Air Quality Monitor

1st Antonio Ribeiro
MEEC, FEUP
up202302879

2nd Miguel Gouveia
MEEC,FEUP
up202302221

3rd Tiago Duarte Correia de Oliveira
MEEC, FEUP
up202005936

***Abstract*—This paper presents the development of an air quality monitoring system created during the SEMB course. The platform utilized for the project was the Arduino UNO. The scheduling follows a First Come First Served (FCFS) approach, with a non-preemptive microkernel implemented to meet the necessary timing requirements. In addition, a comparative analysis was conducted to validate the correct operation and performance of the air quality monitoring system.**

***Index Terms*—Air quality monitor, Arduino UNO, non-preemptive**

I. Introduction

The objective of this project is to implement an Air quality monitor with real-time response. Our program is designed for ideal use within a home or a specific room, providing critical environmental information. It monitors key parameters, including temperature, humidity, air quality, and methane levels, offering a comprehensive overview of the environmental conditions in the area where it is implemented. This localized approach ensures targeted and efficient data collection tailored to the specific indoor environment.

The system is intended to provide efficient user navigation over pages presenting important parameters. Users are directed to a main page that displays a list of available parameter pages and their respective page numbers. The user then selects a particular page by clicking the corresponding number in the keypad, which takes them directly to that page. The necessary information is then clearly presented on the screen. For added convenience, users can continue navigating to other parameter pages by selecting their respective numbers. If a user forgets the page number, they can press any key not listed on the main page to return to the main page, where they can review the list of available pages and their numbers. This intuitive design ensures ease of use and quick access to the required information.

This system requires the following items: an Arduino UNO, an OLED display, four sensors (temperature, humidity, methane, and air quality sensors), and a keypad, presented in Fig. 5.

II. Software Architecture

A non-preemptive system was used alongside a periodic system. This implies that activities will be conducted until completion with no interruptions from higher priority. Tasks will be executed at specified intervals and in the predetermined order, keeping the system simple and deterministic.

A. Tasks

Five tasks were created for this system to run properly. The first three tasks concern the sensors utilized in this system. The first task involves reading data from the DHT11 sensor, which measures temperature and humidity. The task is assigned a deadline of 5 seconds, considering that the theoretical data update intervals for temperature and humidity are 10 seconds and 6 seconds, respectively. Since the sensor provides dual measurements, a compromise of 5 seconds was implemented for efficiency. The received data is displayed on page 1, showing the temperature and humidity values, illustrated in Fig. 2.

The second task focuses on the air quality sensor, which operates at a very fast speed due to its use of I2C communication, allowing a maximum frequency of 1 MHz (1 microsecond per transaction). To leverage this speed, the task has a deadline of 2 seconds, ensuring timely updates. This season has a high sensitivity to ammonia gas, sulfide, and benzene steam, and uses the concentration of these to conclude the air quality measurement. The data from this sensor are displayed on page 2, along with the readings of the methane sensor, demonstrated in Fig. 3.

The third task involves the methane sensor, which detects methane levels ranging from 200 ppm to 10,000 ppm with high accuracy. Similar to the air quality sensor, it operates quickly, and the deadline was also set to 2 seconds. As previously mentioned, the methane sensor's readings are displayed on page 2, and shown in Fig. 3.

The fourth task addresses the keypad, which serves as the primary user interface. To ensure smooth interaction, the task has a short deadline of 90 ms, enabling continuous monitoring of user input. The keypad provides the following functionality:

1) **Pressing 1** switches the display to page 1 (Fig. 2).
2) **Pressing 2** switches the display to page 2 (Fig. 3).
3) **Pressing any other key** returns the system to the main page, which displays page navigation instructions (Fig. 1).

Frequent polling allows for smooth and responsive interaction with the user.

Lastly, the final task is responsible for updating the OLED display (0.96-inch). The display shows sensor information across three pages:

1) **Main Page:** Lists the available pages and their corresponding page numbers, shown in Fig. 1.
2) **Page 1:** Displays temperature and humidity data, presented in Fig. 2.
3) **Page 2:** Displays methane and air quality data, illustrated in Fig. 3.

The deadline for the display was set to 250 ms, which is approximately three times the period of the keypad. This allows for greater stability by ensuring that if multiple keypad inputs are made in quick succession (up to three in a row) only the last input is processed.

Fig. 1. Main page.

Fig. 2. First page.

Fig. 3. Second page.

B. Shared Resources

Various tasks needed to share information during this process, and to achieve this global variables were used. These global variables stored values related to the sensor measurements, as well as the information on the selected page. Initially, the tasks regarding reading the sensors' information are executed. This ensures that the display always has the information to work with and is up to date. Due to the system being non-preemptive, the sensor tasks update these variables periodically based on a predefined interval, ensuring accurate data for the display.

III. Quality of the Operation

To assess the quality of operation of the air quality monitoring system, scheduling time measurements were taken using a logic analyzer. As for memory footprint, the Arduino Integrated Development Platform (IDE) gave the necessary values on compile and upload time as output. Overall, the time measurements were according to the expected theoretical values and the memory footprint was relatively conservative. According to the initial proposal, the objectives of this project were met, with a CPU utilization of approximately 23.18%.

A. Time Measurements and Scheduling

The scheduler runs with a time tick of 10 ms, meaning the minimum period and/or delay a task can have is 10 ms. Every value chosen for the periods of the tasks will have to be a multiple of 10.

The logic analyzer was kept running and capturing signals representative of the task's execution states for about 8 minutes. A small section of that capture can be visualized as an example in Fig. 4. Using the resulting comma-separated value (CSV) file, it was possible to determine the experimental periods and execution times of the five tasks, producing the histograms displayed through Fig. 6 to Fig. 16 in Appendix A. With that data, the jitter of each task was calculated. Table I presents the task's Defined Periods (DP), Worst Case Execution Times (WCET), and Jitter, all in milliseconds.

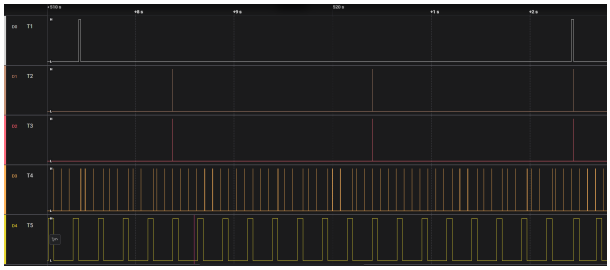

Fig. 4. Small section of logic analyzer capture.

TABLE I
Task Defined and Average Measured Periods, Worst Case Execution Times and Jitter

Task	DP (ms)	WCET (ms)	Jitter (ms)
1	5000	22.089	2470.008
2	2000	0.124	1596.734
3	2000	0.121	1596.987
4	90	0.442	138.688
5	250	55.588	47.193

B. Memory Footprint

Regarding the memory footprint of the program, the Arduino IDE produces an output after completing compiling and uploading. From this source, our program is recorded to use 17424 bytes (54%) of the available storage space.

Furthermore, the global variables used to keep the information on the sensors' readings and current page use 846 bytes (41%) of dynamic memory, leaving 1202 bytes for local variables (the maximum is 2048 bytes).

IV. Complexity

This project operates with a simplistic scheduling. The First Come First Served (FCFS) scheduling strategy consists of queuing task instances upon arrival. The queued task is always given a priority lower than that of all others already in the queue. Also, the priority of ready and executing tasks increases equally for all as time passes. This scheduling strategy has the advantage of causing lower overhead and being easy to implement when compared to other scheduling strategies. On the other hand, it is a non-optimal scheduling that easily causes deadline misses.

As a means to address the inherent problems with deadline misses, an in-depth analysis of the task parameters was conducted, specifically regarding their periods and offsets. This analysis followed the rationale taught during the SEMB course and had its foundation in the explanations presented. By carefully capturing measurements, such as the execution times, and validating these measurements with the theoretical ones expected, it is possible to set periods and deadlines that respect the worst execution times of the other tasks, ensuring that even in the worst-case scenario of a task being the last in the queue, its deadline is safely met.

V. Conclusion

The initial project proposal for a concise weather station focusing on air quality monitoring was successfully achieved. In addition, the monitoring of temperature and humidity is integrated, as well as methane monitoring which was not included in the initial proposal. Furthermore, the initial blueprint for tasks was remodeled, resulting in the exclusion of the buzzer task and the inclusion of a keypad instead of the previous button. In conclusion, the achieved project works as intended, with a CPU utilization of approximately 23.18%, and tested concerning its behavior and responsiveness.

APPENDIX A

TABLE II
GRADING FOR EACH GROUP MEMBER

Name	Percentage
Antonio Ribeiro	33.3%
Miguel Gouveia	33.3%
Tiago Oliveira	33.3%

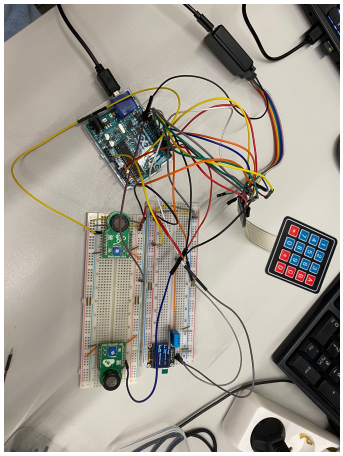

Fig. 5. Project setup.

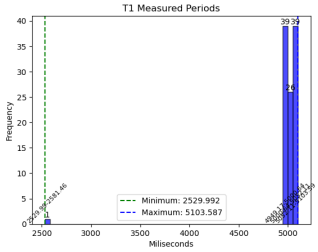

Fig. 6. Task 1 measured periods.

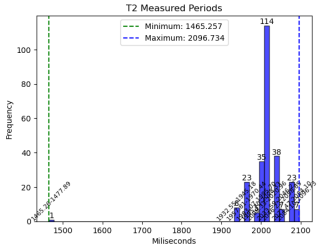

Fig. 7. Task 2 measured periods.

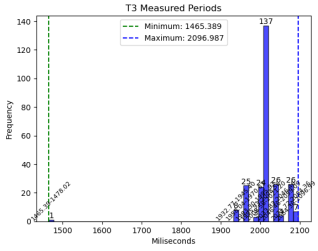

Fig. 8. Task 3 measured periods.

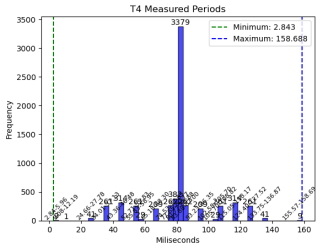

Fig. 9. Task 4 measured periods.

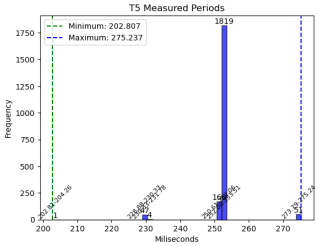

Fig. 10. Task 5 measured periods.

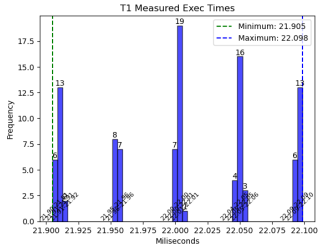

Fig. 11. Task 1 measured execution times.

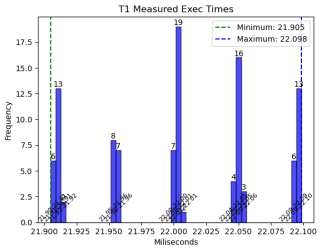

Fig. 12. Task 1 measured execution times.

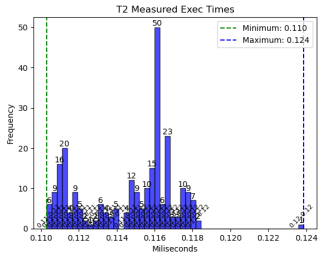

Fig. 13. Task 2 measured execution times.

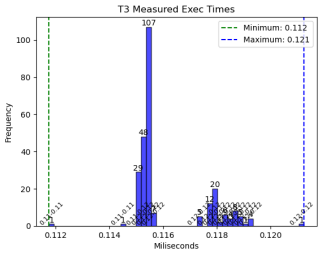

Fig. 14. Task 3 measured execution times.

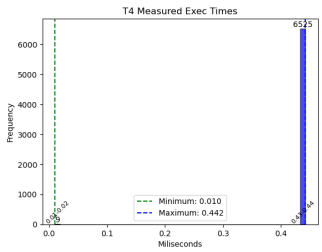

Fig. 15. Task 4 measured execution times.

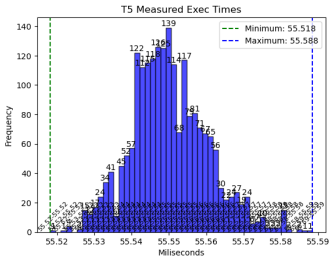

Fig. 16. Task 5 measured execution times.