Co-funded by
the European Union

Report on the implementation of the exercise:
Lab #4: IoT system

CONN’COR, Project No. 2024-1-FR01-KA220-HED-000250882
WP3: Development of new and modernization of existing selected courses as examples of teachers’ collaboration
Activity 3.2: Development of the course Home Compatible Labworks

Lab work duration: 3h00

Implementation date: 02.07.2025

Name of the students:

- Łukasz Forenc (second semester, Automatic Control and Robotics).

Supervisor of this project:

- Assist. Prof. Jarosław Forenc, PhD Eng.

1. Project Objective

The objective of the students’ task is to design and implement a complete IoT system for monitoring CO_2 levels using an ESP32 microcontroller and a CO_2 sensor. Through this project, students are expected to learn how to collect environmental data, process it, and transmit it both locally - using Bluetooth and LED indicators - and remotely via Wi-Fi to the Adafruit IO platform. They should begin by analyzing the provided code, which can serve as a foundation or reference, and then assemble the hardware according to the wiring diagrams, ensuring proper integration of all components. While they are not required to write the entire program from scratch, they should be able to understand, adapt, and improve the existing code, including adding support for additional sensors. Furthermore, the project encourages creativity and extension by collaborating with programming instructors to develop web or Android applications that provide user-friendly interfaces for visualizing CO_2 data. This approach offers students a comprehensive end-to-end IoT experience, from data acquisition to real-time monitoring and visualization across multiple platforms.

2. Project Implementation

The project was conducted by a second-semester student in Automatic Control and Robotics. The student was provided with the following instruments and components:

- ESP32 dev module,
- SenseAir S8 sensor,
- LEDs, connecting wires,
- a breadboard.

3. Student Comments

1. Analysis of the supplied examples

In general, all of the supplied examples follow a similar structure. The CO_2 value is read from the sensor and communicated to the user in various ways.

1.1. Communication with the sensor

All of the supplied examples reuse the same code for communicating with the sensor. The code makes use of the Serial interface directly. This approach is not recommended, as it adds an additional layer of complexity to an otherwise simple task of reading an integer from the device. A better approach would be to use of one of the community-created libraries, such as S8_UART. Using a pre-made library abstracts the sensor interface and lets the programmer focus on their task instead of debugging the communication layer.

1.2. esp32RGB

In this example 4 LEDs and a buzzer are used to communicate the CO_2 levels to the user. This approach creates a simple interface, letting the user know the air quality without them having to understand the meaning of the PPM value. The buzzer is also used to alert the user that immediate action is required when CO_2 levels reach dangerous thresholds. For more verbose interface BLE is used. A BLE server mimicking UART connection is created. Using a smartphone app user is able to read the actual CO_2 level in PPM.

1.3. ESP S8 BLUETOOTH

It’s essentially the same as the esp32RGB example.

1.4. wifi co2

This example makes use of the MQTT protocol to upload the measurement results to the Adafruit IO cloud in addition to the LED and buzzer functionality found in the esp32RGB example. Using the Adafruit cloud enables easy integration with various systems and automations, and allows the user to access measurement results from anywhere in the world, not just on the local network.

1.5 ttgo

This example makes use of the TFT display found on the Lilygo T-Display board. The measurement results are displayed on the TFT display both as a simple and easily readable quality score, and also a verbose PPM result. This example also allows for manual sensor auto-calibration with a press of the button.

1.6 Final Note

Although functional, the supplied examples appear disorganized. Two of the examples are essentially same thing. No clear coding style is followed. Within the same sketch multiple naming conventions are used, there are mismatched indentation depths, some variable names are clear and descriptive and others are vague or sometimes even misleading. Following a clear coding style makes the program easier to read and understand, while also helping students learn more from the supplied examples.

2. Created Firmware

2.1. Features

Newly created firmware contains various features taken from the supplied examples as well as newly developed ones. Features include:

- Measurement of CO_2 levels.
- Displaying the result on the LEDs.

- High CO_2 levels warning with a buzzer.
- Displaying the result of the measurement on the TFT display.
- Displaying the result on the website hosted on the device.
- Ability to set the WiFi credentials from the hosted website.
- Simple REST API allowing for easy reading of the current CO_2 levels in the LAN network.
- High CO_2 levels warning sent over Discord messaging app.

2.2. Explanation of the code

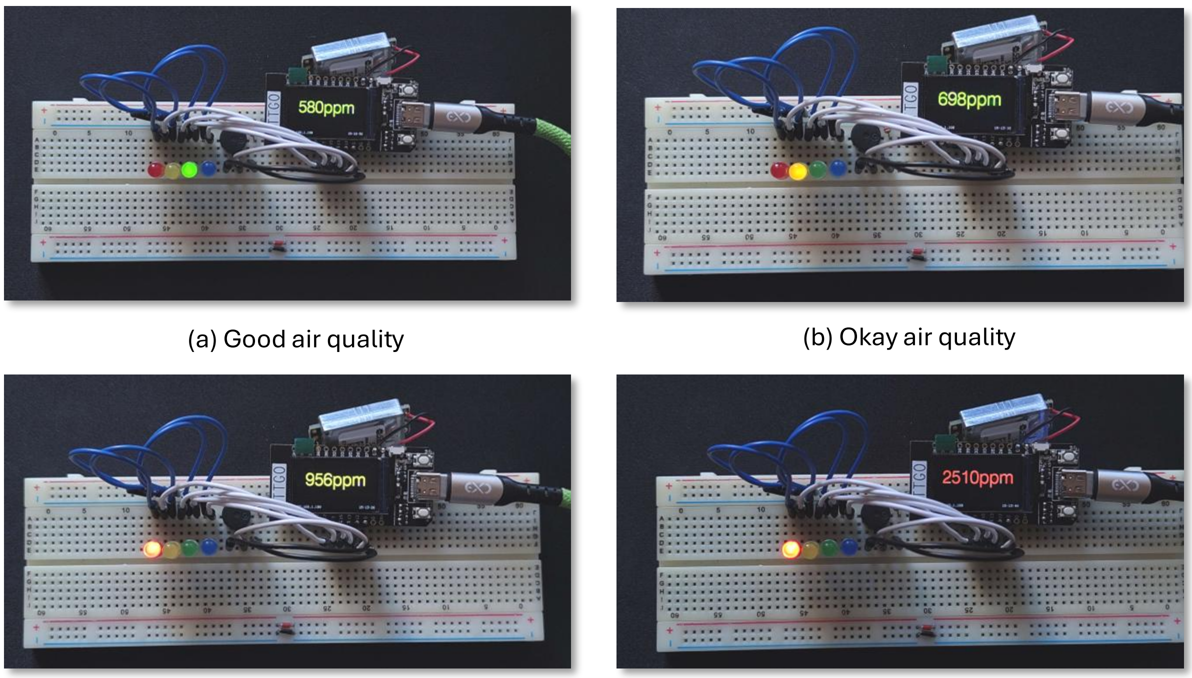
2.2.1. Sensor interface

A simple sensor interface library S8_UART is used. The library handles all of the underlying communication and exposes a simple interface for taking measurements with the get_co2 function.

2.2.2. LEDs and Buzzer

Depending on the air quality different LEDs light up:

- $CO_2 < 450$ ppm - Blue LED,
- 450 ppm $\leq CO_2 <$ 600 ppm - Green LED,
- 600 ppm $\leq CO_2 <$ 800 ppm - Yellow LED,
- 800 ppm $\leq CO_2 <$ 1000 ppm - Red LED,
- $CO_2 >$ 1000 ppm - Red LED blinking, Buzzer alarm turned on.



(a) Good air quality (b) Okay air quality

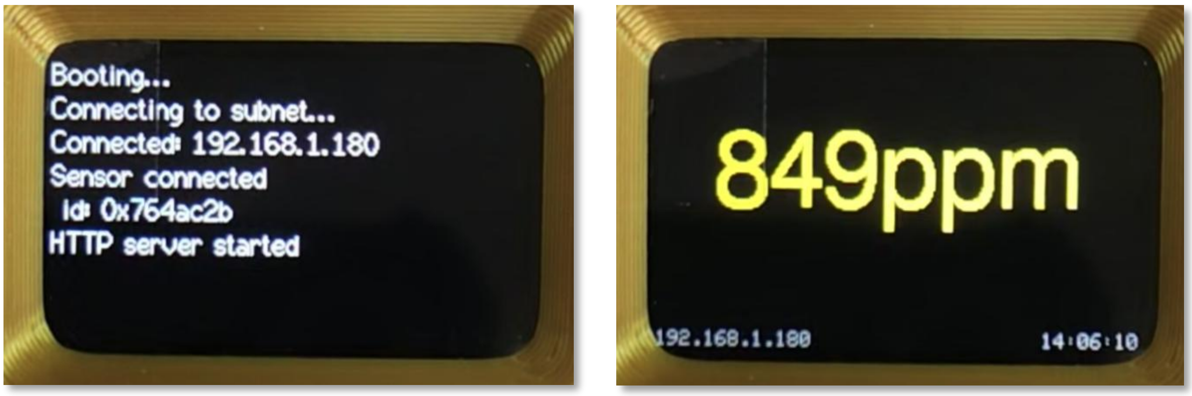
(c) Bad air quality (d) Dangerous air quality

Fig. 1. Measurements displayed on the LEDs

2.2.3 TFT display

Co-funded by
the European Union

During the boot process, logs are shown on the display, and any errors are displayed as well. While in operation the current reading is displayed along with current time and an IP address in the network.



```
Booting...
Connecting to subnet...
Connected: 192.168.1.180
Sensor connected
 Id: 0x764ac2b
HTTP server started
```

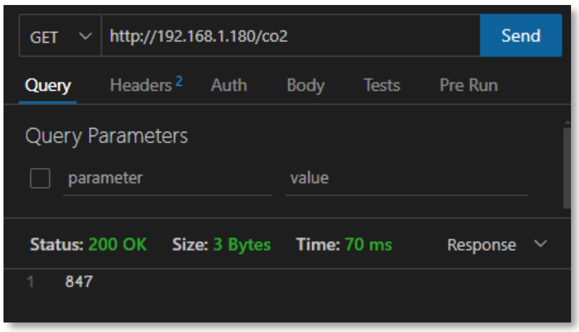
```
849ppm
192.168.1.180                14:06:10
```

(a) During boot (b) While in operation

Fig. 2. TFT display

2.2.4 REST API

A very simple API can be used to access the current reading. Sending a GET request to the /co2 endpoint on the web server returns an integer representing the current reading in PPM. This API is used to get the measurement results and display them on the hosted website.

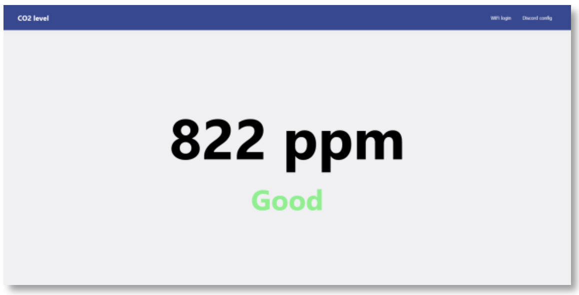


```
GET  http://192.168.1.180/co2
Status: 200 OK   Size: 3 Bytes   Time: 70 ms
847
```

Fig. 3. API request

2.2.5 Web Interface

A simple measurement dashboard is hosted on the ESP32's web server. The user can read the current measurement both in PPM and in a simple ”good or bad air quality” format and change the device configuration.



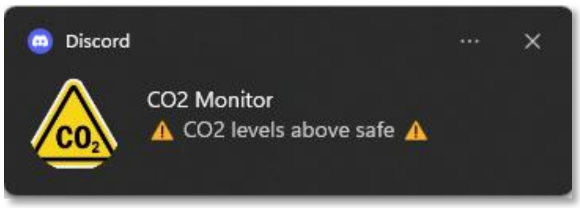
822 ppm

Good

Fig. 4. Web Interface

2.2.6 Discord Notifications

Discord's simple bot API is used to send a notification to the user. If the CO_2 levels go above a set threshold, a notification is sent to the user. In the configuration settings available on the web server the user can set the API token and the channel in which the notification is to be sent.


Discord

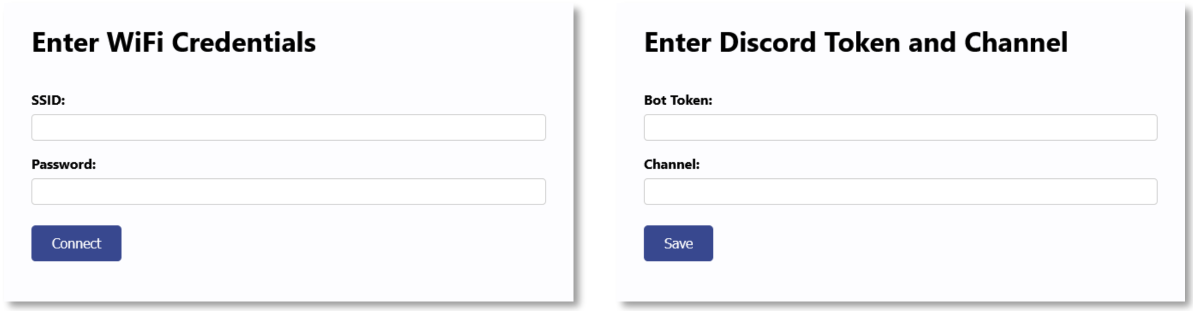
CO2 Monitor

⚠ CO2 levels above safe ⚠

Fig. 5. Discord Notification

2.2.7 Device Configuration

Through the web server the user is able to set the WiFi network credentials for the device to connect to. If the credentials are not set or the device is unable to connect to the set network, an access point is created where the user is able to access the web server and alter the settings. In this configuration the user is also able to change the discord notification settings. The settings are saved to the device flash memory using the LittleFS file system library.


Enter WiFi Credentials

SSID:

Password:

Connect

(a) WiFi credentials

Enter Discord Token and Channel

Bot Token:

Channel:

Save

(b) Discord notification

Fig. 6. Configuration Menus

2.3 Problems while creating the firmware

2.3.1 Direct sensor interface

In the given examples the sketch interfaces directly with the sensor, everything including setting up the sensor and calculating CRC is done in the sketch. Like with many popular sensors, Arduino community has already created a library abstracting the inner workings of the interface and simplifying the program. The library used in this case is S8_UART.

2.3.2 TFT library configuration

The used TFT library TFT_eSPI requires an additional configuration file to be edited. This was not mentioned in any of the given examples. No error or warning hinting at the config being wrong was shown. This led to problem, where the display would not display anything for unknown reason.

Co-funded by the European Union

2.3.3 Web Files

Because the ESP32 does not have an easily accessible file system, the files hosted on the web server must be converted to C strings and included in the source code. A simple python tool was used to do that. There are probably more sophisticated ways of doing that, like for example creating a LittleFS file system image, but they require more advanced tools.

2.3.4 Small system memory

With the given hardware it was impossible to fit both WiFi and BLE servers. When both were included the sketch used 130% of the system memory.