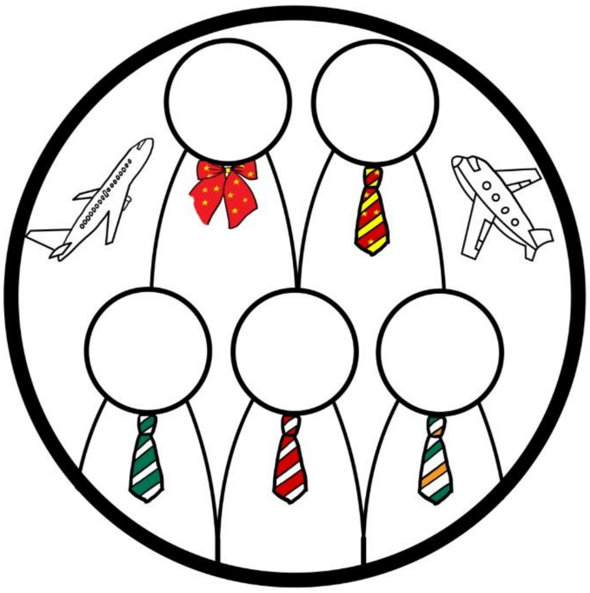
UKXY

Electronic/Signal Project（APP）

Final Report



Group 1：

LIU Yang

Petal Ketul

GUO Xiaofan

KOVAYCIN Umut

JOHN ITOPA ISAH

25/01/2024

Content

1. Summary

The advent of the UKXY3000 Aircraft Cabin Environmental Monitoring and Noise Reduction System marks a significant stride in the aviation industry's ongoing quest to enhance the in-flight experience. This state-of-the-art system is engineered with precision to offer real-time surveillance and management of the aircraft cabin's environmental conditions, thus setting a new benchmark for passenger comfort and safety.

Foremost in its design is the system's ability to continuously assess and adjust cabin temperature, humidity, and air quality, ensuring that passengers are enveloped in a cocoon of comfort. Through the integration of advanced sensors and intelligent algorithms, the UKXY3000 maintains an atmosphere that is meticulously calibrated to the ideal conditions for human occupancy. These sensors detect fluctuations in temperature, CO2 levels, and cabin pressure, and automatically initiate corrective measures to sustain a harmonious balance.

Additionally, the UKXY3000 boasts an innovative noise reduction module that operates at the forefront of acoustic technology. This module employs active noise control techniques to significantly dampen the intrusion of engine roar and aerodynamic noise, thereby curating a serene environment where the hustle of the external world fades into the background. The quietude provided by the system not only enhances passenger relaxation but also serves to protect the hearing of both passengers and crew over long-haul flights.

Beyond its core functionalities, the UKXY3000 integrates seamlessly with the aircraft's existing systems, providing pilots and cabin crew with intuitive interfaces to monitor environmental parameters and adjust as necessary. This proactive approach to cabin management ensures that any deviations from the optimal environment are promptly addressed, thus preserving the well-being and satisfaction of all on board.

As the inaugural element of our project, the UKXY3000 stands as a testament to our dedication to advancing aerospace technology. It encapsulates our vision of a future where air travel transcends mere transportation, offering an experience that is as restful as it is secure. Through meticulous research, collaborative engineering, and an unwavering commitment to innovation, we have forged a path to redefine the standards of in-flight comfort and safety.

2. Project Context

As we chart new horizons in the aviation sector, the Aircraft Cabin Environmental Monitoring and Noise Reduction System emerges as a beacon of innovation in our relentless pursuit of enhancing in-flight comfort and safety. This ambitious project is set against a backdrop of a dynamic industry landscape, where each facet underscores the criticality of our mission.

1) Industry Imperatives

The aviation industry is at a pivotal juncture, marked by a renaissance in aircraft design centred on human-centric considerations. Our project is conceived in an era where technological breakthroughs and passenger expectations converge to redefine what it means to fly. Comfort, once a luxury, is now a necessity, with health and safety being non-negotiable benchmarks that drive design philosophies. The Aircraft Cabin Environmental Monitoring and Noise Reduction System is our answer to these imperatives, embodying an integration of sophisticated sensor technology and advanced acoustic engineering.

2) Market Dynamics

The consumer zeitgeist speaks volumes about the shift towards a more nuanced flying experience. Passengers are increasingly informed and vocal about their preferences, seeking an oasis of tranquillity amidst their travels. Airlines, in response, are not just carriers but curators of experience, looking to distinguish themselves by the quality of their in-flight environments. This system is a strategic investment in customer satisfaction, intended to serve as a differentiator in an airline’s service portfolio, transforming the cabin into a sanctuary of peace and wellness.

3) Regulatory Convergence

Navigating the complex web of aviation regulations is a testament to our commitment to compliance and safety. With the regulatory landscape as a guiding star, our system is designed not just to align with but to transcend global standards. Our proactive stance ensures that as regulations evolve, so too will our system, future-proofing it against the inevitable tightening of standards and safeguarding passengers and crew against even the most subtle environmental adversities.

4) Competitive Horizon:

In the theatre of market competition, innovation is the currency of success. With several players vying for supremacy in the cabin environment space, the uniqueness of our system lies in its integration, intelligence, and interface. It is an ecosystem unto itself, delivering not just a service but an experience. By benchmarking against existing technologies and anticipating future advancements, our project is poised to claim a vanguard position, carving out a niche in a market ripe for disruption.

In this context, the Aircraft Cabin Environmental Monitoring and Noise Reduction System is not just a technological project; it is a vision made manifest. It is the culmination of exhaustive research, collaborative synergy, and a forward-thinking approach. With an eye on the legacy of aviation and a hand steering towards the future, this project is a testament to our dedication to excellence and our promise to elevate the human journey through the skies.

3. Objectives

3.1 Overall Objective

Within the dynamic confines of an aircraft cabin, the UKXY3000 system represents a transformative leap in environmental control and passenger comfort. This intricate network of sensors and controls brings the future of in-flight experience into the present, weaving together a tapestry of technology that caters to the nuanced needs of both passengers and crew.

3.2 Specific Objectives

At the heart of UKXY3000 lies a suite of meticulously engineered functionalities, each serving a pivotal role in harmonizing the cabin atmosphere:

1) **Carbon Dioxide Control**

   Leveraging high-fidelity sensors, the system maintains vigilant oversight of CO2 levels, ensuring air quality remains within healthful bounds, thus safeguarding passenger well-being and alertness.

2) **Temperature Control**

   Through a sophisticated climate control interface, UKXY3000 administers the cabin temperature with surgical precision, creating an enveloping warmth or a refreshing coolness to suit the preferences of those aboard.

3) **Real-time Alerts**

   With swiftness and accuracy, the system communicates with the flight crew, providing critical updates should environmental parameters stray from their predefined sanctuaries, enabling prompt and proactive adjustments.

4) **Active Noise Cancellation**:

   Harnessing the latest in noise cancellation technology, the system actively neutralizes the invasive hum of flight, transforming the cabin into a bastion of silence.

5) **Vibration Dampening**

   The UKXY3000 employs advanced materials and vibration dampening techniques to smother the tremors of travel, thus muting the tactile echoes of the aircraft's operation.

6) **Engine Noise Suppression**

   Targeted acoustic strategies are deployed to insulate the cabin from the roar of the engines, ensuring that the only rumble passengers might perceive is that of their contented sighs.

7) **Enhanced Passenger Experience**

   The confluence of these features culminates in an unparalleled in-flight experience, where tranquillity and comfort reign supreme, leaving passengers disembarking as refreshed as when they embarked.

3.3 Hypotheses

To underpin the UKXY3000's operational efficacy, several hypotheses are posited:

- The seamless interplay between collection, transmission, and notification components will result in an unobtrusive system that enhances environmental awareness without overwhelming the user.
- Real-time data transmission via Bluetooth to passengers' mobile devices will not only inform but also empower passengers with the ability to monitor their immediate surroundings.
- Prompt alert systems will enable flight crews to adjust environmental controls pre-emptively, mitigating potential discomforts and maintaining an optimal cabin environment.
- The integration of active noise cancellation and vibration dampening will significantly reduce the stress of travel, thus fostering a more enjoyable and restful journey.

Through these innovations, the UKXY3000 will set a new industry standard for in-flight environmental management, contributing to heightened customer loyalty and a competitive edge in the market.

The UKXY3000 is not just a system; it is a guardian of comfort, a sentinel of tranquility, and a herald of the future of air travel.

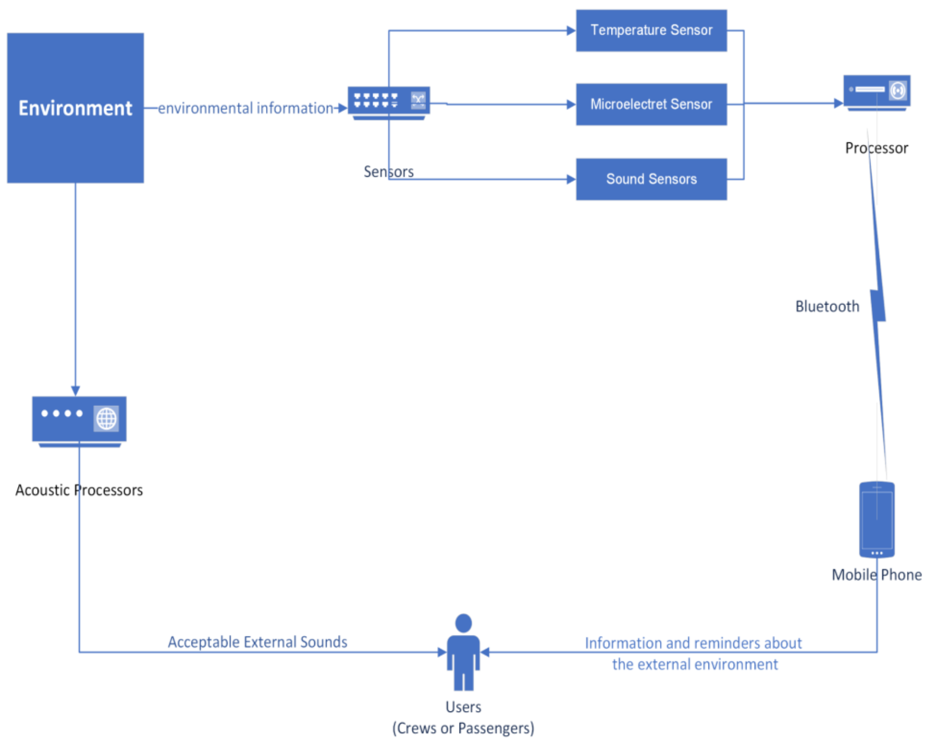


Figure 1 System Workflow.

4. Methodology

4.1 Research Design

This project is made of a closed loop system and displays aircraft compartment environmental indexes on the phone which include Temperature, Carbon Dioxide and Sound.

4.2 Data Collection

The compartment environmental data including temperature, CO_2 , sound will be collected via equipment deployed in the aircraft compartment.

4.3 Data Analysis

The collected data will be stored in the microcontroller. The temperature and CO_2 will transmit to users via Bluetooth immediately.

The data of sound will be filtered. The sound of the compartment speaker and emergency alarm will be tagged green which means passengers should notice. Otherwise, the other source of sound like passenger’s communication, engines running or airflow impact will be defined if those sources of sound exceed the limit. If the sound level exceeds the defined value, the microcontroller will give instructions to passengers to wear hearing protector if possible and send feedback to flight crews at the meantime which they may need to either reduce the thrust to reduce the noisy level or check the compartment if there is any argument.

4.4 Tools and Instruments:

The tools mainly include sensors and electronic boards. But this is the least, required tools may be added later in the process and will be listed in this section.

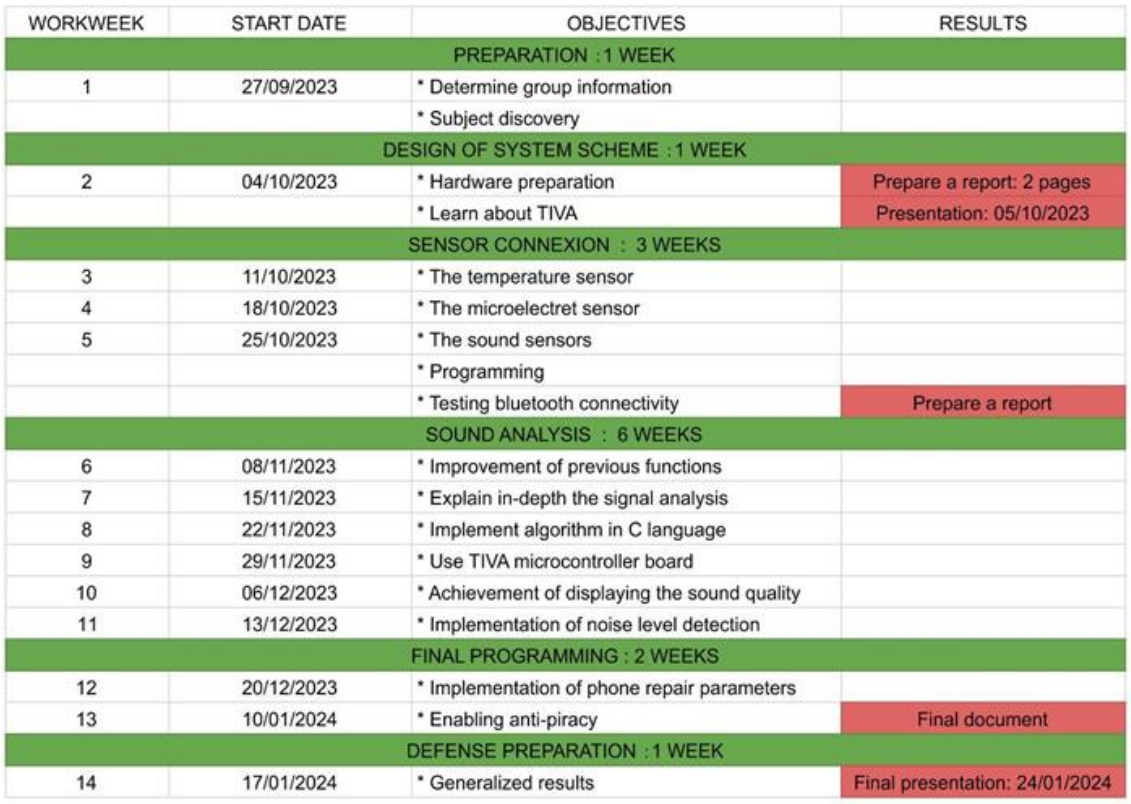
Table 1 Sensors

LM35 (Temperature)
MiCS-VZ-98-TE (CO2)
AMB-707-RC (Microphone-Omni)

4.5 Ethical Considerations

This project is to detect the environmental quality of aircraft compartments and give instructions to passengers and feedback to flight crews. The collected data shall not be leaked and may be encrypted so as aircraft's flight status will not be disclosed for which may cause potential risks. And the data of sound will only be kept during each flight in the microcontroller then will be erased since voiceprint could be a personal account password in nearly years.

4.6 First Timeframe



WORKWEEK	START DATE	OBJECTIVES	RESULTS
		PREPARATION :1 WEEK	
1	27/09/2023	* Determine group information	
		* Subject discovery	
		DESIGN OF SYSTEM SCHEME :1 WEEK	
2	04/10/2023	* Hardware preparation	Prepare a report: 2 pages
		* Learn about TIVA	Presentation: 05/10/2023
		SENSOR CONNEXION : 3 WEEKS	
3	11/10/2023	* The temperature sensor	
4	18/10/2023	* The microelectret sensor	
5	25/10/2023	* The sound sensors	
		* Programming	
		* Testing bluetooth connectivity	Prepare a report
		SOUND ANALYSIS : 6 WEEKS	
6	08/11/2023	* Improvement of previous functions	
7	15/11/2023	* Explain in-depth the signal analysis	
8	22/11/2023	* Implement algorithm in C language	
9	29/11/2023	* Use TIVA microcontroller board	
10	06/12/2023	* Achievement of displaying the sound quality	
11	13/12/2023	* Implementation of noise level detection	
		FINAL PROGRAMMING : 2 WEEKS	
12	20/12/2023	* Implementation of phone repair parameters	
13	10/01/2024	* Enabling anti-piracy	Final document
		DEFENSE PREPARATION :1 WEEK	
14	17/01/2024	* Generalized results	Final presentation: 24/01/2024

Figure 2 Timeframe.

5. Realization

5.1 Mission 1

1) What the V cycle is?

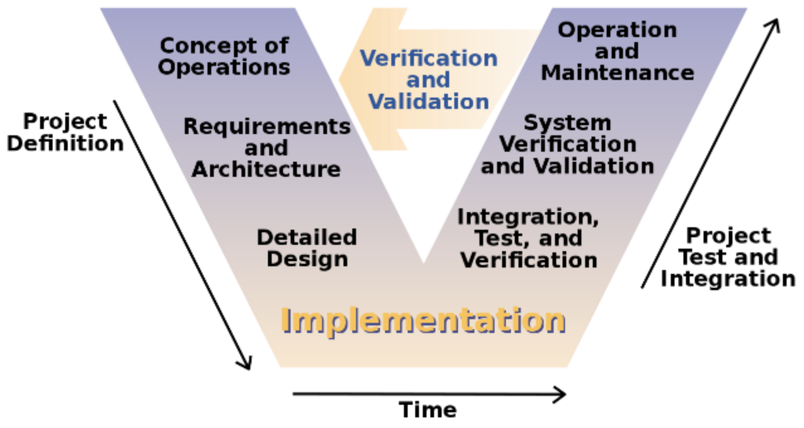

Concept of Operations
Verification and Validation
Operation and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Implementation
Time

Figure 3 V Cycle [1].

- **Concept of Operations**

  Determine the overall concept and operational framework of the project (understand the goals, scope and customer needs and expectations).
- **Requirements and Architecture**

  Determine specific requirements and define the architecture (functions, performance, interfaces, etc.).
- **Detailed Design**

  Create more detailed design documents that describe how software components implement requirements and architecture.
- **Implementation**

  The actual coding and building software phase. Write code based on design documents and ensure the code meets design requirements.
- **Integration, Test, and Verification**

  Individually developed software modules are integrated together and integration tested to ensure that the whole works properly and meets requirements and design standards.
- **System Verification and Validation**

  System-level testing to ensure that the entire software system meets user requirements.
- **Operation and Maintenance**

  The software will undergo maintenance to fix any issues, as well as make updates and improvements.
- **Verification and Validation**

  Throughout the entire V Cycle stages. After each development phase, corresponding verification and verification are carried out to ensure that the project results meet the requirements defined in the earlier phases.

2) Connect a tricolour LED on ports of the microcontroller and generate the following colours: BLUE – RED – GREEN – YELLOW.

- In the function "**setup ()**" used to initialize settings, especially setting pin mode.
- "**pinMode(PIN_NUM, OUTPUT);**" set digital pin PIN_NUM to output mode.
- For the function "**loop ()**" will be executed repeatedly.
- "**digitalWrite(PIN_NUM, HIGH);**" sets the level of pin PIN_NUM to high and the LED lights up;

  "**digitalWrite(PIN_NUM, LOW);**" sets the level of pin PIN_NUM to low and the LED lights off.
- In this project,

PIN_NUM	LED COLOR
33	RED
34	BLUE
35	GREEN

a) Measure the current inside the LED when they are ON and explain it.

- Blue LED (3.8 mA)

Generating blue light requires higher energy and therefore a higher forward voltage. However, due to the high efficiency of blue LEDs, the excitation current required may not be very high.

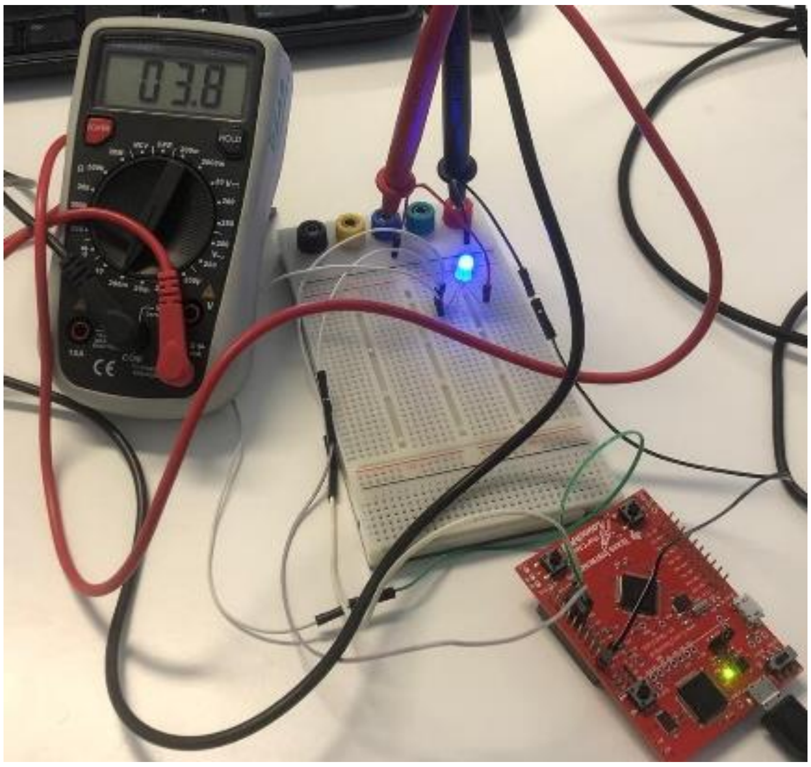


Figure 4 Blue LED turns on.

- Red LED (10.0 mA)

Since red light has lower energy, its forward voltage is correspondingly lower. However, to obtain sufficient brightness, a larger current may be required.

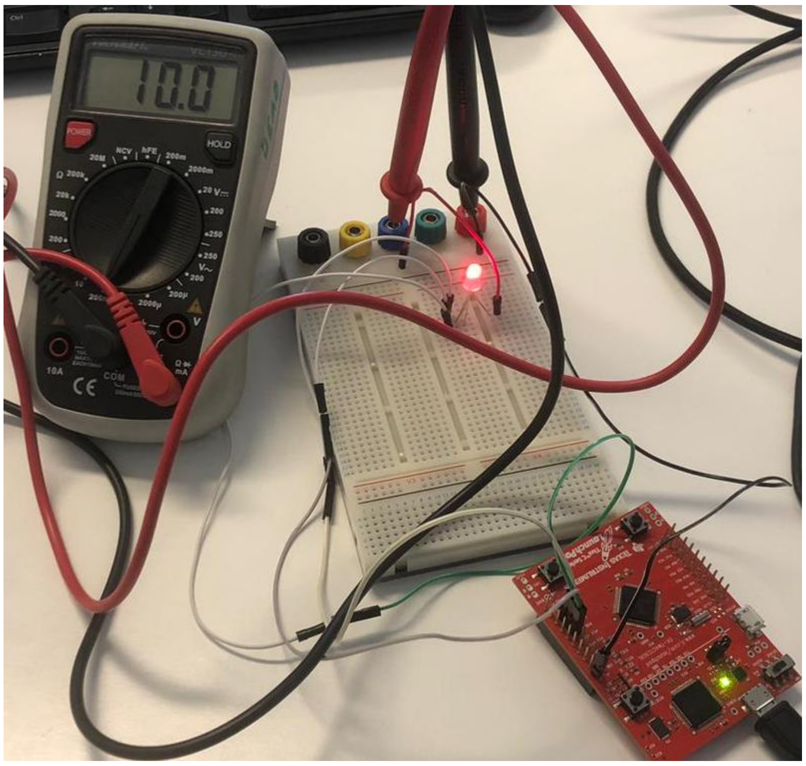


Figure 5 Red LED turns on.

- Green LED (5.1 mA)

Green light has less energy than blue light, but usually more energy than red light. Therefore, the excitation current of green LED is between that of blue and red LED.

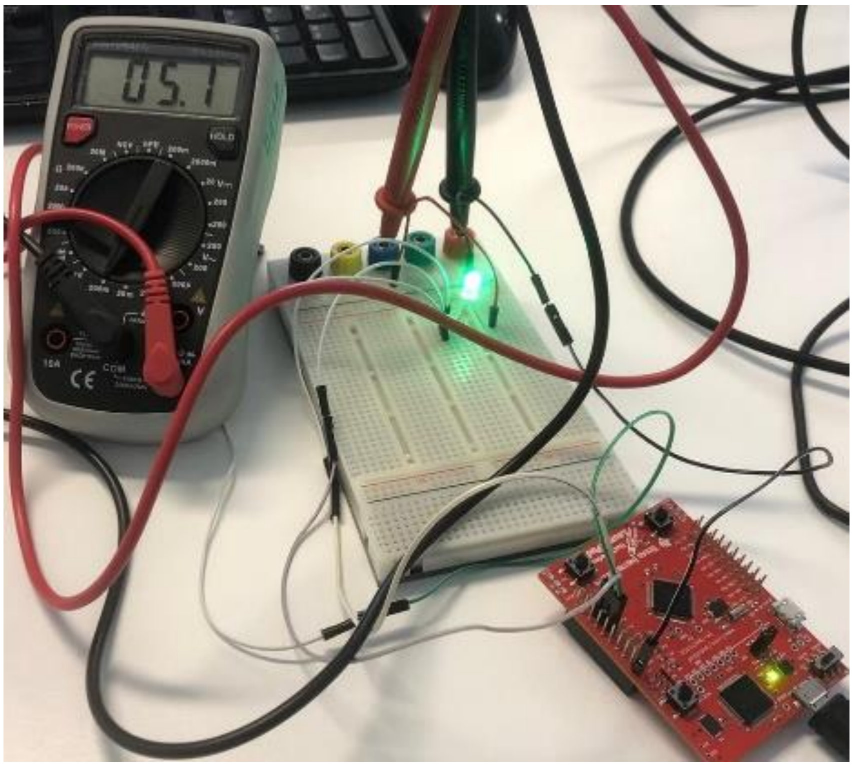


Figure 6 Blue LED turns on.

- Yellow LED (15.0mA)

  Set the red and green LED lights to light up at the same time to mix yellow. The current is the sum of the current that lights up the red LED alone and the current that lights up the green LED alone (within error range).

$$I(Yellow) = I(Red) + I(Green)\ \ (mA)$$

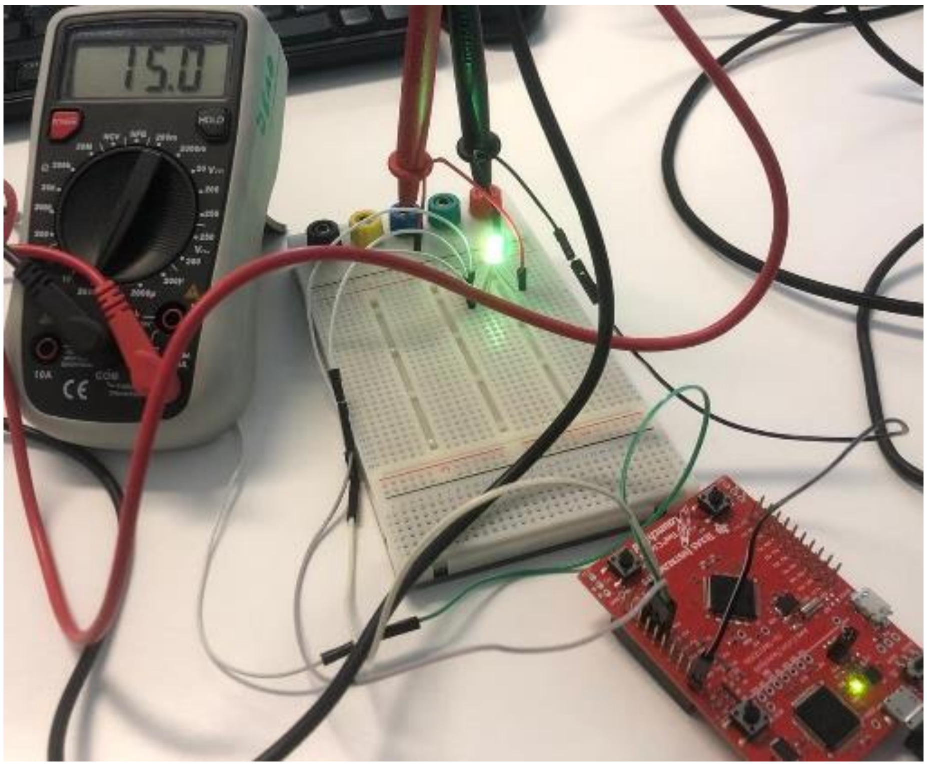


Figure 7 Yellow LED turns on.

b) What is the maximum current provided by a digital port of the microcontroller at level 1 and at level 0.

Parameter	Parameter Name[a]	Value		Unit
		Min	Max	
V_{DD}	V_{DD} supply voltage	0	4	V
V_{DDA}	V_{DDA} supply voltage[b]	0	4	V
V_{BAT}	V_{BAT} battery supply voltage	0	4	V
V_{BATRMP}	V_{BAT} battery supply voltage ramp time	0	0.7	V/µs
V_{IN_GPIO}	Input voltage on GPIOs, regardless of whether the microcontroller is powered[cde]	-0.3	5.5	V
	Input voltage for `PD4`, `PD5`, `PB0` and `PB1` when configured as GPIO	-0.3	V_{DD} + 0.3	V
$I_{GPIOMAX}$	Maximum current per output pin	-	25	mA
T_S	Unpowered storage temperature range	-65	150	°C
T_{JMAX}	Maximum junction temperature	-	150	°C

a. Voltages are measured with respect to GND.

b. To ensure proper operation, VDDA must be powered before VDD if sourced from different supplies, or connected to the same supply as VDD. Note that the minimum operating voltage for VDD differs from the minimum operating voltage for VDDA. This change should be accounted for in the system design if both are sourced from the same supply. There is not a restriction on order for powering off.

c. Applies to static and dynamic signals including overshoot.

d. Refer to Figure 24-16 on page 1386 for a representation of the ESD protection on GPIOs.

e. For additional details, see the note on GPIO pad tolerance in “GPIO Module Characteristics” on page 1385.

Table 2 Maximum Ratings [2]

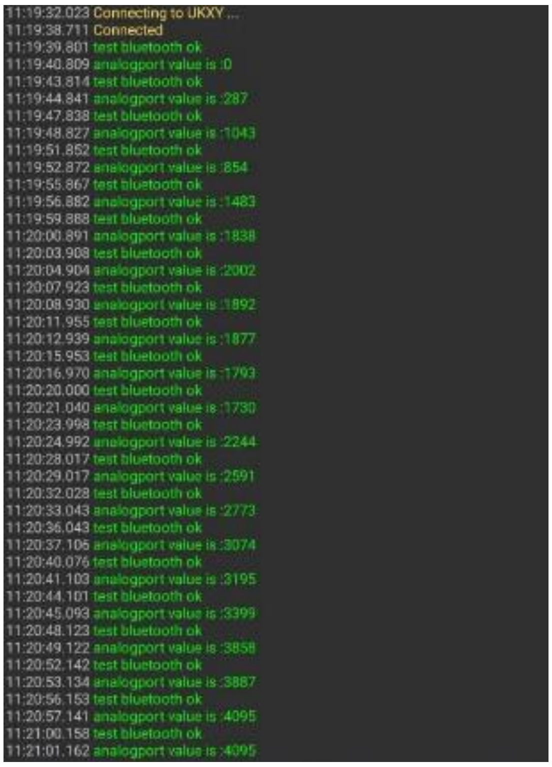
- From the table: $I_{GPIOMAX} \in [0, 25]$ mA.

  I_{max} when digital port of the microcontroller at level 1: 25mA.

  I_{max} when digital port of the microcontroller at level 0: 0mA.

3) Connect a potentiometer to an analog input of the microcontroller.

a) Read the analog value and display the result in volt every second to the console.



```
11:19:32.023 Connecting to UKXY ...
11:19:38.711 Connected
11:19:39.801 test bluetooth ok
11:19:40.809 analogport value is :0
11:19:43.814 test bluetooth ok
11:19:44.841 analogport value is :287
11:19:47.838 test bluetooth ok
11:19:48.827 analogport value is :1043
11:19:51.852 test bluetooth ok
11:19:52.872 analogport value is :854
11:19:55.867 test bluetooth ok
11:19:56.882 analogport value is :1483
11:19:59.888 test bluetooth ok
11:20:00.891 analogport value is :1838
11:20:03.908 test bluetooth ok
11:20:04.904 analogport value is :2002
11:20:07.923 test bluetooth ok
11:20:08.930 analogport value is :1892
11:20:11.955 test bluetooth ok
11:20:12.939 analogport value is :1877
11:20:15.953 test bluetooth ok
11:20:16.970 analogport value is :1793
11:20:20.000 test bluetooth ok
11:20:21.040 analogport value is :1730
11:20:23.998 test bluetooth ok
11:20:24.992 analogport value is :2244
11:20:28.017 test bluetooth ok
11:20:29.017 analogport value is :2591
11:20:32.028 test bluetooth ok
11:20:33.043 analogport value is :2773
11:20:36.043 test bluetooth ok
11:20:37.106 analogport value is :3074
11:20:40.076 test bluetooth ok
11:20:41.103 analogport value is :3195
11:20:44.101 test bluetooth ok
11:20:45.093 analogport value is :3399
11:20:48.123 test bluetooth ok
11:20:49.122 analogport value is :3858
11:20:52.142 test bluetooth ok
11:20:53.134 analogport value is :3887
11:20:56.153 test bluetooth ok
11:20:57.141 analogport value is :4095
11:21:00.158 test bluetooth ok
11:21:01.162 analogport value is :4095
```

Figure 8 Analog Value

4) Measure the power consumption of the board.

a) Precise the test condition (what is active on the board).

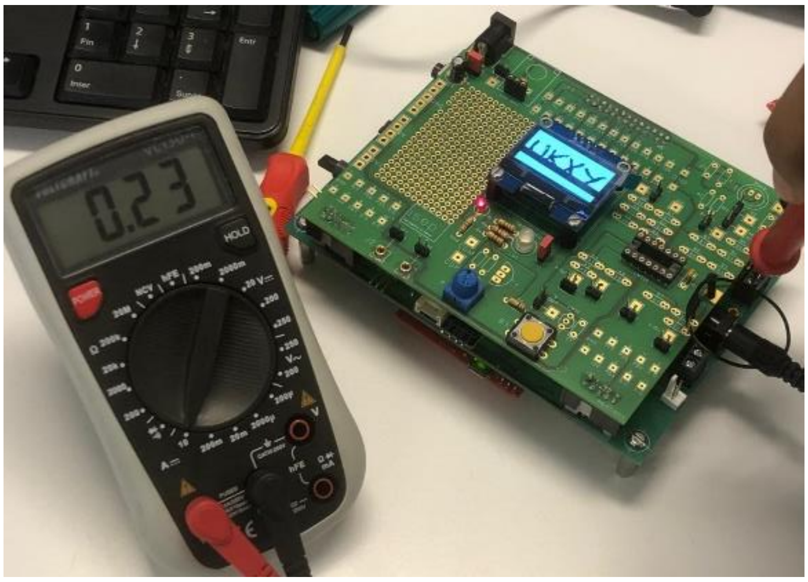


Figure 9 Current when Bluetooth & OLED Active

b) What will be the autonomy with a battery (choose a battery on the WEB)

$$Autonomy(hours) = \frac{Battery\ Capacity(mAh)}{Current\ Draw(mA)} = \frac{5000(mAh)}{0.23(mA)} \approx 21739 \text{ (hours)}$$

c) What is the carbon equivalent of this power consumption?

- Calculate Power:

$$P = 0.23\ mA \times 5V = 0.00115\ W$$

- Calculate Total Energy Consumption:

$$E_{day} = P \times 24\ hours = 0.276\ Wh$$

- Convert to Kilowatt-hours:

$$E_{day} = 0.0276Wh \div 1000 = 0.0000276\ kWh$$

- Calculate Carbon Equivalent:

$$C_{day} = E_{day} \times Carbon\ Intensity$$
$$= 0.0000276kWh\ \times\ 0.35kg\frac{CO2e}{kWh}$$
$$=\ 0.00000966kg\ CO2e$$

5) Indicate the microcontroller parameters:

a) Size of the memories

- Flash Memory: 256 KB
- **SRAM**: 32 KB
- **EEPROM**: 2 KB

b) Clock frequency

- 0 ~ 80 MHz

c) Explain the utility of a floating-point unit.

- A floating-point unit (FPU) is a part of the microcontroller that handles arithmetic operations on floating-point numbers more efficiently than software routines, allowing for better performance in processes that require floating-point calculations.

d) Describe some embedded peripherals.

The TIVA C series microcontrollers feature basic peripherals including:

- **Communication Interfaces**: UART, SPI, and I2C for data exchange with other devices.
- **Timers/Counters**: For measuring intervals and generating precise time delays.
- **Analog-to-Digital Converters (ADC)**: To convert analog signals into digital values.
- **Digital-to-Analog Converters (DAC)**: To convert digital values into analog signals.
- **PWM Controllers**: To generate signals with variable duty cycles, commonly used for motor speed control or LED brightness.
- **GPIO Pins**: General-purpose input/output pins for controlling and monitoring various electronic components.
- **USB Interface**: Provides USB communication capabilities, supporting device or host modes.

- **Floating Point Unit (FPU)**: Accelerates floating-point operations, enhancing mathematical computation performance.

6) Use Serial1.xxx for Bluetooth on the board; pair with the computer first.

a) Display the values of question 2.a via the Bluetooth link.

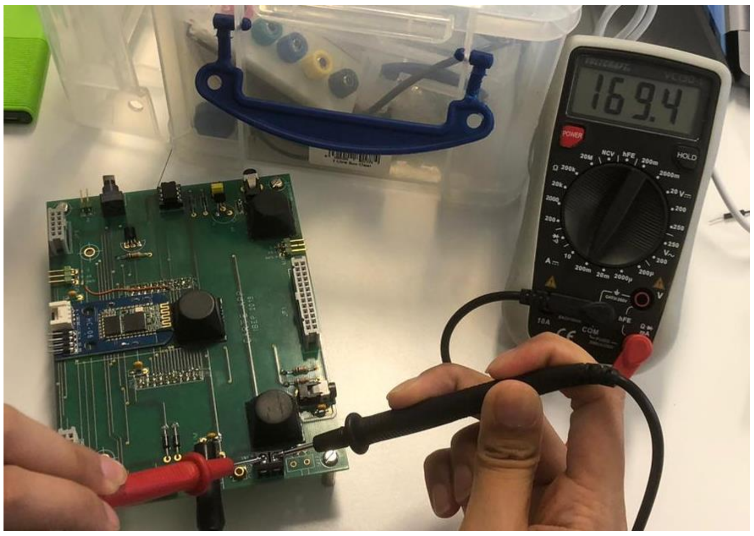
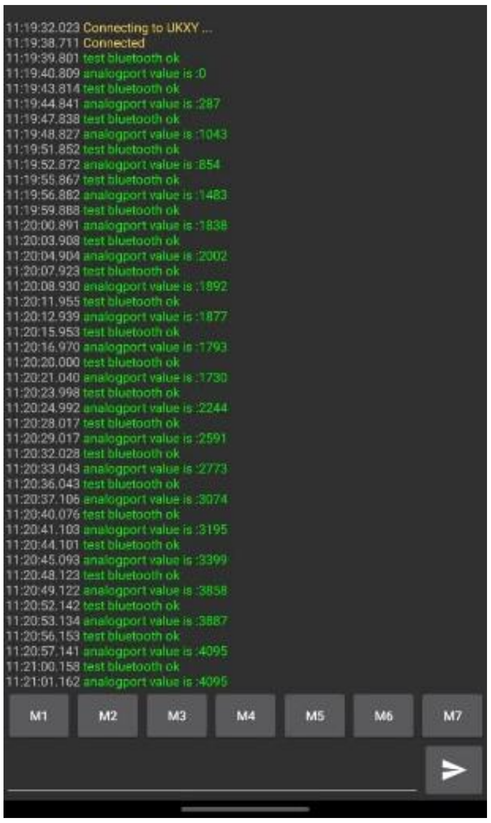


Figure 10 Current when Bluetooth Active

b) Download terminal emulator in the phone and display the value.



```
11:19:32.023 Connecting to UKXY ...
11:19:38.711 Connected
11:19:39.801 test bluetooth ok
11:19:40.809 analogport value is :0
11:19:43.814 test bluetooth ok
11:19:44.841 analogport value is :287
11:19:47.838 test bluetooth ok
11:19:48.827 analogport value is :1043
11:19:51.852 test bluetooth ok
11:19:52.872 analogport value is :854
11:19:55.867 test bluetooth ok
11:19:56.882 analogport value is :1483
11:19:59.888 test bluetooth ok
11:20:00.891 analogport value is :1838
11:20:03.908 test bluetooth ok
11:20:04.904 analogport value is :2002
11:20:07.923 test bluetooth ok
11:20:08.930 analogport value is :1892
11:20:11.955 test bluetooth ok
11:20:12.939 analogport value is :1877
11:20:15.953 test bluetooth ok
11:20:16.970 analogport value is :1793
11:20:20.000 test bluetooth ok
11:20:21.040 analogport value is :1730
11:20:23.998 test bluetooth ok
11:20:24.992 analogport value is :2244
11:20:28.017 test bluetooth ok
11:20:29.017 analogport value is :2591
11:20:32.028 test bluetooth ok
11:20:33.043 analogport value is :2773
11:20:36.043 test bluetooth ok
11:20:37.106 analogport value is :3074
11:20:40.076 test bluetooth ok
11:20:41.103 analogport value is :3195
11:20:44.101 test bluetooth ok
11:20:45.093 analogport value is :3399
11:20:48.123 test bluetooth ok
11:20:49.122 analogport value is :3858
11:20:52.142 test bluetooth ok
11:20:53.134 analogport value is :3887
11:20:56.153 test bluetooth ok
11:20:57.141 analogport value is :4095
11:21:00.158 test bluetooth ok
11:21:01.162 analogport value is :4095
```

Figure 11 Temperature Display on the Phone

7) Sold the OLED display.

a) Use the given library to display messages on it.

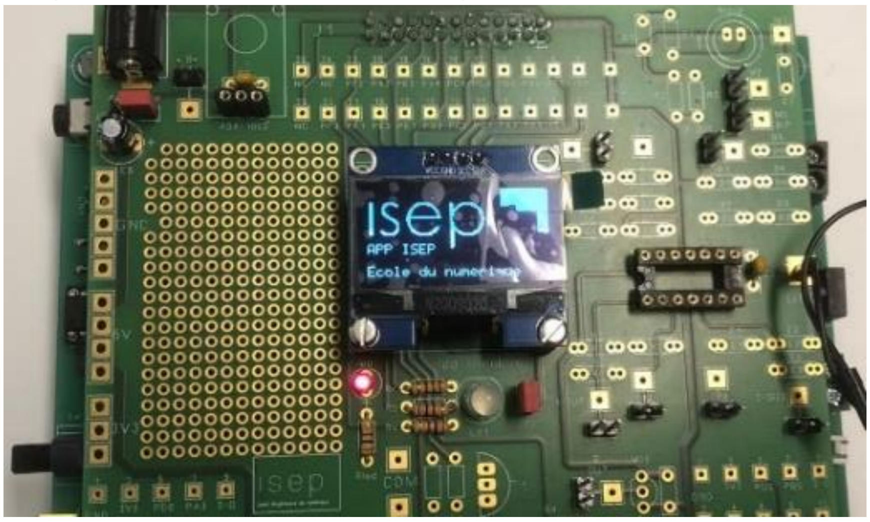


Figure 12 Display ISEP

b) Display a logo on the OLED display.

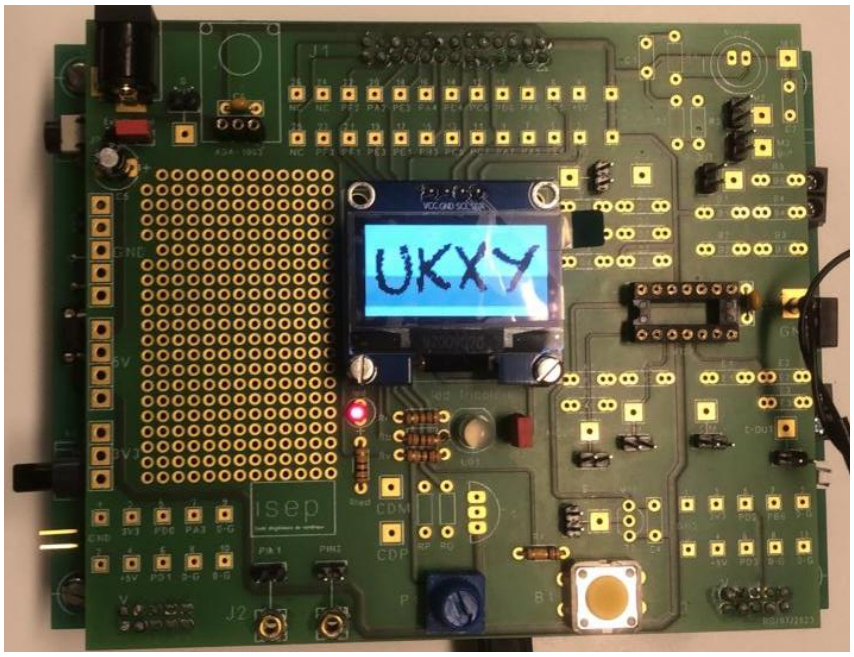
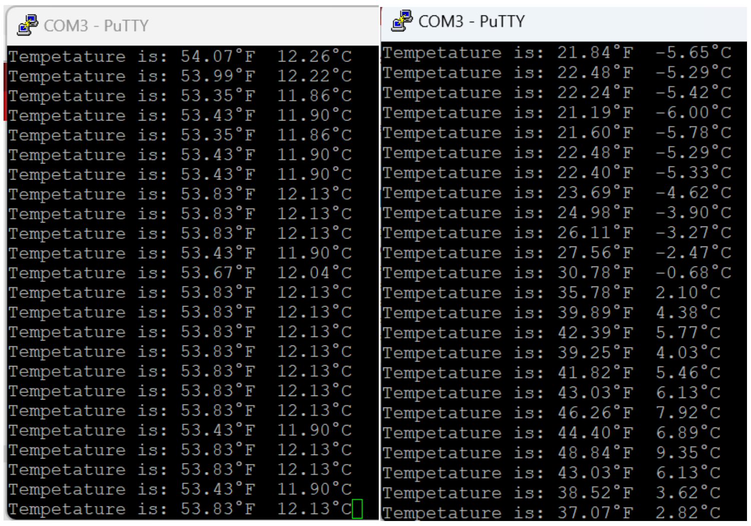


Figure 13 Display UKXY

5.2 Mission 2

1) Connect the temperature sensor on an analog port and display the temperature.



```
Tempetature is: 54.07°F  12.26°C
Tempetature is: 53.99°F  12.22°C
Tempetature is: 53.35°F  11.86°C
Tempetature is: 53.43°F  11.90°C
Tempetature is: 53.35°F  11.86°C
Tempetature is: 53.43°F  11.90°C
Tempetature is: 53.43°F  11.90°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.43°F  11.90°C
Tempetature is: 53.67°F  12.04°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.43°F  11.90°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.83°F  12.13°C
Tempetature is: 53.43°F  11.90°C
Tempetature is: 53.83°F  12.13°C
```

```
Tempetature is: 21.84°F  -5.65°C
Tempetature is: 22.48°F  -5.29°C
Tempetature is: 22.24°F  -5.42°C
Tempetature is: 21.19°F  -6.00°C
Tempetature is: 21.60°F  -5.78°C
Tempetature is: 22.48°F  -5.29°C
Tempetature is: 22.40°F  -5.33°C
Tempetature is: 23.69°F  -4.62°C
Tempetature is: 24.98°F  -3.90°C
Tempetature is: 26.11°F  -3.27°C
Tempetature is: 27.56°F  -2.47°C
Tempetature is: 30.78°F  -0.68°C
Tempetature is: 35.78°F  2.10°C
Tempetature is: 39.89°F  4.38°C
Tempetature is: 42.39°F  5.77°C
Tempetature is: 39.25°F  4.03°C
Tempetature is: 41.82°F  5.46°C
Tempetature is: 43.03°F  6.13°C
Tempetature is: 46.26°F  7.92°C
Tempetature is: 44.40°F  6.89°C
Tempetature is: 48.84°F  9.35°C
Tempetature is: 43.03°F  6.13°C
Tempetature is: 38.52°F  3.62°C
Tempetature is: 37.07°F  2.82°C
```

Figure 14 Temperature Display

a) What is the accuracy of the measurement?

Typical accuracy of the LM35 is approximately ±0.5°C at room temperature (factors such as power supply voltage stability, environmental noise, sensor placement, etc. may affect accuracy).

2) Explain what Co2 measurements are from the provided sensor.

CO2 measurements from a sensor typically refer to the process of detecting and quantifying the amount of carbon dioxide (CO2) present in the environment. The sensor used for this purpose is often a type of gas sensor that can detect CO2 levels in the air. Here's how it generally works:

a) **Detection Principle**: CO2 sensors usually operate based on one of several principles, including infrared gas sensing, chemical gas sensing, or metal-oxide semiconductor technology. Infrared sensors are commonly used for CO2 detection because CO2 molecules absorb infrared light at specific wavelengths.
b) **Measurement Process**: The sensor emits light at these wavelengths and measures the amount of light that is absorbed by the CO2 in the air. The more CO2 present, the more light is absorbed.
c) **Output Signal**: The sensor then converts this information into an electrical signal that can be measured. This signal is proportional to the concentration of CO2 in the environment.
d) **Data Interpretation**: The sensor's output is then displayed or transmitted for further analysis. This can be a digital readout, or the data can be sent to a computer or other device for more detailed analysis and record-keeping.

e) **Applications:** CO2 measurements are crucial in various applications like environmental monitoring, indoor air quality assessment, industrial processes, and in your case, educational projects to understand environmental characteristics.

a) Connect the sensor to your system and display the measures.

Due to sensor issues, this item is currently unfinished.

3) Connect the micro electret sensor.

b) Get a sound:

i. Explain how this sensor works.

The Micro Electret AMB-707-RC is a condenser microphone that uses changes in capacitance to convert sound waves into electrical signals. The capacitor in the microphone consists of a fixed electrode and a vibrating membrane (electrode). When the sound wave hits the vibrating membrane, the distance between the capacitor's changes, thereby changing the capacitance value and generating a corresponding voltage signal.

ii. How to make it work?

- Connect VCC to the positive power pin of the microphone.
- Connect the GND pin to ground.
- Connect the signal output pin to an audio amplifier or analog-to-digital converter (ADC).

iii. What's the voltage generated by the sensor with your assembly?

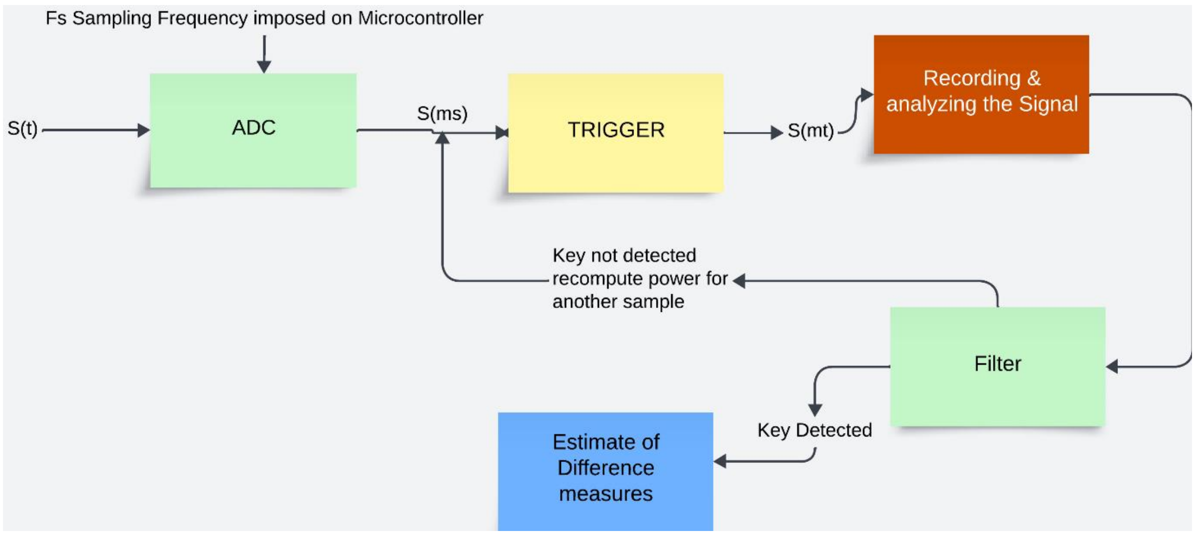
The output of an Electret microphone is usually weak, usually between a few millivolts and tens of millivolts, depending on the loudness and frequency of the sound.

c) Amplified the sound to Increase reception sensitivity.

i. What are the parameters of the amp to be used?

- **Gain**: Depends on the degree of amplification required, usually between 20 and 1000 times.
- **Noise Figure**: as low as possible to maintain sound quality.
- **Frequency response**: Covers at least the audible range of the human ear from 20Hz to 20kHz.
- **Input and output impedance**: match the microphone and next stage circuit.

ii. Design, calculate and simulate an amplifier.



Fs Sampling Frequency imposed on Microcontroller
S(t)
ADC
S(ms)
TRIGGER
S(mt)
Recording & analyzing the Signal
Key not detected recompute power for another sample
Filter
Key Detected
Estimate of Difference measures

Figure 15 Voice Treatment

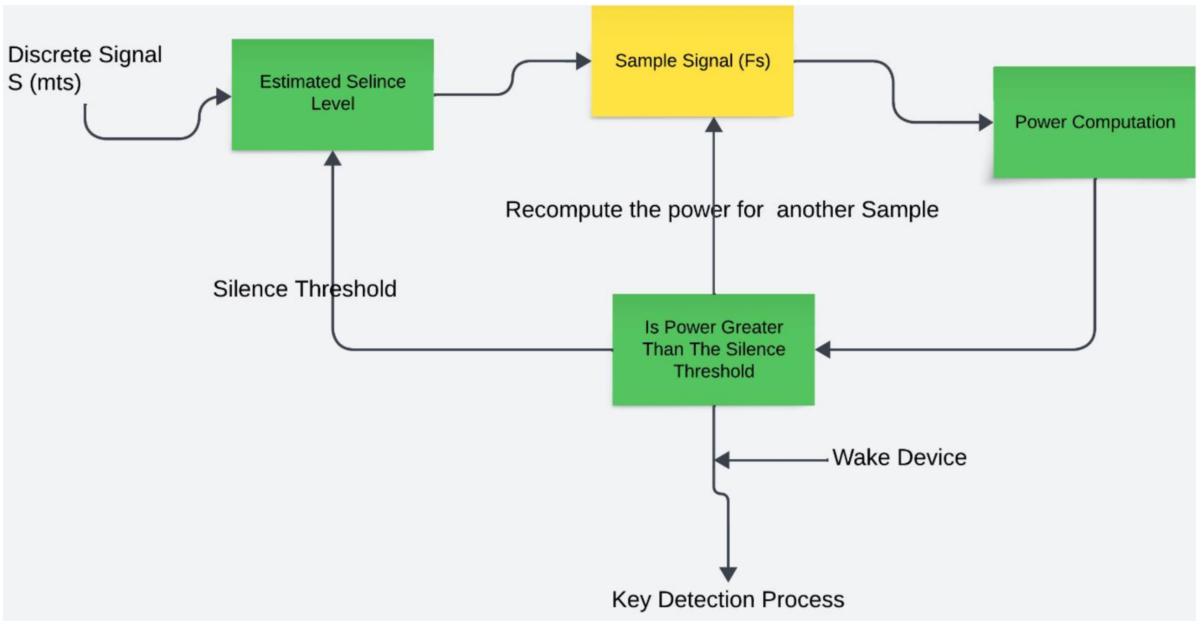


Figure 16 Trigger System

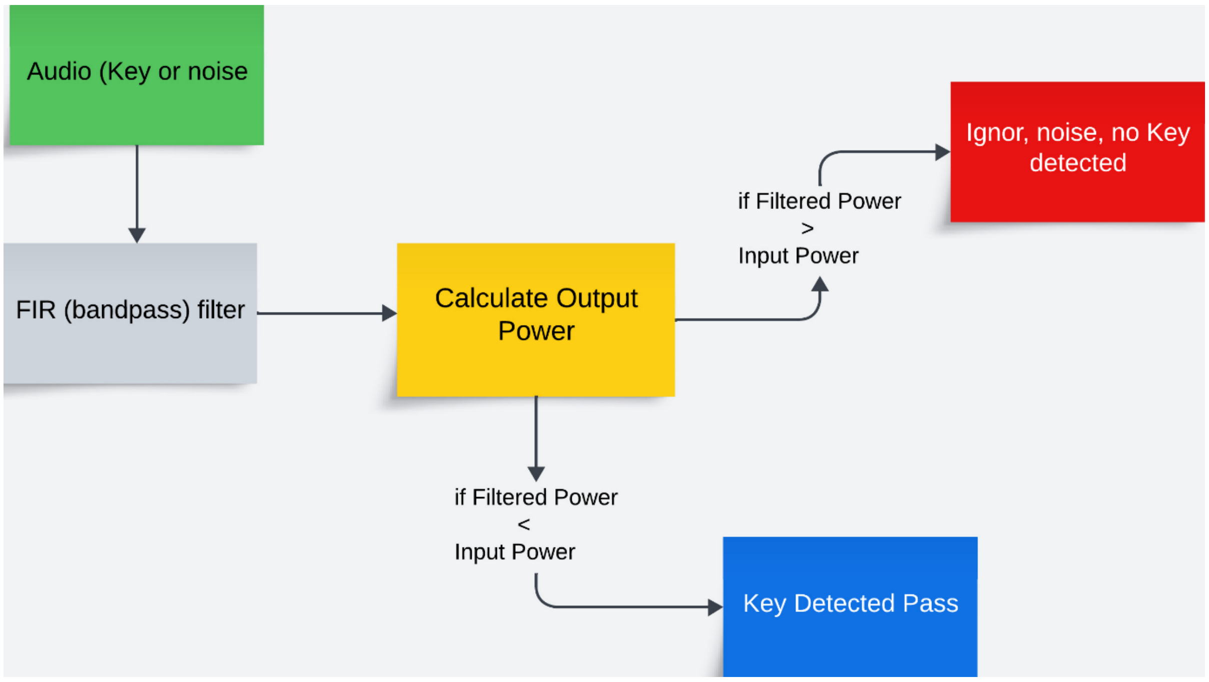

Audio (Key or noise
FIR (bandpass) filter
Calculate Output Power
if Filtered Power > Input Power
Ignor, noise, no Key detected
if Filtered Power < Input Power
Key Detected Pass

Figure 17 Filter Design and Implementation

d) Filter the sound (2nd order filter): The frequencies you process will be less than 2500 kHz.

1)

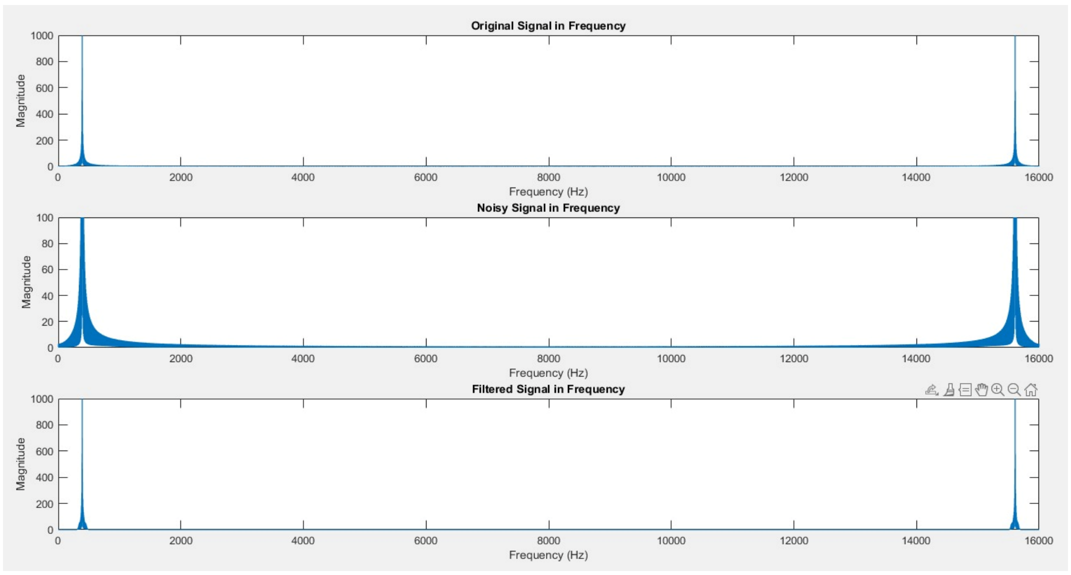


Figure 18 392.wav sound processing

- **Original Signal in Frequency:**
  - ➢ This graph shows the frequency spectrum of the original signal, with two significant peaks indicating strong periodic components or tones at those frequencies. These peaks are evident at the low and high ends of the frequency spectrum.
  - ➢ The horizontal axis is labeled with frequency in Hertz (Hz), indicating the different frequencies present in the original signal.
  - ➢ The vertical axis represents magnitude, which could refer to the amplitude or power of the signal at each frequency.
- **Noisy Signal in Frequency:**
  - ➢ This graph displays what appears to be the original signal corrupted with noise. The spectrum has become broader, showing increased magnitude across a range of frequencies which is indicative of the presence of noise.
  - ➢ The noise is particularly noticeable at the high-frequency end of the spectrum, significantly increasing the magnitude relative to the original signal.
- **Filtered Signal in Frequency:**
  - ➢ The final graph depicts the frequency spectrum after a filtering process has been applied. The filtering has effectively reduced the noise, as can be seen by the diminished magnitudes at frequencies where the noise was prominent.
  - ➢ The result of the filtering process has left us with a spectrum that closely resembles the original signal's spectrum, indicating that the filter has likely been successful in removing the unwanted noise and preserving the signal's key frequency components.
  - ➢ The filter used here might be a bandstop or notch filter targeting the noise frequencies, or a bandpass filter designed to retain the frequencies of the original signal peaks while rejecting others.

2)

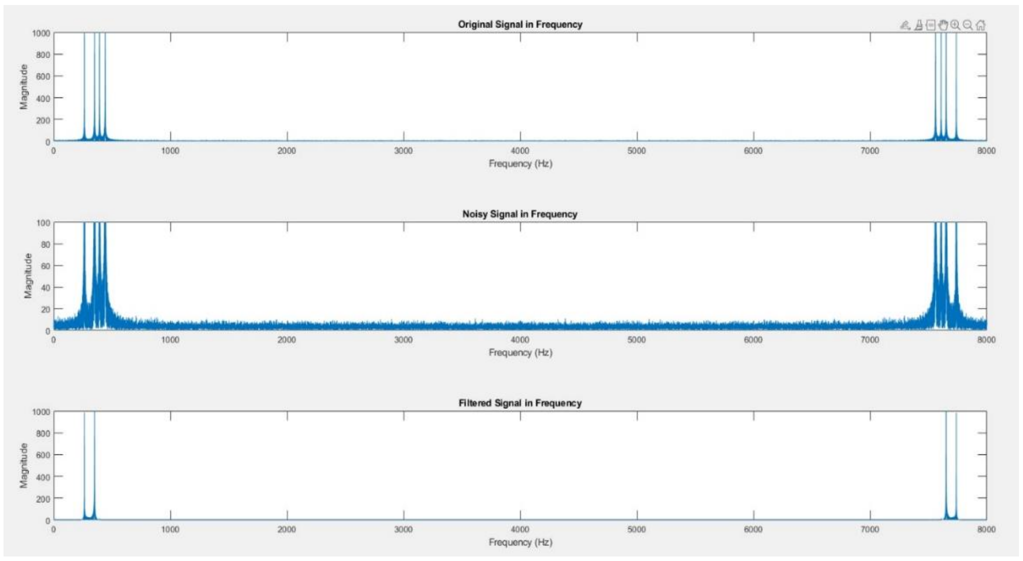


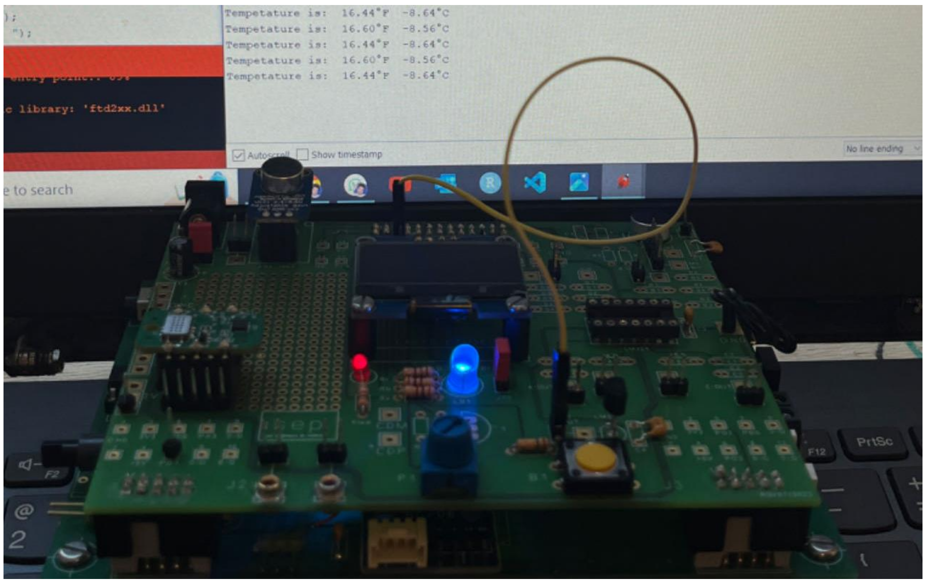
Figure 19 chord1.wav sound processing

- **Original Signal in Frequency:**
  - The graph shows a clean frequency spectrum of an original signal, with distinct peaks at certain frequencies. These peaks represent the dominant frequency components of the signal.
  - The horizontal axis is labelled with frequency in Hertz (Hz), indicating the frequencies present in the original signal.
  - The vertical axis represents magnitude, which likely corresponds to the amplitude or power of the signal at each frequency.
- **Noisy Signal in Frequency:**
  - This graph exhibits the original signal with added noise. Unlike the original signal, the spectrum now shows a dense spread of frequencies indicating the presence of noise throughout, which is especially pronounced at the lower frequencies.
  - The noise adds amplitude across the entire frequency spectrum but maintains the peaks of the original signal, suggesting that the noise is superimposed on the original signal without completely masking the original frequency components.
- **Filtered Signal in Frequency:**
  - The final graph demonstrates the frequency spectrum after noise reduction through filtering. The filter has effectively removed the noise, leaving behind a spectrum that closely resembles the original one, with clear peaks and a clean background.
  - The significant reduction of noise across all frequencies except at the peaks suggests the use of a filter that preserves the primary frequencies of the original signal while suppressing the noise – possibly a bandpass or a multiband filter that selectively allows the frequencies of the original signal's peaks through while attenuating other frequencies, which are likely noise.

5.3 Mission 3

1. Link the temperature sensor and control its data to control the LED.

- **[TEST]**When the temperature is below 20.00 degrees Fahrenheit (-6,67 degrees Celsius), the LED lights up and displays **blue** (reminds of low temperature).
- Low temperature LED blue light will be lit when the temperature is below 60,80 degrees Fahrenheit (16.00 degrees Celsius) when used in the cabin, to remind administrators to appropriately raise cabin temperature.

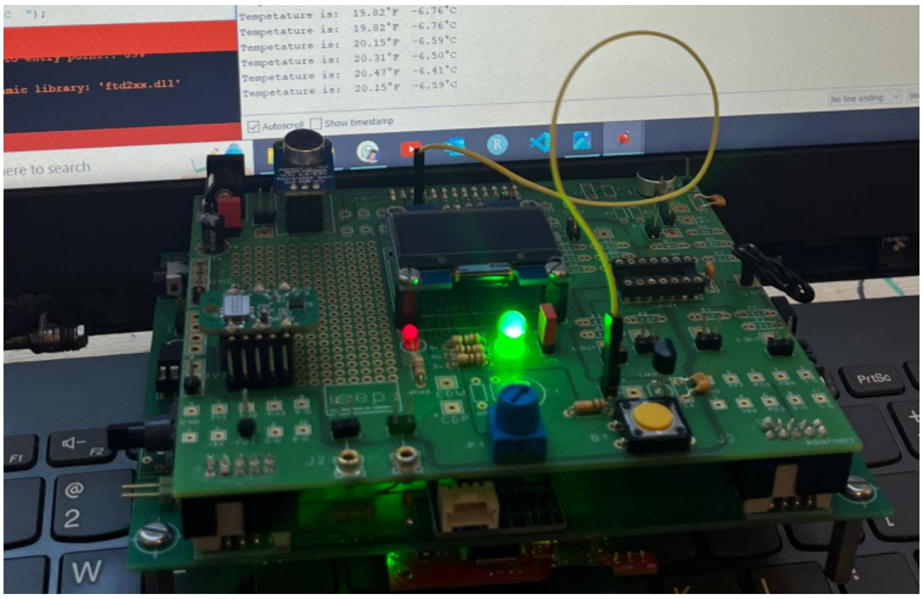


```
Tempetatature is:  16.44°F  -8.64°C
Tempetature is:  16.60°F  -8.56°C
Tempetature is:  16.44°F  -8.64°C
Tempetature is:  16.60°F  -8.56°C
Tempetature is:  16.44°F  -8.64°C
```

```
c library: 'ftd2xx.dll'
```

Figure 20 Reminds of Low Temperature

- **[TEST]**When the temperature is between 20.00 degrees Fahrenheit (-6,67 degrees Celsius) with 37,40 degrees Fahrenheit (3.00 degrees Celsius), the LED lights up and displays **green** (suitable temperature).
- Suitable temperature LED green light will be lit when the temperature is between 60.8 degrees Fahrenheit (16.00 degrees Celsius) with 77.00 degrees Fahrenheit (25.00 degrees Celsius) when used in the cabin, indicates that the current cabin temperature is suitable.

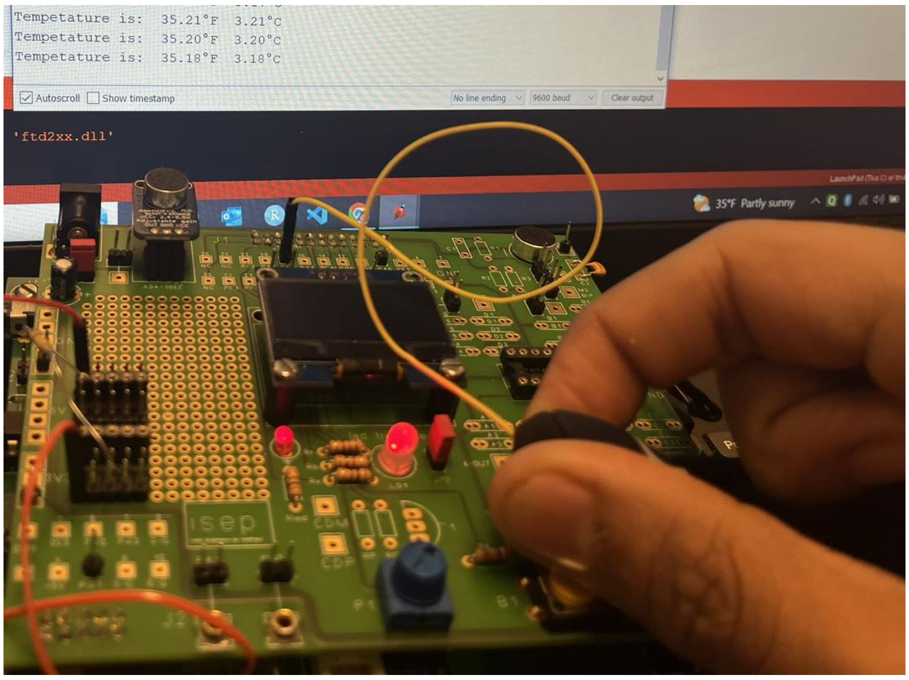


```
Tempetature is:  19.82°F  -6.76°C
Tempetature is:  19.82°F  -6.76°C
Tempetature is:  20.15°F  -6.59°C
Tempetature is:  20.31°F  -6.50°C
Tempetature is:  20.47°F  -6.41°C
Tempetature is:  20.15°F  -6.59°C
```

```
mic library: 'ftd2xx.dll'
```

Figure 21 Reminds of Suitable Temperature

- **[TEST]**When the temperature is over 37,40 degrees Fahrenheit (3.00 degrees Celsius), the LED lights up and displays **red** (reminds of high temperature).
- High temperature LED red light will be lit when the temperature is over 77.00 degrees Fahrenheit (25.00 degrees Celsius) when used in the cabin, to remind administrators to appropriately lower cabin temperature.



```
Tempetature is:  35.21°F  3.21°C
Tempetature is:  35.20°F  3.20°C
Tempetature is:  35.18°F  3.18°C
```

Figure 22 Reminds of High Temperature

2. Link the micro sensor and control its data to control the LED.

- This **blue** light will be activated when the sound level in the cabin falls below 70 decibels (dB), suggesting the environment is very quiet.
- The **green** light will be lit when the sound level in the cabin is between 70 dB and 80 dB. This decibel range is considered acceptable and comfortable for a cabin environment, providing a balance between being audibly perceptible without being overwhelmingly loud.
- The **red** light will be activated when the sound level in the cabin exceeds 80 dB. This indicates a high noise level that could lead to discomfort, stress, or even hearing damage over extended periods. The red light serves as a signal for administrators to take necessary measures to mitigate the noise.

3. Link the CO2 sensor and control its data to control the LED.

- A **blue** light could indicate that CO2 levels are below a certain threshold, such as 400 parts per million (ppm). This would inform the administrators that the cabin air is fresh.
- The **green** light would illuminate when the CO2 concentration is within a range considered healthy for indoor spaces, for example between 400 ppm and 1000 ppm. This indicates good air quality and proper ventilation.
- The **red** light would indicate high levels of CO2, above a level such as 1000 ppm, which could suggest inadequate ventilation and potential discomfort or health issues for people inside the cabin. This is a prompt for administrators to improve air circulation.

4. Connect the sensor to the screen and print relevant information to the screen.

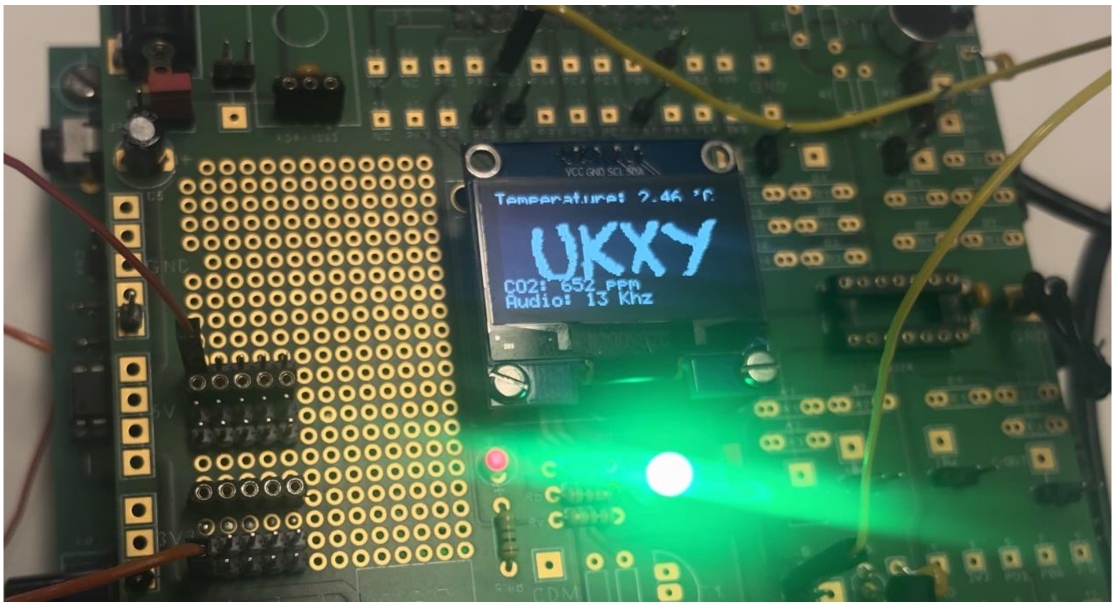


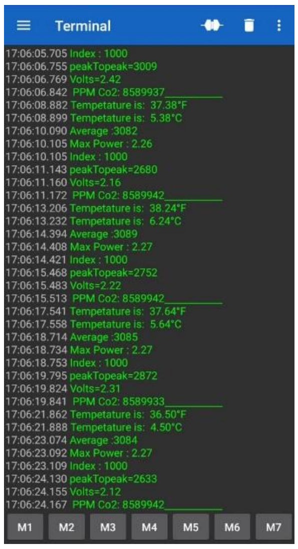
Figure 23 Information on the Screen

- ➢ When the device is well connected and under normal conditions, the temperature, CO2 and sound values will be displayed on the screen in real time.
- ➢ The color of the LED light will give prompts. When the LED light shows red, the crew can understand numerical abnormality information and adjust through the screen.

5. Real-time data transmission via Bluetooth.

As illustrated in the terminal interface, the system diligently relays critical parameters such as temperature readings in both Fahrenheit and Celsius, CO2 levels measured in parts per million (PPM), as well as other relevant indices like peak values and voltage measurements. With each entry time-stamped to the precise second, passengers can observe the fluctuations in their cabin environment as they happen, fostering a sense of control and reassurance.

This seamless integration of environmental monitoring and Bluetooth technology not only informs but also transforms the passenger experience from passive to participatory.



```
17:06:05.705 Index : 1000
17:06:06.755 peakTopeak=3009
17:06:06.769 Volts=2.42
17:06:06.842  PPM Co2: 8589937___________
17:06:08.882 Tempetature is:  37.38°F
17:06:08.899 Tempetature is:  5.38°C
17:06:10.090 Average :3082
17:06:10.105 Max Power : 2.26
17:06:10.105 Index : 1000
17:06:11.143 peakTopeak=2680
17:06:11.160 Volts=2.16
17:06:11.172  PPM Co2: 8589942___________
17:06:13.206 Tempetature is:  38.24°F
17:06:13.232 Tempetature is:  6.24°C
17:06:14.394 Average :3089
17:06:14.408 Max Power : 2.27
17:06:14.421 Index : 1000
17:06:15.468 peakTopeak=2752
17:06:15.483 Volts=2.22
17:06:15.513  PPM Co2: 8589942___________
17:06:17.541 Tempetature is:  37.64°F
17:06:17.558 Tempetature is:  5.64°C
17:06:18.714 Average :3085
17:06:18.734 Max Power : 2.27
17:06:18.753 Index : 1000
17:06:19.795 peakTopeak=2872
17:06:19.824 Volts=2.31
17:06:19.841  PPM Co2: 8589933___________
17:06:21.862 Tempetature is:  36.50°F
17:06:21.888 Tempetature is:  4.50°C
17:06:23.074 Average :3084
17:06:23.092 Max Power : 2.27
17:06:23.109 Index : 1000
17:06:24.130 peakTopeak=2633
17:06:24.155 Volts=2.12
17:06:24.167  PPM Co2: 8589942___________
```

Figure 24 Serial Bluetooth Terminal

6. Conclusion

In this project, we achieved a comprehensive understanding and meticulously crafted the hardware components during the initial phase. Our team became adept at utilizing the associated software tools, which was crucial for the subsequent stages of development.

As we progressed into the intermediate phase, our efforts were centred around leveraging the TIVA platform to bring our design to life. Through proficient use of the Energia software environment, we adeptly programmed the microcontroller to manage the LED indicators, effectively driving visual alerts based on sensor inputs. The display screen interface was also programmed to provide real-time feedback and system status. Concurrently, we harnessed the powerful analytical capabilities of MATLAB for sound signal processing, allowing us to interpret and manipulate audio inputs with precision.

In the project's final stage, we successfully integrated the sensors, display screen, and LED indicators to create a seamless interface. This integration was aimed at ensuring that sensor data could be intuitively accessed and understood by users. The temperature, sound, and CO2 sensors worked in concert, with the LEDs providing immediate visual cues — blue for low temperature or sound level, green for optimal conditions, and red for high readings — thus offering an at-a-glance health check of the cabin environment.

Furthermore, the screen display was programmed to provide a more detailed account of the environmental parameters, offering insights beyond the binary feedback of the LEDs. This enables users to make informed decisions based on precise data readings.

The culmination of this project not only reflects the synthesis of hardware and software expertise but also embodies a commitment to creating a user-friendly system. The knowledge and experience gained through this hands-on project will undoubtedly serve as a valuable foundation for future technological endeavors. This project stands as a testament to the collaborative effort, technical proficiency, and innovative spirit of the team.

7. References

[1] Figure V Cycle, https://en.wikipedia.org/wiki/V-model.

[2] Table Maximum Ratings, Tiva™ C Series TM4C123GH6PM Microcontroller Data Sheet (Rev. E), Page 1358.

8. Appendix

1. Matlab Code.

```
%function myfilter
[S,Fs]=audioread('392.wav');

new_Fs = 1000;
resampled_S = resample(S, new_Fs, Fs);


spectrum= abs(fft(resampled_S));
plot(spectrum);

subplot(2,1,1)
plot(spectrum)

 new_Fs = 1000;  % Sampling Frequency

Fstop1 = 352;            % First Stopband Frequency
Fpass1 = 372;            % First Passband Frequency
Fpass2 = 412;            % Second Passband Frequency
Fstop2 = 432;            % Second Stopband Frequency
Dstop1 = 0.001;          % First Stopband Attenuation
Dpass  = 0.057501127785;  % Passband Ripple
Dstop2 = 0.001;          % Second Stopband Attenuation
dens   = 20;             % Density Factor



% Calculate the order from the parameters using FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fstop1 Fpass1 Fpass2 Fstop2]/(Fs_new/2), [0 1 ...
    0], [Dstop1 Dpass Dstop2]);
% Calculate the coefficients using the FIRPM function.
b  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);


% [EOF]

Outputsignal=filter(Hd,resampled_S);
Spectrum2=abs(fft(Outputsignal));

subplot(2,1,2)
plot(Spectrum2)
title("after filter")

Ps1 = 0;
for i = 1:newFs
    %for i = 1:N
    Ps1 = Ps1 + resampled_S(i) * resampled_S(i);
```

```
end
Ps2 = 0;
for i = 1:newFs
    %for i = 1:N
    Ps2 = Ps2 + Outputsignal(i) * Outputsignal(i);
end
disp(Ps1);
disp(Ps2);

Threshold = 3;

if (Ps2 > Threshold)
    disp("Key is activated")
else
    disp("Key is not activated")
end
```

2. Screen Display LOGO Code.

```
unsigned char motif[128*8] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xe0, 0xf0, 0xf0, 0xf0, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08,
  0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0xf8, 0xf8, 0xf0, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xc0, 0xf0, 0xf0, 0xf0, 0xfc, 0x3e, 0x3c, 0x1e, 0x06,
  0x00, 0x00, 0x1c, 0x7c, 0xf8, 0xf8, 0xfc, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0,
  0xf8, 0xfe, 0xff, 0xff, 0x3f, 0x1e, 0x00, 0x00, 0x60, 0xf0, 0xf0, 0xf8, 0xf8, 0xf8, 0xe0, 0xc0,
  0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0xc0, 0xf8,
  0xf8, 0xfc, 0x7e, 0x3e, 0x0e, 0x06, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0xfc, 0xff, 0xff, 0xff, 0x5f, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x06, 0x07, 0xfe, 0xff, 0xff, 0xfe, 0xe4, 0x00, 0x00, 0x00, 0x00, 0xdf, 0xff, 0xff, 0xfe,
  0xf0, 0xf8, 0xf8, 0xf8, 0xff, 0x7f, 0x1f, 0x0f, 0x0f, 0x07, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x0f, 0x1f, 0x7f, 0xfe, 0xfc, 0xf8, 0xfc, 0xff, 0x7f, 0x1f,
  0x1f, 0x07, 0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x07, 0x0f, 0x0f,
  0x0f, 0x1f, 0x3f, 0x3e, 0x3e, 0x38, 0x7c, 0x78, 0xf8, 0xb0, 0xf8, 0xf8, 0xff, 0xff, 0xff, 0x7f,
  0x1f, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x07, 0xff, 0xff, 0xff, 0xfc, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x17, 0xff, 0xbf, 0xff, 0x1c, 0x00, 0x00, 0x00, 0x80, 0xff, 0xff, 0xff, 0x1f,
  0x3f, 0x3f, 0x7f, 0xbf, 0xfc, 0xf8, 0xf8, 0xc0, 0xe0, 0xc0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xf0, 0xfc, 0xff, 0x7f, 0x7f, 0x7f, 0xff, 0xff, 0xfc,
  0xf0, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0xe0, 0xf8, 0xff, 0xff, 0x7f, 0x1f, 0x07, 0x03, 0x01, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x01, 0x1f, 0x3f, 0x7f, 0x7f, 0xfb, 0xf8, 0xe0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0,
  0xc0, 0xe0, 0xf8, 0x7f, 0x3f, 0x0f, 0x01, 0x00, 0x00, 0x00, 0x01, 0xff, 0xff, 0xff, 0x7f, 0x05,
  0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x07, 0x0f, 0x1f, 0x1f, 0x3f, 0x3f, 0x7c, 0x7e, 0x1c, 0x18,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x1c, 0x3f, 0x3f, 0x07, 0x01, 0x00, 0x00, 0x00, 0x00, 0x03, 0x0f,
  0x1f, 0xff, 0xff, 0xfe, 0x78, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x80, 0x40, 0xc0, 0xe8, 0xfe, 0x7f, 0x3f, 0x0f, 0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x03, 0x01, 0x03, 0x03,
  0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
```

3. Complete Hareware Code.

```
#include "stdio.h"
#include "Iseplogo128.h"
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_i2c.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "fontData.h"
```

```
#define PORTI2C   2

#define SAMPLE_RATE 10000  // Sampling rate in Hz
#define SAMPLE_SIZE 1024   // Size of the array to store signal samples (adjust as needed)
#define WINDOWS     200     // Number of samples used to calculate power
#define SCALEOVERTWO 2048
const int sampleWindow = 50;
unsigned int sample;
#define PWM_PIN 27
#define lm35 28
#define audio 29
int signal[SAMPLE_SIZE];   // Array to store signal samples
unsigned long duration;
float tamp=0.0;

  void InitI2C0(void)
{
    //enable I2C module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);

    //reset module
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);

    //enable GPIO peripheral that contains I2C 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

    // Configure the pin muxing for I2C0 functions on port B2 and B3.
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);

    // Select the I2C function for these pins.
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

    // Enable and initialize the I2C0 master module.  Use the system clock for
    // the I2C0 module.  The last parameter sets the I2C data transfer rate.
    // If false the data rate is set to 100kbps and if true the data rate will
    // be set to 400kbps.
    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);

    //clear I2C FIFOs
    HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;
}

void InitI2C1(void)
{
    //enable I2C module 1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);
    //reset module
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C1);

    //enable GPIO peripheral that contains I2C 1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    // Configure the pin muxing for I2C1 functions on port B2 and B3.
    GPIOPinConfigure(GPIO_PA6_I2C1SCL);
    GPIOPinConfigure(GPIO_PA7_I2C1SDA);

    // Select the I2C function for these pins.
    GPIOPinTypeI2CSCL(GPIO_PORTA_BASE, GPIO_PIN_6);
    GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_7);

    // Enable and initialize the I2C0 master module.  Use the system clock for
    // the I2C0 module.  The last parameter sets the I2C data transfer rate.
    // If false the data rate is set to 100kbps and if true the data rate will
    // be set to 400kbps.
    I2CMasterInitExpClk(I2C1_BASE, SysCtlClockGet(), false);

    //clear I2C FIFOs
    HWREG(I2C1_BASE + I2C_O_FIFOCTL) = 80008000;
}


void InitI2C2(void)
{
    //enable I2C module 2
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C2);

    //reset module
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C2);

    //enable GPIO peripheral that contains I2C 2
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

    // Configure the pin muxing for I2C0 functions on port B2 and B3.
    GPIOPinConfigure(GPIO_PE4_I2C2SCL);
    GPIOPinConfigure(GPIO_PE5_I2C2SDA);

    // Select the I2C function for these pins.
    GPIOPinTypeI2CSCL(GPIO_PORTE_BASE, GPIO_PIN_4);
    GPIOPinTypeI2C(GPIO_PORTE_BASE, GPIO_PIN_5);

    // Enable and initialize the I2C0 master module.  Use the system clock for
    // the I2C0 module.  The last parameter sets the I2C data transfer rate.
    // If false the data rate is set to 100kbps and if true the data rate will
    // be set to 400kbps.
    I2CMasterInitExpClk(I2C2_BASE, SysCtlClockGet(), false);
    //clear I2C FIFOs
    HWREG(I2C2_BASE + I2C_O_FIFOCTL) = 80008000;
}

void InitI2C(void)
{
    if (PORTI2C == 0)
        InitI2C0();
    else
        if (PORTI2C == 1)
            InitI2C1();
        else
            InitI2C2();
}
//sends an I2C command to the specified slave
void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...)
{

    uint8_t i;
    uint32_t PortI2c;

    if (PORTI2C == 0)
        PortI2c = I2C0_BASE;
    else
        if (PORTI2C == 1)
            PortI2c = I2C1_BASE;
        else
            PortI2c = I2C2_BASE;
    // Tell the master module what address it will place on the bus when
    // communicating with the slave.
    I2CMasterSlaveAddrSet(PortI2c, slave_addr, false);

    //stores list of variable number of arguments
    va_list vargs;

    //specifies the va_list to "open" and the last fixed argument
    //so vargs knows where to start looking
    va_start(vargs, num_of_args);

    //put data to be sent into FIFO
    I2CMasterDataPut(PortI2c, va_arg(vargs, uint32_t));

    //if there is only one argument, we only need to use the
    //single send I2C function
    if(num_of_args == 1)
    {
        //Initiate send of data from the MCU
        I2CMasterControl(PortI2c, I2C_MASTER_CMD_SINGLE_SEND);

        // Wait until MCU is done transferring.
        while(I2CMasterBusy(PortI2c));

        //"close" variable argument list
        va_end(vargs);
    }

    //otherwise, we start transmission of multiple bytes on the
    //I2C bus
    else
    {
        //Initiate send of data from the MCU
        I2CMasterControl(PortI2c, I2C_MASTER_CMD_BURST_SEND_START);

        // Wait until MCU is done transferring.
        while(I2CMasterBusy(PortI2c));

        //send num_of_args-2 pieces of data, using the
        //BURST_SEND_CONT command of the I2C module
        for(i = 1; i < (num_of_args - 1); i++)
        {
            //put next piece of data into I2C FIFO
            I2CMasterDataPut(PortI2c, va_arg(vargs, uint32_t));
            //send next data that was just placed into FIFO
            I2CMasterControl(PortI2c, I2C_MASTER_CMD_BURST_SEND_CONT);

            // Wait until MCU is done transferring.
            while(I2CMasterBusy(PortI2c));
        }

        //put last piece of data into I2C FIFO
        I2CMasterDataPut(PortI2c, va_arg(vargs, uint32_t));
        //send next data that was just placed into FIFO
        I2CMasterControl(PortI2c, I2C_MASTER_CMD_BURST_SEND_FINISH);
        // Wait until MCU is done transferring.
        while(I2CMasterBusy(PortI2c));

        //"close" variable args list
        va_end(vargs);
    }
}

void I2C2Send(uint8_t slave_addr, uint8_t num_of_args, ...)
{

    uint8_t i;
    // Tell the master module what address it will place on the bus when
    // communicating with the slave.
    I2CMasterSlaveAddrSet(I2C2_BASE, slave_addr, false);

    //stores list of variable number of arguments
```

```
va_list vargs;

//specifies the va_list to "open" and the last fixed argument
//so vargs knows where to start looking
va_start(vargs, num_of_args);

//put data to be sent into FIFO
I2CMasterDataPut(I2C2_BASE, va_arg(vargs, uint32_t));

//if there is only one argument, we only need to use the
//single send I2C function
if(num_of_args == 1)
{
    //Initiate send of data from the MCU
    I2CMasterControl(I2C2_BASE, I2C_MASTER_CMD_SINGLE_SEND);

    // Wait until MCU is done transferring.
    while(I2CMasterBusy(I2C2_BASE));

    //"close" variable argument list
    va_end(vargs);
}

//otherwise, we start transmission of multiple bytes on the
//I2C bus
else
{
    //Initiate send of data from the MCU
    I2CMasterControl(I2C2_BASE, I2C_MASTER_CMD_BURST_SEND_START);

    // Wait until MCU is done transferring.
    while(I2CMasterBusy(I2C2_BASE));

    //send num_of_args-2 pieces of data, using the
    //BURST_SEND_CONT command of the I2C module
    for(i = 1; i < (num_of_args - 1); i++)
    {
        //put next piece of data into I2C FIFO
        I2CMasterDataPut(I2C2_BASE, va_arg(vargs, uint32_t));
        //send next data that was just placed into FIFO
        I2CMasterControl(I2C2_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);

        // Wait until MCU is done transferring.
        while(I2CMasterBusy(I2C2_BASE));
    }

    //put last piece of data into I2C FIFO
    I2CMasterDataPut(I2C2_BASE, va_arg(vargs, uint32_t));
    //send next data that was just placed into FIFO
    I2CMasterControl(I2C2_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
```

```
        // Wait until MCU is done transferring.
        while(I2CMasterBusy(I2C2_BASE));

        //"close" variable args list
        va_end(vargs);
    }
}



#define LCD_SLAVE_ADDR 0x3C // L'adresse semble être 0x3C
#define LCD_ON 0x4f
#define LCD_OFF 0x4e
#define PUMP    0x14 //0x8d


void mydelay(int duration)
{
    int i;
    while(duration--)
     for (i=0;i<1000;i++);
}

void WriteCmd(int cmd)
{
    I2CSend(LCD_SLAVE_ADDR, 2,0x0, (char) cmd);  // 00 indique le mode commande
}

void WriteData(int data)
{
    I2CSend(LCD_SLAVE_ADDR, 2,0x40, (char) data);  // 0x40 indique le mode data
}



void SetPageAddress(uint8_t add) {
    add &= 0x7;
    add=0xb0|add;
    WriteCmd(add);
    return;
}

void SetColumnAddress(uint8_t add) {
        add += 2;
        WriteCmd((0x10|(add>>4)));
        WriteCmd((0x0f&add));
        return;
}
```

```
 void Fill(int val) {

    int i,j;
    for (i=0;i<8; i++) {
        SetPageAddress(i);
        SetColumnAddress(0);
        for (j=0; j<128; j++) {
            WriteData(val);
        }
    }
}
#define NORMALDISPLAY                   0xA6
#define INVERTDISPLAY                   0xA7
#define ACTIVATESCROLL                  0x2F
#define DEACTIVATESCROLL                0x2E
#define SETVERTICALSCROLLAREA           0xA3
#define RIGHTHORIZONTALSCROLL           0x26
#define LEFT_HORIZONTALSCROLL           0x27
#define VERTICALRIGHTHORIZONTALSCROLL   0x29
#define VERTICALLEFTHORIZONTALSCROLL    0x2A

void InvertDisplay(int inv) {
    if (inv)
    WriteCmd(INVERTDISPLAY);
    else
    WriteCmd(NORMALDISPLAY);
}

void Display(unsigned char *p) {
    int i, j;

    for (i=0; i<8; i++) {
        SetPageAddress(i);
        SetColumnAddress(0);
        for (j=127;j>=0;j--) {
            WriteData(p[i*128+j]);
        }
    }
}




void InitScreen(void)
```

```
{
   WriteCmd(0xAE); // Set display OFF

   WriteCmd(0xD4); // Set Display Clock Divide Ratio / OSC Frequency
   WriteCmd(0x80); // Display Clock Divide Ratio / OSC Frequency

   WriteCmd(0xA8); // Set Multiplex Ratio
   WriteCmd(0x3F); // Multiplex Ratio for 128x64 (64-1)

   WriteCmd(0xD3); // Set Display Offset
   WriteCmd(0x00); // Display Offset

   WriteCmd(0x40); // Set Display Start Line 0

   WriteCmd(0x8D); // Set Charge Pump
   WriteCmd(0x14); // Charge Pump (0x10 External, 0x14 Internal DC/DC)

   WriteCmd(0xA0); // Set Segment Re-Map
   WriteCmd(0xC8); // Set Com Output Scan Direction

   WriteCmd(0xDA); // Set COM Hardware Configuration
   WriteCmd(0x12); // COM Hardware Configuration

   WriteCmd(0x81); // Set Contrast
   WriteCmd(0xCF); // Contrast

   WriteCmd(0xD9); // Set Pre-Charge Period
   WriteCmd(0xF1); // Set Pre-Charge Period (0x22 External, 0xF1 Internal)

   WriteCmd(0xDB); // Set VCOMH Deselect Level
   WriteCmd(0x40); // VCOMH Deselect Level


   WriteCmd(0xb0);




   WriteCmd(0xA4); // Set all pixels OFF
   WriteCmd(0xA6); // Set display not inverted
   WriteCmd(0xAF); // Set display On


}
#define SX  16
#define EX  96

#define SY  3
#define EY  5
void    DrawTest()
```

```
{
    int j;
SetPageAddress(SY);
SetColumnAddress(SX);
for (j=SX; j<EX; j++)
      WriteData(0x80);
SetPageAddress(4);
SetColumnAddress(SX);
WriteData(0x255);
SetColumnAddress(EX);
WriteData(0x255);
SetPageAddress(EY);
SetColumnAddress(SX);
for (j=SX; j<EX; j++)
      WriteData(0x1);
}

void DisplayCarac(int x,int y,char c)
{
    int i,j;
    x = 128-x;
    SetPageAddress(y);
    SetColumnAddress(x-5);
    for(i=4;i>=0;i--)
    {
        j = fontData[5*(c-32)+i];
        WriteData(j);
    }
}

void DisplayString(int x,int y,char *s)
{

       while (*s)
       {
        DisplayCarac(x,y,*s);
        x += 6;
        s++;
       }
}
void DisplayTemperature(int x, int y, float temp) {
    char tempString[10];  // Adjust the size based on your needs
    sprintf(tempString, "%.2f", temp);  // Format the floating-point value
 DisplayString(x, y, tempString);

}

void setup() {
  InitI2C();
  InitScreen();
  Display(motif);
  pinMode(lm35, INPUT);
  pinMode(audio, INPUT);
  pinMode(PWM_PIN, INPUT);
  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(BLUE_LED, OUTPUT);
  Serial.begin(9600);
  Serial1.begin(9600);
}

void loop() {
 {
  tamp = analogRead(lm35);

  float tamp_C = ((tamp*3.3/4096.0)*100/10);
  float tamp_F = tamp_C*(9/5)+32;
  if (tamp_C > 3.00) {
    digitalWrite(RED_LED, HIGH);
    digitalWrite(GREEN_LED, LOW);
    digitalWrite(BLUE_LED, LOW);
  }
 else if ( 0.00 <= tamp_C <= 3.00) {
    digitalWrite(RED_LED, LOW);
    digitalWrite(GREEN_LED, HIGH);
    digitalWrite(BLUE_LED, LOW);
  }
 else if( tamp_C < 0.00){
    digitalWrite(RED_LED, LOW);
    digitalWrite(GREEN_LED, LOW);
    digitalWrite(BLUE_LED, HIGH);
  }
 float temp = tamp_C;
  Serial.print("Tempetature is:  ");
  Serial.print(tamp_F);
  Serial1.print("Tempetature is:  ");
  Serial1.print(tamp_F);
  Serial.println("°F  ");
  Serial1.println("°F  ");
  Serial.print("Tempetature is:  ");
  Serial1.print("Tempetature is:  ");
  Serial.print(tamp_C);
  Serial1.print(tamp_C);
  Serial.println("°C  ");
  Serial1.println("°C  ");
  DisplayString(0,0,"Tempetature F:");
 DisplayTemperature(100, 0, temp);
  delay(1000);
}
  {
```

```
// Reading the audio signal from analog pin A0
 for (int i = 0; i < SAMPLE_SIZE; i++) {
   signal[i] = analogRead(audio);  // Adjust the pin according to your setup
   delayMicroseconds(1000000 / SAMPLE_RATE);  // Pause to adhere to the sampling rate

 }

 // Signal processing code

 int i, j;
 int average = 0;
 float power;
 float maxPower = 0;
 int index = 0;

 // Calculating the average value of the signal
 for (i = 0; i < SAMPLE_SIZE; i++)
   average += signal[i];
 average /= SAMPLE_SIZE;
 printf("Average : %d\n", average);
  Serial.print("Average :");
  Serial1.print("Average :");
 Serial.print(average);
 Serial1.print(average);
  Serial.println();
  Serial1.println();
 // Analyzing the signal using sliding windows
 for (i = 0; i < SAMPLE_SIZE; i += WINDOWS) {
   power = 0.0;

   // Calculating power for each window
   for (j = 0; j < WINDOWS; j++) {
     power = (float)((signal[i + j] - (float)average) / SCALEOVERTWO);
     power = power * power;
   }

   // Updating the maximum power peak
   if (power > maxPower) {
     maxPower = power;
     index = i;
   }
 }

 // Displaying the results
 Serial.print("Max Power : ");
 Serial1.print("Max Power : ");
  Serial.println(maxPower);
   Serial1.println(maxPower);
   Serial.print("Index : ");
   Serial1.print("Index : ");
```

```
    Serial.println(index);
    Serial1.println(index);
   printf("Index : %d Max Power : %f\n", index, maxPower);
//  getc(stdin);

  // Waiting between readings (adjust as needed)
  delay(1000);
}
{
unsigned long startMillis = millis(); // Start of sample window
  unsigned int peakToPeak = 0;   // peak-to-peak level

  unsigned int signalMax = 0;
  unsigned int signalMin = 4095;

  // collect data for 50 mS and then plot data
  while (millis() - startMillis < sampleWindow)
  {
    sample = analogRead(audio);
    if (sample < 4096)  // toss out spurious readings
    {
      if (sample > signalMax)
      {
        signalMax = sample;  // save just the max levels
      }
      else if (sample < signalMin)
      {
        signalMin = sample;  // save just the min levels
      }
    }
  }
     peakToPeak = signalMax - signalMin;  // max - min = peak-peak amplitude
  Serial.print("peakTopeak=");
  Serial1.print("peakTopeak=");
  Serial.println(peakToPeak);
Serial1.println(peakToPeak);
  Serial.print("Volts=");
  Serial1.print("Volts=");
  double volts = (peakToPeak * 3.3) / 4096;  // convert to volts
  Serial.println(volts);
  Serial1.println(volts);
}
  {
    unsigned long ppm; // adding for clarity

duration = pulseIn(PWM_PIN, HIGH); // get the raw reading
ppm = duration - 177000UL; // subtract the 'zero point'
ppm /= 500UL; // divide by the gain
ppm += 350UL; // add the zero point back in
```

```
        Serial.print(" PPM Co2: ");
        Serial.print(ppm,DEC);
        //Serial.print(" Raw Sensor: ");
        //Serial.println(duration,DEC);
        Serial1.print(" PPM Co2: ");
        Serial1.print(ppm,DEC);
        //Serial1.print(" Raw Sensor: ");
        //Serial1.println(duration,DEC);
         // Adjust the delay as needed
        //  int pwmValue = pulseIn(PWM_PIN, HIGH); // Read the pulse width in microseconds
        //
        //  // Print the PWM value to the Serial Monitor
        //  Serial.print("PWM Value: ");
        //  Serial1.print("PWM Value: ");
        //  Serial.println(pwmValue);
        //  Serial1.println(pwmValue);
        //
        //  // Print the corresponding CO2 equivalent levels
        //  Serial.print("CO2 equ (ppm): ");
        //  if (pwmValue >= 0 && pwmValue < 5500) {
        //    Serial.println(400);
        //    Serial1.println(400);
        //  } else if (pwmValue >= 5500 && pwmValue < 7070) {
        //    Serial.println(1027);
        //    Serial1.println(1027);
        //  } else if (pwmValue >= 7071 && pwmValue < 8640) {
        //    Serial.println(1654);
        //    Serial1.println(1654);
        //  } else if (pwmValue >= 9500) {
        //    Serial.println(2000);
        //     Serial1.println(2000);
        //  }
        //
        //  delay(1000); // Adjust the delay as needed
        }
        Serial.println("");
        Serial1.println("");
        delay(2000);
}
```