Environmental sensor

Introduction

The main objective of this project is to develop a portable environmental monitoring system integrated around the ESP32-S3 microcontroller. The system architecture uses multiple communication protocols to ensure efficient data exchange. The BME680 and VL53L4CX sensors and the OLED display share the same I²C protocol, an SD card module for high-speed data logging is connected via SPI, and real-time location data is obtained from the GPS module via the UART serial interface.
The following sections describe the hardware design implementation of this system in detail. This includes a PCB layout and efficient routing strategies. Also, the use of a ground plane on the bottom layer to ensure signal integrity and physical separation of the BME680 sensor from the main controller's heat sources to maintain measurement accuracy. In addition, the report presents the optimization of the design of an easy-to-assemble, screw-free, 3D printed enclosure that provides mobility and physical protection for the system.

2. System Design Overview

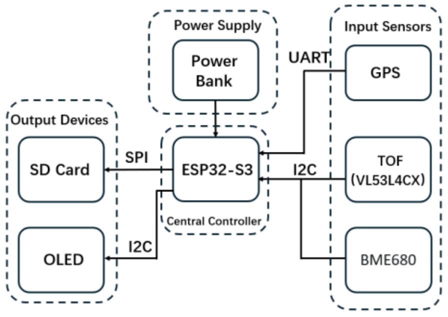


Figure 1. *System framework diagram*

The core of this environmental sensor system is an ESP32-S3 microcontroller, which serves as the central processing unit responsible for coordinating communication, data processing, and storage tasks across all peripheral devices.

The system architecture employs multiple communication protocols to optimize performance and wiring. Specifically:
(1) The I²C (Inter-Integrated Circuit) bus functions as a shared channel, connecting the BME680 environmental sensor, VL53L4CX time-of-flight (TOF) sensor, and OLED display. This design leverages the I²C protocol's capability to communicate with multiple devices using only two signal wires, effectively conserving GPIO pin resources. A dedicated SPI (Serial Peripheral Interface) bus is exclusively employed for high-speed data transfer between the ESP32-S3 and the SD card module, ensuring efficient and reliable data logging. The GPS module connects to the controller via a dedicated UART (Universal Asynchronous Receiver-Transmitter) serial interface to receive real-time geolocation and timestamp data.
(2) The system workflow operates as follows: the ESP32-S3 actively acquires data from various input sensors. It parses positioning information from the GPS module via the UART interface and reads temperature/humidity, barometric pressure data from the BME680, and distance data from the VL53L4CX via the I2C bus. Following processing by the controller, key data is displayed in real-time on the OLED screen. Concurrently, the complete dataset is written on an SD card for subsequent offline analysis. The entire system is powered by a portable power bank, ensuring its mobility as a portable environmental monitoring device.

3. Hardware Design and Implementation

3.1. PCB Design and Sensor Integration

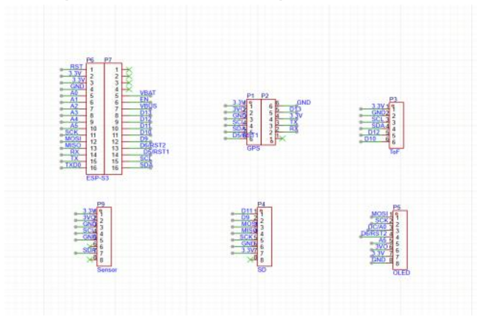


Figure 2. *PCB schematic diagram*

The circuit design of this system centers on the Adafruit ESP32-S3 Feather development board. Leveraging its extensive GPIO pins, we have realized a highly integrated multi-sensor monitoring platform. The development board connects to our customized PCB baseplate via onboard header interfaces (P6 and P7), serving not only as the central processing unit but also supplying power to all peripheral modules. To achieve efficient, reliable communication and optimize pin usage, multiple bus protocols are employed to connect different peripherals. The specific connection scheme is as follows:
(1) I²C Bus Device Integration:
To simplify wiring and conserve I/O resources, several key devices within the system are integrated onto a single I²C bus. The ESP32-S3's SCL (pin P7-15) and SDA (pin P7-16) pins serve as the shared bus, connected in parallel to:
OLED display (P5): For real-time data visualization.
BME680 environmental sensor (P9): For collecting temperature, humidity, barometric pressure, and gas data.
VL53L4CX Time-of-Flight Sensor (P3): for precise distance measurement.
By assigning a unique I²C address to each device, the controller can communicate precisely with specific components.
(2) SPI Bus Device Integration:
Considering the data logging requirements for transmission rates, the SD card module (P4) connects to the controller via a dedicated SPI bus to ensure high-speed, stable read/write performance. The ESP32-S3's hardware SPI pins are utilized, with MOSI (P6-12), MISO (P6-13), and SCK (P6-11) connected to the corresponding pins on the SD card module respectively. The general-purpose GPIO pin D9 (P7-8) is

configured as the Chip Select (CS) signal to uniquely address the SD card module on the bus.

(3) UART Serial Device Integration:

The GPS module (P2) communicates with the ESP32-S3 via an independent UART serial interface. The controller's hardware serial port is enabled, with the TX (P6-15) pin connected to the GPS module's RX pin and the RX (P6-14) pin connected to the GPS module's TX pin. This implements full-duplex asynchronous serial communication for receiving geolocation and time data.

(4) Power Distribution:

The entire system's power supply is centrally managed and distributed by the ESP32-S3 development board. The 3.3V voltage and GND output from the on-board voltage regulator are efficiently distributed via dedicated power traces on the PCB to all sensor modules and the display, providing them with a stable, clean operating power source.

3.2. PCB Layout and Routing Strategies

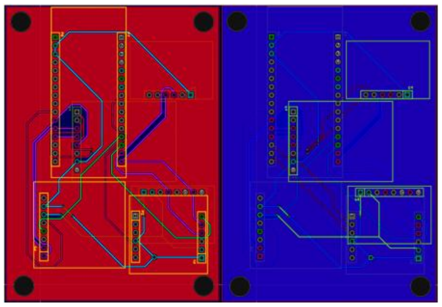


Figure 3. *PCB layout: Top layer (left) & Bottom layer (right)*

(1) Layout Strategy: We positioned the interfaces (P6, P7) of the ESP32-S3 Feather development board towards the center-left of the board. This centralized layout enables the controller to connect to peripherals distributed around the periphery via the shortest average path, effectively reducing signal trace lengths. For practical usability, the SD card module (P4) and OLED display (P5) are strategically positioned at the board's edge. This arrangement ensures users can easily insert/remove SD cards and clearly view display content without obstruction from other components. To mitigate interference from heat generated by the ESP32 main controller and its power management unit during operation, the BME680 environmental sensor (P9) is positioned in the upper-left corner of the circuit board, maintaining a physical distance from the main controller. This layout is crucial for ensuring the accuracy of temperature and humidity readings.

(2) Routing Strategy and Signal Integrity: A core strategy in this PCB design involves employing aa large copper-clad area on the bottom layer (blue) as a complete ground plane. This approach has multiple advantages: it provides all signals with a stable, low impedance return path, effectively suppressing electromagnetic interference (EMI) and crosstalk between signals. It enhances power supply stability, acting as a natural filter. It also facilitates uniform heat dissipation across the entire circuit board.

The top layer (red) primarily accommodates most signal traces and component placement. The bottom layer, beyond serving as a ground plane, is used only for short-distance jumpers where necessary. This minimizes signal trace crossings and detours, maintaining clear signal paths. For high-speed data lines such as the SPI bus, particular attention is paid to ensuring their routing paths are **as short and direct as possible**. Sharp 90-degree turns are avoided, replaced instead with 45-degree angles or curved transitions to minimize issues arising from signal reflection and impedance mismatch. The I²C bus employs a star topology, branching from the ESP32's SCL/SDA pins to connect each I²C device. This ensures roughly equal path lengths from each device to the master controller.

4. Enclosure Design

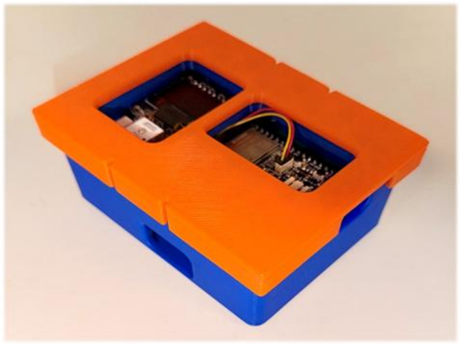


Figure 4.1. *Actual photograph of the enclosure*

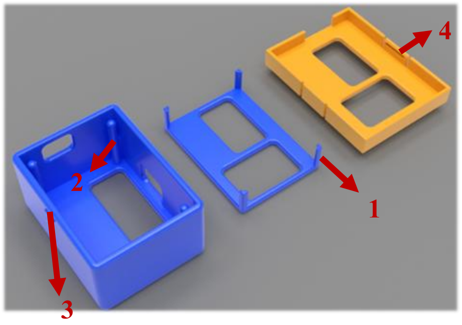


Figure 4.2. *Parts of the enclosure*

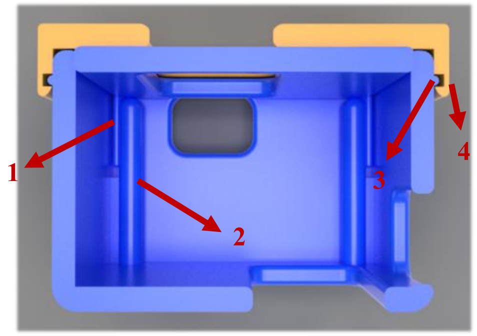


Figure 4.3. *Cross-section of the enclosure*

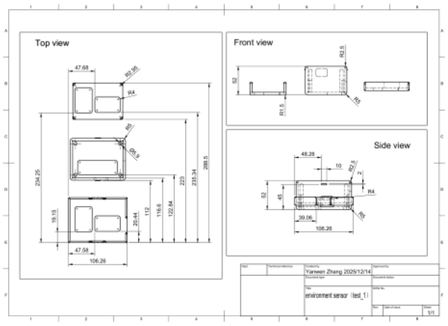


Figure 4.4. *Engineering drawing of the enclosure*

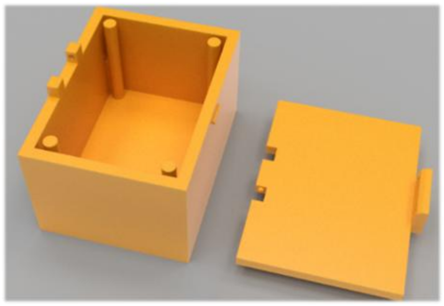


Figure 4.5. *Previous design of the enclosure*

4.1. Design Optimisation

As shown in **Figure 4.5**, in our initial design, we employed a snap-fit hinge mechanism on one side to enable the lid's opening and closing. Unfortunately, our 3D printing materials were limited to TPU and PLA. TPU proved unsuitable for the hinge due to its insufficient rigidity, while PLA's excessive stiffness hindered the design's functionality, as the hinge mechanism required the material's appropriate elasticity for assembly. After evaluating material properties and printing precision, we opted for a dual-side snap-fit design instead of the previous single-side hinge and single-side snap-fit configuration.

4.2. Design Explanation

For our new design, our casing incorporates a simple, detachable design, requiring no external components such as screws for assembly. In our design, the casing comprises three primary components. As illustrated above in the **Figure 4.2**, the first element on the left is the base housing, which accommodates and supports the PCB board while providing lower positioning. The second component is the positioning plate, serving to establish upper positioning. The third element is the top cover, which secures the positioning plate and engages with the base housing via a snap-fit connection.

As illustrated in **Figure 4.3**, Marker 1 denotes the primary limiting structure within the stop plate assembly. This cylindrical component presses against the upper surface of the PCB to prevent upward displacement; Marker 2 denotes the primary functional structure of the base housing. This cylindrical structure features diameters matching the circular apertures at the PCB's four corners. All four cylinders simultaneously pass through these apertures to horizontally secure the PCB. Additionally, a platform structure is incorporated between the lower half of the cylindrical structure and the base housing body. This platform supports the underside of the PCB, preventing downward displacement; Marker 3 denotes a semi-cylindrical protrusion situated on the base shell's side. Marker 4 indicates a recessed structure located on the inner surface of the top cover. Together, these form a snap-fit mechanism. Owing to PLA's slight malleability, the top cover need only be positioned atop the base shell and subjected to moderate force to engage the snap-fit. Similarly, the top cover and base shell may be separated by applying outward pressure.

5. Embedded Software

5.1. Code Organization and Module Partitioning

The firmware is required to read BME680 / ToF / GPS data, show the key outputs on an OLED, provide a web dashboard, and optionally record the measurements to an SD card. Based on these requirements, the program is organised into a main control file and several function modules, and the overall software structure is shown in Fig. 1.

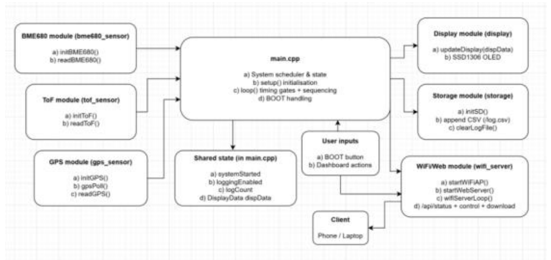


Figure 5.1. *Firmware module partitioning and data flow*

From main.cpp, it can be seen that the integration is handled in one place by including the module headers, and the key runtime variables are also maintained in the same file. In particular, systemStarted is used to control whether the system is still waiting at the welcome stage, loggingEnabled is used to control SD recording, logCount is used as the sample counter, and dispData is used as the shared container for the current system status.

Each hardware function is wrapped using a simple “data struct + interface functions” pattern. For example, BME680 uses BME680Data together with initBME680() and readBME680(), and ToF uses ToFData with initToF() and readToF(). The GPS module follows the same style through GPSData, and it should be noted that serial parsing is not done only at the moment of reading. Instead, gpsPoll() is called in the main loop to continuously decode incoming NMEA sentences, and readGPS() returns the most recently parsed result. In order to avoid maintaining two separate sets of variables for the OLED and the web page, the project defines DisplayData in display.h. At run time, the main program updates dispData, and both the OLED update function and the /api/status JSON output are derived from the same container, so the two interfaces are consistent by construction.

For the network layer, the public interface is intentionally kept short. As shown in wifi_server.h, only startWiFiAP(), startWebServer(), and wifiServerLoop() are exposed, while DNS handling and HTTP routing are implemented in wifi_server.cpp. In order to support web-side control (entering measurement mode and controlling logging), the server synchronises with the main program by referencing shared states via extern (e.g., loggingEnabled, logCount, dispData, and systemStarted).

5.2. Main Program Flow and Timing Control

During setup(), the firmware starts the I2C and serial interfaces, configures the BOOT button input, enables the sensor power rail, and prints the Wi-Fi connection information. The OLED is then initialised and the welcome screen is shown, after which the access point and web server are started. Sensor initialisation is attempted next, and the SD card is initialised afterwards, with the SD result stored in dispData.sd_status.

As can be seen in loop(), the program stays in a standby path while systemStarted is false. In this stage, the system does not sample sensors or write data, and the code simply keeps serving the web interface through wifiServerLoop() while waiting for the BOOT press. Once BOOT is detected, systemStarted is set to true and the normal run path begins. It should be noted that the same transition is also provided through the web endpoint /api/system/start, which sets systemStarted directly in the server handler.wifi_server

After entering run mode, the loop executes in a fixed order. First, the BOOT key is processed by handleButton() using millis() timing, where a short press toggles logging and a long press (>2 s) clears the SD log file and updates the status string. Then GPS is polled through gpsPoll(), and the three sensors are sampled using readBME680(), readToF(), and readGPS().

The timing of recording and display refresh is controlled using millis() gates. SD logging is scheduled at 50 Hz (LOG_INTERVAL_MS = 20 ms), and each log operation builds one CSV row, writes it to the SD card, updates logCount, and updates dispData.sd_status as “Logging x/100”. When the counter reaches 100, logging is stopped automatically and the status is set to “Done”. In parallel, the OLED is updated every 500 ms, where the latest readings are copied into dispData and then rendered by updateDisplay().

5.3 Data Acquisition and Logging Pipeline

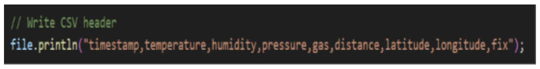
In order to keep the data path simple, the firmware treats each peripheral readout as a “single snapshot” and then builds both the display output and the log row from that snapshot. In loop(), the three modules are sampled in sequence (readBME680(), readToF(), and readGPS()), and the results are held in local structs before being copied into dispData.

For the environmental sensor, readBME680() returns temperature, humidity, pressure, and a gas-related value. It should be noted that the gas value used in the system is not the raw resistance; instead, the code maps the resistance to a relative AQI-style index (0–500) and applies exponential smoothing ($\alpha = 0.1$) before storing the result in data.gas. A fixed temperature offset (−5 °C) and a pressure unit conversion are also applied at the module level, so the main program always receives data in the expected form.

For the ToF sensor, the reading is only updated when a new measurement is ready. If no valid object is reported (or the range status is not OK), the function returns distance = -1, which is then displayed as “-- mm” on the OLED and still logged as a numeric placeholder in the CSV row.

For GPS, the parsing is handled continuously in gpsPoll(), and readGPS() simply returns the most recently parsed fields. If a fix is not available, the latitude/longitude are left as "--", and the timestamp in the log is also set to "--". When a fix is available, the timestamp is built from the GPS time fields and written in the format YYYY-MM-DD HH:MM:SS.

The SD logging format is implemented as a single CSV file (/log.csv). On initialisation, the code checks whether the file exists; if not, it creates the file and writes a header row. Each logging event then appends one comma-separated line using FILE_APPEND, which keeps the write operation short and self-contained.



```
// Write CSV header
file.println("timestamp,temperature,humidity,pressure,gas,distance,latitude,longitude,fix");
```

Figure 5.2. *CSV row construction and append write (main.cpp + storage.cpp)*

5.4. Web Dashboard and Control Interface

In order to provide a self-contained user interface without relying on external infrastructure, the ESP32 runs in SoftAP mode and serves a built-in dashboard page. The access point is configured with SSID Sensor-AP, password 12345678, and a fixed IP address 192.168.4.1. A DNS server is started with a wildcard rule, so that any hostname is redirected to the same address, and unknown paths are handled by returning the dashboard page.

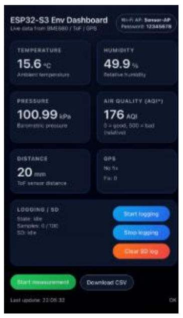


Figure 5.3. *Website page and Dashboard*

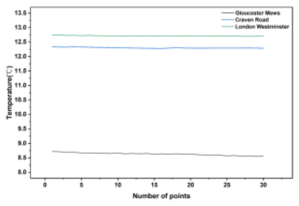
The dashboard uses a simple polling model. Every 1 s, the browser requests /api/status and updates the on-page fields (temperature, humidity, pressure, AQI*, distance, GPS fix, SD status, and the sample counter). For robustness, numeric fields are generated using a helper that converts NaN/Inf to null, which avoids JSON parsing issues on the client side.

Control actions are implemented as POST requests to a small set of endpoints. /api/system/start is used to enter measurement mode (equivalent to pressing BOOT), and logging is controlled through /api/log/start, /api/log/stop, and /api/log/clear. For data export, /download (and the alias /data) streams the CSV file to the browser with the correct content type and download headers.

6. Measurements and results of environmental parameters in different locations of London.

6.1. Analysis and visualization of our sensor data

a. Temperature and pressure were generally stable within the short sampling window, with low visible noise, indicating good short-term repeatability. Humidity at Craven Road showed a step-like change, which may be related to short-term airflow changes, vehicle-induced mixing, or local microclimate differences (e.g., shading and moisture conditions).



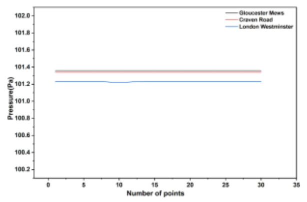
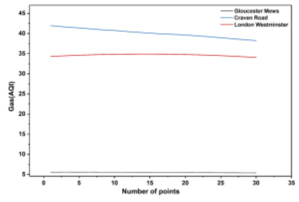


Figure 6.1, 6.2. *Temperature and Pressure results*



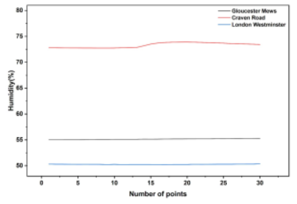


Figure 6.3, 6.4. *Gas (AQI) and Humidity results*

b. Gas (AQI) (our VOC/gas-related indicator derived from the BME680 gas channel) showed a clear and consistent spatial ranking: Craven Road highest, Marylebone Road intermediate, and Gloucester Mews lowest, indicating that the indicator can distinguish location differences.

6.2. Comparison with official data and critical analysis

a. Official website readings for the same sites were: Gloucester Mews AQI = 6 (Good), Craven Road AQI = 66 (Moderate), and Marylebone Road AQI = 46 (Good).
b. The spatial pattern matches our results: Craven Road is the worst site, Gloucester Mews is the cleanest, and Marylebone Road lies in between. This supports that our system can identify relative pollution "hotspots" relevant to exposure assessment.
c. However, Gas (AQI) is not physically equivalent to official AQI. Official AQI is derived from PM2.5/PM10 and regulated gases such as NO2 and O3, whereas the BME680 gas channel reflects a VOC-sensitive MOX resistance signal and is not a particulate sensor. Therefore, one-to-one numerical agreement should not be expected.
d. In addition, the BME680 gas sensor requires stabilization and baseline formation; the documentation recommends extended burn-in and warm-up to obtain a stable baseline. With only tens-of-seconds sampling per site, Gas (AQI) is more suitable for relative comparisons and trend judgement, while absolute values may be affected by response time, humidity/temperature coupling, and baseline drift.
e. Differences can also arise from time-window mismatch: official readings are typically averaged and updated over minutes to hours, whereas our measurements are second-level short windows; if traffic flow or wind changes rapidly, discrepancies are unavoidable.

6.5 Public-health implication

a. Both datasets indicate higher exposure at Craven Road, showing that even over short distances, different streets can significantly change personal pollution exposure, which is especially important for sensitive groups (e.g., asthma patients, people with cardiopulmonary disease, and children).
b. Combining the "adjacent-site micro-control pair" with a major-road reference site supports a practical conclusion: low-cost sensing can provide a useful relative guide for micro-environment monitoring and route/exposure awareness, but without systematic calibration and synchronized sampling it should not be presented as a direct replacement for official AQI.

7. Conclusion

The project successfully implemented the process of measuring and collecting environmental parameters (temperature, pressure, gas, humidity, distance, location) based on the ESP32-S3 microcontroller. During the PCB design phase, the features of the Adafruit ESP32-S3 board were taken advantage of and I2C, SPI and UART protocols were used for different sensors and components (GPS, BME680, VL53L4CX, OLED and SD card).

The use of GND on the bottom layer of the PCB board was an important step to improve signal quality and suppress electromagnetic waves. Placing the environmental sensor away from the controller's heat source helped to increase the accuracy of the measurements.

Significant optimization was made in the mechanical design section, resulting in a screw-free, two-sided snap-fit enclosure design.

8. Reference

[1] https://aqicn.org/map/london/cn/
[2] https://cdn-learn.adafruit.com/downloads/pdf/adafruit-bme680-humidity-temperature-barometic-pressure-voc gas.pdf
[3] https://cdn-learn.adafruit.com/downloads/pdf/adafruit-micro-sd-breakout-board-card-tutorial.pdf
[4] https://cdn-learn.adafruit.com/downloads/pdf/adafruit-vl53l4cx-time-of-flight-distance-sensor.pdf
[5] https://cdn-learn.adafruit.com/downloads/pdf/monochrome-oled-breakouts.pdf
[6] https://cdn-learn.adafruit.com/downloads/pdf/adafruit-esp32-s3-feather.pdf
[7] adafruit-mini-gps-pa1010d-module%20(1).pdf