Livestock Gas Monitor

Tanmay Sarkar, Matthew Owen, Joaquin Salas,
Blake Schwartzkopf

Final Report

Table of Contents

CONCEPT OF OPERATIONS

REVISION – 2
25 September 2024

Concept of Operations
for
Livestock Gas Monitor

Team 28

Approved by:

_____
Matthew Owens Date

_____
Dr. John Lusher II Date

_____
Vishwam Raval Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	9/15/2024	Blake Schwartzkopf		Draft Release
2	12/5/2024	Blake Schwartzkopf	Joaquin Salas	Updating wireless communications protocol

Table of Contents

List of Figures

1. Executive Summary

Being able to provide live data for the living conditions of animals inside farms gives key insight into animals long term health. The purpose of the project is to design a fixed and durable, gas-monitoring device that can track and store data about the amount and types of gas that are present. The full list of gases that the system will track includes ammonia, carbon dioxide, methane, nitrous oxide, and hydrogen sulfide. The system will be as miniaturized as possible, while also prioritizing performance details such as power efficiency, signal processing, and data management. Our design must be able to function efficiently regardless of potential environmental challenges that may occur. The gas monitor will utilize wireless communication to transmit the readings to a database, where it will be stored and visualized. This will not only keep track of the live conditions inside the farm, but will allow the user to track the abundance of certain gases over a long period of time. The ultimate goal of the collected data is to correlate gas concentrations with animal health outcomes. This could provide valuable insights for potential improvements in farm conditions, improved quality of animal meat, and minimized cost of veterinary expenses.

2. Introduction

This Concept of Operations report will provide essential context to the gas monitoring in animal farms. The health of the animals inside a farm is one of the most important aspects of upholding the sustainability of farms in the long term. We are attempting to analyze the gases present in the animals' living spaces, which will allow clients/researchers to improve the long-term health of the animals. The more overlying objective of the system is to be able to provide live monitoring of various gases inside an animal farm. This data will be outputted and stored in a central database, where it can be viewed and analyzed. We hope to be able to make our system completely seamless and easy to use in a flexible manner.

2.1. Background

There are integrated gas sensing systems in place at certain farms, which often come with data-logging features. We commonly see fixed gas sensors that continuously measure specific gases. The point of these systems is to set off an alert if a certain gas exceeds an expected amount. While our system could be used in this manner, the primary purpose is to be able to find long term health impacts of different gases on animals. By being focused on data management and tracking over time, we can improve the conditions in which these animals are living. We plan on building a fixed gas monitor that can handle tracking multiple gases simultaneously. Having one system will benefit the user by allowing the data to be handled easily and actions can be taken while using one seamless system.

○ *2.2. Overview*

Figure 1: Concept picture of fixed sensing device

As shown above, the system will be a fixed device that will be placed in industrial animal farms. It will use sensors on a PCB, placed inside a firm housing system, and gather gas concentration data over a period of time. The microcontroller inside the system will use either LoRaWAN, Wi-Fi, or Bluetooth to transmit data. The data will then be stored inside a central database in a .JSON format, so that it is easy to work with. Each subsystem of the project will be powered from an electrical outlet. The system will be housed in a durable device, made to withstand multiple environmental elements. Finally, it will be visualized in a UI using a web application.

○ *2.3. Referenced Documents and Standards*

- Environmental Protection Agency (EPA) Regulations
- Wireless Communication Regulations (FCC)
    - FCC Part 15: Devices should not interfere with licensed communication devices and must meet low-power standards
    - EMI Compliance: Ensure that the device does not generate harmful EMI that could impact other devices.
- .JSON Standard - RFC 8259 (Microcontroller output)
- IEEE 802.11 Wireless Communications Standards
- MEMS Gas Sensors - A Review
- IEEE 2700 Standard for Sensor Performance Parameter Definitions

3. Operating Concept

3.1. Scope

The project involves the development of a durable, miniaturized environmental sensor that monitors key elements (e.g., ammonia, carbon dioxide, methane, etc.) in agricultural environments. The device will wirelessly transmit data to a central server for future analysis, enabling farmers and researchers to explore connections between gas concentrations and the wellbeing of livestock. The project goal is to create a prototype that operates with minimal power consumption while maintaining high accuracy and reliability.

3.2. Operational Description and Constraints

The environmental sensor will be deployed in agricultural fields to continuously monitor gases. Data will be collected from the sensors and transmitted wirelessly to a centralized server, where the user can access historical gas readings. The sensor must be durable enough to withstand harsh environmental conditions such as extreme temperatures, humidity, foreign objects, and physical impacts. Constraints include optimizing the design to ensure minimal power consumption and maintaining reliable communication despite interference or range limitations. The system must also be able to withstand extreme conditions while appearing sleek and innovative.

3.3. System Description

The Livestock Gas Monitoring System is designed with four key subsystems, each contributing to the detection, transmission, and collection of gas concentrations produced in animal farming environments.

1. Sensors and Signal Conditioning

The system will utilize a series of high-precision sensors connected to the Gas Monitoring System to measure the concentration of common gases emitted by livestock waste and fertilizers. The list gases include ammonia (NH_3), carbon dioxide (CO_2), methane (CH_4), nitrous oxide (N_2O), and hydrogen sulfide (H_2S).

- Signal Conditioning Circuits will amplify and filter the raw signals from the sensors, converting them into digital data. This will ensure that the gas concentration readings are precise, with minimal noise and distortion, before being processed by the microcontroller.

2. Power Distribution and Housing Unit

The system will require an intricate and precise power supply circuitry that converts AC power from an outlet to DC power for various components. It will involve designing a PCB to meet all the power needs of all other subsystems in the project.

Power will be needed for the sensor circuitry, a microcontroller, PCB's, etc. Additionally this subsystem will also have the task of designing a housing unit that will compact, sleek, and provide protection for sensors, PCBs, and the microcontroller.

- The PCB designed in this subsystem will first take the AC power from a wall outlet and implement rectifiers to convert it to DC. Once this voltage has reached the PCB, it will be manipulated to reduce noise in the voltage. A buck-boost converter will then adjust the voltage level according to the specific needs of the rest of the subsystem. Lastly, all power will be routed and delivered to all other parts of the gas monitoring system..
- Development of a compact and most importantly extremely durable housing unit. This housing will serve the purpose of protecting the different sensors, PCB's, circuitry, and the microcontroller. Durability is one of the most important aspects of this design as it must be built to design and withstand harsh environmental conditions. It must stand firm against bumps, drops, and any other possible damages.

3. Microcontroller and Data Transmission

The system will be controlled by a microcontroller that manages data collection from several sensors, processes the signal data, and transmits the data to a centralized server. This subsystem includes:

- Data Formatting and Preprocessing: The microcontroller will apply necessary calibration algorithms from several sensors the device will have to prepare for transmission.
- Wireless Communication Module: The data will be transmitted wirelessly using a suitable communication protocol such as Wi-Fi, LoRaWAN, Zigbee, or Bluetooth. These protocols will be compared based on their reliability, long-range capabilities, and low power consumption.
- Data Rate Control: The user will be able to manually adjust the rate of data transmission. Data rate is important to control due to factors such as network conditions.

4. Data Management and Web Application

The system will interface with a web-based application that will take in the data from the microcontroller, visualize it, and provide the user with an interface that can easily be used. The aim is to have the data outputted in a .JSON format, so that it can be easily handled in the post-measuring phase. Ultimately, after receiving the data from the microcontroller, the web application will provide the user with:

- Real-Time Data Visualization: Users will have access to real-time data on gas concentrations, displayed in dashboards with visual aids.

- Data Storage and Historical Analysis: One major objective of the system is to allow users to track long-term data. We will store the data for a long period of time so the user can utilize this feature to see trends in the gas content of the area.
- AI and Machine Learning for Data Analysis: One long-term goal of the system that could be included in the future is a predictive algorithm that will leverage past trends of gas content in order to predict what the gas abundance might look like in the future. This can be used to preserve the health of the animals ahead of time. This would require a certain amount of data already measured inside a farm, which can be used to train the algorithm.

3.4. Modes of Operations

The Gas Monitoring System will have a single mode of operation which is active monitoring mode (automatic). The system will continuously collect data from various gas sensors found within the monitor. As data is being collected (every 5-30 seconds) it will then transmit to a central server, where readings will be stored and visualized in a user-friendly interface. The device will include a display to show an indicator that the device is transmitting.

3.5. Users

The gas monitoring system will be marketed for farmers and industrial livestock production facilities, it will provide users the ability to monitor the gas emissions in farms. The ability to monitor gases found in livestock farms and access historical data will enable researchers and farmers to gain a deeper understanding of their production environments and make data-driven decisions to improve quality of meat, decrease amount spent on veterinary expenses, and enhance sustainability.

3.6. Support

Support for the gas monitoring system will be provided as a user operators manual providing detailed instructions on installation, maintenance, and sensor calibration guides. The manual will also include instructions on how the user can interact with the web based interface and add/drop monitors to the system. The system as a whole will be designed to be both intuitive and user friendly to ensure it maintains the ability to be easily used and implemented.

4. Scenario(s)

4.1. Livestock Health and Environment monitoring

The main use case for the Gas Monitoring System is to improve the health of livestock by monitoring the air quality in animal housing areas such as barns. These housing areas will be monitored for harmful gases that often show up here. Gases like ammonia can negatively impact animal health by impacting respiratory health. By using the Gas Monitoring System, farmers can swiftly identify the potentially harmful conditions in the housing areas and address the conditions immediately. This use case will help to promote healthier livestock that is more productive.

5. Analysis

5.1. Summary of Proposed Improvements

- The user will be able to control rates of data transmission.
- The system will interface directly with a web-based application that will implement predictive algorithms to forecast future gas concentrations.
- All proposed components will be implemented on a System-On-Chip.
- The system will be easy to install on posts, fences, and other objects common in an agricultural setting.
- Housing for the system will be durable and able to withstand weather conditions and contact from livestock.

5.2. Disadvantages and Limitations

- The proposed system will only measure ammonia, carbon dioxide, methane, nitrous oxide, and hydrogen sulfide.
- The system only interfaces with a web-based application that will be created. Other applications for data analysis will be unsupported.
- The user's device needs to be compatible with the web application.

5.3. Alternatives

Alternatives to the Livestock Gas Monitoring System include:

- No gas monitoring sensors could be present in the livestock farm.
- Less gases could be measured by the sensors.
- Gas concentrations could be sampled with a handheld sensor regularly.
- A battery management system could be implemented to provide redundancy.
- Communications are wired rather than wireless.
- The data could be left to be organized and analyzed by the user.

5.4. Impact

Economic Impacts:

- By determining the relationship between atmospheric conditions and animal health, this data can inform decision making crucial to keeping livestock healthy.
- Expenses are reduced due to less veterinary visits.
- The farmer yields higher quality meat and other animal products.

Health and Safety Impacts:

- Livestock is at less risk of inhaling toxic concentrations of gases that could lead to injury or death.
- Toxic concentrations of gases can be detected before humans are exposed to them.

Social Impacts:

- Ranchers have less reason to worry about dangerous amounts of certain gases harming their livestock.

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – Draft
25 September 2024

Functional System Requirements
FOR
Livestock Gas Monitor

Team 28

Approved by:

_____
Matthew Owens Date

_____
Dr. John Lusher II Date

_____
Vishwam Raval Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	9/25/2024	Blake Schwartzkopf		Draft Release
2	12/5/2024	Blake Schwartzkopf	Joaquin Salas	Updating wireless communications protocol

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Purpose and Scope

The purpose of this document is to outline the detailed system requirements for the Gas Monitoring System. The system aims to measure and store data on the concentrations of gases such as ammonia, carbon dioxide, methane, and hydrogen sulfide. By collecting data about the environments in which animals live, farmers will be able to make informed decisions to improve animal health. Enhanced animal health can lead to reduced veterinary costs, higher meat quality, and increased profits for farmers.

Figure 1. Gas Monitoring System Concept Image

1.2. Responsibility and Change Authority

Requirements can only be changed with the approval of the team leader Joaquin Salas, Professor John Lusher, and Justin Houck. The individual subsystem owner is responsible for the fulfillment of the requirements of their subsystem.

Subsystem	Responsibility
Sensors and Signal Conditioning	Blake Schwartzkopf
Power Distribution and Housing Unit	Matthew Owen
Microcontroller and Data Transmission	Joaquin Salas
Data Management And Web Application	Tanmay Sarkar

Table 1. Subsystem Responsibility

2. Applicable and Reference Documents

2.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Document Number	Revision/Release Date	Document Title
RFC 8259	2017	JSON Data Interchange Format: Defines the standards for formatting the data transmitted by your microcontroller
FCC Part 15	2006	Federal Communications Commission Regulations: Governs wireless communication devices to ensure low power emissions and avoid interference with licensed communication services

Table 2. Applicable Documents

2.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Document Number	Revision/Release Date	Document Title
EN 60529	N/A	Ingress Protection Rating

Table 3. Reference Documents

2.3. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

3.1. System Definition

This section outlines the requirements for the Gas Monitoring System, which is essential to make sure all system needs of design, functionality, and performance are met. The Gas Monitoring System is a durable, accurate, and reliable instrument for monitoring concentrations of various gasses commonly found in industrial livestock environments. It allows users to view concentration levels in a historian to conduct further research and improve livestock production. The Gas Monitoring System has four subsystems: Gas Sensors and Signal Conditioning, Power and Housing Unit, Microcontroller and Data Transmission, and Data Management and Web Application. We have set these requirements outlined in this document to ensure each subsystem will contribute to the overall functionality of the system.

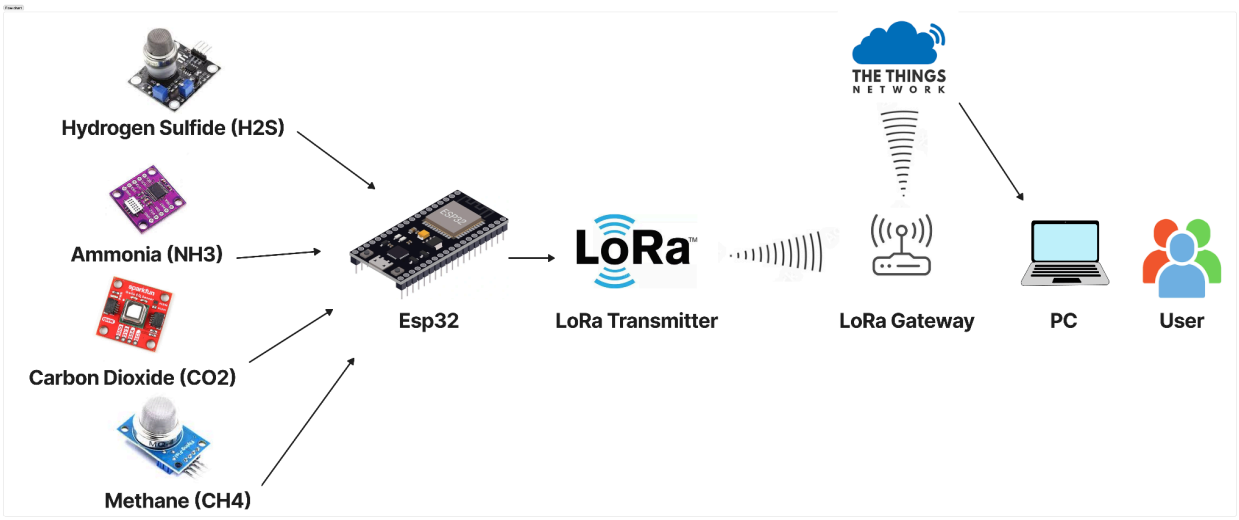


Figure 2. Block Diagram of System

Gas sensors measure methane, ammonia, carbon dioxide, and hydrogen sulfide, with three providing analog outputs and one offering digital I2C output. The analog outputs are converted into digital SPI signals using an AD7718 analog-to-digital converter, allowing the ESP32 microcontroller to gather and process the data. The system is powered by a battery management unit housed in a weather-resistant casing to protect the internal components from harsh farm conditions. The ESP32 microcontroller is also responsible for communicating with the Soft Access Point, which sends the collected data to a nearby ESP32 Client/Station for long-range transmission.

Once transmitted, the data is sent from the ESP32 Client/Station to a central server using TCP/IP and UART. The server stores the data for long-term analysis and provides historical records for real-time monitoring via a web application. Additionally, a machine learning model analyzes the collected data, offering predictive insights into how the gas levels

impact livestock health. This allows farmers to make informed decisions that can improve the living conditions of the animals and increase farm productivity while reducing costs.

3.2. Characteristics

3.2.1. Functional / Performance Requirements

3.2.1.1. Gas Detection Range of Ammonia

The Gas Monitoring System shall be able to detect ammonia between 5 and 500 ppm.

Rationale: The average amount of ammonia found in poorly ventilated animal farms is approximately 100-200 ppm.

3.2.1.2. Gas Detection Range of Carbon Dioxide

The Gas Monitoring System shall be able to detect carbon dioxide between 400 and 5000 ppm.

Rationale: The average amount of carbon dioxide found in animal farms is approximately 1000-3000 ppm.

3.2.1.3. Gas Detection Range of Methane

The Gas Monitor System shall be able to detect methane between 300 and 10000 ppm.

Rationale: The average amount of methane in the respiration air of cattle is 1000 ppm.

3.2.1.4. Gas Detection Range of Hydrogen Sulfide

The Gas Monitor shall be able to detect hydrogen sulfide between 1 and 200 ppm.

Rationale: The average amount of hydrogen sulfide in animal farms is 2-10 ppm.

3.2.1.5. Gas Detection Error

The Gas Monitoring System concentration readings for each gas shall not be more than 10% off from the actual gas concentration.

Rationale: The system should be able to detect a good estimate of the gas concentration in ppm. 10% is a reasonable deviation for the budget allocated to this project.

3.2.1.6. Data Transmission Rate

The microcontroller will transmit data between 30 seconds and 1 minute.

Rationale: The data transmission rate will be adjusted depending on the users requirements.

3.2.2. Physical Characteristics

3.2.2.1. Mass

The mass of the Gas monitoring System shall be less than or equal to 4 kilograms.

Rationale: The gas monitoring system must be lightweight to be mounted easily. It must be easily installed and accessible without too much unnecessary weight.

3.2.2.2. Volume Envelope

The volume of the Gas monitoring System shall be less than or equal to 15 inches in height, 15 inches in width, and 4 inches in length.

Rationale: The gas monitoring system needs to be able to be mounted inside of animal farms without affecting the animals.

3.2.2.3. Water Resistance

The Gas Monitoring System shall have water resistant housing to protect the electrical elements. The sensors should be able to detect gases outside of the case.

Rationale: IP 31 is a standard rating that offers protection against water and foreign objects. Some animal farms will not be indoors. Electrical components need to be protected in outdoor conditions.

3.2.2.4. Collision Resistance

The Gas Monitoring System shall have collision resistant housing to protect the electrical elements. The sensors should be able to detect gases outside of the case.

Rationale: Animals will likely collide with the system, the housing needs to protect the electrical components from these collisions.

3.2.2.5. Mounting

The Gas Monitoring System will support multiple options for mounting, including but not limited to pole, wall, and fence installations. The exact method of mounting will be determined after a tour of the animal farms has taken place.

Rationale: Many of the animal farms may vary significantly from one another, thus a flexible mounting system will be needed to adapt to the specific needs of each farm.

3.2.3. Electrical Characteristics

3.2.3.1. Voltage Input Range

The input voltage for the Gas Monitoring System will be AC voltage. The voltage will be 120V ± 10%, 60 Hz.

Rationale: Standard outlets where this system will be plugged in are typically 120v AC at 60 Hz. The system will need to accommodate this input voltage.

3.2.3.2. AC to DC Conversion

The Gas Monitoring System will have an internal AC to DC converter circuit that will output voltages ranging from 1V to 12V of DC power to internal systems.

Rationale: The internal subsystems will operate and require lower DC voltages than what is inputted. An internal conversion is required to provide stable voltage to the rest of the system.

3.2.3.3. Buck-Boost Voltage Regulation

The system will include the required voltage converters that can step down the DC voltage received to varying values of voltage needed to power other internal components.

> *Rationale: Buck-boost converter is needed in order to provide a consistent and stable voltage supply to important components of the system.*

3.2.3.4. Voltage Filters

The power subsystem will have the required voltage filter circuits with the purpose of suppressing noise and any interference from the output of the buck boost circuits.

> *Rationale: We expect some farm environments to have noisy electrical systems so we must ensure all components are free from anything that can negatively affect the performance of electronics.*

3.2.4. Output

3.2.4.1. Data Output

The Gas Monitoring System shall include a wireless interface for transmitting gas concentration data to a central server. The system will support Wi-Fi for long-range communication.

> *Rationale: Data will be transmitted over Wi-Fi to a centralized server for visualization and analysis. Wi-Fi offers long-range, low-power communication, which is essential for agricultural applications where devices may be far from the receiver.*

3.2.4.2. Diagnostic Output

The Gas Monitoring System shall include a diagnostic interface to monitor system health and sensor functionality.

> *Rationale: The diagnostic output will help with debugging and allow for real-time error detection.*

3.2.4.3. Reliable Data handling & Error checking

The Gas Monitoring System will ensure that the data received from the gateway is all in the correct format (JSON) and that none of it was lost during transmission.

> *Rationale: This could be done using buffer handling and timestamps for when the measurements are being taken. The efficiency of getting the data from the gateway to the server is also crucial in reliable data handling. Error handling is also included in this requirement. It will be crucial to ensure that the data being received has not been corrupted or contains invalid values.*

3.2.4.4. Scalable Storage

The Gas Monitor will have a scalable server for increased amounts of data over time.

> *Rationale: This requirement is crucial because one of the key components of the system is being able to handle large volumes of data over time. The long-term goal of this is to see the effects of certain gas amounts on the health of the animal. MySQL will be used*

because it supports horizontal scaling. In order to optimize storage use, the system shall also automatically delete older data.

3.2.4.5. Database and Web Application Security

The Gas Monitoring system will encrypt communication between the transmitter, server, and web application.

Rationale: It is important that the gas data is encrypted so that there is no unauthorized use. In addition to this, the web application will have a secure authentication mechanism so that only the intended users can access the data.

3.2.4.6. Overall System Latency

The Gas Monitoring System will aim to concurrently update the web application with gas value updates as they are ready.

Rationale: The aim is to have the data be updated in an efficient manner by using real time API handling. It may not be live updating but we aim to have a refresh of data every 5 minutes at the maximum. This means that the user can have up to date information regarding the gas abundances in a certain location.

3.2.4.7. User Login Privacy

The Gas Monitoring System will comply with several user breach laws that apply in Texas.

Rationale: Because the system will have role-based access and the user will have to create an account, the system will have to adhere to regulations such as the Texas Privacy Protection Act (HB 3746) and the Texas Identity Theft Enforcement and Protection Act (ITEPA).

3.2.4.8. Thermal

The Gas Monitoring System will need to work properly in an environment with temperatures ranging from - 20°F to 110°F

Rationale: Animal farms can have a wide range of temperatures. Sponsor specified the need for the gas monitoring system to be durable and have the ability to work in a large range of temperatures.

3.2.4.9. Humidity

The Gas Monitoring System will need to work properly in an environment with humidity ranging from 0% to 100%.

Rationale: The outdoor environment will have extreme humidity variations, from dry to high moisture levels. The system must operate reliability in these conditions to ensure accurate gas measurements.

4. Support Requirements

The system will include all the sensors, a microcontroller which will communicate with the central database, and a web application that will gather the data and visualize it. The system will also be placed in a housing system that will be able to be plugged into a wall outlet for power distribution. The only requirement for the user of the system will be to have a device with an internet connection. If possible, the user being able to have basic technological skills could be necessary. This would help the user be able to read the data from the web application and draw a conclusion based on it. An additional goal is to have the system be able to output a small summary of what the raw data shows, along with possible actions that could be taken based on it.

• Appendix A: Acronyms and Abbreviations

V	Voltage
AC	Alternating Current
DC	Direct Current
EMC	Electromagnetic Compatibility
PPM	Parts Per Million
PCB	Printed Circuit Board
Hz	Hertz
kHz	Kilohertz
A	Amps
mA	Milliamp
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
MQTT	MQ Telemetry Transport
HTTP	Hypertext Transfer Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol

- # Appendix B: Definition of Terms

INTERFACE CONTROL DOCUMENT

REVISION – Draft
25 September 2024

Interface Control Document
for
Livestock Gas Monitor

Team 28

Approved by:

_____
Matthew Owens Date

_____
Dr. John Lusher II Date

_____
Vishwam Raval Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	9/25/2024	Blake Schwartzkopf		Draft Release
2	12/5/2024	Blake Schwartzkopf	Joaquin Salas	Updating wireless communications protocol

Table of Contents

List of Tables

List of Figures

1. Overview

The Interface Control Document (ICD) for the Gas Monitoring System will outline how each of the subsystems within the project will interface with one another. It will provide a detailed summary of all physical, electrical, and data interfaces between each subsystem. This document explains all inputs and outputs of each subsystem to ensure that there is compatibility between each subsystem. Lastly, this document acts as a verification tool during development and testing, helping ensure the system works as expected when all subsystems are integrated.

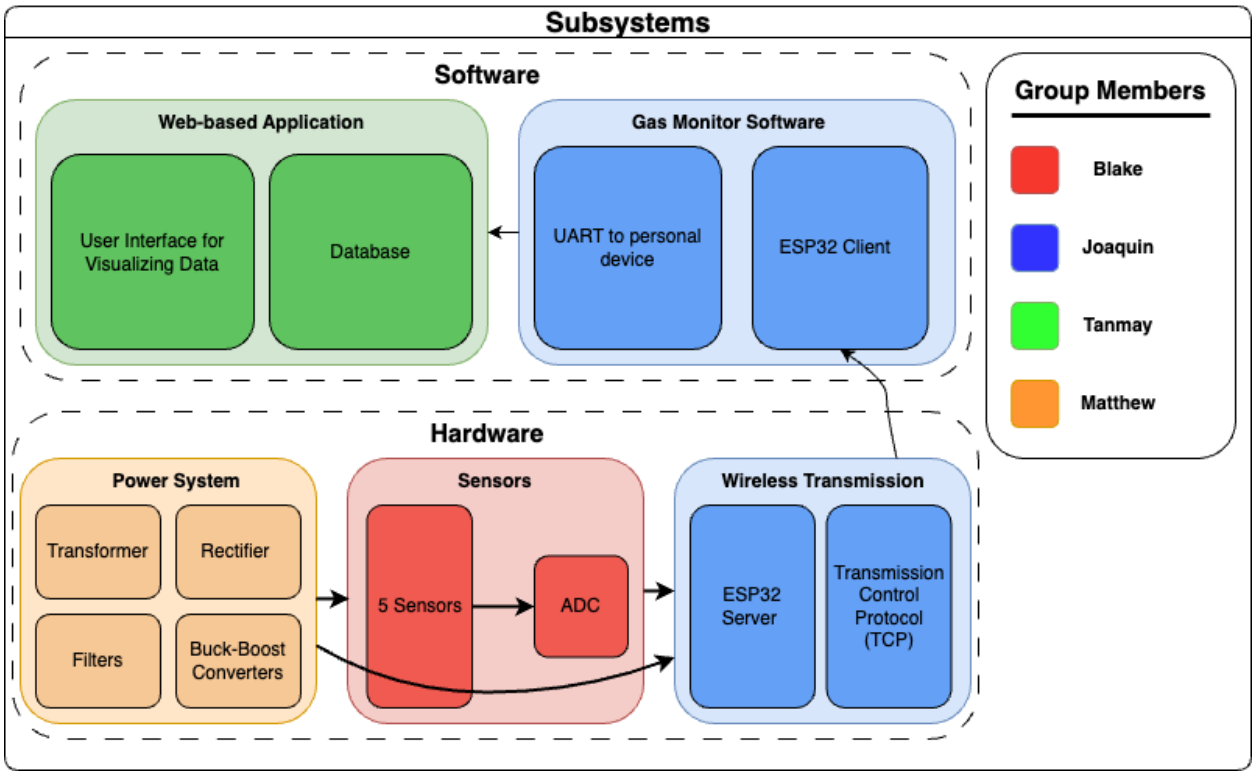


Figure 1. Subsystems

2. References and Definitions

2.1. References

- Wireless Communication Regulations (FCC):
    - FCC Part 15: Devices should not interfere with licensed communication devices and must meet low-power standards
    - EMI Compliance: Ensure that the device does not generate harmful EMI that could impact other devices.

2.2. Definitions

V	Voltage
AC	Alternating Current
DC	Direct Current
EMC	Electromagnetic Compatibility
PPM	Parts Per Million
PCB	Printed Circuit Board
Hz	Hertz
kHz	Kilohertz
A	Amps
mA	Milliamp
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface

3. Physical Interface

Provide details on the physical interface. Examples are:

3.1. Weight

3.1.1. Main PCB

Component	Weight	Number of Items	Total Weight
MQ-137 Ammonia Sensor	4.070 g	1	4.070 g
SCD41-D-R2 Carbon Dioxide Sensor	0.600 g	1	0.600 g
MQ-4 Methane Sensor	5.000 g	1	5.000 g
MQ136 Hydrogen Sulfide Sensor	9.000 g	1	9.000 g
HDC1080DMBR Temperature Sensor	0.160 g	1	0.160 g
AD7718 Analog to Digital Converter	0.403 g	1	0.403 g
esp32 dev board v4	10.000 g	1	10.000 g
LoRa Module	8.000 g	1	8.000 g

Table 1: Main PCB Weight

3.1.2. Power PCB

Component	Weight	Number of Items	Total Weight
Transformer TCT50-03E07K	907.2 g	1	907.2 g
Diode Bridge Rectifier	36g	1	36g
Buck-Boost	TBD	TBD	TBD

Table 2: Power PCB Weight

3.2. Dimensions

3.2.1. Dimensions of Sensor Subsystem

Component	Length	Width	Height	Diameter
MQ-137 Ammonia Sensor	N/A	N/A	24 mm	19 mm
SCD41-D-R2 Carbon Dioxide Sensor	10.1 mm	10.1 mm	6.5 mm	N/A
MQ-4 Methane Sensor	N/A	N/A	24 mm	19 mm
MQ136 Hydrogen Sulfide Sensor	N/A	N/A	24 mm	19 mm
HDC1080DMBR Temperature Sensor	3.1 mm	3.1 mm	0.8 mm	N/A
AD7718 Analog to Digital Converter	9.7 mm	4.4 mm	1.05 mm	N/A

Table 3: Dimensions of Sensor Subsystem

3.2.2. Dimension of Microcontroller Subsystem

Component	Length	Width	Height	Diameter
ESP32-S2-WROVER-I	18mm	31mm	3.3 cm	N/A
LoRa Module	28 mm	20.3 mm	5 mm	N/A

Table 4: Main PCB Weight

3.2.3. Dimensions of Power Subsystem

Component	Length	Width	Height
Transformer TCT50-03E07K	92.46mm	56.67mm	57.40mm
Diode Bridge Rectifier	TBD	TBD	TBD
Buck-Boost	TBD	TBD	TBD

Table 5: Dimensions of Power Subsystem

3.3. Mounting Locations

The Gas Monitoring System will be installed across industrial agriculture settings in the state, ensuring unobstructed access to the air being measured. The monitor will be mounted on posts, fences, or other structures commonly found in animal farms. Using the LoRaWAN wireless protocol, the system can transmit data efficiently from distances of 10-15 km, making it ideal for deployment in remote locations.

Figure 2. Concept Image of Mounting Location

4. Thermal Interface

The Gas Monitoring System will operate in industrial agricultural environments where temperature fluctuations are common. Due to the low power consumption of the sensors, microcontroller, and transmitter, active cooling is not required. Passive thermal management through the housing material will ensure any heat generated internally is dissipated and effectively. The design will allow the system to function reliably in both high and low-temperature conditions.

5. Electrical Interface

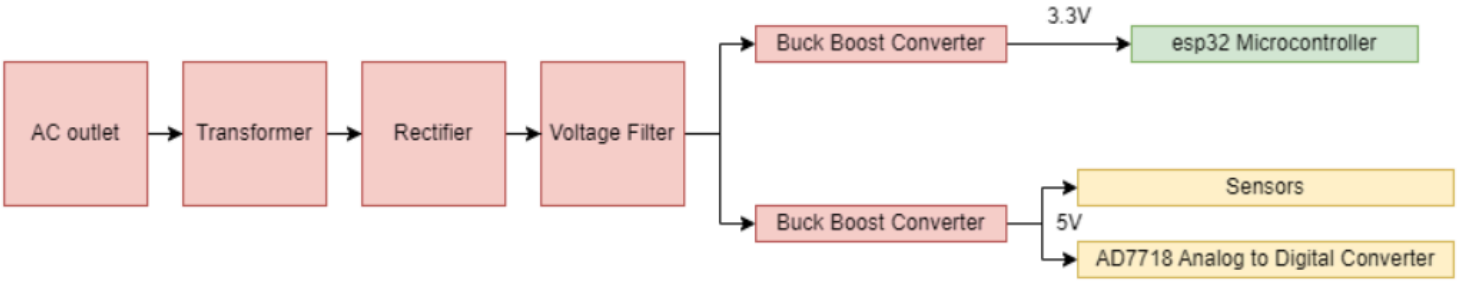


Figure 3: Electrical Interface

5.1. Primary Input Power

All power necessary for the Gas Monitoring System will be supplied through a 120 V AC connection with a standard US wall outlet. Inputted power will be sent to the power PCB subsystem where it will be stepped down, rectified, and filtered. The DC voltage will then be stepped down to 3.3V for the microcontroller subsystem and 5V for the sensor subsystem.

5.2. Voltage and Current Levels

5.2.1. Maximum Values

Component	Voltage (V)
MQ-137 Ammonia Sensor	5
SCD41-D-R2 Carbon Dioxide Sensor	5
MQ-4 Methane Sensor	5
MQ136 Hydrogen Sulfide Sensor	5
HDC1080DMBR Temperature Sensor	5
AD7718 Analog to Digital Converter	5
ESP32-S2-WROVER-I	3.3

Table 6: Voltage and Current Level

5.3. Signal Interfaces

The Gas Monitoring System utilizes both analog and digital signal interfaces. Three gas sensors provide analog outputs, which are converted to digital signals using AD7718

analog-to-digital converter (ADC). These SPI digital outputs, along with the I2C digital outputs from other sensors, are transmitted to the ESP32 microcontroller for processing. The LoRa module will be connected to the appropriate GPIO pins on the ESP32 and will transmit the processed data using the LoRaWAN wireless protocol to a nearby gateway, which forwards the data to a database for data visualization.

5.4. User Control Interface

The user control interface is a web application that interacts with the system's database. It allows the user to adjust the data transmission rate, view live data streams, access historical data, and interact with the diagnostic interface. On the web application, the user will also have access to downloading the current data that is on it, and have the freedom to draw conclusions based on that.

6. Communications / Device Interface Protocols

6.1. Wireless Communications

6.1.1. Wi-Fi

The microcontroller will use Wi-Fi to transmit sensor data. The microcontroller will connect to a ESP32 Server transmitter, which sends the data to a ESP32 Client/Station.

6.1.2. Database

The collected data will be received in the ESP32 Client, where a python script will be used to forward data to a local database using the UART to pull data from the ESP32 Client for storage and accessing.

6.1.3. Web Application

The web application will retrieve the data from ESP32 Client using UART, and it will be sent in a .JSON format, making the data easy to handle once it reaches the web application.

SCHEDULE AND VALIDATION

REVISION – 2
25 September 2024

Schedule:

Understand project and requirements	8/27/2024	All	
Research	9/13/2024	All	
Complete Conops	9/15/2024	All	
Complete FSR	9/26/2024	All	
Complete ICD	9/26/2024	All	
Complete Validation Plan	9/26/2024	All	
Complete Schedule	9/26/2024	All	
Order Parts	9/27/2024	All	
Learn about HTML/JavaScript/MySQL and compare	9/30/2024	Tanmay	
Learn about JSON/Begin basic structure for Web App	9/30/2024	Tanmay	
Complete Subsystem Introduction Project	9/30/2024	All	
Mid-Term Presentation	10/2/2024	All	
Create part symbols	10/3/2024	Blake	
Create part footprints	10/3/2024	Blake	
Begin creating a test environment for data handling	10/7/2024	Tanmay	
Ensure that Web App can retrieve data from Database	10/7/2024	Tanmay	
Design schematic	10/15/2024	Blake	
Implement I2C for Sensors	10/9/2024	Joaquin	
Wireless Communications to database	10/9/2024	Joaquin	
Design Circuit Schematic	10/10/2024	Matthew	
PCB routing	10/17/2024	Blake	
Web Application be able to retrieve/visualize test data	10/14/2024	Tanmay	
Final design check	10/22/2024	Blake	
Order PCB	10/22/2024	Blake	
Build Circuit In Altium (FootPrints, Schematic, routing)	10/18/2024	Matthew	
Status Update Presentation	10/21/2024	All	

Learn to solder	10/22/2024	Blake	
Program Microcontroller for Demo	11/21/2024	Blake	
Final Design Check	10/23/2024	Matthew	
Order PCB from PCB Way	10/25/2024	Matthew	
Order parts	10/25/2024	Matthew	
Improve Web App Aesthetics/Functionality/Efficiency	10/28/2024	Tanmay	
Build function for sensor inputs	10/29/2024	Joaquin	
Finnish Communications to database	10/29/2024	Joaquin	
Test wireless communications with data	10/29/2024	Joaquin	
Solder parts onto PCB	11/5/2024	Blake	
Solder parts onto PCB	11/5/2024	Matthew	
Test op amp and voltage divider	11/7/2024	Blake	
Test SPI communication from analog to digital converter	11/24/2024	Blake	
Test I2C communication from digital sensors	11/24/2024	Blake	
Test and Debug PCB	11/26/2024	Matthew	
Final Presentation	11/18/2024	All	
Final Demo	11/26/2024	All	
Final Report	12/5/2024	All	

Validation Plan:

Task	Specification	Result
Data Updating	<5 Mins	Validated
Data Transmission	<5 Seconds	Validated
Op Amp Voltage Same on Both Sides	< .1%	Validated
Voltage Divider cuts Voltage in Half	< .2%	Validated
Analog to Digital Conversion	Conversion successful	Validated
I2C Communications to Microcontroller	Data received from digital sensors	Validated
SPI Communications from ADC	Data received from ADC	Validated
Output Voltage	5V, 3.3V, -5V	Validated
Output Current	Max of 1A	Validated
Voltage Rectified	12VDC +-2V	Validated
Voltage From Transformer	12V AC +-2V	Validated
MCU Data Transmission	Data transmission to database	Yes, first tests up to 45 yards
Transmitter Range	100 feet	Yes
Transmitter Frequency	30 sec - 1 min	5 second frequency

SUBSYSTEM REPORTS

REVISION – Draft
25 September 2024

SUBSYSTEM REPORTS
FOR
Livestock Gas Monitor

TEAM 28

APPROVED BY:

_____
Matthew Owen Date

_____
Dr. John Lusher II Date

_____
Vishwam Raval Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	12/5/2024	Blake Schwartzkopf		Draft Release

Table of Contents

List of Tables

List of Figures

1. Introduction

This part of the report is a comprehensive overview of each of the subsystems that have been developed throughout time in ECEN 403. The four main subsystems that will be addressed are the power subsystem, sensor subsystem, microcontroller and data transmission subsystem, and the database and web application subsystem. Each subsystem serves an important role in ensuring the functionality and integration of the Gas monitoring system. This section will go in depth into the validation progress each subsystem has made. Additionally this report will address the current difficulties we have encountered and the readiness for full integration. The overall goal of the system is to monitor and analyze the gas concentrations in agricultural environments to improve animal health, optimize farm operations, and provide long term data.

2. Power Subsystem Report (Matthew Owen)

2.1. Subsystem Introduction

The power subsystem is responsible for supplying voltage that can handle adequate load across the entire system. Included in this subsystem is a transformer that converts 120VAC to 12VAC, a rectification circuit that converts 12VAC to 12VDC, and a variety of voltage regulating circuits that output 5V, -5V, and 3.3V. This system was designed and fabricated to ensure the voltage had minimal load and could handle up to 1A of load. Below is an image of the full power subsystem, a printed circuit board wired from the transformer to the board, and then connected to a power outlet.

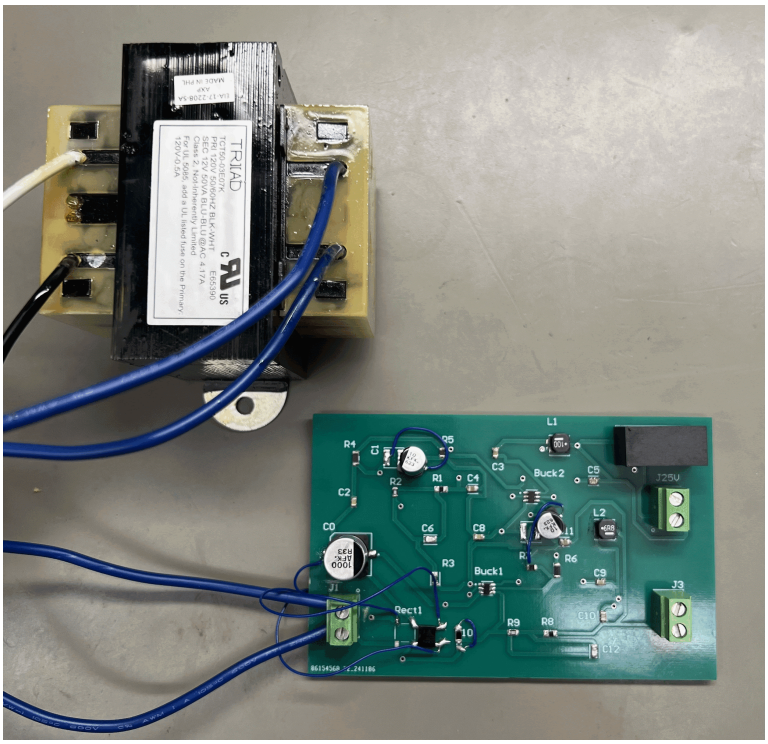


Figure 1: Power Subsystem Final

2.2. Validation

2.2.1. *Transformer*

The transformer that I used for this part of the project was a TCT50-03E07K. I have it wired to a cord that plugs directly into a wall outlet. I have verified that it worked as intended, with a voltage output of 12V at 3A. When the transformer is connected to load, it can successfully handle loads up to 3A, which is more than what is necessary for this project.

2.2.2. Rectification

To handle the conversion between AC and DC voltage I trusted the ABS210 bridge rectifier. The main reason it was chosen was due to the high current load availability. The voltage that was fed into this circuit correctly output 10.287V under a load of 1A. This voltage was more than enough to supply the voltage regulator circuits.The image below shows the rectification circuit that was implemented along with a 1000uf filtering capacitor.

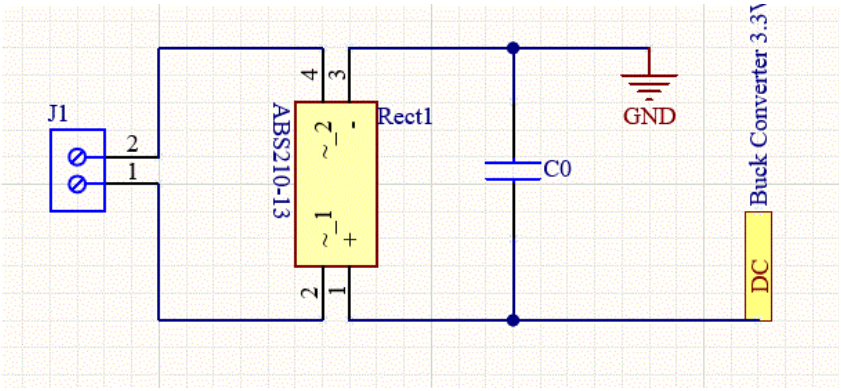


Figure 2: Rectification Circuit

2.2.3. Noise

During the validation process, I thoroughly tested the noise levels that were present in the system to ensure that the noise in the output voltages is minimal. This is a vital part of the subsystem to ensure clean voltage to the microcontroller and the sensors. I have attached 4 images below: two are for the 3v output and two for the 5v output. As shown in the images noise peaks around nearly 100mV, but the ripple after quickly decreases to next to nothing. This is what we expected as the output and is largely due to how I implemented the rectification circuit. The filtering capacitors successfully minimize the majority of the potential noise. For the requirements of this power supply, this noise level is well suited to ensure the reliability of the system.

Figure 3/4: Oscilloscope screenshots of Noise in 5V output

Figure 5/6: Oscilloscope screenshots of Noise in 2.7V output

2.2.4. Voltage Outputs

In this part of the validation, I verified that my voltage outputs would be sufficient to power the rest of the system. The output voltages from my PCB were 5.206V and 2.952V. The 5V output is within the tolerance to power multiple sensors, while the 2.9524 is between the 1.9V and 3.3V range necessary to supply the ESP32. However, I was unable to get the -5V output due to a faulty inverter component. Moving forward, I plan to replace this part and ensure the system gets the -5V output necessary to gain full functionality.

2.2.5. Load Testing

The final part of my validation was testing the voltage outputs under load, up to 1A. I have attached images below of the voltage outputs under 0.5A of load. As demonstrated during the demo, the voltages are able to sufficiently supply the necessary voltage at this load level. However, due to constraints in some of the buck converter chips, the current max load that the board can supply was about 0.5A. Moving forward next semester I plan to integrate the same chips but in a different version that can supply up to 1A. The images below are from a demonstration of my board working using the electronic load machine.

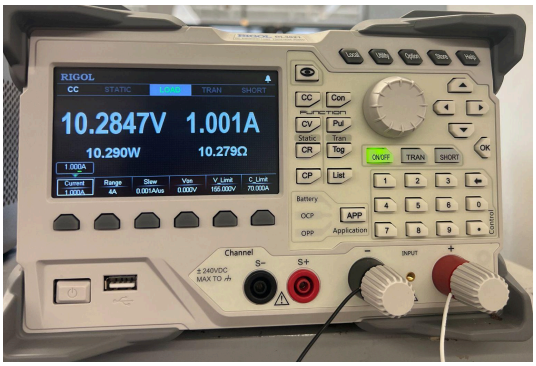


Figure 7: rectified voltage under 1A load

Figure 8: 5V output under 0.419A of load

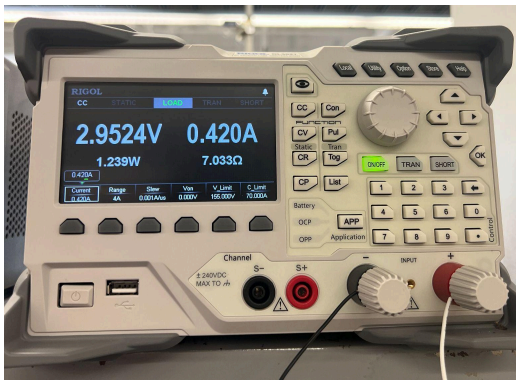


Figure 9: 3V output under 0.420A of load

2.3. Conclusion

The power subsystem has successfully supplied the critical voltage levels that are required in the rest of the system. The main corrections that I need to fix in ECEN 404 would be to enhance the board's ability to handle load levels up to 1A and also replacing the inverter to ensure -5V output voltage. With these main corrections the power subsystem is fully ready for the integration with the other subsystems in the project.

3. Sensor Subsystem Report

3.1. Subsystem Introduction

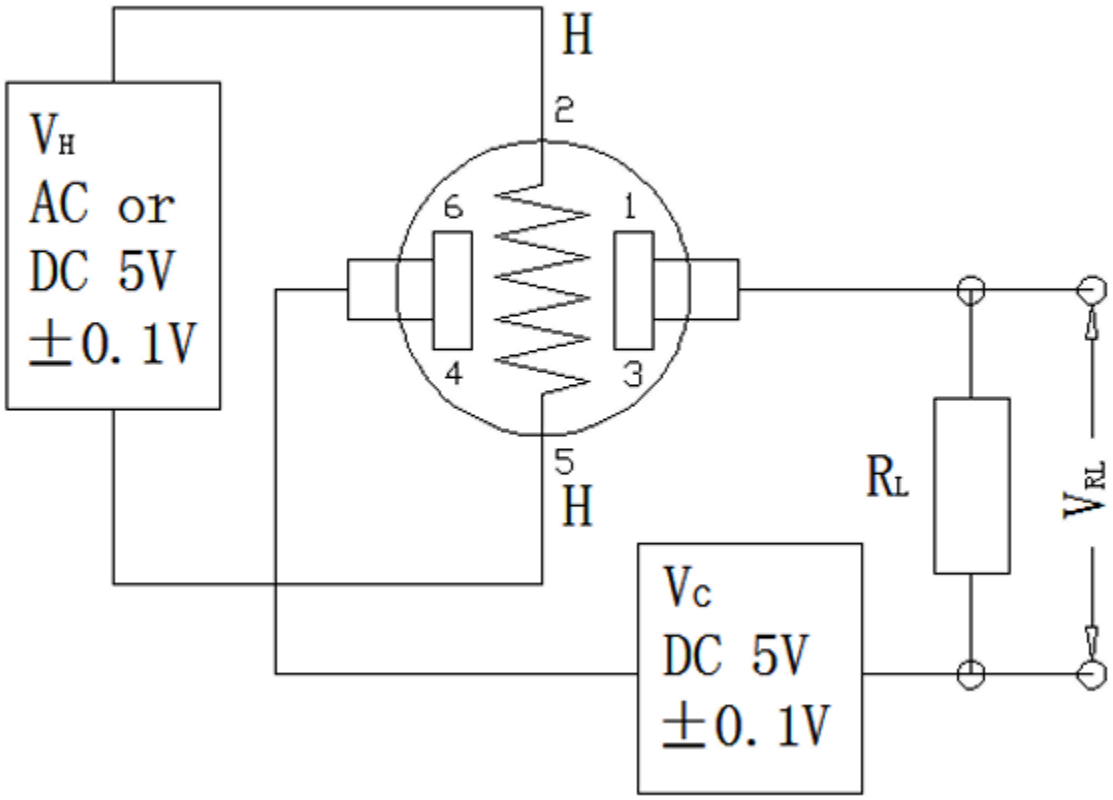
The goal of the sensor is to detect carbon dioxide, ammonia, methane, hydrogen sulfide, temperature, and humidity concentrations and send these measurements to the MCU. The ammonia, methane, and hydrogen sulfide sensors have analog outputs and the carbon dioxide and temperature/humidity sensors have digital outputs. The sensor subsystem was tested to confirm the validity of communications of sensor signals to the MCU.

3.2. Analog Sensors

3.2.1. Operation

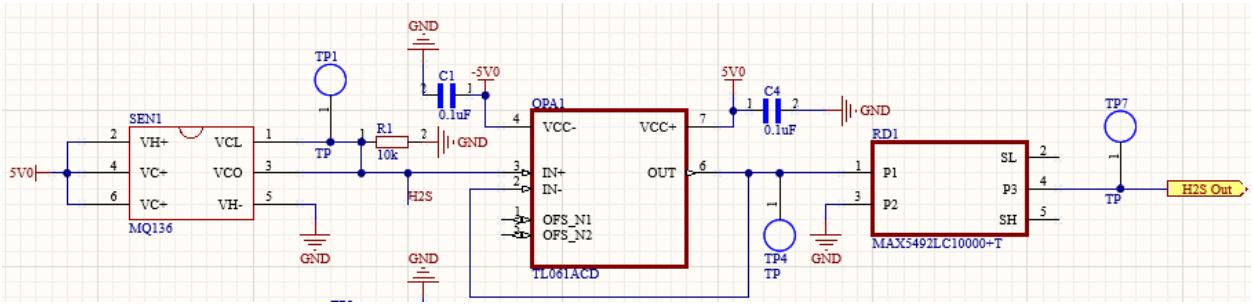
MQ series sensors were used for the ammonia, methane, and hydrogen sulfide sensors. These sensors have a heating resistor inside of the sensor and a load resistor, these two resistors act as a resistor divider.

Figure 10: Diagram of MQ Series Test Circuit



The maximum voltage output possible is 5V and the maximum input voltage for the analog to digital converter is 2.56V. Therefore, the output voltage needs to be divided in 2 to account for all possible values. A unity buffer op amp and voltage divider will be needed to divide the voltage. The op amp is needed as two voltage divider in series would not yield an accurate measurement for the sensor.

Figure 11: Op Amp and Voltage Divider Schematic



3.2.2. Validation

Test voltages were inputted into the op amp. Test voltages from .25V to 5V were tested to validate the complete range of possible sensor output voltages. The op amp input, op amp output, and voltage divider output were recorded.

Table 1: Op Amp and Voltage Divider Measurements

Op Amp Input (V)	Op Amp Output (V)	Voltage Divider Output (V)
0.2520	0.2517	0.1264
0.5011	0.5007	0.2515
1.0006	1.0004	0.5023
1.5001	1.5000	0.7530
2.0013	2.0009	1.0050
2.5010	2.5006	1.2560
3.0004	2.9999	1.5068
3.5001	3.4996	1.7577
3.9995	3.9991	2.0084
4.4991	4.2860	2.1524
4.9987	4.2858	2.1526

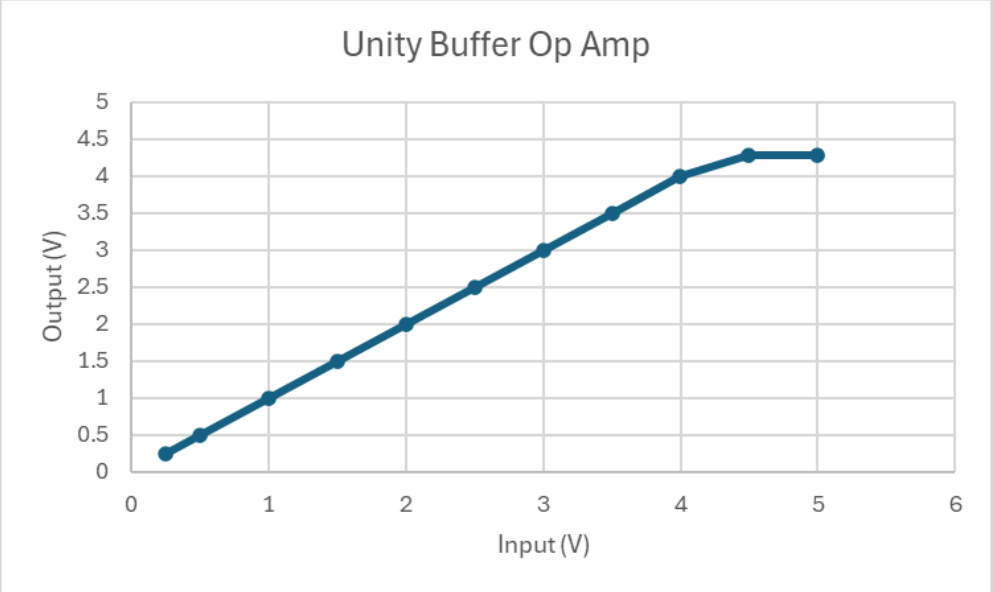
The unity buffer error is found by subtracting the output of the op amp from the input and dividing by the input. The voltage divider error is found by subtracting the expected from the measured value and dividing by the expected.

Table 2: Op Amp and Voltage Divider Error

Unity Buffer Error (%)	Voltage Divider Error (%)
.119	.435
.080	.457
.020	.418
.007	.398
.020	.453
.016	.454
.017	.455
.014	.449
.010	.441
4.972	.437
16.63	.541

The error for the unity buffer op amp is within reason for all values except for the last two inputs of 4.5V and 5V. The corresponding outputs plateau at around 4.28V. The error for the voltage divider is reasonable but can be made smaller by using higher precision resistors of 0.1%. They are less precise because lower precision resistors were used on a breadboard when troubleshooting the PCB.

Figure 12: Unity Buffer Op Amp Graph



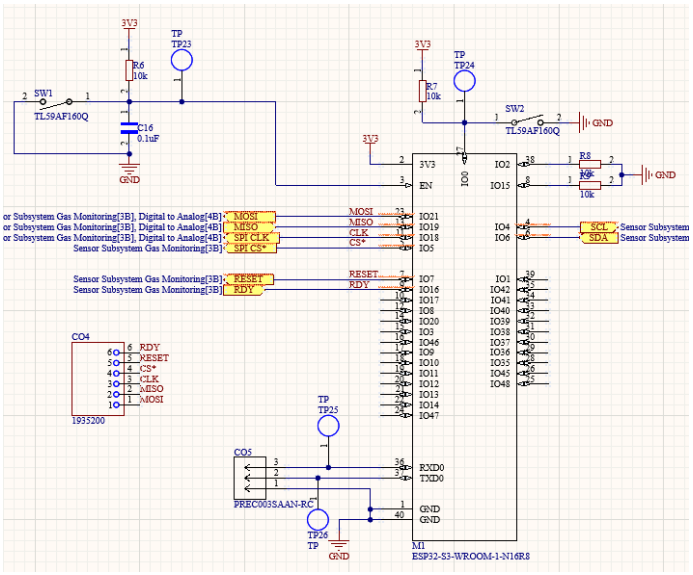
The graph above shows the plateau of the voltage at approximately 4.28V. This leads me to believe that an inadequate supply voltage was given to the op amp, leading to saturation. For the next iteration of the PCB, a supply voltage larger than 5V will be supplied.

3.3. Microcontroller Hardware

3.3.1. Operation

An ESP32-S3-WROOM-1-N16R8 is used as the MCU. A push button for the boot and enable pins, power and power bypass capacitors, and a USB header were implemented to give the ability to upload code.

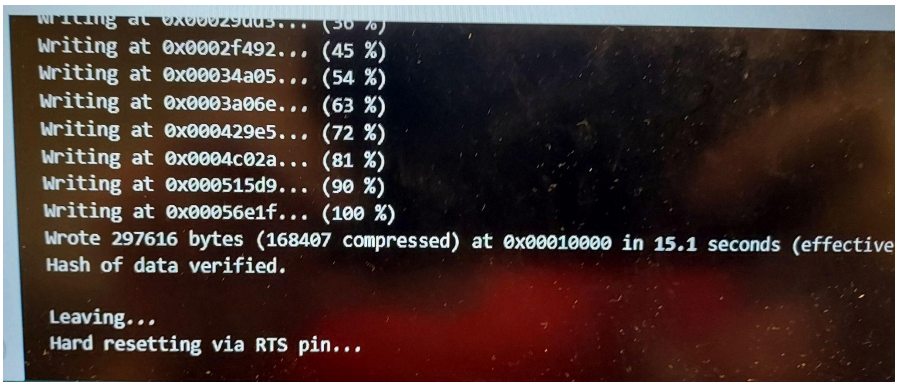
Figure 13: Microcontroller Hardware Schematic



3.3.2. Validation

Test code that printed “Hello World” was written with Arduino was used to validate that code can be uploaded to the microcontroller.

Figure 14: Code being Uploaded

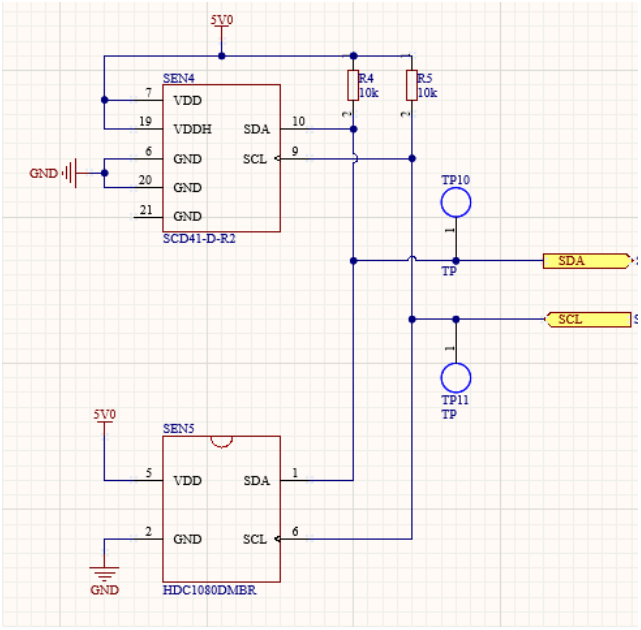


3.4. Digital Sensors

3.4.1. Operation

A SCD41-D-R2 was used for the CO2 sensor and a HDC1080DMBR was used for the temperature/humidity sensor. Both of these sensors communicate to the MCU with the I2C protocol. Commands can be sent to pull data from registers that contain the data needed. The SDA line is used for sending data and the SCL line is used as a clock to synchronize the data transfers.

Figure 15: Digital Sensors Schematic



3.4.2. Validation

I2C Scanner code was written to detect I2C addresses on the PCB. The code loops through all possible I2C addresses and tries to transmit to them. The I2C code attempts to pull the temperature and humidity bytes from the registers.

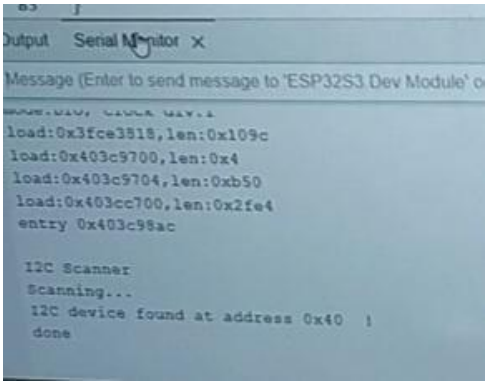
Figure 16: I2C Scanner Code and I2C Code

```
#include <Wire.h>


void setup()
{
  Wire.begin(6,4);

  Serial.begin(115200);
  while (!Serial);             // Leonardo: wait for serial monitor
  Serial.println("\nI2C Scanner");
}


void loop()
{
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");

      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknown error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);           // wait 5 seconds for next scan
}
```

```
#include "Wire.h"
byte temp1;
byte temp2;
float realtemp;

void setup() {
  Wire.begin(6,4);
  Serial.begin(115200);


}

void loop() {

   Wire.beginTransmission(0x40);
   Wire.write(0x00); //register starting at 0x00



  Wire.requestFrom( 0x40, byte(2));

   delay(30);
   temp1 = Wire.read();
   temp2 = Wire.read();
   Serial.println(temp1); //read one byte
   Serial.println(temp2); //read second byte
  // realtemp = ( (float)temp )*165/65536 - 40;
   //Serial.println(realtemp);

  Wire.endTransmission( );
  delay(1000);
}
```

The I2C scanner code detects the temperature/humidity sensor but not the CO2 sensor. The suspected reason for this is I forgot to connect the center ground pin or the chip was not soldered properly. The I2C code returned maxed out register values and is not working properly. The oscilloscope was used for further analysis.

Figure 17: I2C Scanner Results



When probing the SDA and SCL lines with the oscilloscope for the I2C scan code, a digital waveform at around 3.3V was detected. When probing with the I2C code running, a digital waveform at around 5V was detected. This voltage mismatch is the likely reason as to why the register data is not being read properly.

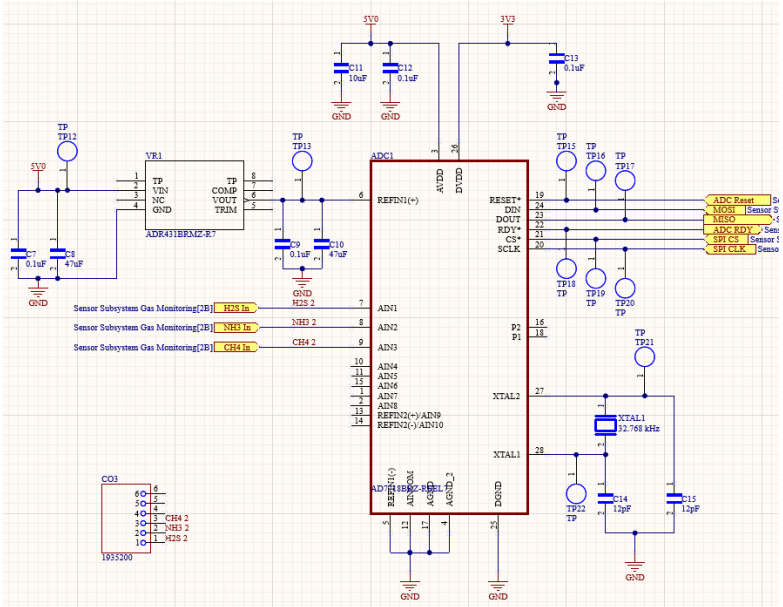
Figure 18: I2C Oscilloscope

3.5. Analog to Digital Converter

3.5.1. Operation

The outputs from the analog sensors go to the analog to digital converter. The converted value is then transferred to the MCU with the SPI interface. The MISO and MOSI lines communicate data back and forth. The SCLK line synchronizes communication with a digital clock. The CS line is a chip that selects what analog input to read. A voltage reference, digital and analog power, watch crystal, and power bypass capacitors are used.

Figure 19: Analog to Digital Converter Schematic



3.5.2. Validation

SPI code was written using the SPI library to convert from analog to digital and then pull the value from the register.

Figure 20: SPI Code

```
void setup()
{

  pinMode(SPI_MOSI, OUTPUT);
  pinMode(SPI_MISO, INPUT);
  pinMode(SPI_SCK, OUTPUT);

  SPI.begin(5);

  myADC.initializeADC();
  Serial.begin(115200);

}

void loop()
{

  Serial.println(myADC.isADCReady());
  myADC.startConversion(0x0);
  Serial.println(myADC.readADCValue(0x0));
  delay(1000);

}

AD7718::AD7718(int ssPin, int rdyPin) : m_ssPin(ssPin), m_rdyPin(rdyPin)

{

}
```

A value of 0 was returned. Nothing was found when probing the MOSI, MISO, and SCLK lines with the oscilloscope. It is unknown what the problem is. I will rewrite the code from scratch and get an ADC development board to test the code on.

3.6. Subsystem Conclusion

About half of the system is working as intended. Most problems have been identified and will be fixed for next semester. The unknown problems will be troubleshooted further over the break. An external development board will be used to determine if the hardware or software is the issue.

4. Microcontroller and Data Transmission Subsystem

4.1. Subsystem Introduction

5.The Microcontroller and Data Transmission Subsystem is responsible for collecting gas sensor data and transmitting it to a database for historical access. The subsystem uses two ESP32 development kits, one configured as a Server and the other as a Station/Client. The subsystem interfaces with gas sensors using I2C and SPI protocols, processes the sensor data, and transmits it over Wi-Fi using TCP/IP. This setup allows for reliable data transfer and storage for further analysis and monitoring.

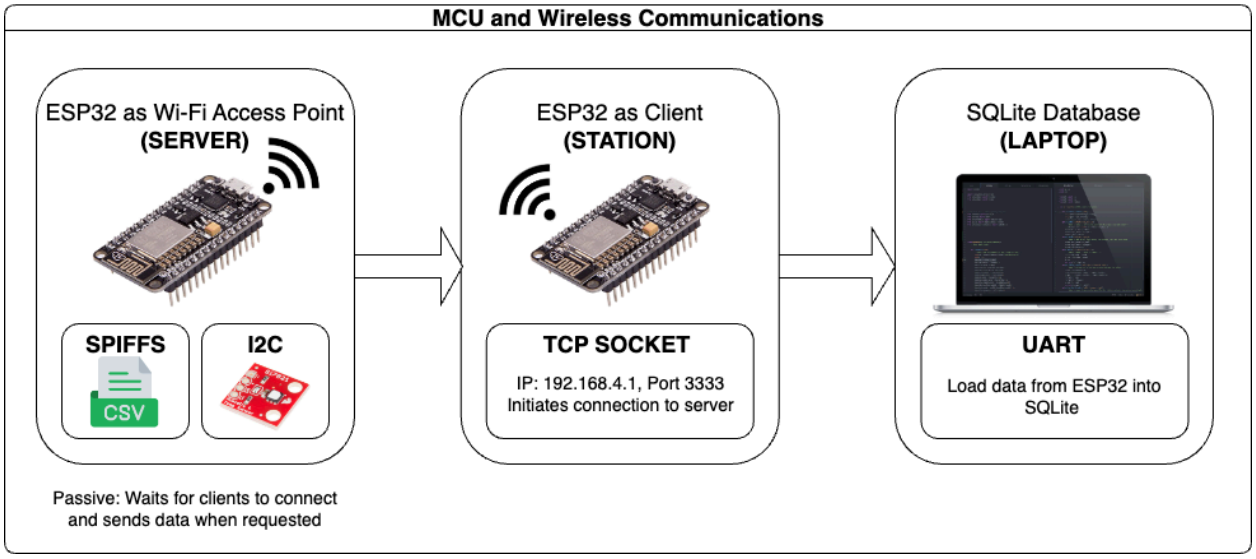


Figure 21. MCU and Wireless Communications workflow

5.1. Sensor Data Collection and Simulation

Sensor data collection and simulation refers to the real and simulated sensor readings that are interpreted by the microcontroller and prepared for transmission.

5.1.1. Si7021 Sensor

The Si7021 temperature and humidity sensor was selected for real-time monitoring of environmental conditions. The ESP32 interfaces with the Si7021 over I2C to collect temperature (in °C) and humidity (in %) values to be transmitted over Wi-Fi.

5.1.1.1. Wiring Diagram for the Si7021

The visual aids below show the connections between the Si7021 sensor and the ESP32. The SDA and SCL pins of the Si7021 are connected to D21 and D22 respectively on the microcontroller for communication over the I2C bus.

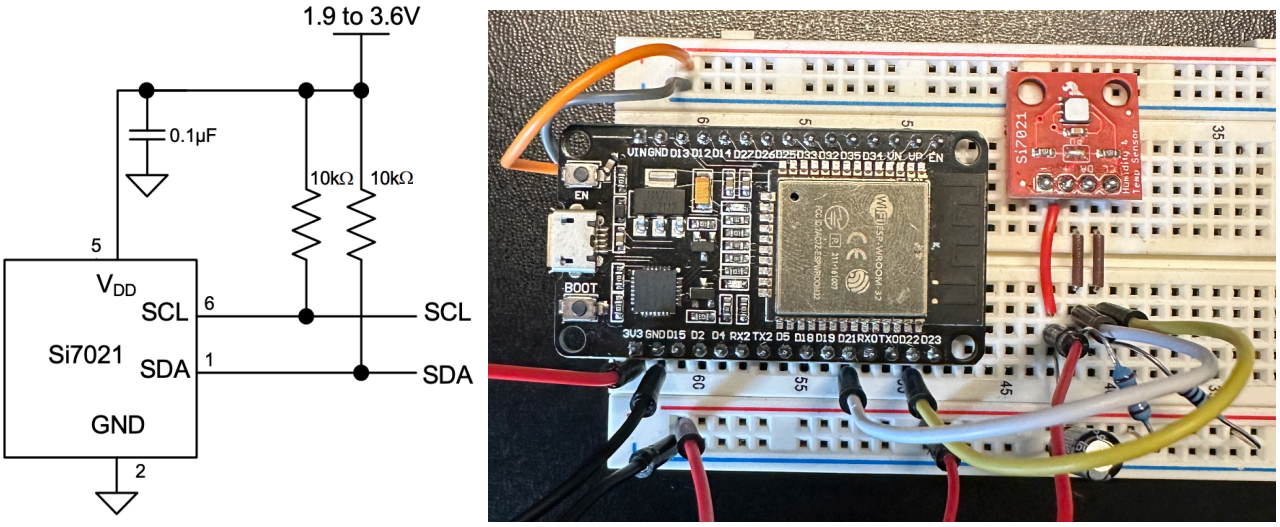


Figure 22. Si7021 Circuit and physical layout

5.1.1.2. Si7021 Software

The sensor is controlled using two files: Si7021.c and Si7021.h, which handle the sensor's data reading functions. Below is the code snippet in the main program which access temperature and humidity from the sensor every 5 seconds:

```
// Infinite loop to keep reading and printing sensor data
while (1) {
    // Read temperature and humidity using the library functions
    float temperature_c = si7021_read_temperature();
    temperature = (temperature_c * 9.0 / 5.0) + 32.0; // Converting from Celsius to Farenheit
    humidity = si7021_read_humidity(); // Read humidity

    // Log the temperature and humidity values to the terminal
    ESP_LOGI("SI7021", "Temperature: %.2f°F, Humidity: %.2f%%", temperature, humidity);

    // Wait for 5 seconds before the next reading
    vTaskDelay(5000 / portTICK_PERIOD_MS);  // Adjust the delay time as needed
}
```

Figure 23. Infinite loop to read and print Si7021 sensor data

5.1.2. Simulated Analog Sensor Data

For development and testing, simulated analog sensor data is generated to mimic the outputs of real sensors connected to the ESP32. This data is stored in a CSV file within the SPI Flash File System (SPIFFS). When the program prompts the Chip Select function, the sensor data is read and transmitted. This setup allows the system to process and transmit sensor data as though it were coming from real sensors.

5.1.2.1. Reading Dummy Sensor Data

```
    // Initialize SPI, chip select pin, and open CSV file
    sensor_data_file = fopen("/storage/sensor_data.csv", "r"); // Open CSV file in read mode
    if (sensor_data_file == NULL) { // Check if the file opening fails
        printf("Failed to open CSV file.\n"); // print it
    } else {
        char header[256]; // Array to hold header line
        fgets(header, sizeof(header), sensor_data_file); // Skip the header line that shows sensor type
    }
    //Print message indicating that the ADC setup is complete
    printf("ADC set up complete. Reading test data from CSV file.\n");
}

void read_sensor_data_csv() {
    char line[256]; // Array to hold each line read from CSV file
    if (fgets(line, sizeof(line), sensor_data_file) != NULL) { // Read one line from the CSV file

        // Read sensor data values from the CSV and store them in current_data structure
        sscanf(line, "%*[^,],%f,%f,%f,%f",
               &current_data.ammonia, &current_data.methane, &current_data.co2, &current_data.h2s);

        // Print the read sensor data to the console for debugging
        printf("Sensor data read from CSV: %.2f, %.2f, %.2f, %.2f\n",
               current_data.ammonia, current_data.methane, current_data.co2, current_data.h2s);
    } else { // If end of file reached
        printf("End of CSV file reached.\n");
        //rewind(sensor_data_file);  // Restart reading from the beginning if at the end
    }
}
```

Figure 24. Read sensor data function

The code snippet above opens the sensor_data.csv, ignores the header line that shows the sensor type, and proceeds to read the first line in the file and stores them in a string to be transmitted.

5.2. Wireless Communications

This subsystem is set up using two ESP32 Development kits. One acts as the Wi-Fi client and the other as the Wi-Fi server. This setup allows for efficient and reliable data communication between the two devices in a private Wi-Fi network.

5.2.1. Wi-Fi Server Setup

The first ESP32 creates a Wi-Fi server, enabling private communications and data reception. The ESP32 server is set up as an AP, allowing the client to connect directly to it. The code shown in Figure X below makes the ESP32 Server create a Wi-Fi network with a specified SSID and password which the customer may set to their preference, enabling communication between the Wi-Fi client and server without the need of an external access point.

```c
void wifi_init_softap(void) {

    ESP_ERROR_CHECK(esp_netif_init()); // Initialize esp32 network interface and create default event loop
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_ap();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT(); // Default Wi-Fi configuration
    ESP_ERROR_CHECK(esp_wifi_init(&cfg)); // Initialize Wi-Fi with the default config
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT, ESP_EVENT_ANY_ID,
                        &wifi_event_handler, NULL, NULL)); // Register Wi-Fi event handler

    wifi_config_t wifi_config = {

        .ap = {
            .ssid = EXAMPLE_ESP_WIFI_SSID, // Set SSID for AP
            .ssid_len = strlen(EXAMPLE_ESP_WIFI_SSID), // SSID length
            .channel = EXAMPLE_ESP_WIFI_CHANNEL, // Wi-Fi channel
            .password = EXAMPLE_ESP_WIFI_PASS, // Wi-Fi password
            .max_connection = EXAMPLE_MAX_STA_CONN, // Max number of client connections
            .authmode = WIFI_AUTH_WPA2_PSK, // Set WPA2 PSK authentication
            .pmf_cfg = {.required = true}, // Set Protected Management Frames
        },
    };
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP)); // Set device as an access point
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &wifi_config)); // Set AP configuration
```

Figure 25. Wi-Fi Soft Access Point Initialization

5.2.2. Wi-Fi Station Setup

The ESP32 client is responsible for connecting to the Wi-Fi access point (AP). This is achieved by inputting the SSID and password of the network created by the ESP32 server. Once connected, it establishes communication between the server to transmit sensor data.

```c
// Initialize Wi-Fi in STATION mode
void wifi_init_sta(void) {
    wifi_event_group = xEventGroupCreate(); // Create event gorup for Wi-Fi events
    esp_netif_init(); // Initialize the esp32 network interface
    esp_event_loop_create_default(); // Create default event loop
    esp_netif_create_default_wifi_sta(); // Create the Wi-Fi STA interface
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT(); // Set default Wi-Fi configurations
    esp_wifi_init(&cfg); // Initialize Wi-Fi

    wifi_config_t wifi_config = { // Configure Wi-Fi credentials with the ones defined
        .sta = {
            .ssid = WIFI_SSID,
            .password = WIFI_PASS,
        },
    };
    esp_wifi_set_mode(WIFI_MODE_STA); // Set Wi-Fi mode to station (STA)
    esp_wifi_set_config(WIFI_IF_STA, &wifi_config); // Set Wi-Fi configuration
    esp_wifi_start(); // Start the Wi-Fi
}
```

Figure 26. Wi-Fi Station Initialization

The microcontroller scans for the network defined in WIFI_SSID and connects automatically. This ensures that when connected, the device remains on the network for continuous data transmission. The ESP32 client has a feature for automatic reconnection in the case that Wi-Fi connection is lost. The following code snippet handles the reconnection logic within the event handler:

```
// Wi-Fi event handler for different Wi-Fi events
static void wifi_event_handler(void* arg, esp_event_base_t event_base, int32_t event_id, void* event_data) {
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
        esp_wifi_connect(); // Connect to the Wi-Fi network
    } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) {
        esp_wifi_connect(); // Reconnect if disconnected
        ESP_LOGI(TAG, "Reconnecting to the AP...");
```

Figure 27. Event handler for Wi-Fi Reconnection if disconnected

5.2.3. TCP Communications

5.2.3.1. Socket Communication

TCP (Transmission Control Protocol) ensures reliable data transmission between the server and client over an IP network. The client creates a socket connection to the server using port 3333, where the server listens to incoming connections. The sensor data is formatted into packets when read from the sensors and then the client receives the packets. The code below creates a socket in the ESP32 Client by listening at the predefined server IP address which is provided by the ESP32 Server after Wi-Fi initialization, in this case SERVER_IP is 192.168.4.1.

```
while (1) { // While esp32 is initializing
    // Create socket
    int sock = socket(addr_family, SOCK_STREAM, ip_protocol);
    if (sock < 0) { // Have a check for socket creation error
        ESP_LOGE(TAG, "Unable to create socket: errno %d", errno); // Log error if socket creation fails
        vTaskDelay(1000 / portTICK_PERIOD_MS); // Wait a second before retrying
        continue;
    }
    ESP_LOGI(TAG, "Socket created, connecting to %s:%d", SERVER_IP, PORT); // If successful log the message

    // Connect to server
    int err = connect(sock, (struct sockaddr *)&dest_addr, sizeof(dest_addr));
    if (err != 0) { // Check for connection error
        ESP_LOGE(TAG, "Socket unable to connect: errno %d", errno); // Log error if connection fails
        close(sock); // Close the socket
        vTaskDelay(1000 / portTICK_PERIOD_MS); // Wait a second before retrying
        continue;
    }
    ESP_LOGI(TAG, "Successfully connected");  // If successful log the message
```

Figure 28. Creating Socket

5.2.3.2. Error Handling

During testing, an issue arose where packet transmissions were incomplete, causing the next packet to compensate by including data from the previous one. To account for this, the Python script that reads data from the ESP client through UART was modified to use a buffer that stores incomplete data until the full packet is received. If the data does not match the expected format, the buffer is cleared, and the failure count is incremented. This ensures that only valid and complete data is processed and stored in the database.

```
# define a string that mimics the sensor output format.
data_pattern = re.compile(r"Received data: Temp:(\d+\.\d+),Humidity:(\d+\.\d+),NH3:(\d+\.\d+),H2S:(\d+\.\d+),CO2:(\d+\.\d+),CH4:(\d+\.\d+)")
```

```
# Append new line data to the buffer
buffer += line

# Check if the buffer contains a full line of sensor data
match = data_pattern.search(buffer)
if match:
    temperature = float(match.group(1))
    humidity = float(match.group(2))
    nh3 = float(match.group(3))
    h2s = float(match.group(4))
    co2 = float(match.group(5))
    ch4 = float(match.group(6))

    # Insert data into the database
    cursor.execute('''
        INSERT INTO sensor_data (temperature, humidity, nh3, h2s, co2, ch4)
        VALUES (?, ?, ?, ?, ?, ?)
    ''', (temperature, humidity, nh3, h2s, co2, ch4))
    conn.commit()

    print(f"Stored data - Temp: {temperature}, Humidity: {humidity}, NH3: {nh3}, H2S: {h2s}, CO2: {co2}, CH4: {ch4}")

    # Clear the buffer since data has been processed
    buffer = ""
else:
    # If the buffer does not contain a full match, check for malformed data
    # If a transmission is failed, increment the counter
    if len(buffer) > 100: # threshold to detect invalid data length
        failed_transmission += 1
        print(f"Failed Transmissions: {failed_transmission}")
        buffer = "" # Clear the buffer and wait for the next valid data
```

Figure 29. Error handling for writing to database

5.3. SQLite Database

This Microcontroller and Data Transmission subsystems final demonstration is to accurately store sensor data into a database. This subsystem uses a simple python script to read data packets from the ESP32 client through UART and store them into a database with timestamps for each packet.

5.3.1. Packet Format

Readings from the different sensors are combined and packetized before being transmitted by server and received by client. The format is a string which has the title of the value, followed by a colon and the value itself. These readings are all comma separated:

Received data: Temp:74.99,Humidity:57.84,NH3:175.48,H2S:5.57,CO2:1327.13,CH4:183.17

5.3.2. Database

Each entry in the database covers all readings from the ESP32 Server (transmitter) with its respective timestamp. The order of the columns is as follows:

timestamp	temperature	humidity	NH3	H2S	CO2	CH4

5.4. Subsystem Validation

5.4.1. Stable Wi-Fi and TCP Socket Connection

Below is validation of the ESP32 Server creating its own soft access point with SSID: ESP32-Access-Point and password: joaquinsalas. It also outputs its own IP address which is shown to be 192.168.4.1. The Server then creates a socket defined as port 3333.

```
I (4649) TCP_SOCKET_SERVER: wifi_init_softap finished. SSID:ESP32-Access-Point password:joaquinsalas channel:1
I (4659) esp_netif_lwip: DHCP server started on interface WIFI_AP_DEF with IP: 192.168.4.1
I (4749) ADC: Partition size: total: 956561, used: 103663
ADC set up complete. Reading test data from CSV file.
I (4749) TCP_SOCKET_SERVER: Socket created
I (4749) TCP_SOCKET_SERVER: Socket bound, port 3333
I (4759) TCP_SOCKET_SERVER: Socket listening
```

Figure 30. ESP32 Server initialization

Here, the ESP32 Client is connecting to the access point by finding the network SSID and matching password for security. It then successfully connects with the IP Address and port initialized by the ESP32 Server. Below the interface shows that a socket has been created and connected to 192.168.4.1.3333 and proceeds to begin receiving the data packets.

```
I (77544) wifi:connected with ESP32-Access-Point, aid = 1, channel 1, 40U, bssid = c8:c9:a3:c7:a8:cd
I (77544) wifi:security: WPA2-PSK, phy: bgn, rssi: -43
I (77554) wifi:pm start, type: 1

I (77554) wifi:dp: 1, bi: 102400, li: 3, scale listen interval from 307200 us to 307200 us
I (77634) wifi:dp: 2, bi: 102400, li: 4, scale listen interval from 307200 us to 409600 us
I (77634) wifi:AP's beacon interval = 102400 us, DTIM period = 2
I (78564) esp_netif_handlers: sta ip: 192.168.4.2, mask: 255.255.255.0, gw: 192.168.4.1
I (78564) TCP_SOCKET_CLIENT: Connected with IP Address:192.168.4.2
I (78564) TCP_SOCKET_CLIENT: Socket created, connecting to 192.168.4.1:3333
I (78574) main_task: Returned from app_main()
I (78584) wifi:<ba-add>idx:0 (ifx:0, c8:c9:a3:c7:a8:cd), tid:0, ssn:0, winSize:64
I (78594) TCP_SOCKET_CLIENT: Successfully connected
I (80594) TCP_SOCKET_CLIENT: Received data: Temp:73.71,Humidity:59.03,NH3:195.24,H2S:4.54,CO2:1828.71,CH4:412.46
```

Figure 31. ESP32 Client Initialization

5.4.2. Reading Actual and Simulated Sensors

The terminal snippet below which is monitoring the ESP32 Server shows successful acquisition of temperature and humidity values from the Si7021 and CSV file containing dummy values. This validates that sensor data is read and is formatted for transmission to the database.

Figure 32. Reading values from sensor and csv

```
ADC set up complete. Reading test data from CSV file.
I (4749) TCP_SOCKET_SERVER: Socket created
I (4749) TCP_SOCKET_SERVER: Socket bound, port 3333
I (4759) TCP_SOCKET_SERVER: Socket listening
Sensor data read from CSV: 195.24, 412.46, 1828.71, 4.54
I (4769) main_task: Returned from app_main()
I (4859) SI7021: Temperature: 75.22°F, Humidity: 56.60%
I (9959) SI7021: Temperature: 75.26°F, Humidity: 57.43%
Sensor data read from CSV: 9.14, 301.98, 1868.06, 11.38
I (15059) SI7021: Temperature: 75.28°F, Humidity: 57.70%
Sensor data read from CSV: 93.88, 389.83, 2743.87, 1.25
I (20159) SI7021: Temperature: 75.34°F, Humidity: 57.88%
```

5.4.3. Data Transmission Range

The gas monitor needs to transmit data from animal pins to a local database, so the system must be able to transmit data over a reasonable distance. The target transmission range for the microcontroller is set to 25 meters. To validate this, the ESP32 Server prototype was elevated on a lawn chair, and the ESP32 Client was placed across a parking lot. The results showed that the system can reliably send data packets up to 60 meters, after which the network begins to disconnect and reconnect. The optimal data transmission range is 42 meters. Obstructions were found to shorten the range including vehicles and long grass if placed on the ground.

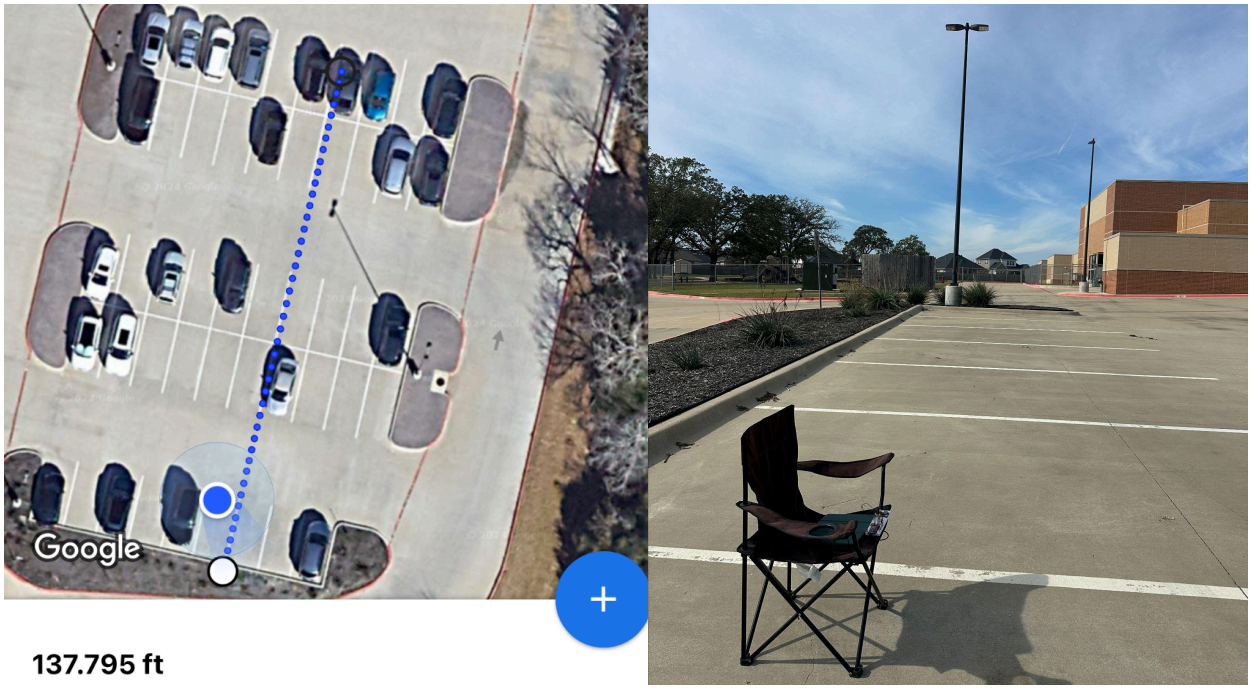


Figure 33. Data Transmission Range Validation

5.4.4. Database

The final validation for this subsystem is to ensure that data can be accurately transmitted from the ESP32 Server and stored in a database with its corresponding timestamp. Below is a snippet of the database on the left, along with the dummy sensor csv file on the right, which demonstrates that the data reaches the database with minimal error. The values for the four gases in the database match those generated by the CSV file, confirming successful transmission and storage of sensor data.

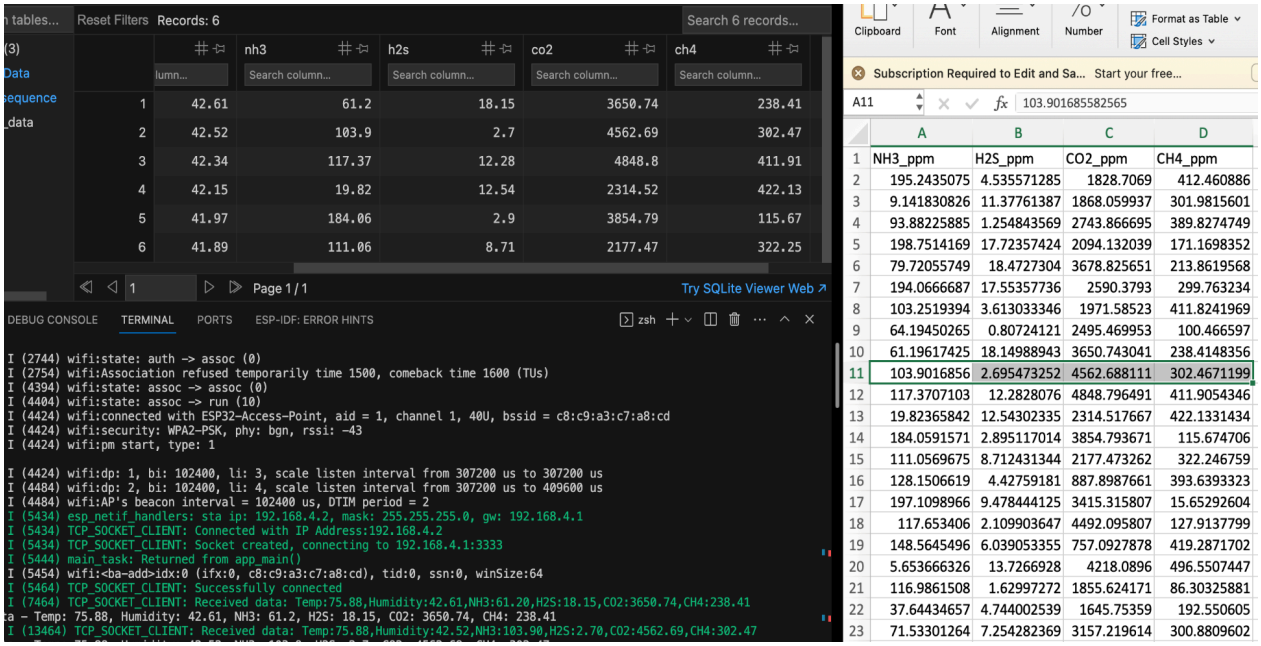


Figure 34. Comparison of Database and CSV file

5.5. Subsystem Conclusion

The Microcontroller and Data Transmission subsystem successfully demonstrated data collection from the hardware Si7021 using I2C communications, and read data from simulated analog sensor readings. This data was packetized in the ESP32 server and transmitted to the ESP32 client every 5 seconds. It is also able to effectively upload test data into the SQLite database. Moving forward, the code will be updated to interface with hardware gas sensors and seamlessly integrate with the Database subsystem to reliably monitor common gasses found in animal farms.

6. Database + Web Application Subsystem Report

6.1. Backend - MySQL/AWS Database

The software that handles all the data will be Amazon Web Services. This was chosen because of its relative cost-effectiveness for a project such as this, where there are only a couple tables to be handled. The tables that will be stored in the database are the gas data and the table with user information (username, password). The data will be stored through MySQL. MySQL workbench was used for ease of usability. It essentially creates a user interface that can be used to program MySQL to work with the AWS server.

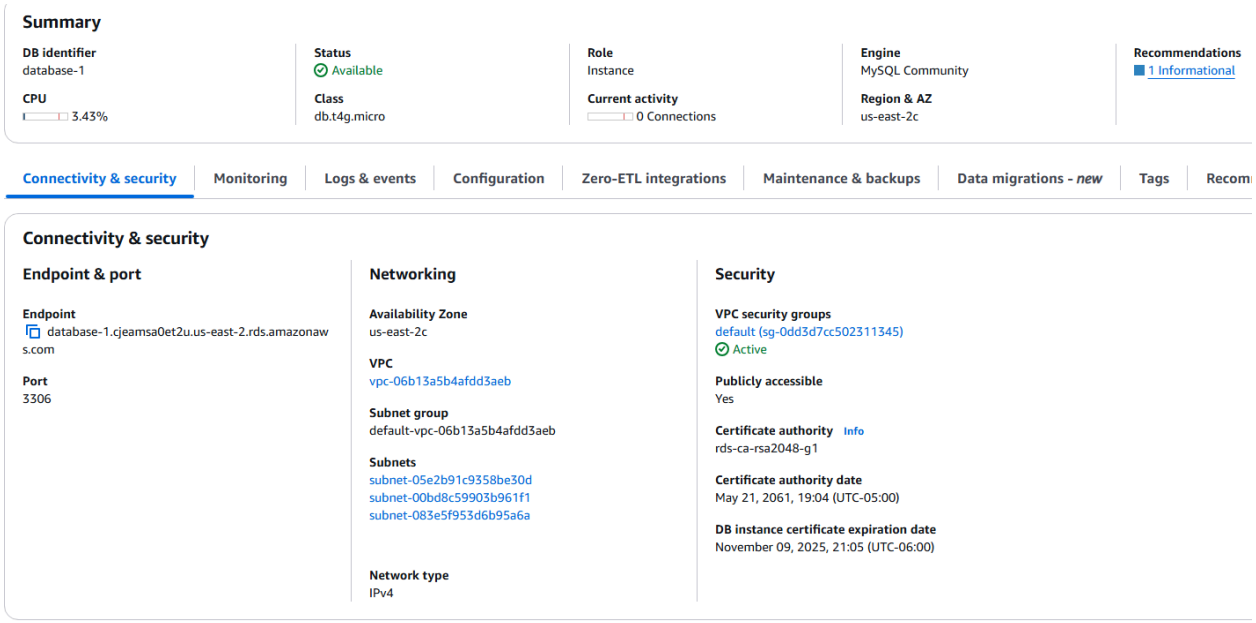


Figure 35 Screenshot of AWS Database main screen

6.2. Backend - Database Security

The database is secured through a couple measures. The first is through an AWS function in which only certain IP addresses are allowed to communicate with the database. The below screenshot shows AWS security group feature. Each inbound rule is one IP address that is allowed to connect to the database currently.

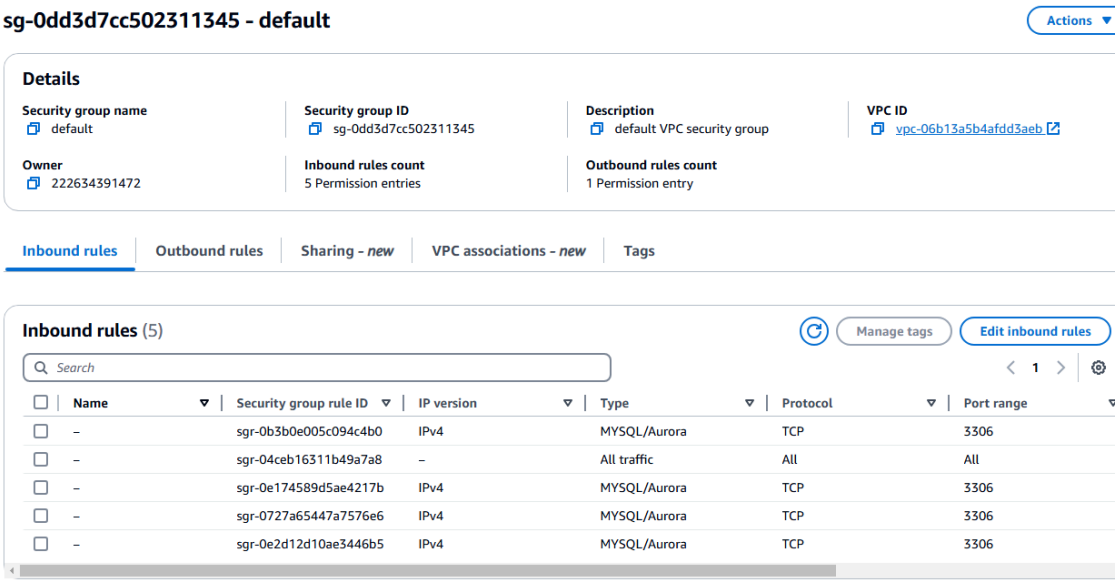


Figure 36. Screenshot of security inbound rules

6.3. Backend - Data Manipulation - Adding, Subtracting from data tables

The way testing the database was handled throughout the semester was primarily by using randomly generated, or manually inserted test data points. Throughout the next semester, this will be integrated with the microcontroller and rest of the system to use real datapoints. There was additional functionality to manually enter and remove all data points from the database, which shows the ability to read and write from it.

```python
def insert_gas_data():
    try:
        # Connect to the MySQL database
        connection = mysql.connector.connect(**db_config)
        cursor = connection.cursor()

        # Generate a single datapoint for each gas
        ammonia = round(random.uniform(0, 25), 4)  # Ammonia in ppm, safe range typically 0-25 ppm
        carbon_dioxide = round(random.uniform(350, 1000), 4)  # Carbon Dioxide in ppm, safe range 350-1000 ppm
        methane = round(random.uniform(0, 10), 4)  # Methane in ppm, safe range typically 0-10 ppm
        nitrous_oxide = round(random.uniform(0, 1), 4)  # Nitrous Oxide in ppm, safe range typically 0-1 ppm
        hydrogen_sulfide = round(random.uniform(0, 5), 4)  # Hydrogen Sulfide in ppm, safe range typically 0-5 ppm
        timestamp = datetime.now(tz).strftime('%Y-%m-%d %H:%M:%S')  # Local timestamp with timezone

        # Insert the data into the gas_data table
        insert_query = '''
        INSERT INTO gas_data (ammonia, carbon_dioxide, methane, nitrous_oxide, hydrogen_sulfide, timestamp)
        VALUES (%s, %s, %s, %s, %s, %s)
        '''
        cursor.execute(insert_query, (ammonia, carbon_dioxide, methane, nitrous_oxide, hydrogen_sulfide, timestamp))

        # Commit the transaction
        connection.commit()
        print("Inserted 1 new gas data point.")

    #Error checking after the try block
    except Error as e:
        print(f"Error: {e}")
```

Figure 37. Snippet of code showing ability to add test data points

```
PS C:\Users\tanma\OneDrive\Desktop\WebApp\server> cd C:\Users\tanma\OneDrive\Desktop\WebApp\data_generator
PS C:\Users\tanma\OneDrive\Desktop\WebApp\data_generator> python datagen.py
Inserted 1 new gas data point.
```

Figure 38. Snippet of inserting new gas point manually

6.4. Backend - Handling of user information

```python
# Route: User login
@app.route('/login', methods=['POST'])
def login():
    data = request.json  # Get JSON data from the frontend request
    username = data.get('username')
    password = data.get('password')

    # Validate input fields
    if not username or not password:
        return jsonify({"error": "Username and password are required."}), 400

    try:
        # Connect to the database
        connection = mysql.connector.connect(**db_config)
        cursor = connection.cursor()

        # Query the database for the user's hashed password
        query = "SELECT password FROM users WHERE username = %s"
        cursor.execute(query, (username,))
        result = cursor.fetchone()  # Fetch one result; we only expect one user entry

        if result:
            # Extract the hashed password from the query result
            stored_password = result[0]
            # Verify if the provided password matches the hashed password
            if check_password_hash(stored_password, password):
                return jsonify({"message": "Login successful!"}), 200
            else:
                return jsonify({"error": "Invalid password."}), 401
        else:
            return jsonify({"error": "Username does not exist."}), 404

    except Error as e:
        # Handle any database error
        return jsonify({"error": str(e)}), 500
```

Figure 39. Snippet of code showing user handling

6.5. Backend - Handling of test data table

Below is a snippet of code that shows the following functionality in regards to the test data table: Creation of the table, showing average, min and max values from the last 30 minutes, and displaying the concentration of each gas, including a timestamp for when the data was measured.

```
# Route: Get gas data for monitoring
@app.route('/get_gas_data', methods=['GET'])
def get_gas_data():
    try:
        # Connect to the database
        connection = mysql.connector.connect(**db_config)
        cursor = connection.cursor()

        # Delete records older than 30 minutes to maintain recent data
        thirty_minutes_ago = datetime.now(tz) - timedelta(minutes=30)
        formatted_time = thirty_minutes_ago.strftime('%Y-%m-%d %H:%M:%S')
        delete_query = 'DELETE FROM gas_data WHERE timestamp < %s'
        cursor.execute(delete_query, (formatted_time,))
        connection.commit()  # Commit changes to delete old records

        # Fetch all rows from the gas_data table, ordered by timestamp (latest first)
        cursor.execute('SELECT * FROM gas_data ORDER BY timestamp DESC')
        rows = cursor.fetchall()  # Fetch all records from the query

        # Calculate statistics (average, min, max) for each gas type in the last 30 minutes
        cursor.execute('''
            SELECT AVG(ammonia), MIN(ammonia), MAX(ammonia),
                   AVG(carbon_dioxide), MIN(carbon_dioxide), MAX(carbon_dioxide),
                   AVG(methane), MIN(methane), MAX(methane),
                   AVG(nitrous_oxide), MIN(nitrous_oxide), MAX(nitrous_oxide),
                   AVG(hydrogen_sulfide), MIN(hydrogen_sulfide), MAX(hydrogen_sulfide)
            FROM gas_data
            WHERE timestamp >= %s
        ''', (formatted_time,))
        stats = cursor.fetchone()  # Fetch the calculated statistics for each gas type

        # Convert database rows to a list of dictionaries for JSON response
        data = [{
            'id': row[0],  # Record ID
            'ammonia': round(row[1], 4),  # Ammonia concentration rounded to 4 decimal places
            'carbon_dioxide': round(row[2], 4),  # Carbon Dioxide concentration
            'methane': round(row[3], 4),  # Methane concentration
            'nitrous_oxide': round(row[4], 4),  # Nitrous Oxide concentration
            'hydrogen_sulfide': round(row[5], 4),  # Hydrogen Sulfide concentration
            'timestamp': row[6].astimezone(tz).strftime('%Y-%m-%d %H:%M:%S')  # Convert timestamp to specified timezone
        } for row in rows]
```

Figure 40. Snippet of code showing handling of gas data.

6.6. Frontend - Landing page

The first screen the user will see when trying to access the gas data will be a login page. Here, they can either log in or create an account. Later, when the web application is deployed onto a domain, it will be able to use email verification so that only authenticated users can access the data. The login screen is shown below.

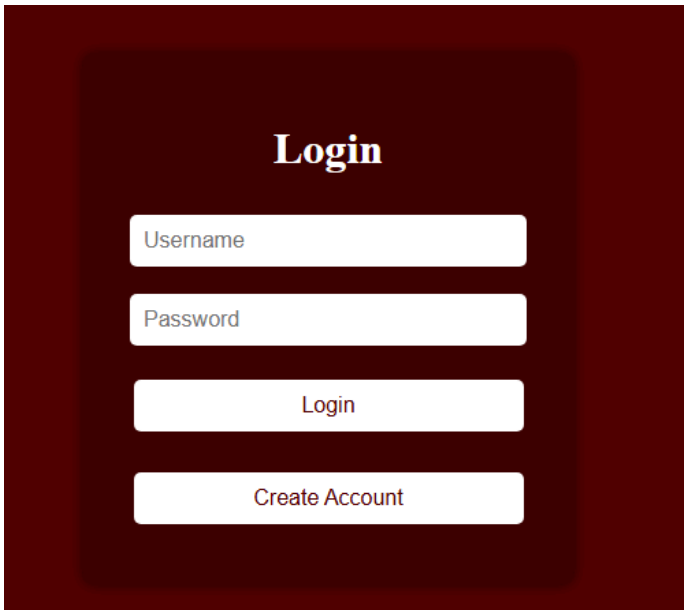


Figure 41. Login Screen

6.7. Frontend - Post-Login screen

After logging in, the user is then taken to a different UI, where there are several tabs for navigation. This UI could be improved in terms of graphics, but all the functionality was completed. Screenshots for each tab in which the user can navigate are shown in the following sections.

6.7.1. Welcome Tab

Figure 42. Welcome Tab

6.7.2. Live Data Tab

Figure 43. Live Data Tab

6.7.3. Contact Information Tab

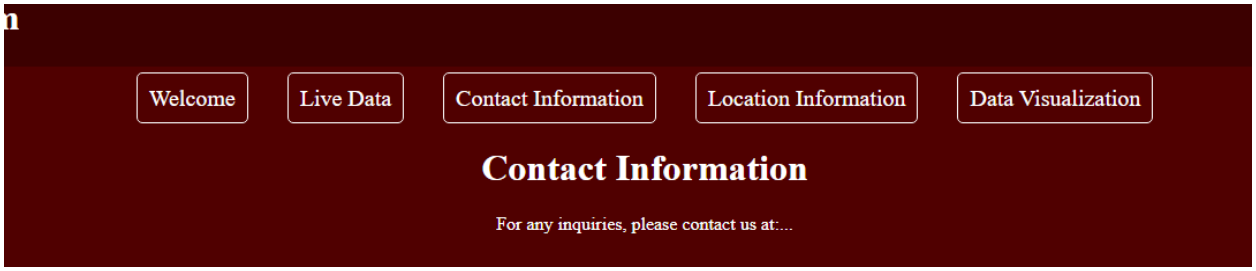


Figure 44. Contact Info

6.7.4. Location Information Tab

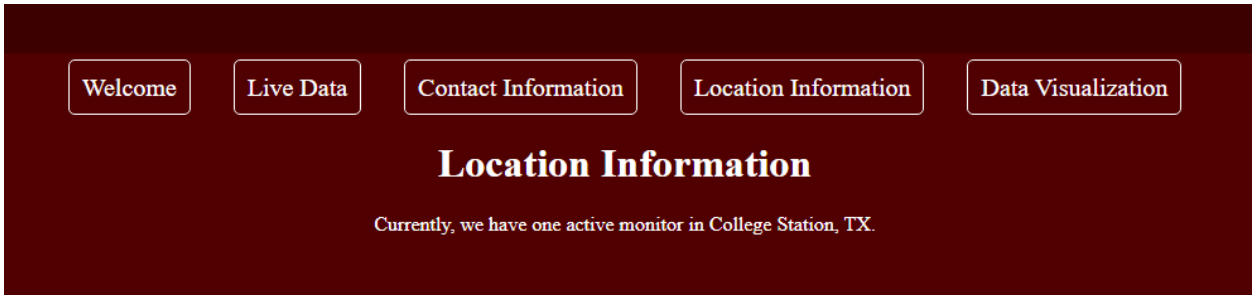


Figure 45. Location Info

6.7.5. Data Visualization Tab

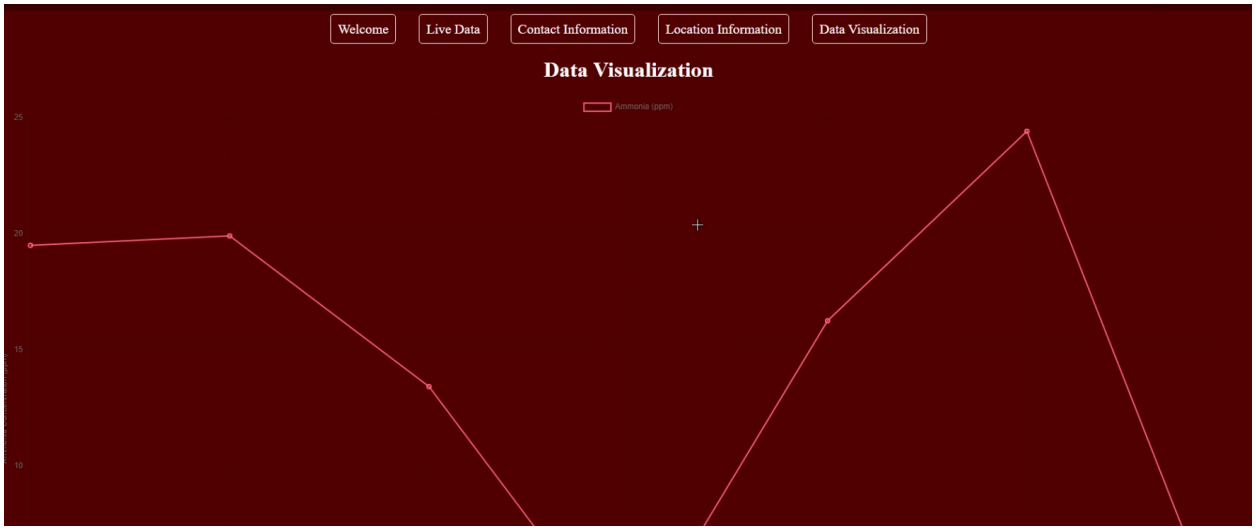


Figure 46. Data Visualization Tab

6.8. Frontend - Ability to export to CSV

Once the Export to CSV button is pressed as seen in the main Live Data tab from section 5.7.2, the user will be able to download the current gas table in excel sheet format.

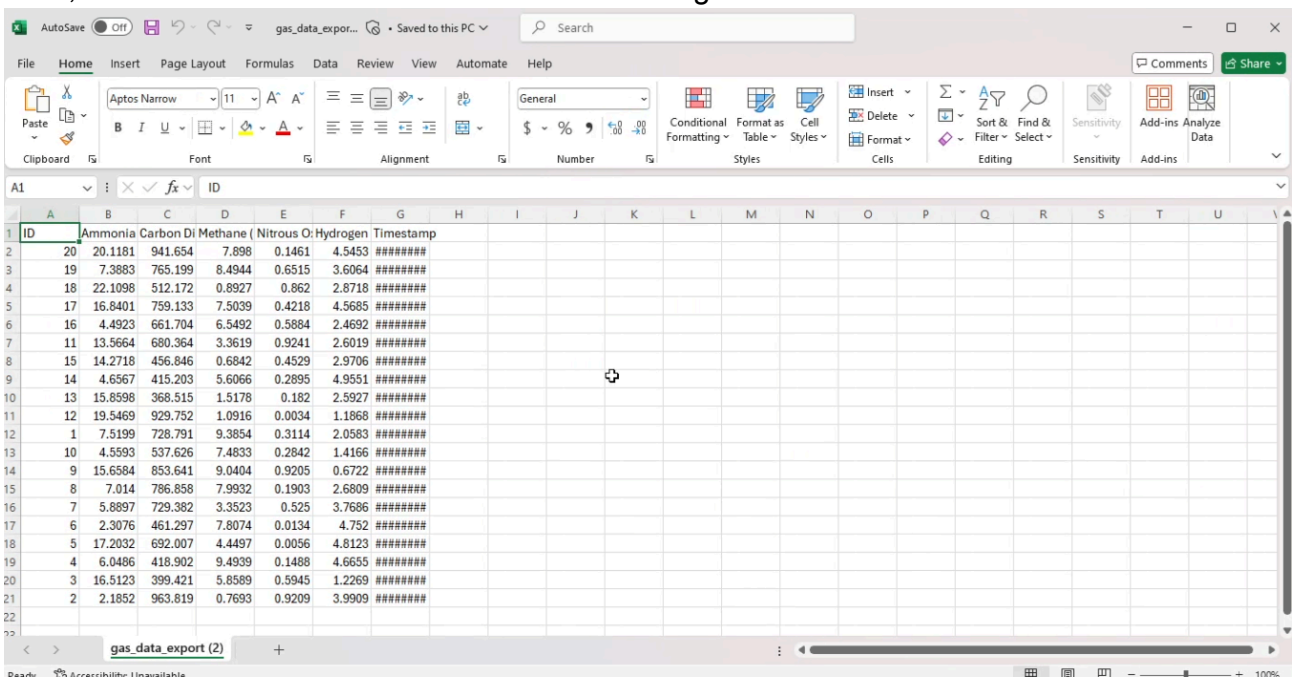


Figure 47. Exporting to CSV

6.9. Readiness for Integration + Current Limitations

A majority of the functionality for the web application is completed, with the ability to integrate with the microcontroller so that the live data can be sent in. There will obviously need to be script updates made so that the database and backend server can communicate with the microcontroller, instead of just working with randomly generated test data, which is the way it works now. There are no current concerns with the actual ability to integrate with the microcontroller itself.

In terms of limitations with the database, web application subsystem on its own, there are a couple that can be fixed. First, it is unknown how much the final database in AWS will cost monthly. Once there is lots of data to be handled, it is possible that the cost increases by a lot (Currently only $3 per month). There is also an option to do a year long free trial, which is a main reason why AWS was chosen as the database host, along with its ease of use and cost effectiveness. In addition to this, the front end security is currently not fully implemented. Once the web application is deployed (it is currently privately hosted), email verification will be used. Currently, anyone can just create an account. This will be fixed by the time the system is fully implemented. Other smaller limitations include the graphics of the website when the data updates, and the functionality of the export to CSV button, which can sometimes take multiple clicks.

System Description and Development

Table of Contents

List of Figures

1. Overview

In this final section, we reflect on how our Gas Monitoring System has evolved since the ConOps and FSR. We will detail specific changes to the project that have been made since our initial designs, outline the integration process, and prove how we validated and demonstrated our portions of the project. Lastly, we will provide some concluding thoughts and feedback we received from our sponsor.

2. Development Plan and Execution

2.1. Subsystem Changes

2.1.1. Sensor Subsystem

There were very little changes to the sensor subsystem. One change made was rerouting the analog sensor 1 signal to analog input 4 of the ADC. Another change was powering the digital sensors with 3.3V rather than 5V. Majority of the changes were just removing test points and making the board as compact as possible.

2.1.2. Microcontroller + Data Transmission Subsystem Changes

The most notable change to the subsystem is moving away from two MCUs that communicate with each other, requiring a hard wire for UART, to sending sensor data from the MCU on the gas monitor directly to AWS IoT. A more subtle change that became necessary while testing is Wi-Fi provisioning, which allows the user to connect to any Wi-Fi network.

2.1.3. Database + Web Application Subsystem Changes

There were a few changes to the database and web application subsystem since the initial plan in 403. These primarily changes were made to the security of the database. Originally, the web application was supposed to have a landing page that required the user to create a username and password, and verify through email 2-factor authentication. However, due to software errors and difficulty of debugging during the time that the web application was locally hosted, this proved to be a difficult task. The database security is now handled by AWS, in which if the user wants to change the database + web app infrastructure, they must create an account through AWS and join our project as a root user (must be approved by Joaquin). The only other changes since 403 were subtle changes to the front end, including the design of the data visualization tab.

2.2. Integration

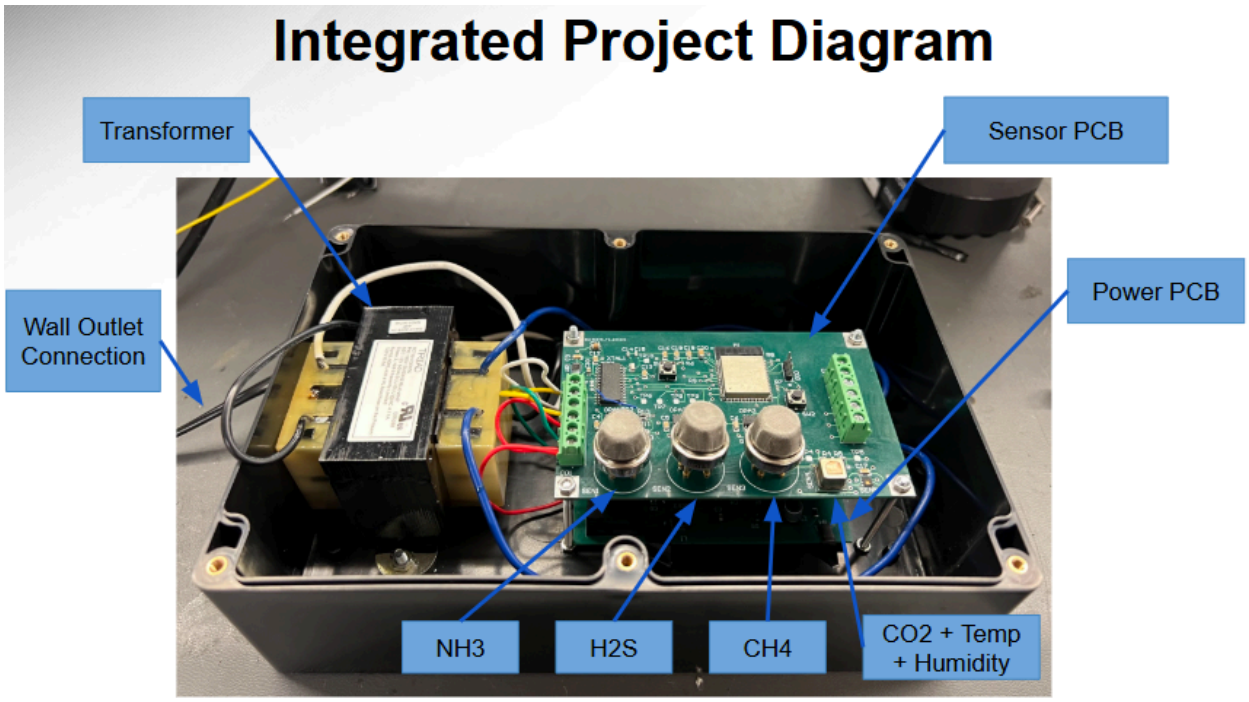


Figure 1. Integrated project diagram

2.2.1. Power - Sensor Integration

The power subsystem was wired to send 5, 3.3, and -8.5 V into the sensor PCB. The 5V powered the analog sensors and ADC. The 3.3V powered the digital sensors, microcontroller, and ADC. The -8.5V was the negative saturation voltage for the op amp.

2.2.2. Sensor - MCU - Database Integration

Code was written and uploaded to the microcontroller to interface with I2C based sensors, capturing temperature, humidity, and CO2 data. Additionally, firmware was created to interact with the AD7718 ADC via SPI to access analog voltage readings. The voltages from the ADC were calibrated and converted to gas concentrations (PPM). Wi-Fi provisioning and error-handling routines were implemented, managing reconnection in case of weak Wi-Fi signal or MQTT broker disconnections. Data is securely transmitted to AWS IoT Core using authenticated MQTT communication, using security credentials (public/private keys and root certificates), and formatted in JSON to easily load sensor readings into the cloud-based database.

2.2.3. MCU - Database - Web App Integration

Initially, the ESP32 sends data to AWS' IoT Core feature. The MCU passes the data in through MQTT messages in a JSON format, which is handled by AWS IoT. Then, AWS Lambda is used to connect these messages to the MySQL database using a python script. This script contains the IP and password data of the database, along with the format in which the database is expecting the data to be inputted. Now that the database has the data, a Flask-based web application is used to get the data from the database in real time. Flask serves up API endpoints which is used by the web application regularly. The application handles the data by dynamically updating them on the table shown on the "Live Data" tab, and graphing them on the "Data Visualization" tab.

```
I (359770) gasmonitor: Publishing: {"temperature":81.53, "humidity":48.59,
 "co2":636.00, "NH3":4.431723, "H2S":0.636742, "CH4":1.900668}
I (364980) AD7718: AIN1 Voltage: 0.515357 V
I (365180) AD7718: AIN2 Voltage: 0.481850 V
I (365380) AD7718: AIN3 Voltage: 0.331905 V
I (366380) gasmonitor: Publishing: {"temperature":81.25, "humidity":49.01,
 "co2":572.00, "NH3":4.391113, "H2S":0.668618, "CH4":1.914688}
I (370580) AD7718: AIN1 Voltage: 0.517552 V
I (370780) AD7718: AIN2 Voltage: 0.482742 V
I (370980) AD7718: AIN3 Voltage: 0.331237 V
I (370980) gasmonitor: Publishing: {"temperature":80.85, "humidity":49.54,
 "co2":575.00, "NH3":4.432902, "H2S":0.662237, "CH4":1.932789}
```

Figure 2. ESP32 Terminal showing data collection

ID	Temperature (°F)	Humidity (%)	CO2 (ppm)	NH3 (ppm)	H2S (ppm)	CH4 (ppm)	Timestamp
17135	80.85	49.54	575	4.4329	0.662237	1.93279	2025-04-17 12:06:39
17134	81.25	49.01	572	4.39111	0.668618	1.91469	2025-04-17 12:06:36
17133	81.53	48.59	636	4.43172	0.636742	1.90067	2025-04-17 12:06:28

Figure 3. Web application displaying data

The figures above depict the ESP32 terminal fetching the data from the sensors, and sending this data along with the timestamp to the database, which is eventually received by the web application.

2.3. Validation

Make database/web app more secure	Password/2FA security through AWS	YES	Tanmay
Integrate MCU w/ database	Successful data transmission < 5 seconds	YES	Tanmay
Deploy web application	Web application stays online 24/7	YES	Tanmay
Read Analog Sensor Data	Successfully convert analog data through the ADC by testing different voltage inputs.	YES	Joaquin
Read Digital Sensor Data	Successfully collect accurate and consistent data from I2C sensors.	YES	Joaquin
Transmit Sensor Data	Transmit sensor data wirelessly over a Wi-Fi network to AWS.	YES	Joaquin
Wi-Fi Range and Stability	Maintain stable transmission over extended time periods (30 min, 1 hr, 2 hrs, 3 hrs)	YES	Joaquin
I2C Communications	Code can read CO2 and temperature/humidity sensor.	YES	Blake
SPI Communications	Code can read analog input from analog to digital converter.	YES	Blake
Microcontroller Flash	Microcontroller can upload code.	YES	Blake
Voltage Divider and Op Amp	Voltage divider cuts voltage in half. Unity gain op amp does not change the voltage. Works from range of .25 - 5V.	YES	Blake
Correct Output Voltages	Specified voltages of 5,-5,3.3 V are supplied	YES	Matthew
Load Requirements Met	Power subsystem can adequately provides 0.5-1A of current	YES	Matthew
Entire project is safely housed	Successfully designed a housing unit that will protect the sensors, pcbs, transformers.	YES	Matthew

2.3.1. Sensor Subsystem Validation

To validate the sensor subsystem, code was uploaded onto the microcontroller successfully, data was received from the sensors and ADC from the microcontroller, and the voltage levels on the op amp and voltage divider were tested.

2.3.2. Power Subsystem Validation

The power subsystem was validated by testing the performance of the voltages under full load conditions. We began by verifying the board's operation with no sensors attached, then added each sensor sequentially until the system ran as expected. Below I have added images of the ossicope testing all voltages when they were under full load conditions.

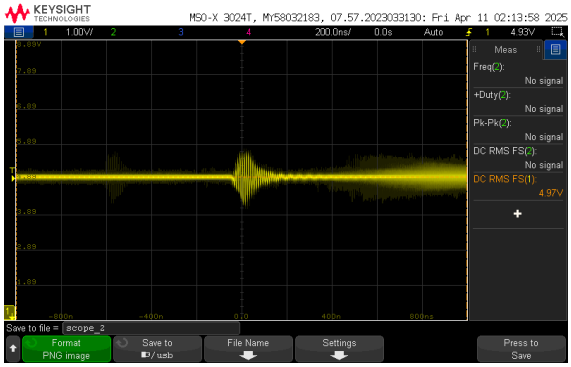


Figure 4. Oscilloscope screenshot at 5V

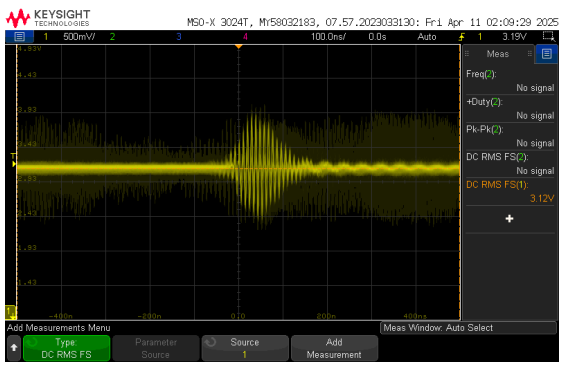


Figure 5. Oscilloscope screenshot at 3.3V

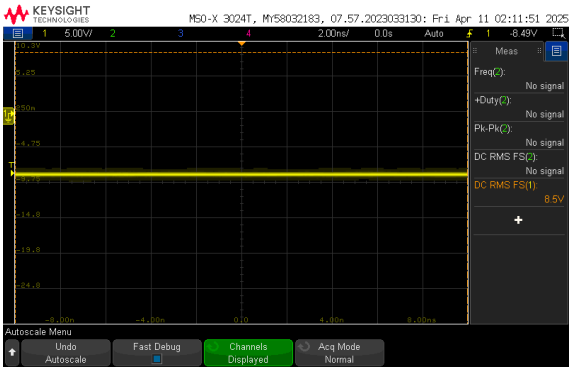


Figure 6. Oscilloscope screenshot at -8.5V

2.3.3. Database + Web Application Subsystem Validation

Due to this subsystem being software-based, there were no physical tests done to validate that the goals were met. Rather, it was validated through simply completing each task and then leaving the system up and running to make sure it stays consistent. For example, in order to validate that the web application was staying up and running 24/7, I deployed it through AWS and consistently checked the logs to ensure that it was not down for an extended period of time. In addition, the data is successfully transferring 1 data point every 5 seconds.

Time	Type	Details
April 22, 2025 01:37:36 (UTC-5)	INFO	Environment health has transitioned from Severe to Ok.
April 22, 2025 01:36:37 (UTC-5)	WARN	Environment health has transitioned from Ok to Severe. 96.0 % of the requests are erroring with HTTP 4xx.
April 22, 2025 01:25:38 (UTC-5)	INFO	Environment health has transitioned from Severe to Ok.
April 22, 2025 01:24:38 (UTC-5)	WARN	Environment health has transitioned from Ok to Severe. 98.5 % of the requests are erroring with HTTP 4xx.
April 22, 2025 01:13:39 (UTC-5)	INFO	Environment health has transitioned from Severe to Ok.
April 22, 2025 01:12:39 (UTC-5)	WARN	Environment health has transitioned from Ok to Severe. 93.5 % of the requests are erroring with HTTP 4xx.
April 21, 2025 19:26:09 (UTC-5)	INFO	Environment health has transitioned from Severe to Ok.
April 21, 2025 19:25:09 (UTC-5)	WARN	Environment health has transitioned from Ok to Severe. 92.3 % of the requests are erroring with HTTP 4xx.
April 19, 2025 20:01:21 (UTC-5)	INFO	Environment health has transitioned from Severe to Ok.
April 19, 2025 20:00:21 (UTC-5)	WARN	Environment health has transitioned from Ok to Severe. 92.9 % of the requests are erroring with HTTP 4xx.
April 19, 2025 15:53:43 (UTC-5)	INFO	Environment health has transitioned from Warning to Ok.

Figure 7. AWS error logging

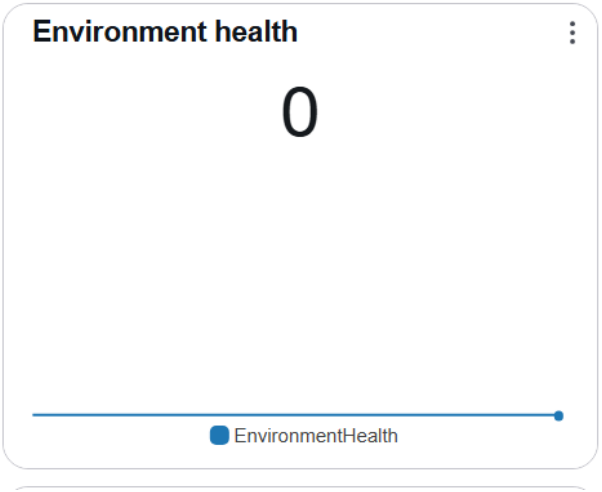


Figure 8. AWS Environment health logging

The first log shows all the times the website had health issues over 3 days. By analyzing the timestamps, we can see that it typically went down for one minute at a time, and remained up otherwise. The environment health picture indicates that there were not any fluctuations in the health of the web application over the last 3 hours.

2.3.4. Microcontroller + Data Transmission Subsystem Validation

```
I (359770) gasmonitor: Publishing: {"temperature":81.53, "humidity":48.59,
 "co2":636.00, "NH3":4.431723, "H2S":0.636742, "CH4":1.900668}
I (364980) AD7718: AIN1 Voltage: 0.515357 V
I (365180) AD7718: AIN2 Voltage: 0.481850 V
I (365380) AD7718: AIN3 Voltage: 0.331905 V
I (366380) gasmonitor: Publishing: {"temperature":81.25, "humidity":49.01,
 "co2":572.00, "NH3":4.391113, "H2S":0.668618, "CH4":1.914688}
I (370580) AD7718: AIN1 Voltage: 0.517552 V
I (370780) AD7718: AIN2 Voltage: 0.482742 V
I (370980) AD7718: AIN3 Voltage: 0.331237 V
I (370980) gasmonitor: Publishing: {"temperature":80.85, "humidity":49.54,
 "co2":575.00, "NH3":4.432902, "H2S":0.662237, "CH4":1.932789}
```

Figure 9. Data collection

The figure above demonstrates full functionality as outlined in the validation plan. A JSON-formatted message contains all sensor values when they are successfully transmitted to AWS IoT. This verifies that the gas monitor can read analog and digital sensor data, reliably transmit sensor data, and maintains a stable connection to Wi-Fi. The tests conducted at the animal farm are also validation that with robust error-handling developed through troubleshooting edge cases, ensuring continuous and stable operation of the gas monitoring system for extended data collection points.

2.4. Data Collection Results

Once integrated, the gas monitor was taken to a farm on west campus two separate times to conduct tests. The sensor was plugged in and connected to TAMU wifi and data was collected for 2-3 hours and collected over 2160 data points. Different stimuli were introduced to attempt to cause change in the sensor readings. Different stimuli introduced include hay, cow patty, and damp cow bedding. The base state for the test was sitting in the room with no elements placed next to it. When damp cow bedding was introduced, there was a spike in ammonia and a smaller spike in methane and hydrogen sulfide. This is consistent as ammonia is produced from damp cow bedding at 10x the magnitude of the other gases.

Figure 10: Stimuli Introduced

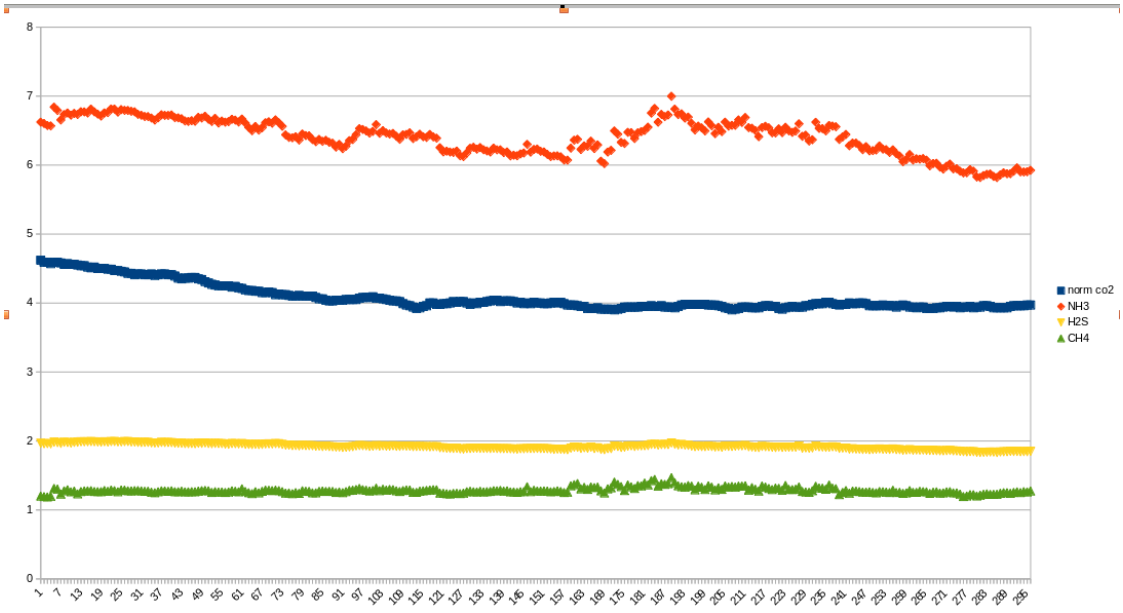


Figure 11: First Farm Visit Data

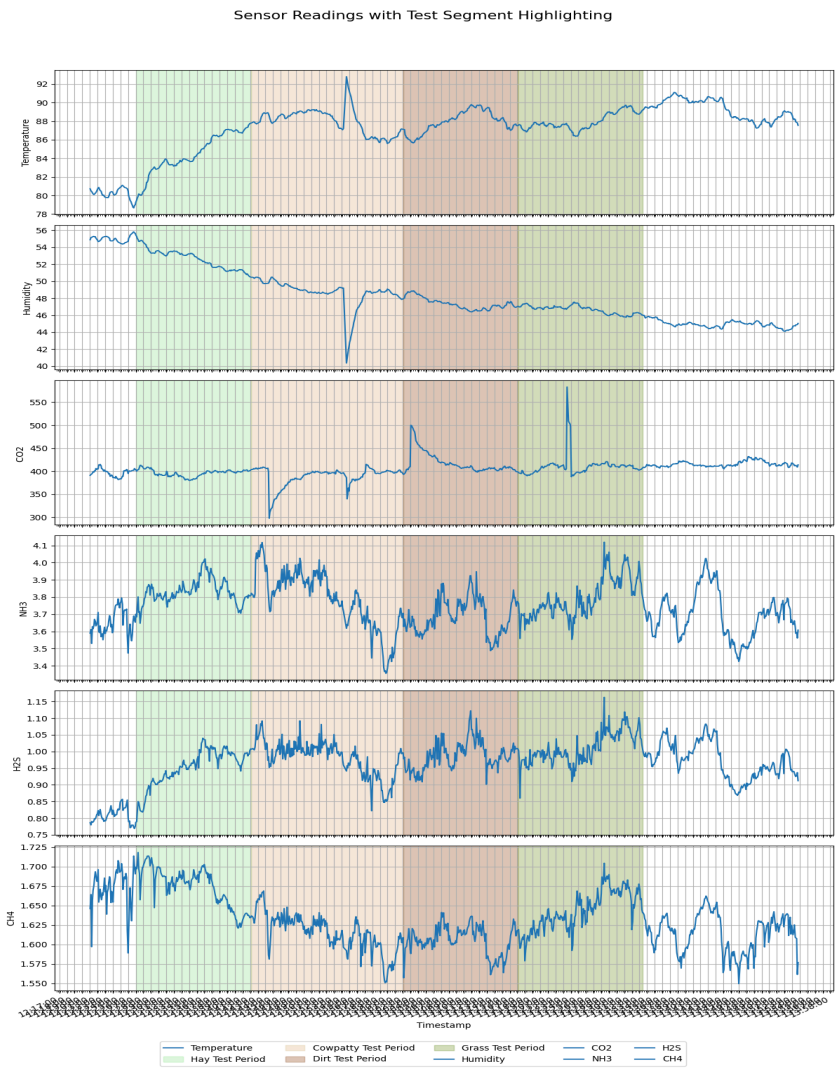


Figure 12. Data collection from all tests second farm visit

3. Conclusion

Overall the final system met all of our sponsors expectations. An accurate proof of concept for our sponsor's vision was created and a framework for future refinement and changes to the system was developed.

The system measured the gases that were part of the requirements. Real time measurements are taken every 5 seconds, the data is transmitted wirelessly, ultimately being displayed on a website for users to access remotely. The system is intentionally easy to set up for our sponsor made possible due to features such as wifi provisioning.

A few flaws were found while testing the system, but they can be easily fixed in the future. The first flaw is the analog sensors heat up the board when powered, leading to skewed temperature readings. The solution to this would be either moving the sensor positions on

the board or using an external sensor. Another area that requires future work in further calibration of the analog sensors or buying better sensors with a bigger budget. The sensors need to be run for 24+ hours and put in a chamber with specific concentrations of gases to be calibrated. Another way to improve the system could be buying higher quality sensors.

Another flaw that our gas monitor has experienced is the Wi-Fi signal strength. The ESP32-S3 with the built in antenna was selected with the intent to use LoRaWAN. When the scope was changed to Wi-Fi due to ease of use and lower cost, the Wi-Fi range was not taken into account. A way to improve Wi-Fi transmission range would be adding Wi-Fi repeating infrastructure to an animal farm and upgrading the antenna on the ESP32.

3.1. Sponsor Feedback and User Manual

In order for our sponsor to continue his work and move forward with this project, our team has created a user manual which walks through setup, how to update code, and wifi provisioning. The document is located below and shared with the sponsor:

Livestock Gas Monitor - User Manual

Our sponsor, Justin Houck, is satisfied with our work. Justin stated “The project operates within expected operational scope and exceeds expectations in IoT connectivity and hotspot capability.” Moving forward, more tests will be conducted over the summer by Justin in collaboration with the animal science and cybersecurity departments at Texas A&M University.