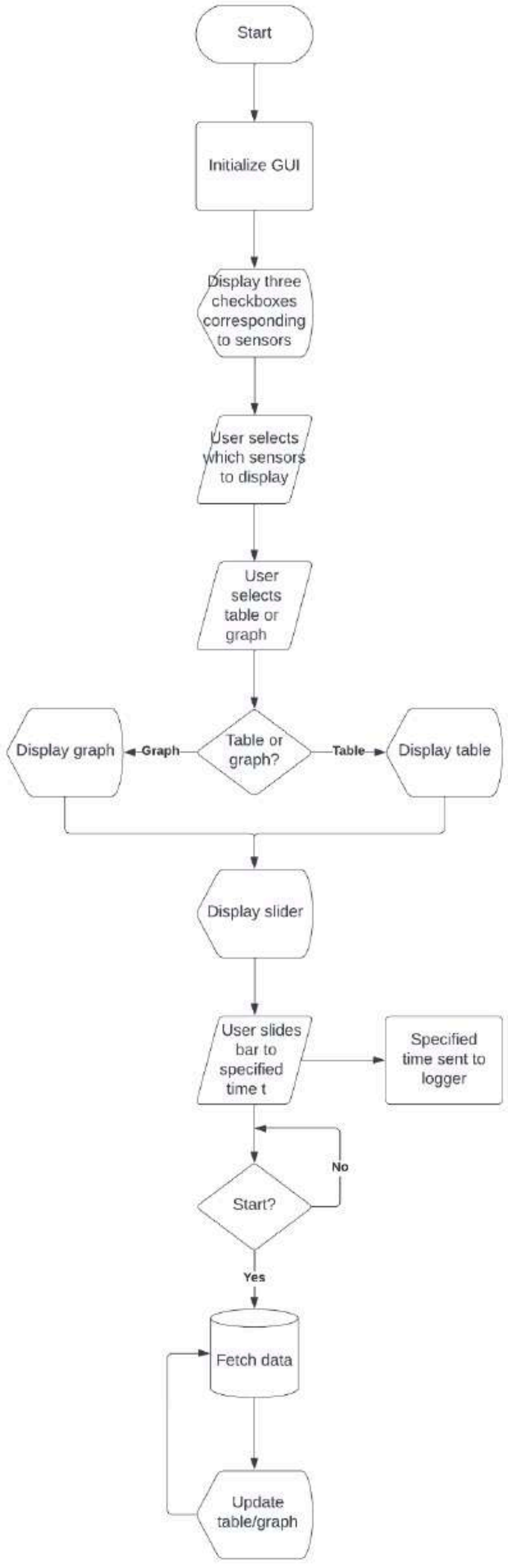

Start
Initialize GUI
Display three checkboxes corresponding to sensors
User selects which sensors to display
User selects table or graph
Table or graph?
Graph
Table
Display graph
Display table
Display slider
User slides bar to specified time t
Specified time sent to logger
Start?
No
Yes
Fetch data
Update table/graph

Fig. 2. UI Pseudocode/Process Flowchart

4.8.3 General Validation

The reason for choosing EGUI as our GUI library was for a few reasons. Firstly, EGUI provides a simple library to program with for rust. As most of our project aims to be programmed in rust, this allows for a simple GUI integration with the rest of the project. Secondly, the GUI needs to be running fast and efficiently to correctly display the incoming data to the user. The database will be storing data at a maximum rate of 100Hz. Although the user will not see data being updated at a rate of 100Hz, the system will still be receiving information at this rate. At minimum, the UI must at least update every 30 seconds to provide mostly accurate and real-time data. However, EGUI aims to achieve around 60Hz, the average refresh rate of a typical monitor [1]. It is crucial that the system does not get overwhelmed with querying the database, EGUI allows for a simple and fast immediate mode GUI for this occasion [1].

In figure 2 above, the rough pseudocode/process flowchart is shown. This roughly describes how the UI will operate when finally implemented. At start, the UI will initialize and present the user with three checkboxes on the screen. These three checkboxes will be interactable by the user and represent the sensor selection. As the user marks the sensors they would like to display, the UI records this as a member variable and stores it. For each sensor selection made, the user will be displayed the choice to select a table or graph. After selections have been made, the UI will display the requested table or graph. A “slider” bar will display to allow the user to record an integer for the UI which corresponds to the length of time they would like to record data for. After the user confirms the location on the “slider” it will record the integer in that position and send that value to the logger. At this point the user will be displayed a start button. If that button is pressed, the UI will query from the database and update the corresponding tables/graphs present. This will continue until the specified time has passed.

An alternative to this block would be to set up a remote server that the UI is hosted on. This would allow for a wider range of devices to connect and interact with the UI like a smartphone. For this to work, however, the database must be stored on a server as well instead of locally. The logger must also have an internet connection to be able to send data to the database. This would also allow for a larger size database that is not limited to the storage size of the logger drive. Stored data on a server would ensure that data is not lost in the event of the logger breaking.

4.8.4 Interface Validation

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**
Otsd_u_usrin: Input		
Type: interactable "slider" bar to record integer	Provides user with the ability to specify length of data recording in visually appealing way	A “slider” bar will be programmed to record the integer the user lands on.

Type: three interactable checkboxes lined up 1x3 to record member variable	Provides user with the ability to select sensors in a simple way and provides data for the UI to present	Three boxes will be programmed to record member variables the user clicks on.
Usability: understandable by 9/10 users	Easy to use by the average person	A google form will be handed to the user testing the UI. 9/10 users tells us that our UI is easy to use.
U_otsd_usrout: Output		
Other: UI updates at least every 30 seconds	Provides the user with accurate data readings.	EGUI provides an immediate mode where the UI updates every single frame [1]. The target response is 60Hz exceeding the requirement.
Type: Graph of data vs time	Presents the data to the user in a visually appealing and organized form.	In the code, data is presented on a line plot which will be data over the time.
Usability: Human-readabl e numeric output used to give specific values	The user is able to understand the data that is output	In the code, all data presented is specified as a float type which is human-readable.
Lggr_u_data: Input		
Messages: float type	Incoming sensor data is stored as a float type. The UI presents data with a float value for compatibility and accuracy of values	The UI code is only programmed to take in float values.
Messages: data from all sensors	The data incoming is from the connected sensor modules	The UI code will only grab data from the database which should only contain sensor data.
Other: SQLite Query	Database used to store sensor data uses SQLite	To fetch data from an SQLite database, a query is used.

4.8.5 Verification Process

1. Provide power to raspberry pi and launch the database testbench script (this will store random values in the database.

2. Launch GUI on raspberry pi.

3. After initialization, select ADC and choose graph, confirm that three checkboxes appear and that when clicked, a member variable is stored, this confirms the second property on otsd_u_usrin.

4. Confirm that a slider bar appears after choosing graph and slide bar to 2 and that this is being stored as an integer. This confirms the first property on otsd_u_usrin.

5. Click start and observe data being recorded.

6. Confirm u_otsd_usrout interface property of data vs time by viewing graph.

7. Confirm u_otsd_usrout interface property of UI updates at least every 30 seconds by viewing new data withing 30 seconds.

8. Confirm u_otsd_usrout interface property of human readable output by verifying that the number is readable.

9. After 2 minutes of “data recording” confirm that incoming data is of float type and queried from database this confirms lggr_u_data interface properties.

10. Answer questionnaire stating if the UI was easy to understand this will confirm the final property in otsd_u_usrin.

4.8.6 References and File Links

[1] emlik, "egui: an easy-to-use GUI in pure Rust," 7 March 2024. [Online]. Available: https://github.com/emilk/egui.

4.8.7 Revision Table

3/8/2024	Jaxon Young: Updated verification process, added pseudocode process explanation in general validation, added alternative to block in general validation, sources updated.
2/18/2024	Jaxon Young: Updated interface properties to match portal, added flowchart
12/7/2023	Jaxon Young: Updated section, drafted section
12/6/2023	Jaxon Young: Initial section creation

4.9 Power

4.9.1 Description

This section will discuss the contents of the power block. This block will provide power to the entire system including the GPS block, the ADC block, the Mikroe sensor block, and the logger block. The power block will be rechargeable and able to support the system with all blocks using their peak current for an hour on one charge. This means that it must have a charge capacity greater than the sum of all the peak currents required by every block multiplied by one hour. According to the block interfaces, each of the three sensor blocks has a peak input current of 170mA, and the logger has a peak of 1.75A. This gives: (170mA + 170mA + 170mA + 1.75A)*(1 hour) = 2.26A*1h = 2.26 Ah, or 2260 mAh. To accomplish this, the power block features a commercially available portable charger.

4.9.2 Design

The power block comprises a commercially available portable charger chosen for its safety, reliability, and capacity. Its model number is PB-HOUMI-1W. It must also include cables used to connect it to the different interfaces. This includes a charging cable to input power to the block, as well as four or more output cables used to connect the block to the rest of the system that it powers. See figures 1 and 2 below for the black box image of this block, as well as a diagram showing the physical setup of the block.

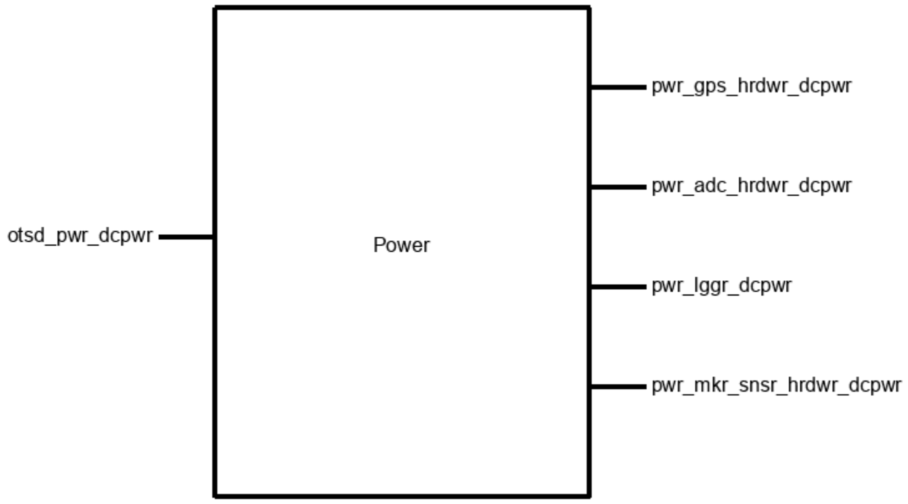


Fig. 1. Power block black box image

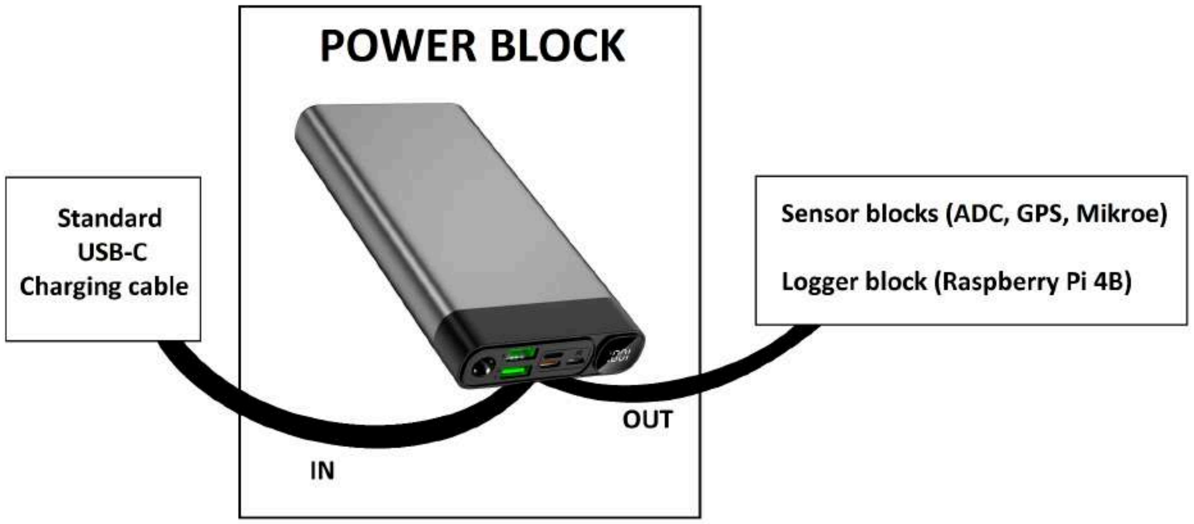

POWER BLOCK
Standard
USB-C
Charging cable
Sensor blocks (ADC, GPS, Mikroe)
Logger block (Raspberry Pi 4B)
IN
OUT

Fig. 2. Power block physical setup using portable charger PB-HOUMI-1W

As Fig. 2. Shows, the portable charger supplies power to the logger block, the ADC block, and the Mikroe Sensor block. Due to the number of outputs on our selected portable charger, we may have to implement the use of a USB splitter cable to supply power for the required amount of blocks.

4.9.3 General Validation

This section will describe why our chosen design details fit the needs of the system as a whole, especially with regards to the block's function, cost, and engineering time. This section will also include an alternate solution for this block.

The design presented was chosen to meet the needs of our system and work reliably in conditions that the system will face, namely strapped to a go-kart. The block being a commercially available portable charger was a decision made with the project partner to ensure both that the system was as safe as it could be and that there could be plenty of time to work on the other aspects of the system that don't already have affordable, easy to use, and commercially available counterparts.

The power block must be able to supply power to the logger block, which will need around 1.75A, the GPS block, needing a max of 170mA, the ADC block which should only need up to 170mA, and the Mikroe Sensor block, of which the current draw will depend greatly on the Mikroe Click sensor board connected, but shouldn't take more than 170mA. To calculate the minimum required capacity to supply the mentioned blocks for an hour, the current is multiplied by the time. This gives: (170mA + 170mA + 170mA + 1.75A)*(1 hour) = 2.26A*1h = 2.26 Ah, or 2260 mAh. Fortunately, the specific portable charger that will be used for the system has a capacity of 30Ah. Since the charge capacity of our power block is over ten times the required amount, it should easily be able to support our system. Additionally, the chosen portable charger is small enough to be mobile and fit with the rest of the system, comes with a metal aluminum alloy casing, ensuring that it is resistant to falling, friction and anti-flaming, and includes three output ports and three input ports including a Micro-USB input port, a Lighting input port, a Type-C input/output port, and two USB-A output ports [1]. These interfaces will ensure that the connections from the power block to the rest of the system will be safe, secure, and that the cables will be easily available. However, since there are only three output ports and four blocks that need to be charged, we will need to employ the use of a USB splitter or device that accomplishes the same purpose. This will be an easy fix as that sort of product is cheap and easily available [2].

An alternate solution for this block is to create a custom PCB that includes power distribution circuitry, an appropriate secondary battery, and safety measures that ensure our system isn't a danger to its users. This option was dismissed by our project partner due to the extra time, effort, cost, and potential unreliability that it would add to our project.

4.9.4 Interface Validation

This section includes a table formatted to show each interface and each interface property relating to the power block, as well as an explanation of why the interface property values were chosen and why the block will work with that value.

TABLE I
Power block interface validation table

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**
otsd_pwr_dcpwr : Input		
Inominal: 1A	This value was chosen based on the information on the product page [1].	This value will be met by our block because we came up with it by using the portable charger’s product page [1].
Ipeak: 1.75A	This value was chosen based on the information on the product page [1].	This value will be met by our block because we came up with it by using the portable charger’s product page [1] and tested it in the lab.
Vmax: 5.25V	This value was chosen based on the information on the product page [1].	This value will be met by our block because it was tested it in the lab.
Vmin: 4.75V	This value was chosen based on the information on the product page [1].	This value will be met by our block because it was tested it in the lab.
pwr_gps_dcpwr : Output		
Inominal: 33mA	This value was chosen based on the nominal current of the microcontroller being used [3] and an estimate of the nominal current that should be needed by the connected GPS [4] .	This value will be met by our block because USB 2.0 can deliver 1.5A [5], far exceeding the required Inominal.
Ipeak: 170mA	This value was chosen based on the peak current of the microcontroller being used [3] and an estimate of the peak current that should be needed by the connected GPS [4].	This value will be met by our block because USB 2.0 can deliver 1.5A [5], far exceeding the required Ipeak.
Vmax: 5.2V	This interface will use a USB 2.0 capable cable, which operates at 5V but allows for some wiggle room [5].	This value will be met by our block because USB 2.0 operates at 5V [5].
Vmin: 4.8V	This interface will use a USB 2.0 capable cable, which operates at 5V but allows for some wiggle room [5].	This value will be met by our block because USB 2.0 operates at 5V [5].
pwr_mkr_snsr_dcpwr : Output		
Inominal: 33mA	This value was chosen based on the nominal current of the microcontroller	This value will be met by our block because USB 2.0 can

	being used [3] and an estimate of the nominal current that should be needed by the connected Mikroe Sensor board.	deliver 1.5A [5], far exceeding the required Inominal.
Ipeak: 170mA	This value was chosen based on the peak current of the microcontroller being used [3] and an estimate of the peak current that should be needed by the connected Mikroe Sensor board.	This value will be met by our block because USB 2.0 can deliver 1.5A [5], far exceeding the required Ipeak.
Vmax: 5.2V	This interface will use a USB 2.0 capable cable, which operates at 5V but allows for some wiggle room [5].	This value will be met by our block because USB 2.0 operates at 5V [5].
Vmin: 4.8V	This interface will use a USB 2.0 capable cable, which operates at 5V but allows for some wiggle room [5].	This value will be met by our block because USB 2.0 operates at 5V [5].
pwr_adc_dcpwr : Output		
Inominal: 33mA	This value was chosen based on the nominal current of the microcontroller being used [3] and an estimate of the nominal current that should be needed by the connected ADC [6] .	This value will be met by our block because USB 2.0 can deliver 1.5A [5], far exceeding the required Inominal.
Ipeak: 170mA	This value was chosen based on the peak current of the microcontroller being used [3] and an estimate of the peak current that should be needed by the connected ADC [6].	This value will be met by our block because USB 2.0 can deliver 1.5A [5], far exceeding the required Ipeak.
Vmax: 5.2V	This block will be powered by a USB 2.0 capable cable, which operates at 5V but allows for some wiggle room [5].	This value will be met by our block because USB 2.0 operates at 5V [5].
Vmin: 4.8V	This block will be powered by a USB 2.0 capable cable, which operates at 5V but allows for some wiggle room [5].	This value will be met by our block because USB 2.0 operates at 5V [5].
pwr_lggr_dcpwr : Output		
Inominal: 1A	Supply powering a Raspberry Pi 4 Model B at 5V must be at least be capable of supplying 2.5A [7], however, a nominal current was not shared. After testing, 1.5A seemed to be a reasonable value.	This value will be met by our block because USB-C can deliver 3A [5], exceeding the required Inominal.

Ipeak: 1.25A	Supply powering a Raspberry Pi 4 Model B at 5V must be capable of supplying 3A max [7], thus 3A was chosen as the peak current.	This value will be met by our block because USB-C can deliver 3A [5], meeting the required Ipeak.
Vmax: 5.2V	Maximum input voltage for a Raspberry Pi 3 B+ is 5.25V, the same is true for the Raspberry Pi 4 Model B [8].	This value will be met by our block because USB-C can operate at 5V [5].
Vmin: 4.8V	Minimum input voltage for a Raspberry Pi 3 B+ is 4.75V, the same is true for the Raspberry Pi 4 Model B [8].	This value will be met by our block because USB-C can operate at 5V [5].

4.9.5 Verification Process

1. Use a benchtop power supply to supply the portable charger with voltage and current meeting the interface properties of otsd_pwr_dcpwr. The portable charger will show it is charging for each value, proving the Inominal, Ipeak, Vmax, and Vmin properties.
2. Connect a USB-C output pin from the portable charger to an electrical load. Configure the load to require voltage and current meeting the interface properties of pwr_lggr_dcpwr. The power block will source each value as confirmed by the electrical load, proving the Inominal, Ipeak, Vmax, and Vmin properties.
3. Connect a USB 2.0 output pin from the portable charger to an electrical load. Configure the load to require voltage and current meeting the interface properties of pwr_adc_dcpwr. The power block will source each value as confirmed by the electrical load, proving the Inominal, Ipeak, Vmax, and Vmin properties.
4. Connect a USB 2.0 output pin from the portable charger to an electrical load. Configure the load to require voltage and current meeting the interface properties of pwr_gps_dcpwr. The power block will source each value as confirmed by the electrical load, proving the Inominal, Ipeak, Vmax, and Vmin properties.
5. Connect a USB 2.0 output pin from the portable charger to an electrical load. Configure the load to require voltage and current meeting the interface properties of pwr_mkr_snsr_dcpwr. The power block will source each value as confirmed by the electrical load, proving the Inominal, Ipeak, Vmax, and Vmin properties.
6. Tests may be conducted concurrently to prove that the block can output the sum of all the Inominal, Ipeak, Vmax, and Vmin properties at the same time.

4.9.6 References and File Links

4.9.6.1 References (IEEE)

[1] "Portable Charger Info Page," Amazon, https://www.amazon.com/gp/product/B07PG6C5C7 (accessed Dec. 6, 2023).

[2] "USB splitter Search," Amazon, https://www.amazon.com/s?k=USB+splitter&crid=1TAOAX32KJ6GS&sprefix=usb+splitte%2Caps%2C137&ref=nb_sb_noss_2 (accessed Jan. 28, 2024).

[3] STMicroelectronics, "STM32F072x8 STM32F072xB," st.com, https://www.st.com/resource/en/datasheet/stm32f072c8.pdf, (accessed Jan. 23, 2024).

[4] u-blox, “NEO-M9N Integration Manual,” u-blox.com, https://content.u-blox.com/sites/default/files/NEO-M9N_Integrationmanual_UBX-19014286.pdf, (accessed Nov. 28, 2023).

[5] “USB,” Wikipedia, https://en.wikipedia.org/wiki/USB (accessed Jan. 28, 2024).

[6] Analog Devices, “AD7124-8 (Rev. F),” analog.com, https://www.analog.com/media/en/technical-documentation/data-sheets/ad7124-8.pdf, (accessed Nov. 28, 2023).

[7] “DATASHEET - Raspberry Pi,” Raspberry Pi 4 Model B Datasheet, https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf (accessed Dec. 6, 2023).

[8] T. Thurium, “Three easy ways to brick a Raspberry Pi,” Twilio Blog, https://www.twilio.com/blog/3-ways-brick-raspberry-pi (accessed Dec. 6, 2023)

4.9.7 Revision Table

TABLE II
Subsection 4.6 Revision Table

1/28/2024	Julian Henry: Updated section with new current information
1/21/2024	Julian Henry: Updated section with current information
12/7/2023	Julian Henry: Section 4.6 minor revisions and updates
12/6/2023	Julian Henry: Drafted section 4.6

5. System Verification Evidence

5.1 Universal Constraints

The Universal Constraints are a set of specific boundaries that every Capstone project must adhere to, ensuring consistent quality across each team’s work. This section will explain how our project meets all of the Universal Constraints.

5.1.1 The system may not include a breadboard

The system is physically composed of one or more sensor blocks connected to a Raspberry Pi. Each sensor and its associated circuitry is housed on one or more PCBs. The use of a breadboard would compromise the system’s functionality due to the non-permanency of its connections, so breadboards were excluded in the final system. The system can be seen below in figure 1.

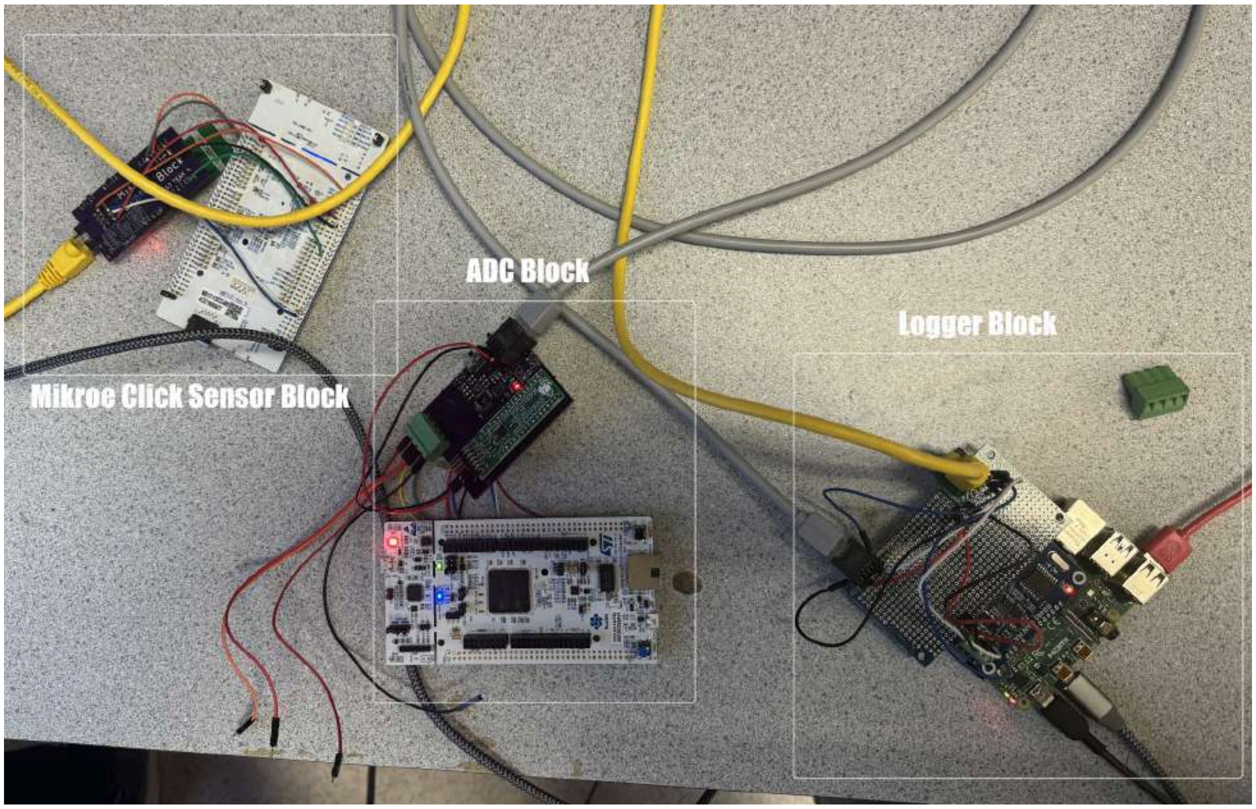


Fig. 1. Physical system

5.1.2 The final system must contain a student-designed PCB
The system includes multiple student-designed PCBs that were designed to interface sensors to microcontrollers that output data over a CAN bus connection using RJ-45 connectors. Unfortunately, we were unable to successfully program the microcontrollers on the PCBs, so we had to connect the student-designed PCBs to microcontroller development boards as shown below.

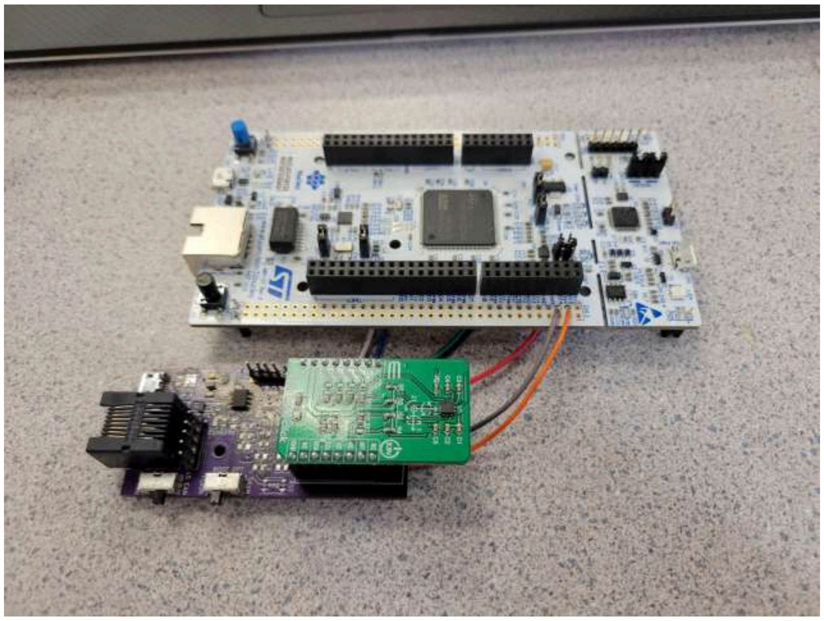


Fig. 2. Mikroe Click Sensor Block Custom PCB (bottom) connected to a microcontroller development board (top).

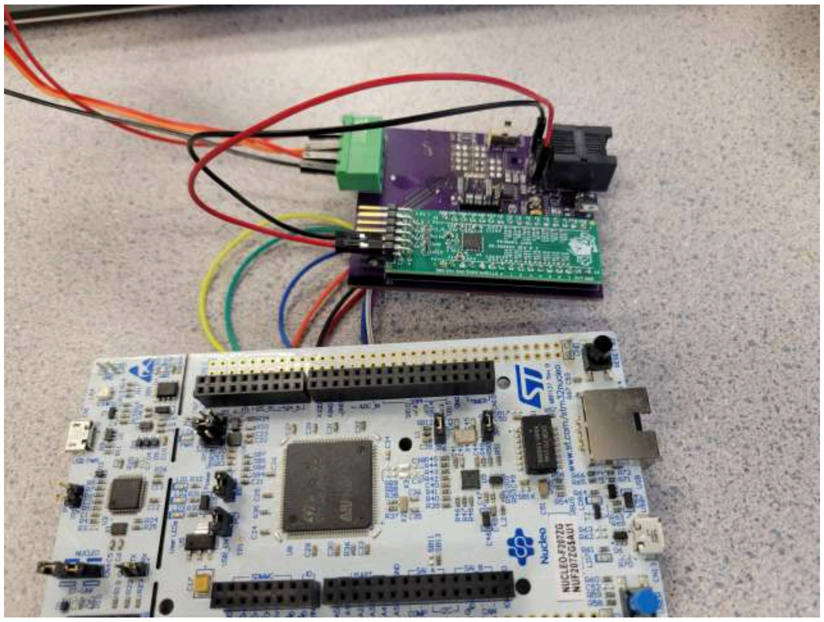


Fig. 3. ADC Block Custom PCB (top) connected to a microcontroller development board (bottom).

5.1.3 All connections to PCBs must use connectors.
The system is modular, as described by the requirement in section 5.2.3 below. Thus, the sensor blocks which mainly comprise student-designed PCBs, must be able to be connected and disconnected by the end user. To achieve this, the initial PCB design used connectors to connect the sensor blocks to the logger block. Unfortunately, the original sensor module PCBs could not be successfully programmed, so wires needed to

be soldered between the sensor PCBs and microcontroller dev boards to achieve a functioning system.

5.1.4 All power supplies in the system must be at least 65% efficient.
The system uses a commercially available portable charger for a power supply. Any portable charger available on the market should be much more than 65% efficient [1]. This is confirmed on the amazon page of the portable charger we are using [2].

5.1.5 The system may be no more than 50% built from purchased 'modules.'
The physical system comprises three sensor hardware blocks, three sensor code blocks, a logger block, and a power bank. Each sensor hardware block includes a custom PCB. Each code block will also be custom. The logger block includes a Raspberry Pi and a CAN hat, and the power bank was also be purchased. Thus, the system is less than 50% built from purchased ‘modules’ and passes this constraint.

5.2 Requirements
This section contains all of the project requirements, including the expected verification process and the evidence of completion [3, 4, 5].

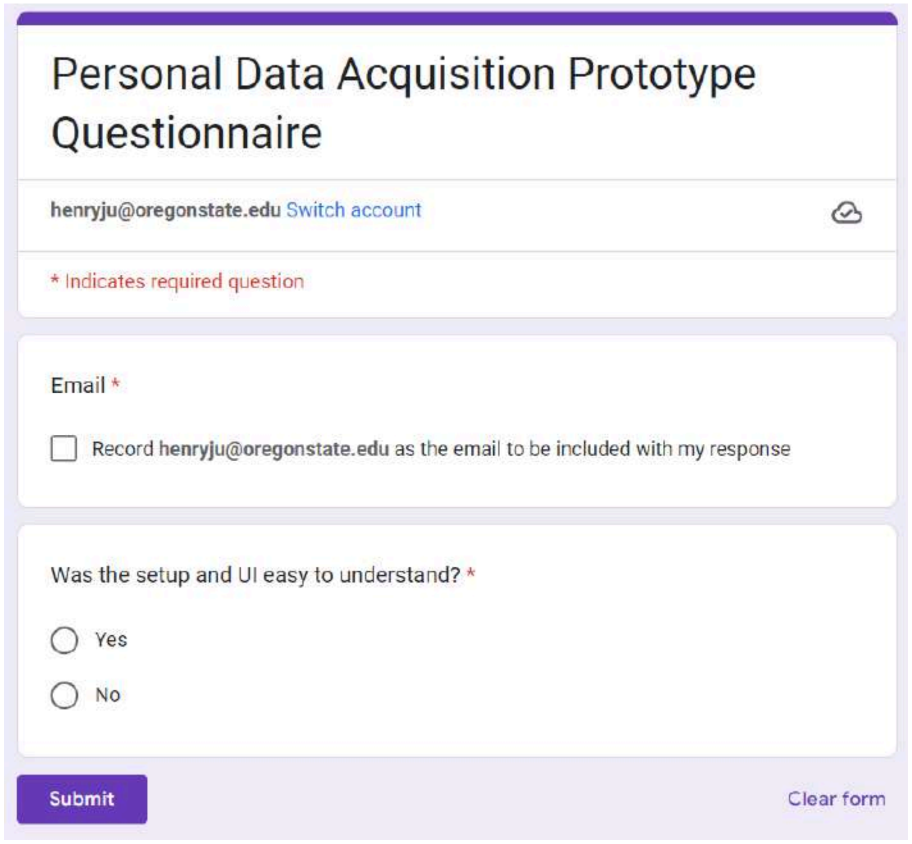
5.2.1 Easy to understand

5.2.1.1 Project Partner Requirement
The system must be usable for a non-technical consumer.

5.2.1.2 Engineering Requirement
9 out of 10 users will configure the sensors and afterward report "The configuration process was easy."

5.2.1.3 Verification Process
- Supply 10 people with the system and its UI
- Ask them to configure one or more sensors
- Afterward, ask them to sign a document asking if they found the configuration process easy

5.2.1.4 Testing Evidence
This section shows evidence that the requirement has been met. To verify this requirement, the verification process from section 5.2.1.3 was followed. A total of eleven people provided responses. Below is an image of the questionnaire used to gauge if the system’s configuration process was user friendly, as well as an image showing the results from the questionnaire.



Personal Data Acquisition Prototype Questionnaire

henryju@oregonstate.edu Switch account

* Indicates required question

Email *

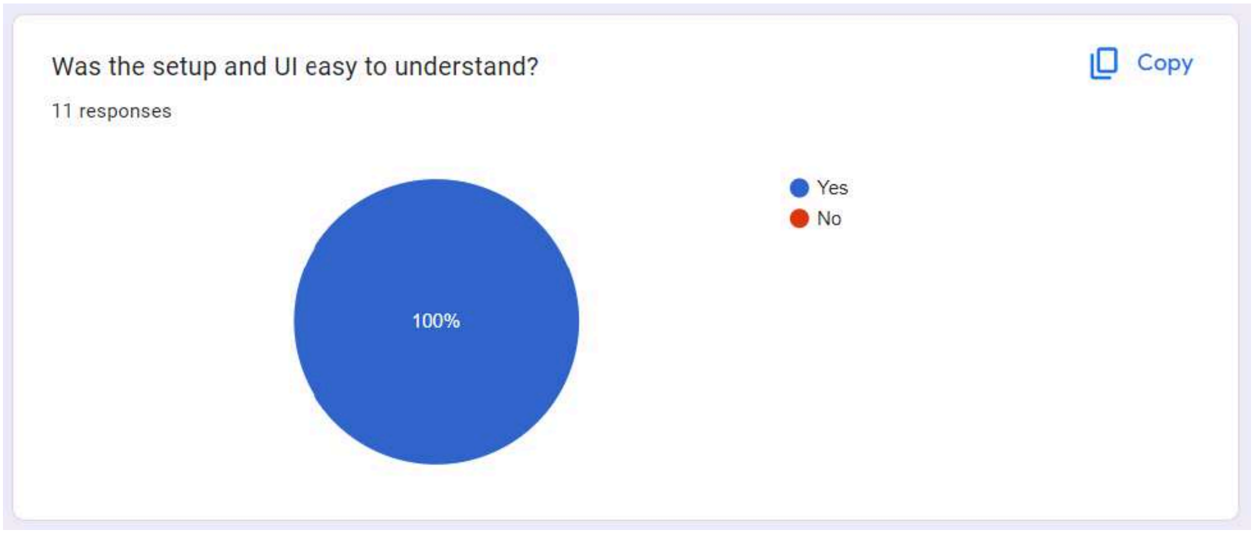
☐ Record henryju@oregonstate.edu as the email to be included with my response

Was the setup and UI easy to understand? *

◯ Yes

◯ No

Fig. 4. Questionnaire



Was the setup and UI easy to understand?

11 responses

Fig. 5. Questionnaire results

5.2.2 Mobile

5.2.2.1 Project Partner Requirement

The system must be mobile.

5.2.2.2 Engineering Requirement

The system will operate normally for at least 1 hour on a single charge.

5.2.2.3 Verification Process

- Connect at least one sensor to the system
- Ensure battery is fully charged
- Set a timer for one hour
- Start data collection
- Wait one hour
- Confirm the system is still operating normally

5.2.2.4 Testing Evidence

This section shows evidence that the requirement has been met. To verify this requirement, the verification process from section 5.2.2.3 was followed. The system was set up with the Mikroe Click sensor block recording data. Below is an image of the system before it was run for an hour, as well as an image of the system after the hour. In that time, the power supply lost 6% of its battery, meeting the requirement. In addition, a timelapse of the system operating over an hour is provided in section 5.3.2.

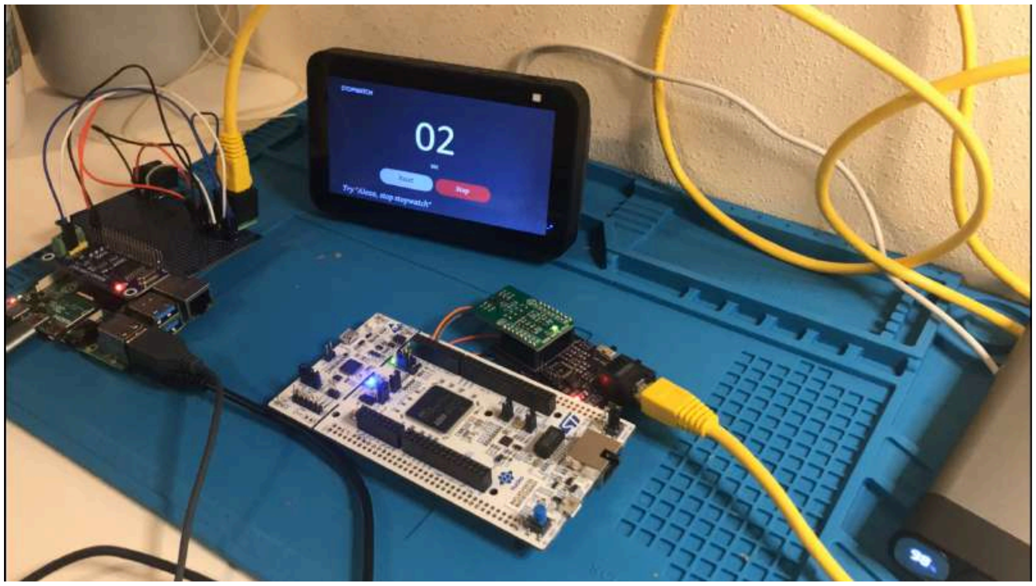


Fig. 6. System before test. Power supply shows 98% charge.

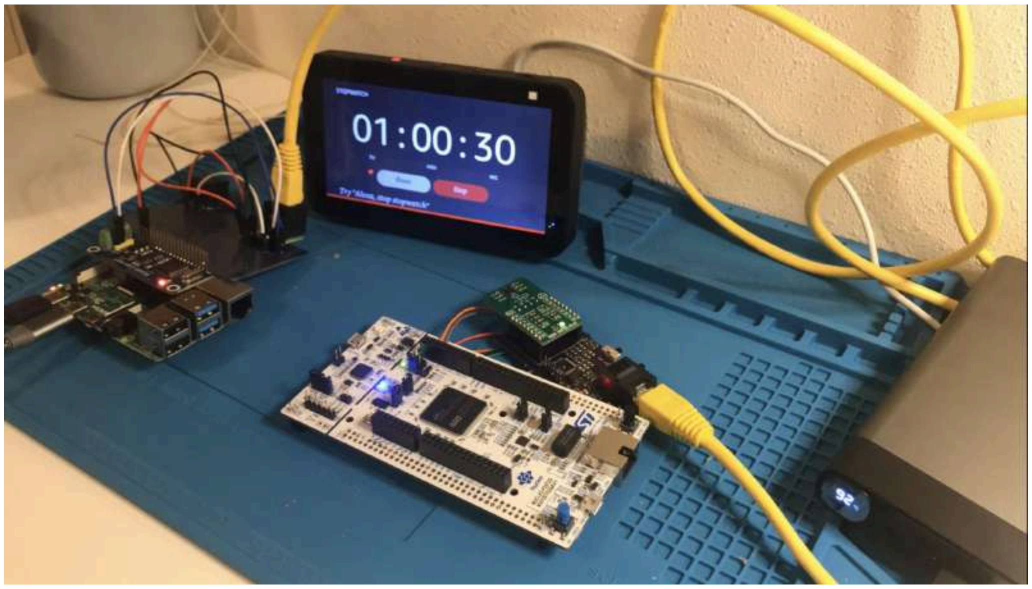


Fig. 7. System after test. Power supply shows 92% charge.

5.2.3 Modular

5.2.3.1 Project Partner Requirement

The system must be modular.

5.2.3.2 Engineering Requirement

The system will operate normally after having the sensor module pulled against the logger with at least 5lbs of force.

5.2.3.3 Verification Process

- Connect at least one sensor to logger
- Attach 5lb weight to sensor
- Suspend sensor in air from logger for 5 seconds
- Confirm that no damage has been done to the system

5.2.3.4 Testing Evidence

This section shows evidence that the requirement has been met. To verify this requirement, the verification process from section 5.2.3.3 was followed. Below is a picture of the test being performed on the system. The 5lb weight was suspended from the Mikroe Click Sensor block by a fishing line. A video can be found in section 5.3.2 which does a better job of showing the temporal aspect of the test.

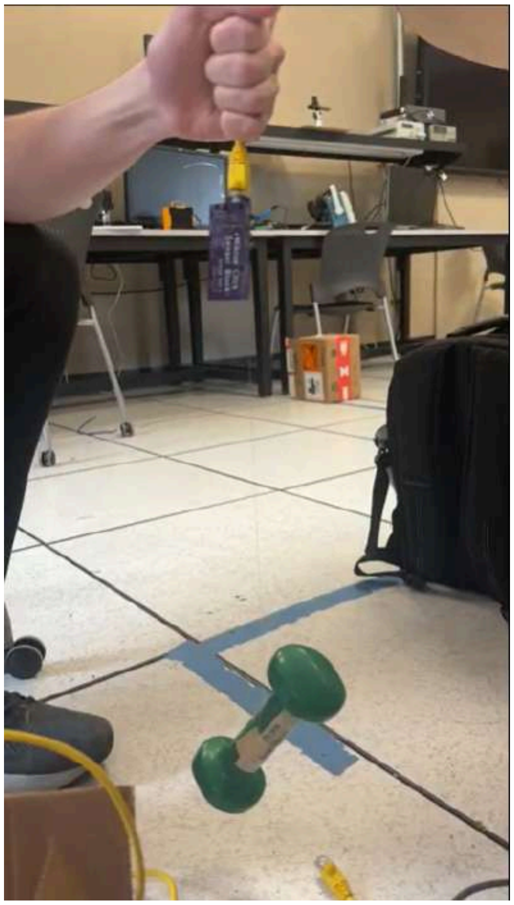


Fig. 8. System during test.

5.2.4 Precise

5.2.4.1 Project Partner Requirement

The system must record precise data.

5.2.4.2 Engineering Requirement

The system will output data with a resolution of at least 8 bits.

5.2.4.3 Verification Process

- Run data collection for a sensor
- Review the transmitted CAN data
- Verify that the outputted data is 8 or more bits

5.2.4.4 Testing Evidence

This section shows evidence that the requirement has been met. To verify this requirement, the verification process from section 5.2.3.3 was followed. The Mikroe Click sensor module was used with an accelerometer click sensor. Figure 9 below shows the test setup, with all the important components labeled. Figure 10 shows the UI. The Graph on the UI shows that the sensor data is being received, stored, and interpreted by the logger. Figure 11 shows the CAN data being received at the logger. The middle section of each row of data shows that the data length (DL) is 2 bytes (i.e., 8 bits), and what hex value is stored. One of these sections is outlined in red on figure 11 for clarity. Since the DL is 8 bits, the system passes the precise requirement.

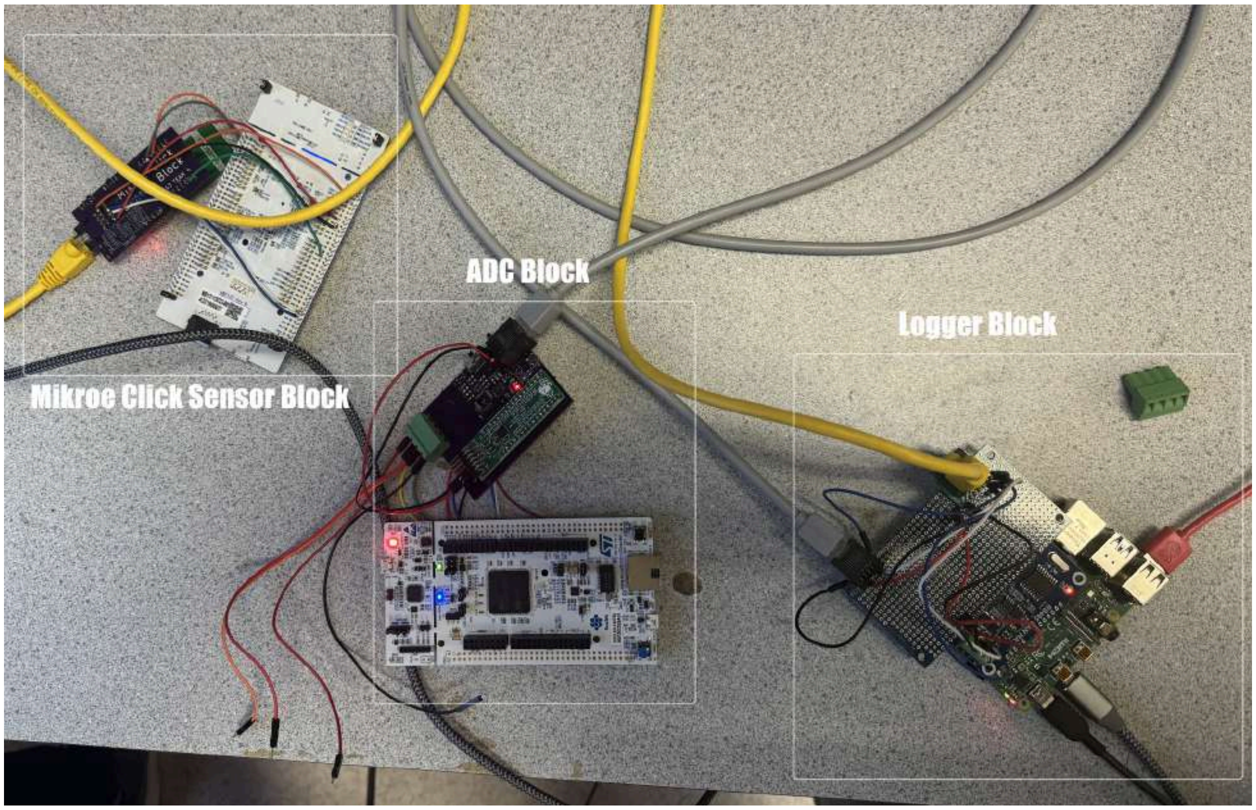


Fig. 9. Physical setup for precision test. Shows Mikroe Click and ADC sensor blocks connected to the logger block.

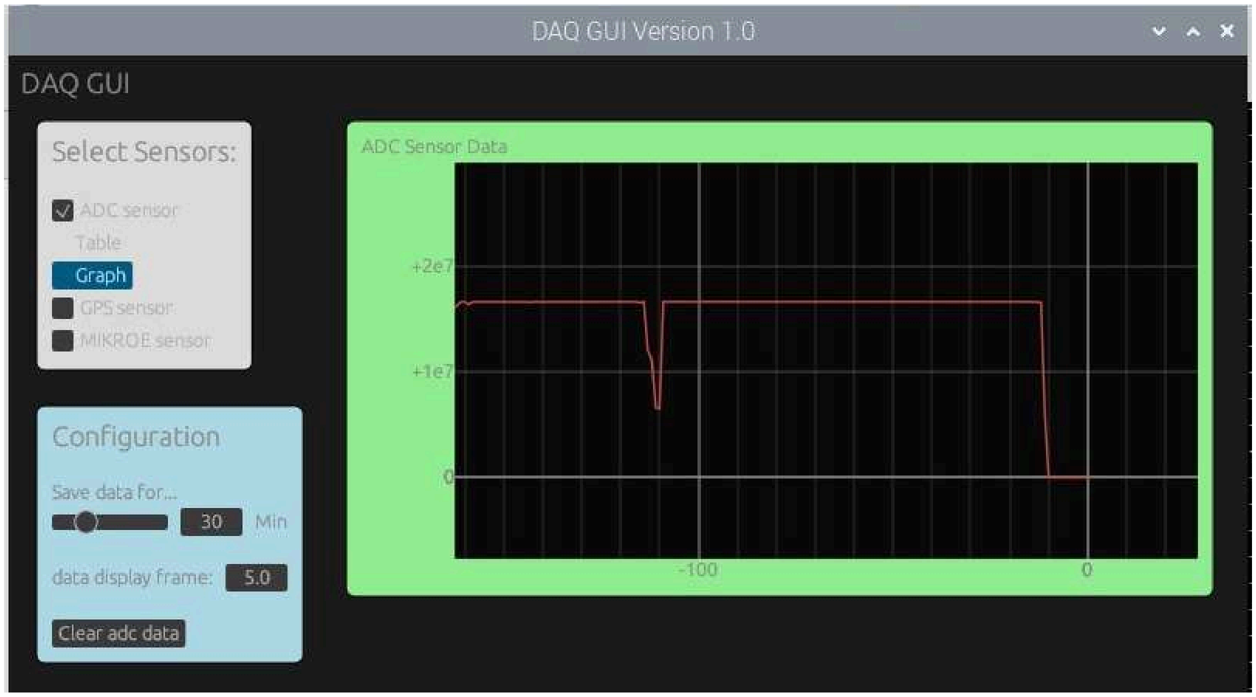
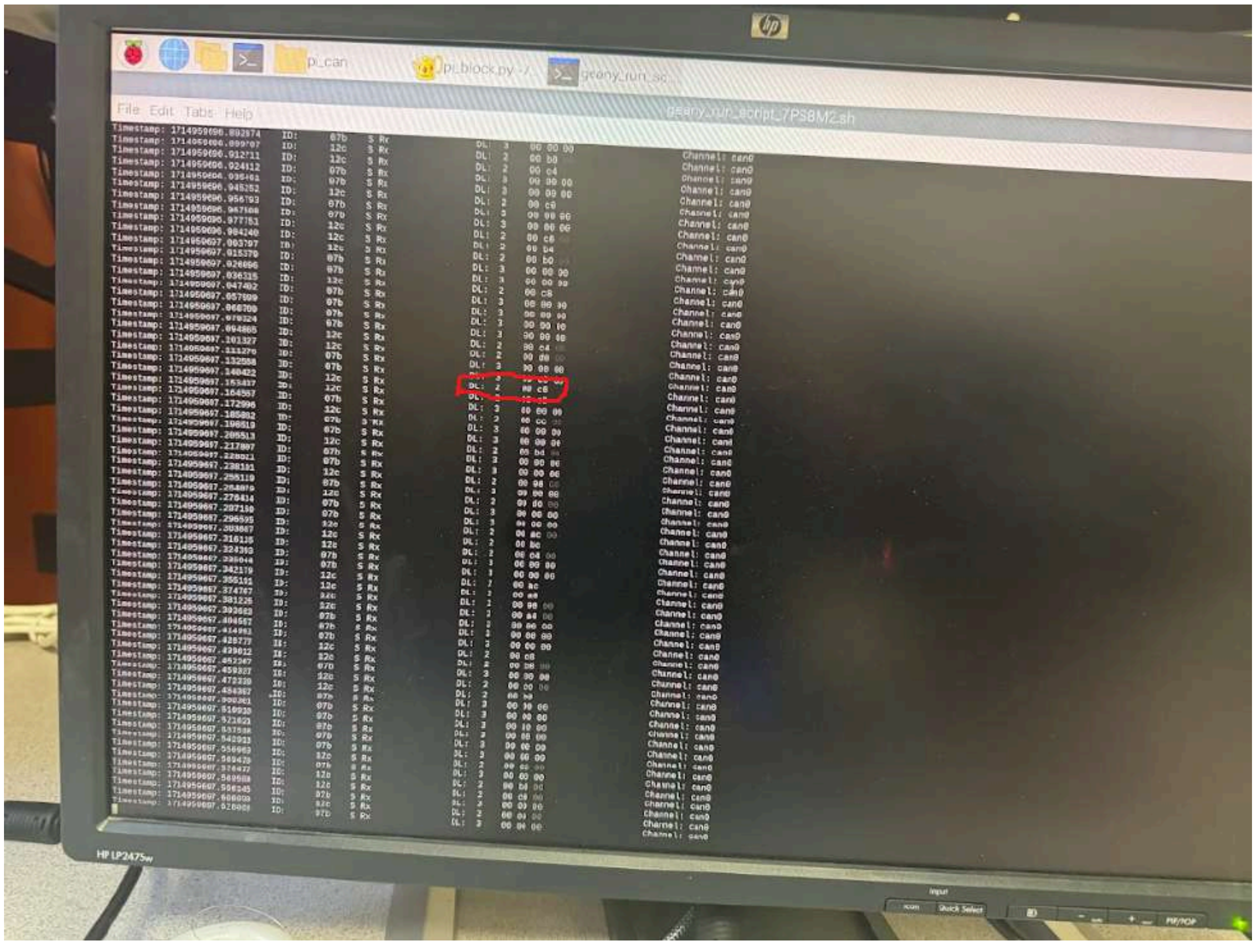


Fig. 10. UI showing output graph for the ADC block



```
Timestamp: 1714959696.892874   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959696.899707   ID:  12c   S Rx   DL:  2   00 b0      Channel: can0
Timestamp: 1714959696.912711   ID:  12c   S Rx   DL:  2   00 c4      Channel: can0
Timestamp: 1714959696.924112   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959696.936461   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959696.945252   ID:  12c   S Rx   DL:  2   00 c8      Channel: can0
Timestamp: 1714959696.956793   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959696.967508   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959696.977751   ID:  12c   S Rx   DL:  2   00 c8      Channel: can0
Timestamp: 1714959696.984240   ID:  12c   S Rx   DL:  2   00 b4      Channel: can0
Timestamp: 1714959697.003797   ID:  12c   S Rx   DL:  2   00 b0      Channel: can0
Timestamp: 1714959697.015370   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.026096   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.036315   ID:  12c   S Rx   DL:  2   00 c8      Channel: can0
Timestamp: 1714959697.047402   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.057599   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.068700   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.079324   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.094865   ID:  12c   S Rx   DL:  2   00 c4      Channel: can0
Timestamp: 1714959697.101327   ID:  12c   S Rx   DL:  2   00 d0      Channel: can0
Timestamp: 1714959697.111270   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.132558   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.140422   ID:  12c   S Rx   DL:  2   00 c8      Channel: can0
Timestamp: 1714959697.153417   ID:  12c   S Rx   DL:  2   [illegible] Channel: can0
Timestamp: 1714959697.164567   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.172996   ID:  12c   S Rx   DL:  2   00 cc      Channel: can0
Timestamp: 1714959697.185882   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.196519   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.205513   ID:  12c   S Rx   DL:  2   00 bd      Channel: can0
Timestamp: 1714959697.217807   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.228021   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.238191   ID:  12c   S Rx   DL:  2   00 98      Channel: can0
Timestamp: 1714959697.255119   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.264879   ID:  12c   S Rx   DL:  2   00 d0      Channel: can0
Timestamp: 1714959697.276414   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.287150   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.296595   ID:  12c   S Rx   DL:  2   00 ac      Channel: can0
Timestamp: 1714959697.303667   ID:  12c   S Rx   DL:  2   00 bc      Channel: can0
Timestamp: 1714959697.316116   ID:  12c   S Rx   DL:  2   00 c4      Channel: can0
Timestamp: 1714959697.324393   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.336044   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.342179   ID:  12c   S Rx   DL:  2   00 ac      Channel: can0
Timestamp: 1714959697.355191   ID:  12c   S Rx   DL:  2   00 a8      Channel: can0
Timestamp: 1714959697.374767   ID:  12c   S Rx   DL:  3   00 98 00   Channel: can0
Timestamp: 1714959697.381226   ID:  12c   S Rx   DL:  3   00 a4 00   Channel: can0
Timestamp: 1714959697.393683   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.404567   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.414993   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.428777   ID:  12c   S Rx   DL:  2   00 c8      Channel: can0
Timestamp: 1714959697.439012   ID:  12c   S Rx   DL:  2   00 b8      Channel: can0
Timestamp: 1714959697.452267   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.459327   ID:  12c   S Rx   DL:  2   00 00      Channel: can0
Timestamp: 1714959697.472330   ID:  12c   S Rx   DL:  2   00 b0      Channel: can0
Timestamp: 1714959697.484267   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.500261   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.510930   ID:  07b   S Rx   DL:  3   00 10 00   Channel: can0
Timestamp: 1714959697.521621   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.537538   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.542933   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.556969   ID:  12c   S Rx   DL:  2   00 c0      Channel: can0
Timestamp: 1714959697.569470   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.576477   ID:  12c   S Rx   DL:  2   00 b0      Channel: can0
Timestamp: 1714959697.589508   ID:  12c   S Rx   DL:  2   00 c8      Channel: can0
Timestamp: 1714959697.596545   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
Timestamp: 1714959697.606000   ID:  12c   S Rx   DL:  2   00 c4      Channel: can0
Timestamp: 1714959697.620000   ID:  07b   S Rx   DL:  3   00 00 00   Channel: can0
```

Fig. 11. Record of CAN data being received by system at logger block. The red outline shows that one message has a data length (DL) of 2 bytes, and stores a value, in hex, of 0x00C8.

5.2.5 Sample Rate

5.2.5.1 Project Partner Requirement

The system must record data in real-time with a reasonable sampling resolution.

5.2.5.2 Engineering Requirement

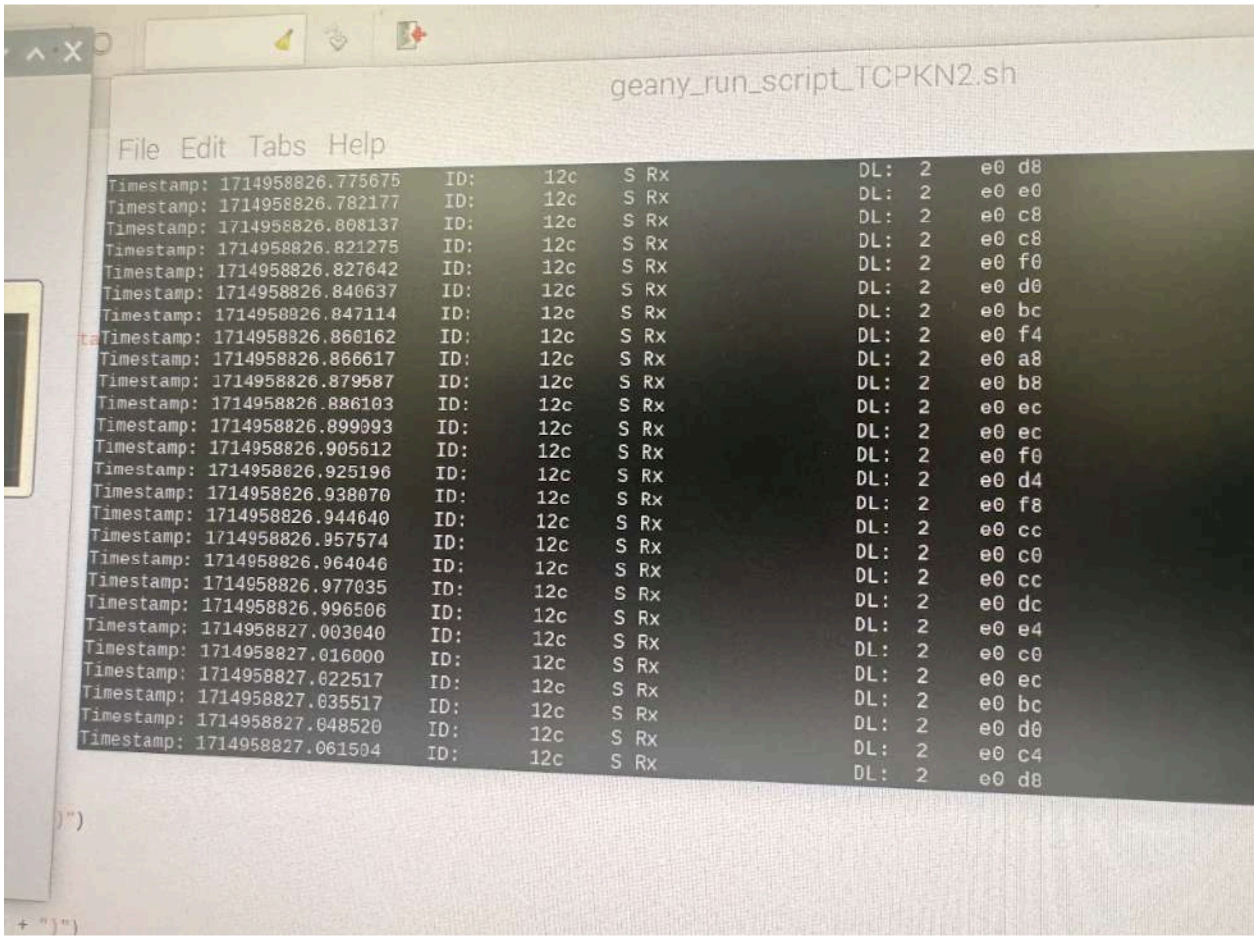
The system will output data at least at 100HZ for analog voltage and acceleration data and at least 5Hz for GPS data.

5.2.5.3 Verification Process

- Collect data from at least one sensor block
- View the data on the UI
- Show that there is at least one data point per 10ms for analog voltage and acceleration measurements
- Show that there is at least one data point per 200ms for GPS measurements

5.2.5.4 Testing Evidence

This section shows evidence that the requirement has been met. To verify this requirement, the verification process from section 5.2.5.3 was followed. As shown below in figure 12, not all data has a 10 ms or less difference in its timestamp value compared to the previous data value due to limitations of how much data the raspberry pi can receive over CAN bus. However, some data values such as samples received at 1714958826.782177 and 1714958826.775675, resulting in a difference of 6.502 ms, show that the accelerometer on the Mikro Click module is able to take measurements at over 100 Hz.



```
Timestamp: 1714958826.775675   ID:   12c   S Rx      DL:  2   e0 d8
Timestamp: 1714958826.782177   ID:   12c   S Rx      DL:  2   e0 e0
Timestamp: 1714958826.808137   ID:   12c   S Rx      DL:  2   e0 c8
Timestamp: 1714958826.821275   ID:   12c   S Rx      DL:  2   e0 c8
Timestamp: 1714958826.827642   ID:   12c   S Rx      DL:  2   e0 f0
Timestamp: 1714958826.840637   ID:   12c   S Rx      DL:  2   e0 d0
Timestamp: 1714958826.847114   ID:   12c   S Rx      DL:  2   e0 bc
Timestamp: 1714958826.860162   ID:   12c   S Rx      DL:  2   e0 f4
Timestamp: 1714958826.866617   ID:   12c   S Rx      DL:  2   e0 a8
Timestamp: 1714958826.879587   ID:   12c   S Rx      DL:  2   e0 b8
Timestamp: 1714958826.886103   ID:   12c   S Rx      DL:  2   e0 ec
Timestamp: 1714958826.899093   ID:   12c   S Rx      DL:  2   e0 ec
Timestamp: 1714958826.905612   ID:   12c   S Rx      DL:  2   e0 f0
Timestamp: 1714958826.925196   ID:   12c   S Rx      DL:  2   e0 d4
Timestamp: 1714958826.938070   ID:   12c   S Rx      DL:  2   e0 f8
Timestamp: 1714958826.944640   ID:   12c   S Rx      DL:  2   e0 cc
Timestamp: 1714958826.957574   ID:   12c   S Rx      DL:  2   e0 c0
Timestamp: 1714958826.964046   ID:   12c   S Rx      DL:  2   e0 cc
Timestamp: 1714958826.977035   ID:   12c   S Rx      DL:  2   e0 dc
Timestamp: 1714958826.996506   ID:   12c   S Rx      DL:  2   e0 e4
Timestamp: 1714958827.003040   ID:   12c   S Rx      DL:  2   e0 c0
Timestamp: 1714958827.016000   ID:   12c   S Rx      DL:  2   e0 ec
Timestamp: 1714958827.022517   ID:   12c   S Rx      DL:  2   e0 bc
Timestamp: 1714958827.035517   ID:   12c   S Rx      DL:  2   e0 d0
Timestamp: 1714958827.048520   ID:   12c   S Rx      DL:  2   e0 c4
Timestamp: 1714958827.061504   ID:   12c   S Rx      DL:  2   e0 d8
```

Fig. 12. Recorded values with timestamps of Mikroe Click data received by the logger module.

5.2.6 Storage

5.2.6.1 Project Partner Requirement

The system must store data.

5.2.6.2 Engineering Requirement

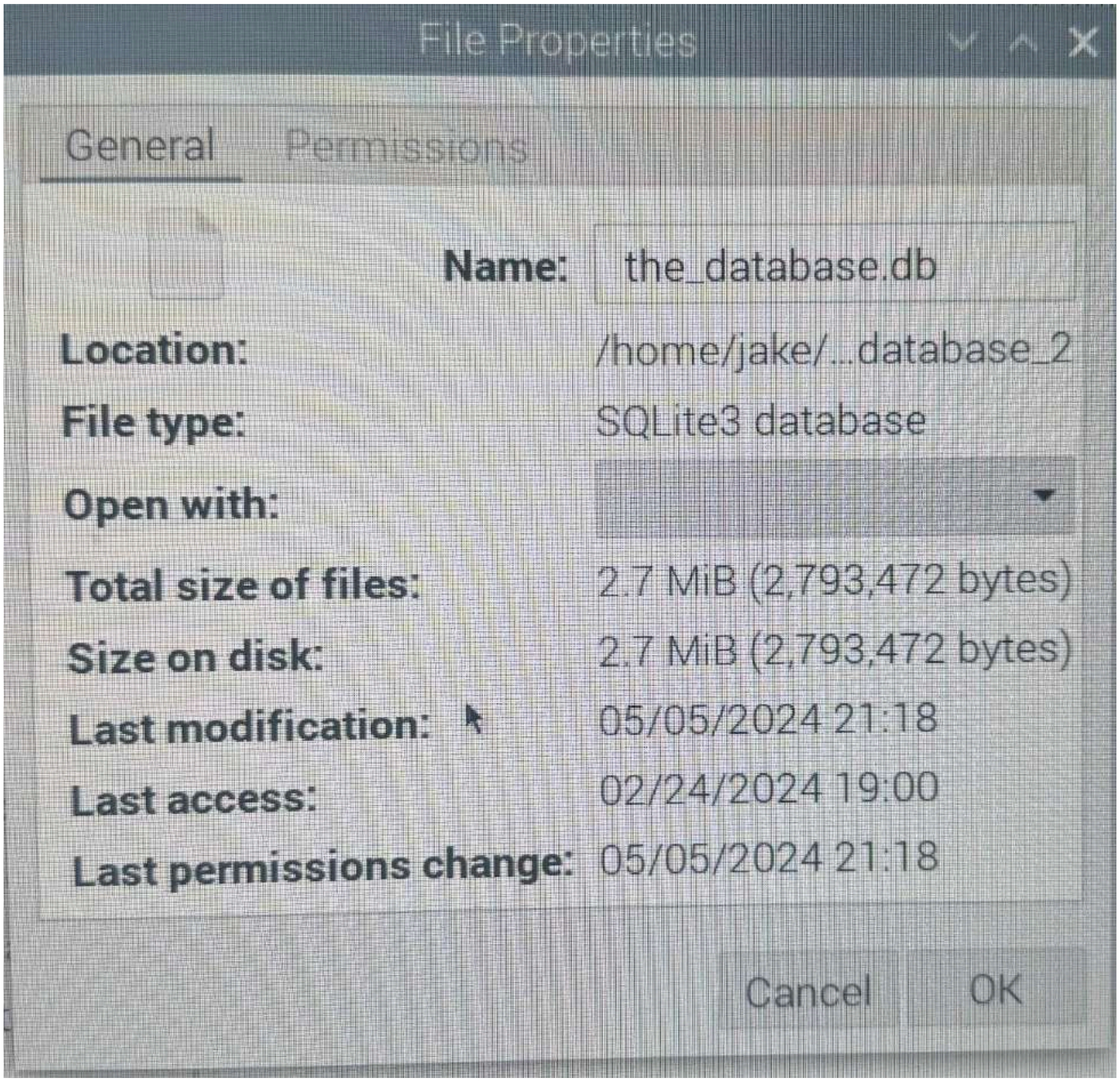
The system will store an hour's worth of data collected by the system.

5.2.6.3 Verification Process

- Connect one or more sensors to the system
- View the database file size
- Run the system for an hour
- Verify that the system is still functioning
- Verify that the database file size has increased, showing that data is being stored

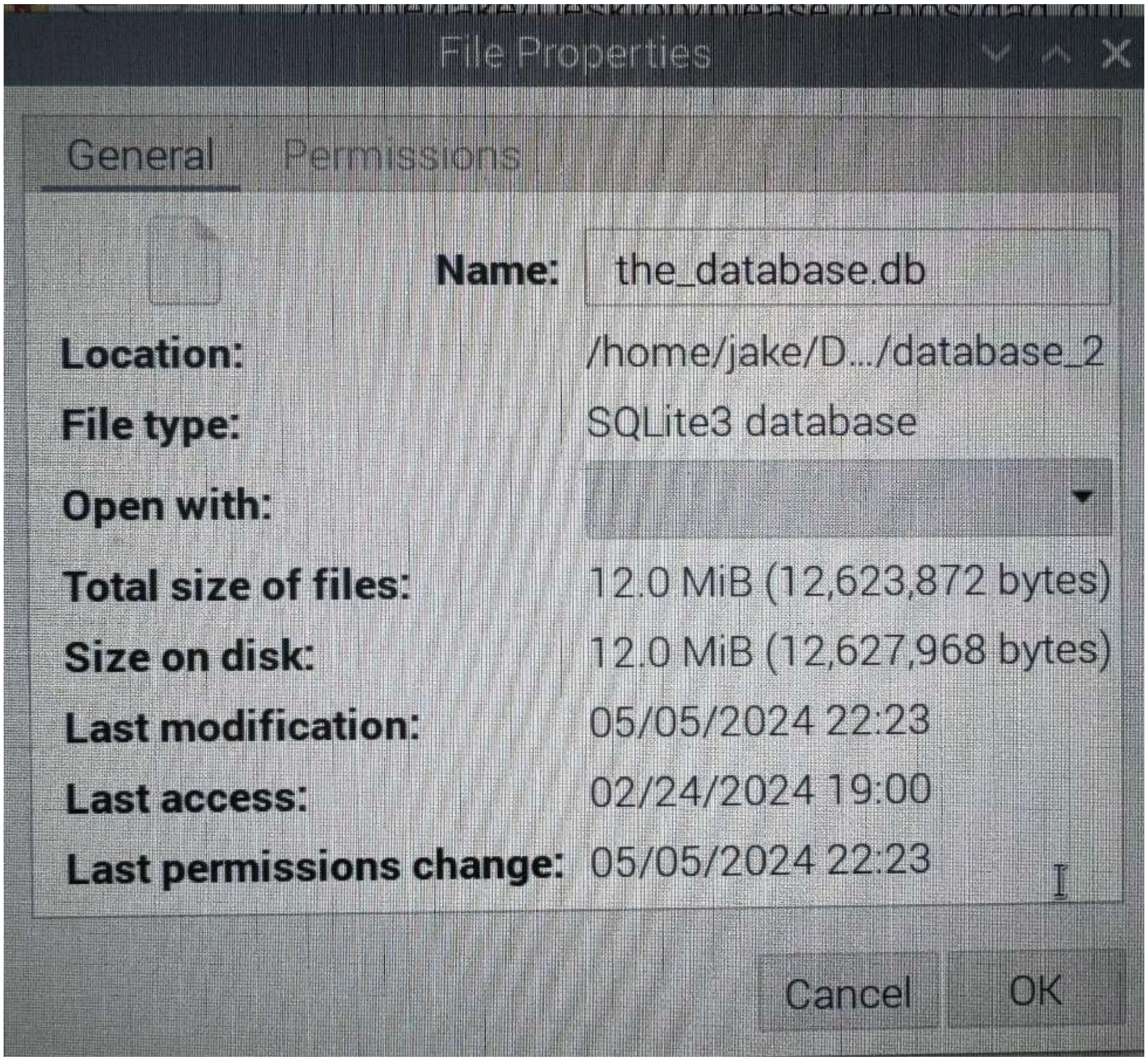
5.2.6.4 Testing Evidence

This shows that the “storage” system requirement has been met. To verify this requirement, the process used in 5.2.6.3 was followed. The image in figure 13 below shows the total size of the file before recording data for an hour, the time is also included. The image in figure 14 below, shows the size of the file after an hour of data being recorded. The size and time demonstrate the amount of data recorded in an hour. In addition, the timelapse used for the “mobile” requirement demonstrates the system recording data for an hour, this can be found in section 5.3.2.



Name: the_database.db
Location: /home/jake/...database_2
File type: SQLite3 database
Open with:
Total size of files: 2.7 MiB (2,793,472 bytes)
Size on disk: 2.7 MiB (2,793,472 bytes)
Last modification: 05/05/2024 21:18
Last access: 02/24/2024 19:00
Last permissions change: 05/05/2024 21:18

Fig. 13 Photo before recording showing data and time.


File Properties

General Permissions

Name: the_database.db
Location: /home/jake/D.../database_2
File type: SQLite3 database
Open with:
Total size of files: 12.0 MiB (12,623,872 bytes)
Size on disk: 12.0 MiB (12,627,968 bytes)
Last modification: 05/05/2024 22:23
Last access: 02/24/2024 19:00
Last permissions change: 05/05/2024 22:23

Fig. 14 Photo after recording showing size and time.

5.2.7 User Feedback

5.2.7.1 Project Partner Requirement

The system must output a stream of data to the user.

5.2.7.2 Engineering Requirement

The system will display data on the UI that updates according to the state of the sensors within 30 seconds of the real-world sensor state change.

5.2.7.3 Verification Process

- Collect data from at least one sensor block
- Apply some input that results in a clear spike in data output graphs, such as the quick movement of a GPS or accelerometer

- Confirm that the time between the real-world event and when the change is viewable on the UI is less than 30 seconds

5.2.7.4 Testing Evidence

This section shows evidence that the "user feedback" requirement has been met. To verify this requirement, the system was set up with the Mikroe Click sensor module with an accelerometer click board. This is the same setup used to verify the "precise" requirement. Figure 9 above shows the test setup, with all the important components labeled. Similarly, figure 15 below shows the UI. The Graph on the UI shows that the sensor data is being received, stored, and interpreted by the logger. The sharp change in the graph shows when the accelerometer was shaken. It is difficult to convey the real-time aspect of this test through text, so a video showing this requirement is included in the file links section below for clarity [7]. In the video, the graph on the UI responds to the change in the sensor's state within a few seconds, passing the user feedback requirement because the time between real world event and UI update is less than 30 seconds.

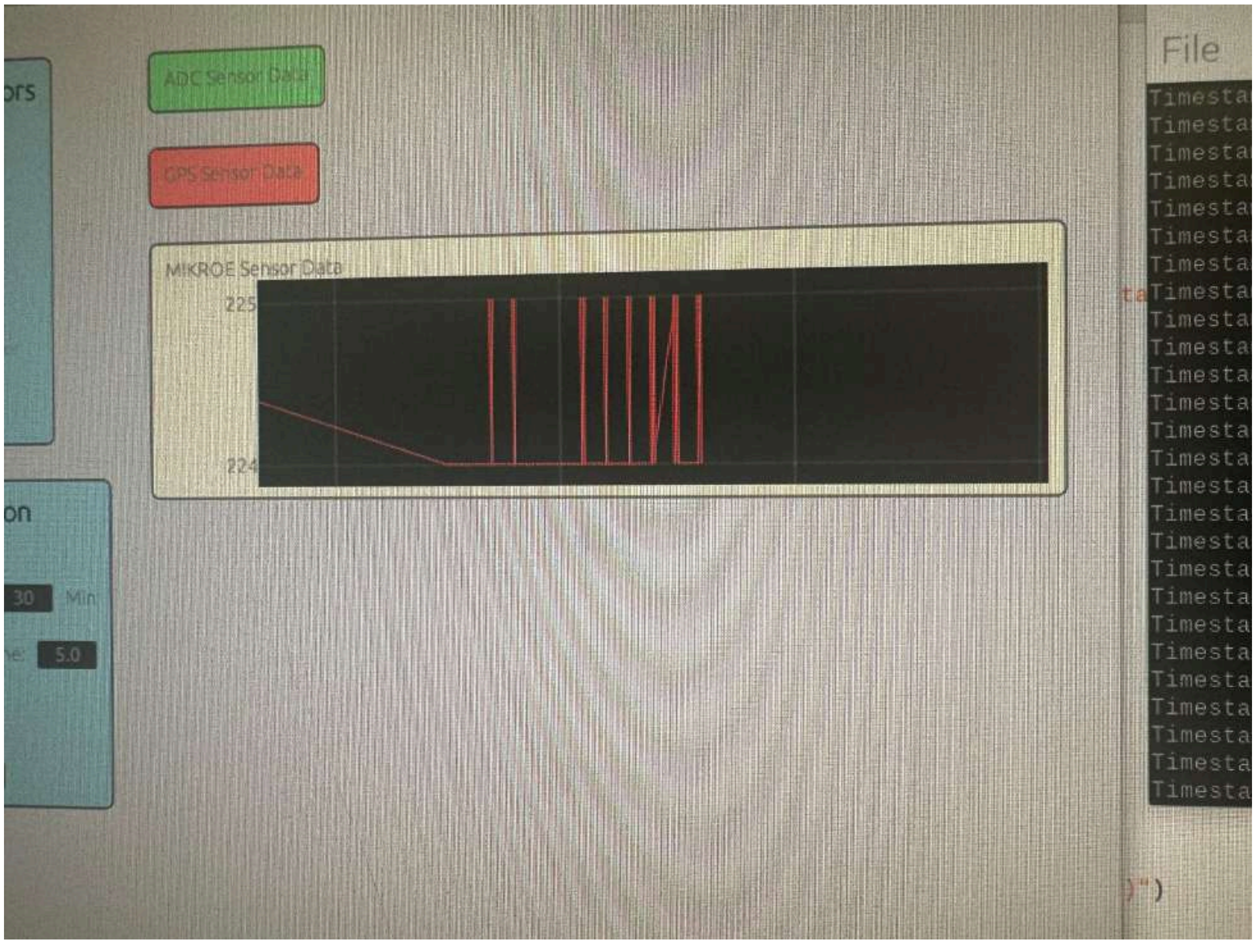


Fig. 15 shows the UI receiving data values

5.2.8 Works with Mikroe Sensors

5.2.8.1 Project Partner Requirement

The system must interface with Mikroe click sensors.

5.2.8.2 Engineering Requirement

The system will include at least 1 “Mikroe click” connection that will work with at least 1 Mikroe product.

5.2.8.3 Verification Process

- Set up the Mikroe Sensor block with a Mikroe Click Sensor board
- Connect the Mikroe Sensor block with the logger block
- Confirm that the sensor's data shows up on the UI

5.2.8.4 Testing Evidence

This section shows evidence that the “works with Mikroe sensors” requirement has been met. To verify this requirement, we set up our system with the Mikroe sensor block. Connected to the Mikroe sensor block is an accelerometer Mikroe Click sensor board. This is the same setup as in sections 5.2.4 and 5.2.7. Figure 9 above shows the test setup, with all of the important components labeled. Figure 15 shows the UI. The graph on the UI shows that the accelerometer data is being received, stored, and interpreted by the logger. Just before the screenshot was taken, the accelerometer was shaken. This data translates to the sharp spikes in the graph. Since the Mikroe Click sensor's data shows up on the system’s UI, the system passes the “works with mikroe sensors” requirement. A video showing this requirement is included in the file links section below.

5.3 References and File Links

5.3.1 References (IEEE)

[1] Claire, “5 things you need to know before buying a Power Bank,” PITAKA, https://www.ipitaka.com/blogs/news/5-things-you-need-to-know-before-buying-a-power-bank (accessed Dec. 4, 2023).

[2] “Portable Charger Info Page,” Amazon, https://www.amazon.com/gp/product/B07PG6C5C7 (accessed Dec. 4, 2023).

5.3.2 File Links

[3] https://drive.google.com/file/d/1Rb8SYO5wn9-coknF8h87nI6pCTVOj4j6/view?usp=sharing
The content for section 5.2 was partly ideated during lecture time. This link shows an image of part of the document created at that point.

[4] https://drive.google.com/file/d/1Rkp_Z6PSPDJ8OKCMw7IM13SLEuRUrTy9/view?usp=sharing
The content for section 5.2 was partly ideated during lecture time. This link shows an image of part of the document created at that point.

[5]
https://drive.google.com/file/d/1Rh9rn6szjOvJ5uw3Naci62TEKqzs8xSp/view?usp=drive_link
The content for section 5.2 was partly ideated during lecture time. This link shows an image of part of the document created at that point.

[6]
https://drive.google.com/file/d/1kIVIOczVyN85P0foZndL4XGRKNIh`nwdl/view?usp=sharing
This link shows a video of our system, consisting of a prototype ADC sensor module connected to the logger module through CAN. The video highlights the system operating with data that is 3 bytes wide, and also shows our UI updating to changes in the sensor state.

[7]
https://drive.google.com/file/d/1BZ8Sr3KelZRa4JThaurde0CNvzWp0fN1/view?usp=sharing
This link shows a video of our system, consisting of a prototype Mikroe Sensor module with an accelerometer Mikroe Click board attached, connected to the logger module through CAN. The video highlights the UI updating to changes in the sensor state, and storing data in a database.

[8]
https://drive.google.com/file/d/1r8Mxas8BscOjt5KL9w-FDPt_fqGO0t3z/view?usp=sharing
This link shows a video of the modular requirement being tested by attaching a 5 lb weight to a board connected to the system and holding it up for 5 seconds. As shown in the video, the board is undamaged by the weight.

[9]
https://drive.google.com/file/d/1LbEe9r-Y8YPQSoNRJNtvOvBdidsyeJEJ/view?usp=sharing
This link demonstrates the mobile requirement. A timelapse of the system operating for an hour is shown.

5.4 Revision Table

TABLE XIII
Section 5 Revision Table

5/5/2024	Team: Updated all sections with current verification information
3/10/2024	Team: Updated requirements to meet current project wording Julian Henry: Wrote verification sections for 5.2.4, 5.2.7, and 5.2.8
12/4/2023	Alex Gibson, Julian Henry: Updated section 5.2 with project partner feedback
11/30/2023	Julian Henry: Section 5.1 drafted, section 5.2 imported and formatted from Project Portal
1129/2023	Julian Henry: Section 5 outline created

6. Project Closing

6.1 Future Recommendations

6.1.1 Technical Recommendations

6.1.1.1 Improve User Interface Functionality

In the future, we recommend that the user interface (UI) should be reworked. Currently, the UI is written in Rust using the EGUI framework. This framework was suggested by the project partner due to their familiarity with Rust. Unfortunately, neither Rust nor the EGUI framework was familiar to the team. Due to the complexity of learning Rust and EGUI, as well as the fact that EGUI is a newer framework and lacks some important features, the UI came out rather basic. While it is functional, it lacks some of the desired features and is not very aesthetically pleasing. Some specific deficiencies of the current UI include that graphs sometimes have to be manually scaled by a user instead of fitting the data, data cannot be displayed in tables and instead relies on formatted text output, the UI uses static pixel layouts which cannot be easily stretched or scaled, and that there is inadequate support for different operating systems, leading to inconsistencies in colors and fonts. To address these deficiencies, we have two recommended starting points. The first would be to invest significant dedicated time into learning the complexities of the Rust language and EGUI framework, then updating the current UI. This could be started by going through the Rust Cookbook [1] to learn Rust, then reading the EGUI documentation [2]. The other recommendation is to restart from scratch using a more user-friendly and versatile framework like PyQt5, which is based on Python. This could significantly enhance the GUI capabilities and address the shortcomings identified with EGUI. A good starting point for learning PyQt is pythonguis.com [3].

6.1.1.2 Choose a Different Microcontroller

In the future, we recommend that a different microcontroller is selected for the sensor modules. Currently, the sensor modules rely on STM32F072C8T6TR microcontrollers. This microcontroller was chosen because of its relatively low cost and support for the required communication protocols. However, working with the STM32 microcontrollers presented certain challenges that hindered the design and programming processes. One issue was the dependency on an ST-Link programmer for flashing and debugging, which added more complexity and was cumbersome during the development and testing phases of our PCBs. Another reason to change microcontrollers is that there are several unused pins with the current setup, which represent inefficiency in resource utilization. We recommend looking into a different line of microcontrollers in the future. One starting point would be to look at the ESP32 microcontrollers [4]. ESP32 microcontrollers can be programmed directly over USB, reducing the complexity and cost of development, and also features built-in support for Wi-Fi and Bluetooth [4], enabling future versions of the Data Acquisition System to utilze wireless communication between sensor modules and the logger which would also reduce the cost of the system.

6.1.1.3 Choose a Different Logger

In the future, we recommend choosing a different approach for implementing the logger module. Currently, the logger is implemented using a Raspberry Pi 4B. A Raspberry Pi 4B was initially chosen for its versatility and the availability of add-on boards such as the CAN Hat that the current system uses to interface with the sensor modules. However, the use of the Raspberry Pi 4B for the logger module has introduced several drawbacks and inefficiencies such as its high power consumption that limits the amount of time the system can run on one charge, its complexity which is largely unneeded besides the ability to communicate with the sensor modules and host the UI and explains its high power consumption, and its cost which makes the system less accessible to its target audience. To address these issues and optimize the logger module design, we recommend exploring alternative solutions such as using an Arduino or ESP32 microcontroller board [5, 6].

6.1.1.4 Ensure Backup Components Exist

In the future, we recommend ensuring that there exists at least one alternative component for every critical part needed in the system. Currently, for each critical component in our system, we have only found one part that we know works. This reliance on specific parts could be an issue in the future if there are supply chain or manufacturing issues leading to component shortages for the companies providing our parts. To address this issue, we recommend conducting a thorough investigation of our parts list and identifying suitable alternative parts for the critical components. A good starting point for searching for these parts is Digikey [7].

6.1.2 Global Impact Recommendations

One global impact recommendation for future work on this project is to reduce the amount of times that components are ordered for the system. This is important because the process of shipping components across the country results in carbon emissions that have negative impacts on the planet. The total number of component orders could be reduced by better planning of what the system requires and communication between group members so that orders can be combined.

Another global impact recommendation would be to put a larger emphasis on safety information for the user of the system. As described in section 2.1.2.2, since the system is intended for an outdoor application such as mountain biking, it is important to inform the user of the risks of using the system such as cables getting tangled in the bike if not installed properly. One way to do this would be to create basic instructions for the user on how to properly attach the system so that it doesn’t have any loose cables.

6.1.3 Teamwork Recommendations

One teamwork/communication recommendation is to establish communication platforms early and choose specific platforms based on the type of communication. This is important as poor communication methods can lead to miscommunication, delays, and hinder the process of the project. A recommendation to increase communication is to designate platforms based on the type of communication. For example, having discord for messages and discussions and using microsoft projects for scheduling meetings, and creating a project timeline.

Another recommendation is to have weekly meetings to discuss the project with your team. Without an effective method to actively discuss and listen to other members' feedback, this may lead to design issues that are overlooked or that may go unresolved. By implementing weekly check-ins, this gives each member of the team a chance to discuss their progress on the project as well as any challenges they are facing. In addition to having members provide feedback/ideas for each other, this allows for each member to become comfortable with communicating to the team.

6.2 Project Artifact Summary with Links

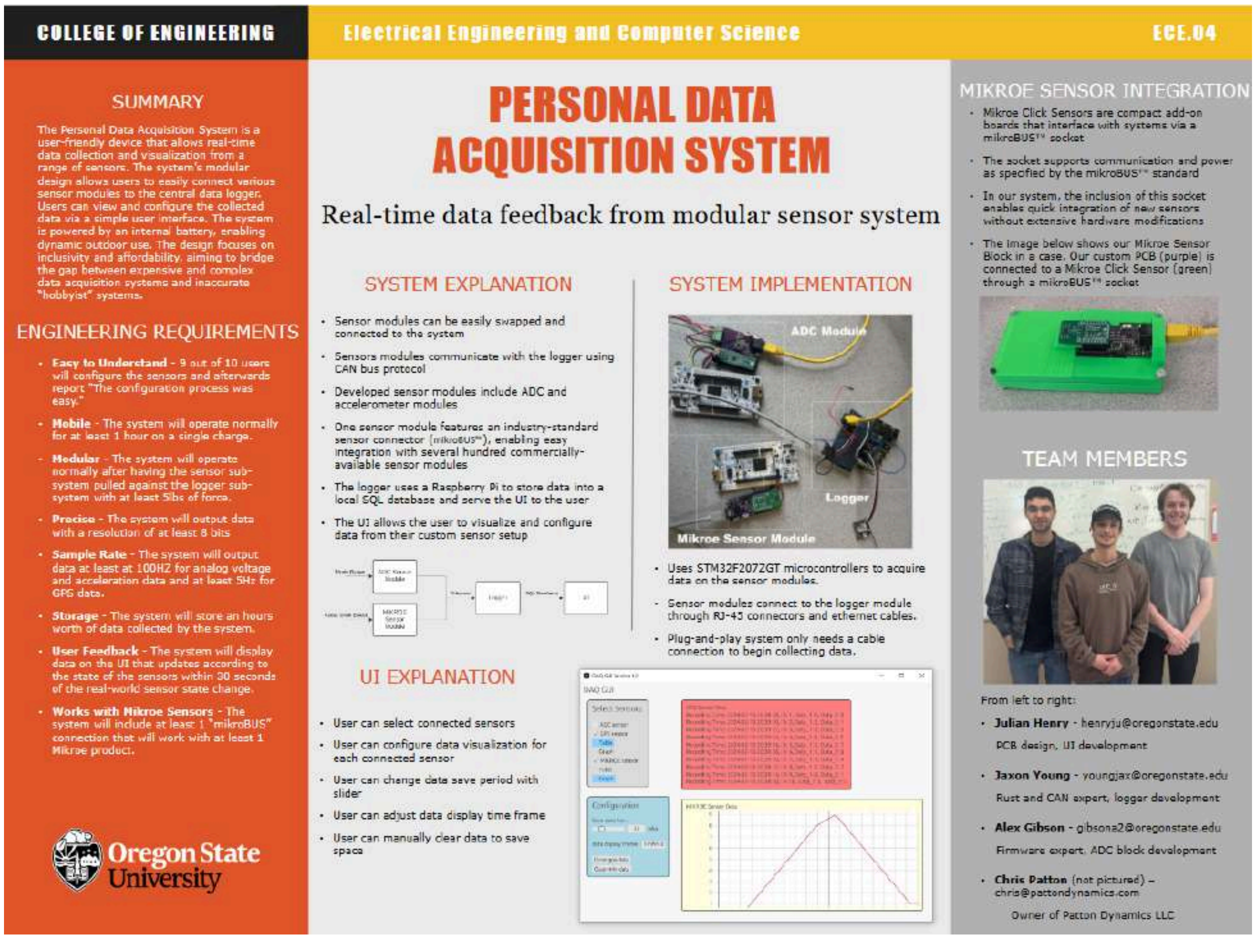
This section has the main project artifacts, including the KiCAD project directories used to create the sensor modules which contain the schematics, layouts, and links to each component purchased for the PCBs, the source code for the user interface, and the firmware written for the microcontrollers. Each of these artifacts has a link below. Additionally, a guide for setting up the logger module with the correct software was created and is provided below.

- Mikroe Sensor Module KiCAD project
- ADC Module KiCAD project
- User interface source code
- Microcontroller firmware
- Logger setup guide

6.3 Presentation Materials

This section has a link to the project on the capstone showcase site as well as an image of the project poster for the Engineering Expo.

- Link to the project showcase site for the project


COLLEGE OF ENGINEERING

Electrical Engineering and Computer Science

ECE.04

PERSONAL DATA ACQUISITION SYSTEM

Real-time data feedback from modular sensor system

SUMMARY

The Personal Data Acquisition System is a user-friendly device that allows real-time data collection and visualization from a range of sensors. The system's modular design allows users to easily connect various sensor modules to the central data logger. Users can view and configure the collected data via a simple user interface. The system is powered by an internal battery, enabling dynamic outdoor use. The design focuses on inclusivity and affordability, aiming to bridge the gap between expensive and complex data acquisition systems and inaccurate "hobbyist" systems.

ENGINEERING REQUIREMENTS

- **Easy to Understand** - 9 out of 10 users will configure the sensors and afterwards report "The configuration process was easy."
- **Mobile** - The system will operate normally for at least 1 hour on a single charge.
- **Modular** - The system will operate normally after having the sensor sub-system pulled against the logger sub-system with at least 5lbs of force.
- **Precise** - The system will output data with a resolution of at least 8 bits
- **Sample Rate** - The system will output data at least at 100HZ for analog voltage and acceleration data and at least 5Hz for GPS data.
- **Storage** - The system will store an hours worth of data collected by the system.
- **User Feedback** - The system will display data on the UI that updates according to the state of the sensors within 30 seconds of the real-world sensor state change.
- **Works with Mikroe Sensors** - The system will include at least 1 "mikroBUS" connection that will work with at least 1 Mikroe product.

Oregon State University

SYSTEM EXPLANATION

- Sensor modules can be easily swapped and connected to the system
- Sensors modules communicate with the logger using CAN bus protocol
- Developed sensor modules include ADC and accelerometer modules
- One sensor module features an industry-standard sensor connector (mikroBUS™), enabling easy integration with several hundred commercially-available sensor modules
- The logger uses a Raspberry Pi to store data into a local SQL database and serve the UI to the user
- The UI allows the user to visualize and configure data from their custom sensor setup

SYSTEM IMPLEMENTATION

- Uses STM32F2072GT microcontrollers to acquire data on the sensor modules.
- Sensor modules connect to the logger module through RJ-45 connectors and ethernet cables.
- Plug-and-play system only needs a cable connection to begin collecting data.

UI EXPLANATION

- User can select connected sensors
- User can configure data visualization for each connected sensor
- User can change data save period with slider
- User can adjust data display time frame
- User can manually clear data to save space

MIKROE SENSOR INTEGRATION

- Mikroe Click Sensors are compact add-on boards that interface with systems via a mikroBUS™ socket
- The socket supports communication and power as specified by the mikroBUS™ standard
- In our system, the inclusion of this socket enables quick integration of new sensors without extensive hardware modifications
- The image below shows our Mikroe Sensor Block in a case. Our custom PCB (purple) is connected to a Mikroe Click Sensor (green) through a mikroBUS™ socket

TEAM MEMBERS

From left to right:

- **Julian Henry** - henryju@oregonstate.edu
  PCB design, UI development
- **Jaxon Young** - youngjax@oregonstate.edu
  Rust and CAN expert, logger development
- **Alex Gibson** - gibsona2@oregonstate.edu
  Firmware expert, ADC block development
- **Chris Patton** (not pictured) – chris@pattondynamics.com
  Owner of Patton Dynamics LLC

Fig. 1 Expo poster

6.4 References (IEEE) and File Links

6.4.1 References (IEEE)

[1] "Rust Cookbook," rust-lang-nursery.github.io, https://rust-lang-nursery.github.io/rust-cookbook/ (accessed April 25, 2024).
[2] "egui Crate," docs.rs, https://docs.rs/egui/latest/egui/ (accessed April 25, 2024).
[3] "The complete PyQt5 tutorial," PythonGUIs, https://www.pythonguis.com/pyqt5-tutorial/ (accessed April 25, 2024).
[4] "ESP32," Espressif, https://www.espressif.com/en/products/socs/esp32 (accessed April 25, 2024).
[5] "Arduino Hardware," Arduino, https://www.arduino.cc/en/hardware (accessed April 25, 2024).
[6] "ESP DevKits," Espressif, https://www.espressif.com/en/products/devkits (accessed April 25, 2024).
[7] "DigiKey Main Page," DigiKey, https://www.digikey.com/en (accessed April 25, 2024).

6.5 Revision Table

TABLE I
Section 6 Revision Table

5/10/2024	Julian Henry: Updated links and images with current information. Updated section 6.1.1 with instructor feedback
04/25/2024	Alexander Gibson: Completed section 6.1.2 and added a link to the firmware folder. Julian Henry: Wrote section 6.1.1.
4/24/2024	Julian Henry: Filled out sections 6.2 and 6.3.
04/20/2024	Julian Henry: Section 6 outline created.