Wrocław University of Science and Technology

Optoelectronics Project
CO2 Monitoring Device

Instructor: Kinga Kaczmarek

Ata Goker 276828

Onurcan Icen 269517

Submission Date: 13.01.2025

Table of Contents

1 Introduction

1.1 Project Objective:

The primary goal of this project is to develop a functional system that measures and displays CO2 concentration, temperature, and humidity using optoelectronic sensors and an integrated hardware setup. The project aims to provide accurate environmental monitoring data, which can be used in various applications such as indoor air quality control, greenhouse management, and smart home systems.

1.2 Subject of the Report:

This report details the development and functionality of the CO2 sensor system. It covers the integration of the selected hardware components, the design of the software for data processing and visualization, and the evaluation of the system's performance under different environmental conditions.

1.3 Report Contents:

The report starts with an overview of the theoretical principles behind CO2 detection and environmental monitoring. It then provides a detailed explanation of the hardware components and software architecture utilized in the project. Following this, the methods for system calibration and testing are described, along with an analysis of the collected data. The report concludes with a discussion of the outcomes, a summary of findings, and references to relevant sources.

2 Theoretical Introduction

2.1 Theoretical Background

CO2 detection is based on the principles of non-dispersive infrared (NDIR) technology, where infrared light absorption is used to measure the concentration of CO2. The interaction between CO2 molecules and specific wavelengths of infrared light enables precise monitoring of air quality, making it a widely adopted method in environmental sensing applications.

2.2 Domain

The domain of this project lies in environmental monitoring and smart sensing. These systems are critical in applications such as indoor air quality management, greenhouse optimization, and industrial process monitoring, where real-time data about CO2 levels, temperature, and humidity are essential.

2.3 Phenomena

The key phenomenon utilized in this project is the absorption of infrared radiation by CO2 molecules. This behavior is quantifiable and forms the basis for measuring CO2 concentration. Additionally, the project considers the impact of environmental factors like temperature and humidity on the accuracy of measurements.

2.4 Realizations

The realization of CO2 detection involves the integration of an NDIR sensor module with a microcontroller for data acquisition and processing. The processed data is visualized on an OLED display for easy interpretation. Calibration ensures accuracy, while the system is tested under various conditions to validate its reliability and performance.

3 Assumptions

3.1 Functional Assumptions: Capabilities and Features

The system is assumed to accurately measure CO2 concentration, temperature, and humidity in real-time. It should display the collected data on an OLED screen for immediate visualization and allow consistent monitoring over extended periods without significant drift in accuracy.

3.2 Design Assumptions: Implementation and Architecture

The design assumes seamless integration of the NDIR sensor, microcontroller, and OLED display, with efficient data processing and communication between components. It is also assumed that the system will be powered reliably, and that the software will handle data calibration and noise filtering effectively.

3.3 Environmental Assumptions: Operating Conditions and Limitations

The system is assumed to function optimally within standard environmental conditions, such as indoor temperature and humidity ranges. It is also expected to remain unaffected by moderate external disturbances like electrical noise or minor temperature fluctuations.

4 Components

4.1 FeatherWing OLED Module (1.3" 128x64 Display)

This module features a 1.3-inch OLED screen with a resolution of 128x64 pixels, providing a high-contrast display for real-time data visualization. It utilizes the I2C communication protocol, making it easy to interface with the Raspberry Pi Zero 2 WH. The display is lightweight, power-efficient, and compact, making it suitable for portable applications. Its ability to render text and simple graphics ensures clear and concise representation of CO2 levels, temperature, and humidity data.

4.2 STEMMA QT SCD-30 NDIR CO2 Sensor Module

The SCD-30 sensor is a key component of the project, measuring CO2 concentration, temperature, and humidity. Utilizing non-dispersive infrared (NDIR) technology, the sensor provides high accuracy and reliability in detecting CO2 levels. It supports both I2C and UART communication protocols, allowing flexibility in integration. Its built-in temperature and humidity compensation enhances the accuracy of CO2 readings, making it ideal for various environmental monitoring scenarios.

4.3 Raspberry Pi Zero 2 WH

The Raspberry Pi Zero 2 WH serves as the main processing unit, offering a quad-core 1GHz processor and 512MB RAM. Its built-in Wi-Fi and Bluetooth capabilities enable remote monitoring and system updates. The compact design and GPIO pins facilitate integration with the sensor and display, while the device provides sufficient computational power to handle real-time data acquisition, processing, and visualization.

5 Circuit Design and Connections

The circuit design revolves around the Raspberry Pi Zero 2 WH, which serves as the central processor, integrating the SCD-30 CO2 sensor and the FeatherWing OLED display. The Raspberry Pi runs a program to acquire environmental data, such as CO2 concentration, temperature, and humidity, from the SCD-30 sensor. This data is processed and transmitted to the OLED display for real-time visualization. The I2C communication protocol is utilized to connect these components, requiring only two lines—SDA for data and SCL for the clock. This protocol simplifies wiring while ensuring efficient data transfer.

Power is distributed through the Raspberry Pi, which provides a stable 3.3V power supply and a common ground to both the sensor and the display. This centralized setup reduces circuit complexity and ensures consistent voltage levels. Grounding is implemented carefully to minimize electrical noise, which could compromise the accuracy of the sensor or clarity of the OLED display.

The modular design makes the system easy to assemble, debug, and scale. This setup also leaves room for future enhancements, such as adding sensors or enabling wireless data logging.

6 Connections

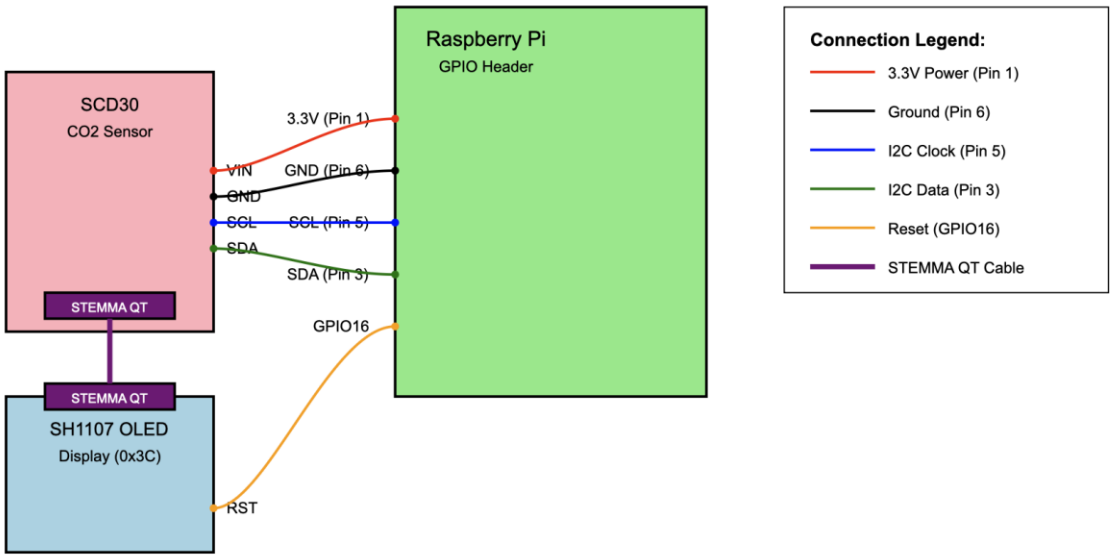


Figure 1: Connection Diagram

6.1 Power and Ground Connections

All components share a common power and ground line. The Raspberry Pi Zero 2 WH supplies 3.3V and GND to the SCD-30 sensor and the OLED display, ensuring stable power delivery to all parts of the circuit.

6.2 Data Communication Pins

The I2C protocol is used for communication between the components. The Raspberry Pi's SDA (data line) and SCL (clock line) pins are connected to the corresponding SDA and SCL pins on both the SCD-30 sensor and the OLED display is connected to SCD30 using STEMMA QT connectors. This arrangement allows the Raspberry Pi to send and receive data efficiently.

6.3 Sensor Data Flow

The CO2 sensor collects environmental data (CO2 concentration, temperature, and humidity) and transmits it to the Raspberry Pi over the I2C bus. The Raspberry Pi processes the received data and sends it to the OLED display, where it is shown in real time for user interpretation.

This setup ensures a streamlined flow of data from the sensor to the display, with the Raspberry Pi acting as the central hub for processing and control.

6.4 Working of the Circuit

The Raspberry Pi Zero 2 WH serves as the controller, running a program to retrieve data from the SCD-30 CO2 sensor via the I2C protocol. The sensor measures CO2 concentration, temperature, and humidity, compensating for environmental variations for accurate results. The data is processed by the Raspberry Pi and sent to the FeatherWing OLED display, which shows the values in real-time. This ensures users can easily monitor environmental conditions. The circuit operates in a continuous loop, automatically updating the display as new data is received, providing seamless functionality with minimal user input.

7 Code Explanation

```
import board
import displayio
import terminalio
import adafruit_displayio_sh1107
from adafruit_display_text import label
import adafruit_scd30
import time
from datetime import datetime
import digitalio

# Initialize DisplayIO
displayio.release_displays()

# Initialize I2C
i2c = board.I2C()

# Setup reset pin
reset_pin = digitalio.DigitalInOut(board.D16)
reset_pin.direction = digitalio.Direction.OUTPUT
```

```
reset_pin.value = True # Start with reset inactive

# Initialize display once
display_bus = displayio.I2CDisplay(i2c, device_address=0x3C, reset=board.D16)
display = adafruit_displayio_sh1107.SH1107(display_bus, width=128, height=64)

# Initialize SCD-30
scd = adafruit_scd30.SCD30(i2c)

def try_reset():
print("Trying reset...")
reset_pin.value = False # Active LOW
time.sleep(0.5) # Hold reset
reset_pin.value = True # Release reset
time.sleep(0.5) # Wait for initialization
print("Reset complete")

# Create initial display group
splash = displayio.Group()

# Create initial text labels
co2_text = label.Label(
terminalio.FONT,
text="CO2: ---",
color=0xFFFFFF,
x=0,
y=10
)
splash.append(co2_text)

temp_text = label.Label(
terminalio.FONT,
text="Temp: ---",
```

```
color=0xFFFFFF,
x=0,
y=32
)
splash.append(temp_text)

hum_text = label.Label(
terminalio.FONT,
text="Hum: ---",
color=0xFFFFFF,
x=0,
y=54
)
splash.append(hum_text)

# Set initial display
display.root_group = splash

print("Starting CO2 Sensor Monitor...")
print("Press Ctrl+C to exit")
print("----------------------")

while True:
if scd.data_available:
# Get current time
current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
# Read and format sensor data
co2 = int(scd.CO2) # Round to integer
temp = round(scd.temperature, 2) # 2 decimal places
humidity = round(scd.relative_humidity, 2) # 2 decimal places
# Try hardware reset
try_reset()
# Clear display
```

```
display.root_group = None
splash = displayio.Group()
# Create new labels with formatted values
co2_text = label.Label(
terminalio.FONT,
text=f"CO2: {co2} PPM",
color=0xFFFFFF,
x=0,
y=10
)
splash.append(co2_text)
temp_text = label.Label(
terminalio.FONT,
text=f"T: {temp} C",
color=0xFFFFFF,
x=0,
y=32
)
splash.append(temp_text)
hum_text = label.Label(
terminalio.FONT,
text=f"H: {humidity} %",
color=0xFFFFFF,
x=0,
y=54
)
splash.append(hum_text)
# Update display
display.root_group = splash
# Print to terminal
print(f"Time: {current_time}")
print(f"CO2: {co2} ppm")
print(f"Temperature: {temp}°C")
```

```
print(f"Humidity: {humidity}%")
print("------------------------")
time.sleep(2)
```

7.1 Main Code Overview

In this code, the primary goal is to read CO2, temperature, and humidity data from an SCD-30 sensor and display that information on a SH1107 OLED screen. The workflow starts by importing the necessary libraries, such as board, digitalio, displayio, and the sensor's own library, adafruit_scd30. We then create an I2C bus object, wire up a reset pin for the display, and initialize both the display and the sensor.

After these components are set up, the program enters an infinite loop (while True:) where it continuously checks if new sensor data is available. If so, the code reads the data (CO2, temperature, humidity), attempts a hardware reset (to ensure the display is functioning properly), and finally updates both the terminal and the OLED display with the fresh sensor readings. The loop repeats with a small delay, allowing the user to observe real-time updates.

7.2 Pin and Variable Definitions

I2C Pins: We use the default board.I2C() pins for communication with the SCD-30 sensor and the SH1107 display. Typically, these map to SCL and SDA on our board.

•**Reset Pin (board.D16)**: This digital pin is used to control the hardware reset line of the SH1107 display. We set it as an output pin and keep it high (inactive) by default. Toggling this pin low triggers a reset signal to the display.

•**display_bus**: This variable holds the I2C display object (displayio.I2CDisplay) that abstracts away the lower-level communication details with the SH1107.

•**display**: An instance of adafruit_displayio_sh1107.SH1107 representing the actual OLED display, configured to a specific width (128) and height (64).

•**scd**: The sensor object (adafruit_scd30.SCD30), which handles the reading of CO2, temperature, and humidity from the connected SCD-30 module.

7.3 Initialization and Setup

1.**Releasing any Active Displays**: At the beginning, we call displayio.release_displays() to ensure no other display sessions are hogging resources.

2.**Establishing I2C Connection**: We create an instance of board.I2C(). This sets up the pins needed for communicating with I2C devices (the SH1107 and the SCD-30).

3.**Configuring the Reset Pin**: A DigitalInOut object is created for board.D16, set as an output, and held high initially.

4.**Initializing the Display**: We create an I2CDisplay object (display_bus) using our I2C instance, specifying the device address for the SH1107 (0x3C) and the reset pin we just configured. Then we create the main display object, giving it the dimensions of our OLED.

5.**Initializing the Sensor**: The scd object is instantiated to start reading from the SCD-30 sensor. Once this is done, the sensor is ready to provide CO2, temperature, and humidity data.

7.4 Continuous Operation Loop

Inside the while True: loop, we have several important operations that happen repeatedly:

1.**Check Data Availability**: The code checks scd.data_available. Only when this is True do we proceed to read the sensor's latest CO2, temperature, and humidity data.

2.**Try Reset**: We call our try_reset() function, which briefly pulls the reset pin low, then brings it back high. This ensures the display doesn't lock up or freeze.

3.**Clear and Rebuild the Display**: First, we set display.root_group = None to clear out whatever was previously on the screen. Then we create a fresh display group (splash) and add new label objects with the most recent sensor readings.

4.**Update Terminal and Display**: We print the data to the serial console so you can watch it in your terminal. Then, by setting display.root_group = splash, we make the new labels visible on the OLED.

5.**Delay**: We wait two seconds (time.sleep(2)) before repeating the cycle. This allows enough time for each reading to be clearly displayed and helps prevent excessive updates.

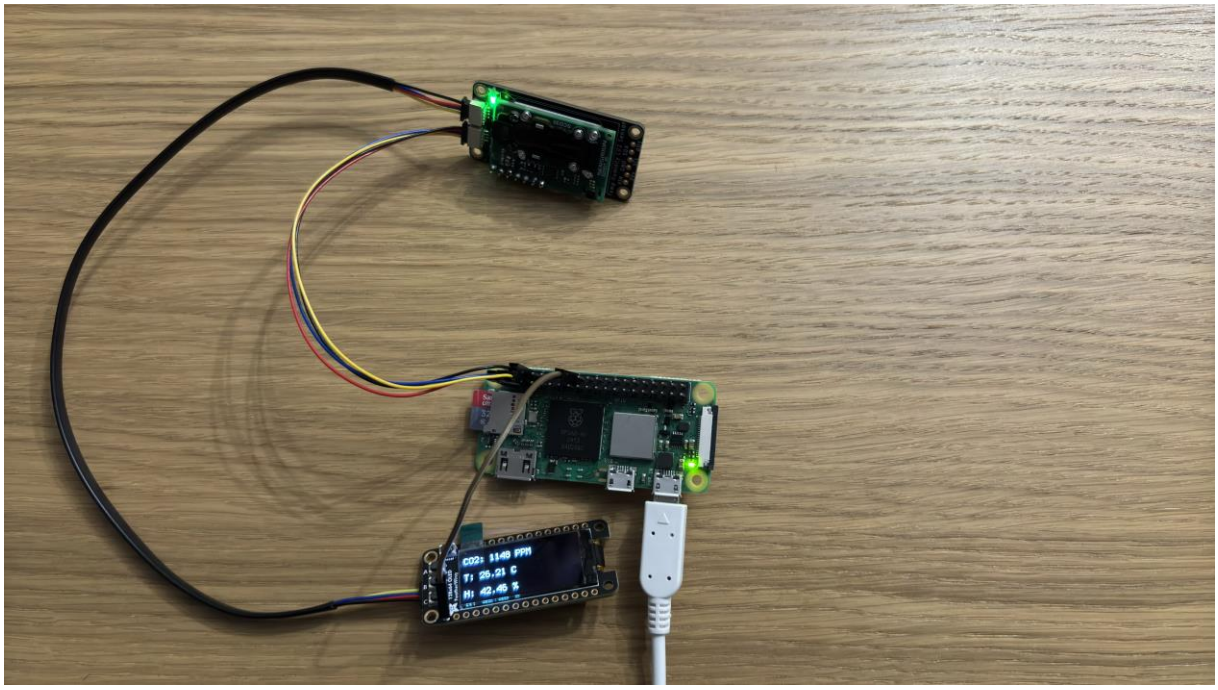
7.5 Data Processing and Display Functions

We convert the sensor data (scd.CO2, scd.temperature, scd.relative_humidity) into human-readable formats. For example, CO2 is cast to an integer, while temperature and humidity are rounded to two decimal places.

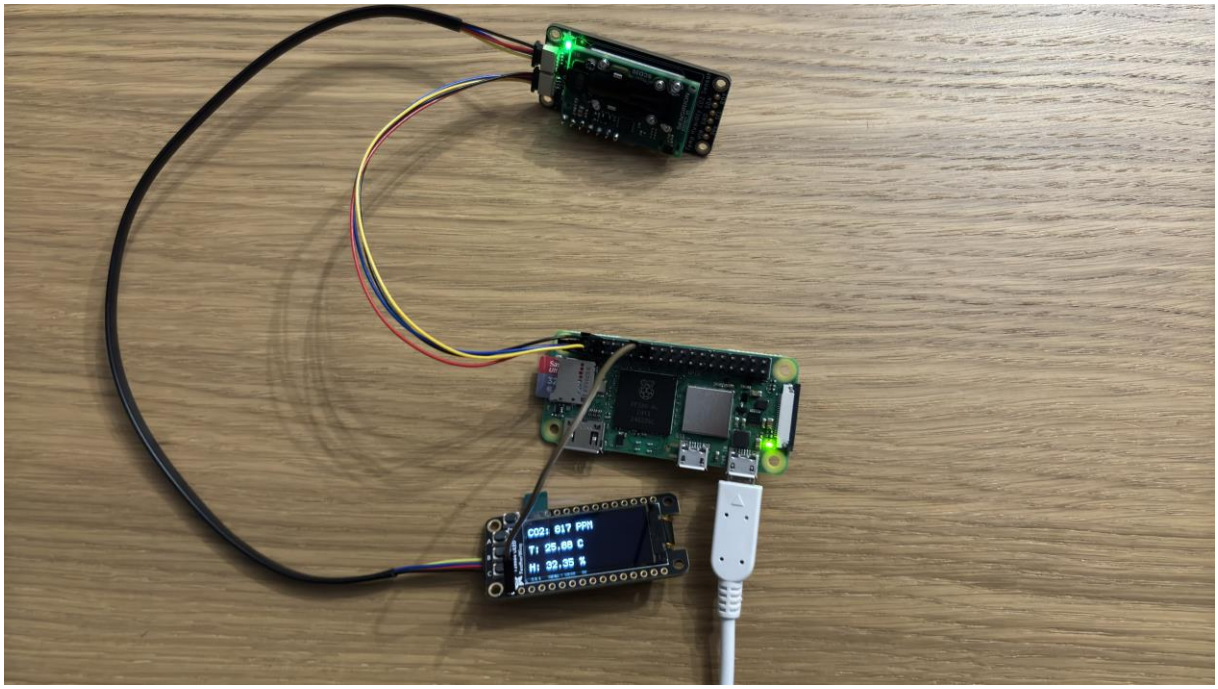
- **Labels and Display Group**: We create new label.Label instances for CO2, temperature, and humidity. Each label is positioned on the display using x-y coordinates. These labels are added to a single displayio.Group() so they can all be rendered together.

•**Printing to Terminal**: We also print a timestamp, CO2, temperature, and humidity to the serial console. This provides a convenient way to monitor or debug the sensor readings without looking directly at the OLED.

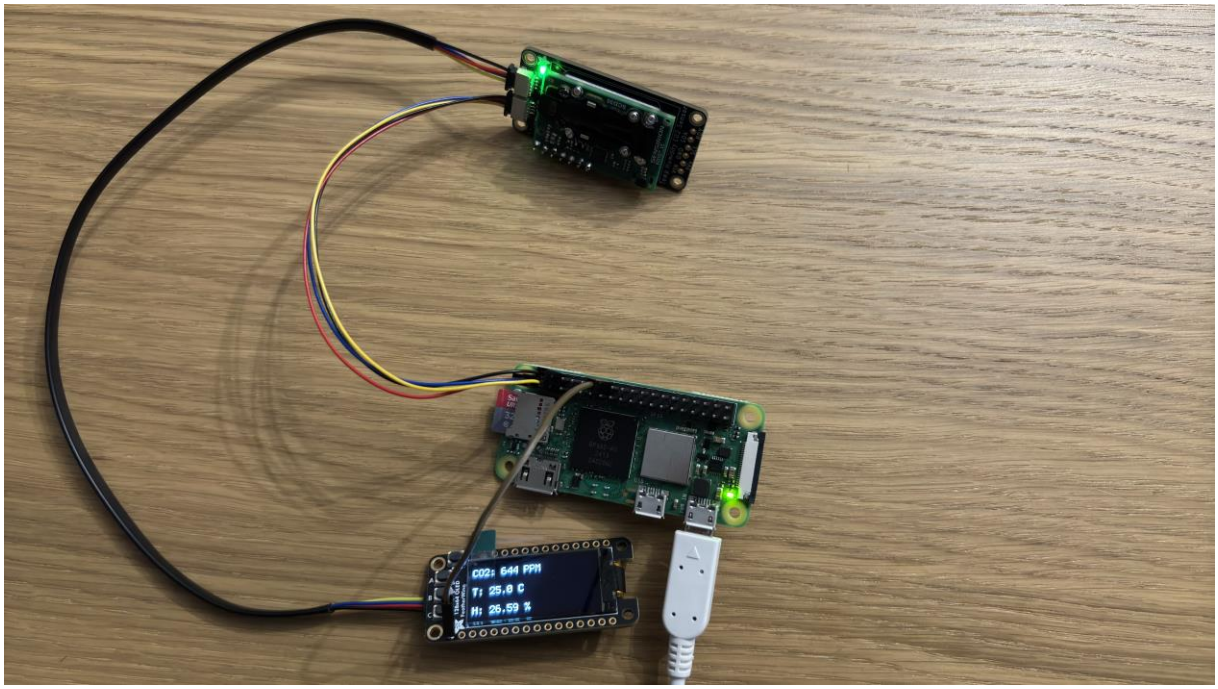
8 Testing the Project



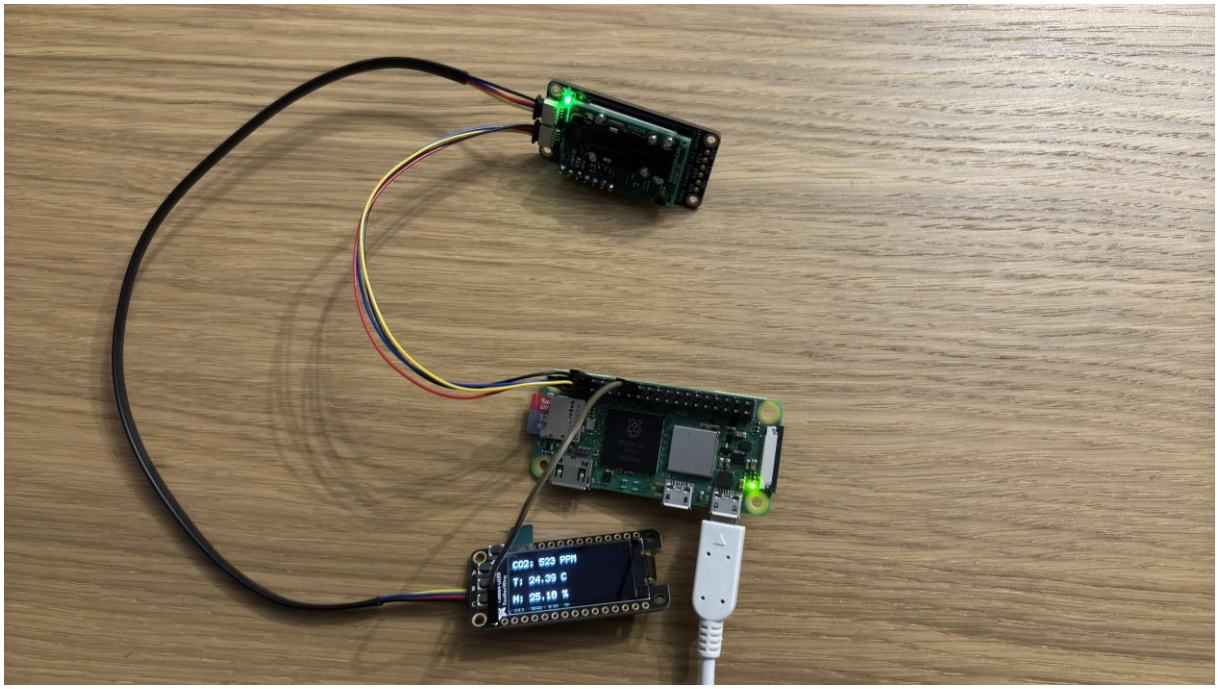
Picture 1: Initial Measurement



Picture 2: One Minute After Opening the Window



Picture 3: Five Minutes After Opening the Window



Picture 4: Ten Minutes After Opening the Window

Below are the observations made during testing in a room without mechanical ventilation, where the only method of introducing fresh and cold air was by opening a window. Initially, with the room sealed, CO2 levels were measured at **1148 ppm**, temperature at **26.21 °C**, and humidity at **42.46%**, indicating significant buildup due to the lack of fresh air. After **1 minute** of opening the window, CO2 dropped to **817 ppm**, temperature decreased slightly to **25.88 °C**, and humidity fell to **32.35%**, showing how quickly fresh, cooler air can change indoor conditions. By **5 minutes** post-opening, CO2 levels had reduced further to **644 ppm**, temperature settled at **25.00 °C**, and humidity dipped to **26.59%**, reflecting continued ventilation. Finally, at **10 minutes**, CO2 measured **523 ppm**, temperature **24.39 °C**, and humidity **25.18%**, demonstrating a return to near-typical fresh-air conditions and confirming the sensor's ability to accurately track real-time environmental changes.

Condition	CO2 (ppm)	Temperature (C)	Humidity (%)
Initial Measurement (Not Ventilated)	1148	26.21	42.46
1 Minute After Opening the Window	817	25.88	32.35
5 Minutes After Opening the Window	644	25.00	26.59
10 Minutes After Opening the Window	523	24.39	25.19

9 Conclusion

9.1 Project Focus and Objectives

This project focused on designing and implementing a system for monitoring CO2 concentration, temperature, and humidity using a Raspberry Pi Zero 2 WH, an SCD-30 CO2 sensor, and an OLED display. The primary goal was to develop an accurate, real-time monitoring solution for environmental conditions.

9.2 Outcomes and Achievements

The system successfully measured and displayed environmental parameters in real-time, with reliable data acquisition and visualization. Testing showed that the integration of components and the use of the I2C protocol provided seamless operation, meeting the project's objectives effectively.

9.3 Challenges and Issues Faced

Several challenges were encountered during development, such as sensor calibration to ensure accuracy under varying environmental conditions and managing electrical noise that initially affected data integrity. Troubleshooting these issues required adjustments to both hardware and software.

9.4 Future Enhancements and Prospects

Potential future improvements include adding wireless capabilities for remote monitoring, integrating additional sensors for expanded functionality, and developing a mobile application for enhanced user interaction. These enhancements could make the system more versatile and user-friendly.

10. References

1. SCD-30 CO2 Sensor Datasheet
   https://cdn-shop.adafruit.com/product-files/4867/Sensirion_CO2_Sensor_SCD30_Datasheet.pdf
2. FeatherWing OLED Display Datasheet
   https://learn.adafruit.com/adafruit-oled-featherwing/overview
3. Raspberry Pi Zero 2 WH Documentation
   https://www.raspberrypi.com/documentation/computers/raspberry-pi.html
4. I2C Protocol Overview
   https://learn.sparkfun.com/tutorials/i2c/all

5. Python Libraries for Hardware Communication
   https://learn.adafruit.com/circuitpython-on-raspberrypi-linux
6. Adafruit CircuitPython SCD30 Library Documentation
   https://github.com/adafruit/Adafruit_CircuitPython_SCD30
7. OLED Display Python Library Documentation
   https://github.com/adafruit/Adafruit_CircuitPython_SSD1306
8. Non-Dispersive Infrared (NDIR) CO2 Sensing Principles
   https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/8_NDIR_CO2_Principle_of_Operation.pdf

These references provide essential resources for understanding the components, communication protocols, and coding tools used in this project.