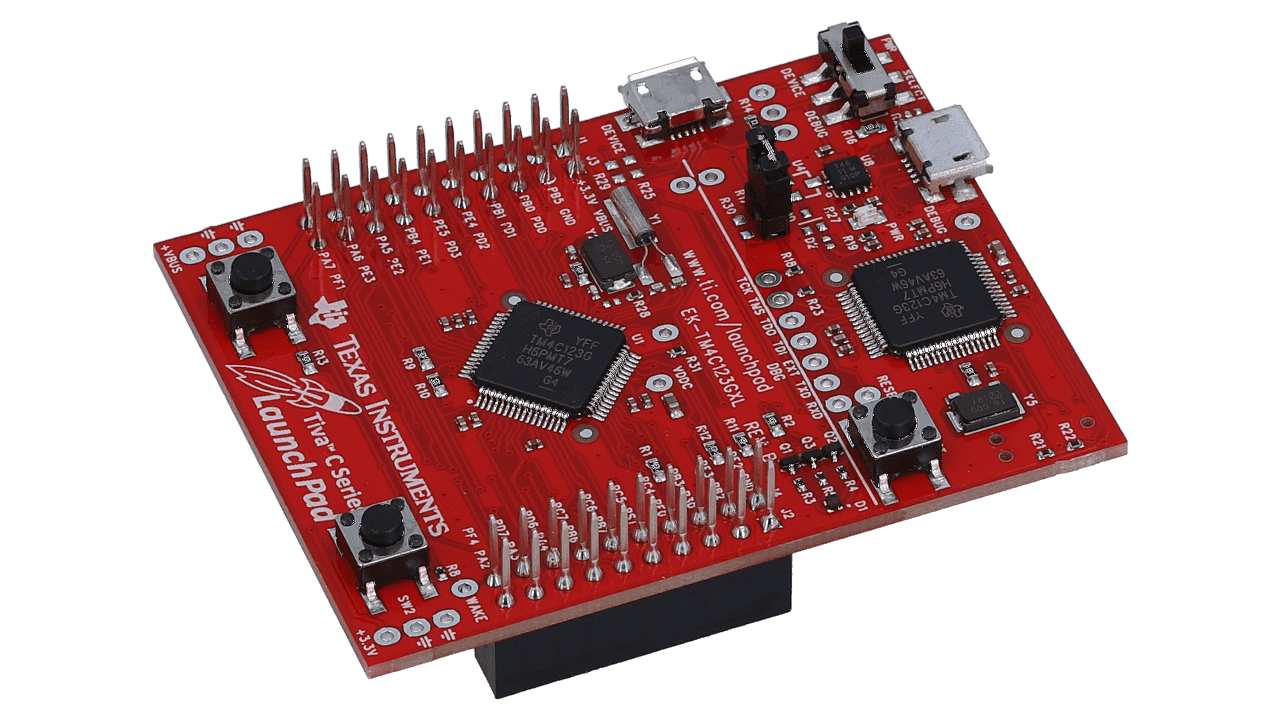
**AASTMT**

*Arabic Academy for Science, Technology and Maritime Transport*

**College of Engineering / Electronics and Communications Department   
*Intro. To Microprocessor***

**Smart House**

(Via the TM4C123G Launchpad)

**Participants Names:**

Team A: Team B:

1. Ahmad Adham Badawy **Reg. No.** /231018156 6. Mohamed Sayed **Reg. No.** /231008761

2. Ali Abd El Nasser Ali **Reg. No.** /231018210 7. Abdelrahman Hamdy **Reg. No.** /231018182

3. Abdallah Fahmy Rabea **Reg. No.** /231008522 8. Mohammed Ehab Badr **Reg. No.** /231008607

4. Abdelrahman Mostafa **Reg. No.** /231008579 9. Mostafa Roshdy **Reg. No.** /222008507

5. Eslam Mohammed **Reg. No.** /231018107

**Table of Contents**

1. **Introduction 3**
   1. What is Tiva C Launchpad?
   2. Why It Is Useful for Our Project?
   3. Why did we use ESP-32S?
   4. Brief Summary on the project Idea
   5. Project Vision
2. **Tools & Components 5**
   1. Motion Sensor (PIR) 2.8. LCD 16x2 (I2C)
   2. LDR 2.9. Mini Submersible Water Pump
   3. Laser Transmitter 2.10. 2N2222 – NPN Transistor
   4. Buzzer 2.11. Soil Moisture Sensor
   5. Temperature Sensor (LM35) 2.12. UltraSonic Sensor
   6. Fire Sensor 2.13. ESP-32S
   7. Bi-Directional Logic Level Shifter
3. **Smart House Concept/Idea 8**
   1. Project Design Vision
   2. Drafts
   3. Final Design
4. **Planning Wise X**
   1. House Electronics Infrastructure (Interior & Exterior)
   2. Demo vs Theoretical
5. **Coding Wise X**
   1. TM4C123G (Launchpad) & ESP-32S Codes
   2. Challenges & Solutions
6. **Conclusion X**
   1. Project Accomplishments
   2. Future Improvements
7. **References X**

**What is Tiva C Launchpad?**

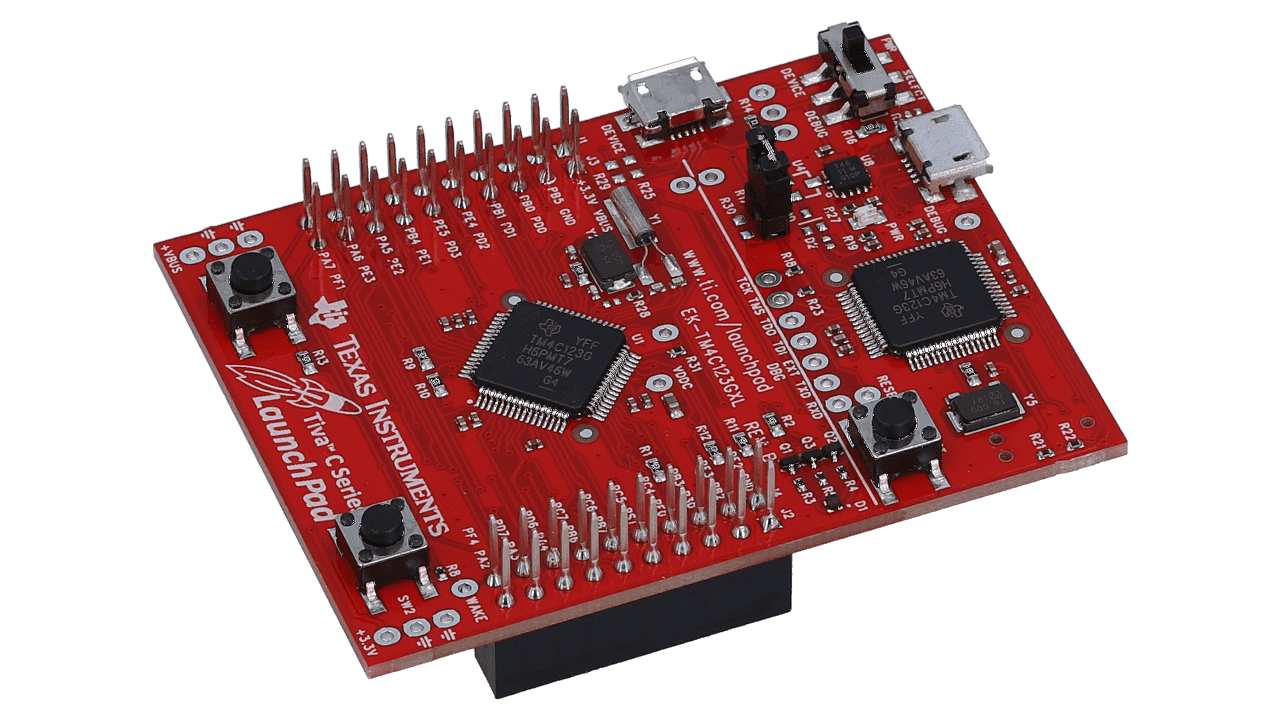
The Tiva C Launchpad (Specifically the TM4C123GXL Model) **(Figs.1,2,3)** is a development board by Texas Instruments based on an ARM Cortex-M4 microcontroller. It's designed for embedded systems projects, offering features like GPIOs, communication interfaces (UART, SPI, I2C), timers, ADCs, and a USB port. It's low-cost, easy to program, and ideal for learning, prototyping IoT devices, robotics, automation, and real-time applications.

Figure 1 - Tiva C Launchpad

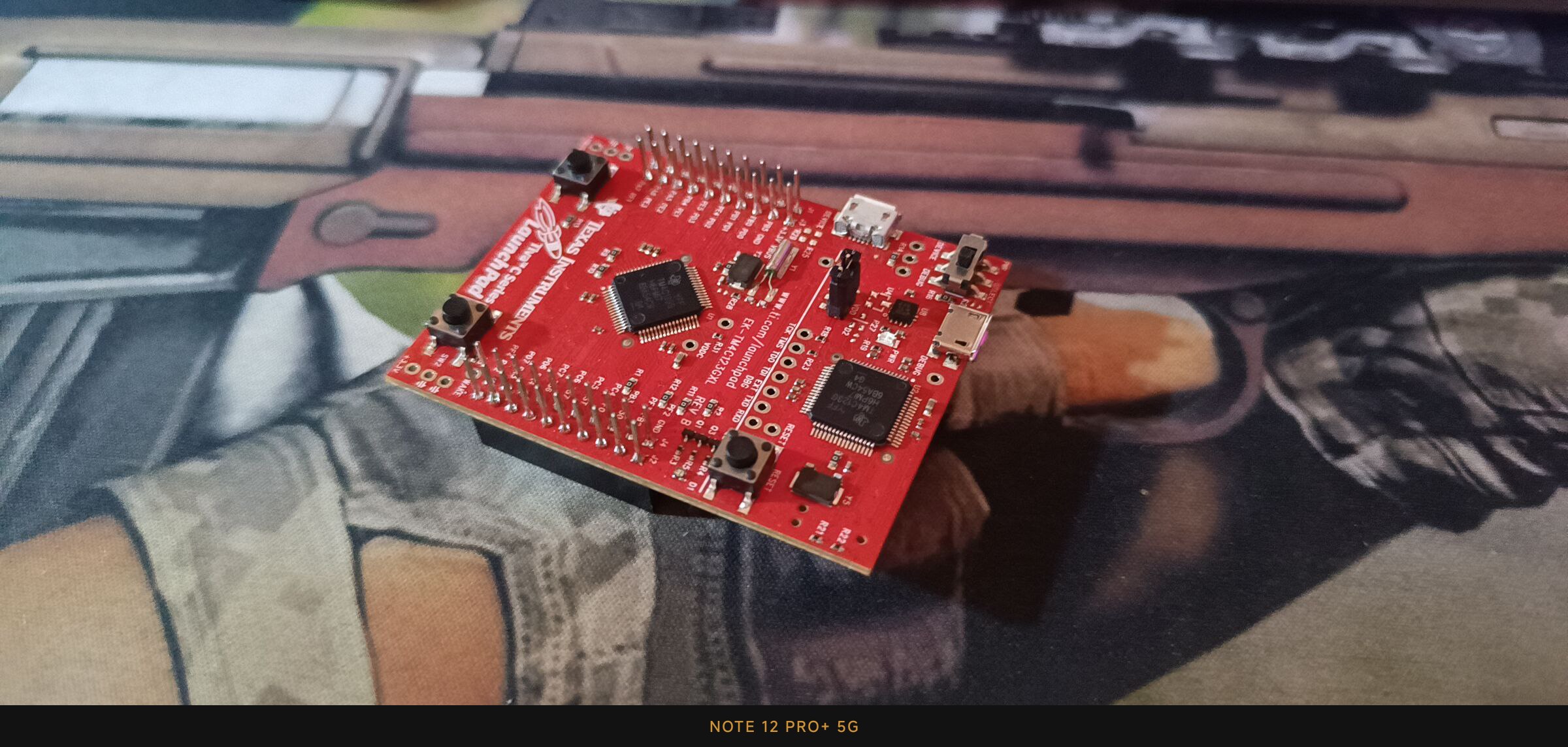


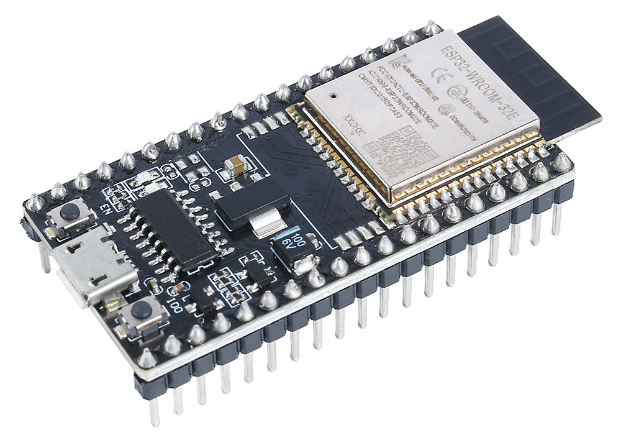
Figure 3 - Tiva C Launchpad IRL

Figure 2 - Tiva C Launchpad IRL

**Why It Is Useful for Our Project?**

* It offers powerful processing capabilities with low power consumption.
* Easy to program and debug, reducing development time.
* Affordable yet provides professional-grade features.
* **Scalable:** It allows easy expansion with external modules.
* **Community Support:** Plenty of documentation, example codes, and libraries are available

**Why did we use ESP-32S?**

 We did use the ESP-32S **(Fig.4)** because of its wireless capabilities and its ability to interact with it using the WIFI or Bluetooth, so we did use it to make a web interface to control the system and receive information about the smart house status. It is interactable via any device that can access the internet and connect to the ESP-32S’s WIFI network.

**Project Vision**

Figure 4 - ESP-32S

The vision of this project is to create a modular, low-cost, and scalable smart house system that enhances home automation, safety, and environmental efficiency. By integrating sensor-based automation, wireless connectivity, and real-time monitoring, the system aims to provide homeowners with greater control, security, and resource management both locally and remotely. The goal is to develop a prototype that can be expanded into a fully functional, user-friendly smart home solution capable of adapting to different environments and user needs.

**Brief Summary on the project Idea**

This project presents the design of a smart house system controlled by the TM4C123G Tiva C Launchpad. The system includes monitoring and automation for both the outside and inside of the house. The outside features a garden with an automated watering system and a water tank with level indicators, along with an LCD screen to monitor the outdoor conditions. The inside consists of four rooms, each demonstrating part of a complete sensor network for security, environment monitoring, and automation. A local Wi-Fi network hosted by an ESP32 enables remote monitoring and control through a web-based dashboard. The project demonstration simplifies the full system design while maintaining the theoretical concept of a fully automated smart house.

**Motion Sensor (PIR)**

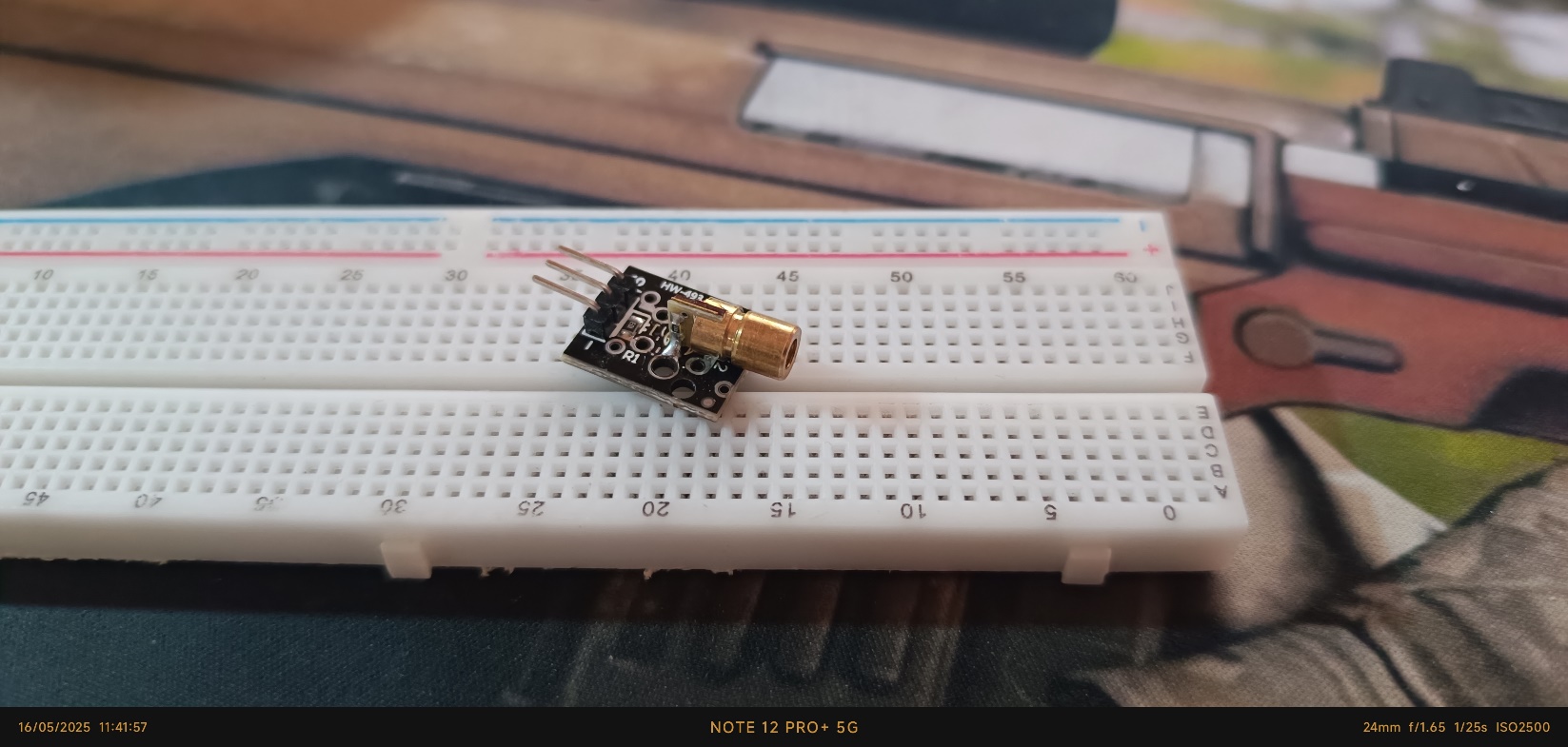
The Passive Infrared (PIR) sensor **(Fig.5)** detects motion by sensing infrared radiation changes caused by moving objects like humans or animals. It is commonly used in security systems or automated lighting to detect presence or activity.

Figure 5 - PIR Sensor IRL

**LDR (Light Dependent Resistor)**

An LDR **(Fig.6)** is used to detect the intensity of light in the environment. Its resistance changes based on the amount of light falling on it. This makes it useful for detecting day or night conditions or controlling light-based triggers in the system.

Figure 6 - LDR IRL

**Laser Transmitter**

The laser module **(Figs.7,8)** emits a focused beam of light, which can be used for alignment, obstacle detection, or object counting when paired with a light sensor. In some cases, it helps detect intrusions by monitoring if the beam is interrupted.

Figure 7 - Laser Module IRL

Figure 8 - Laser Module IRL

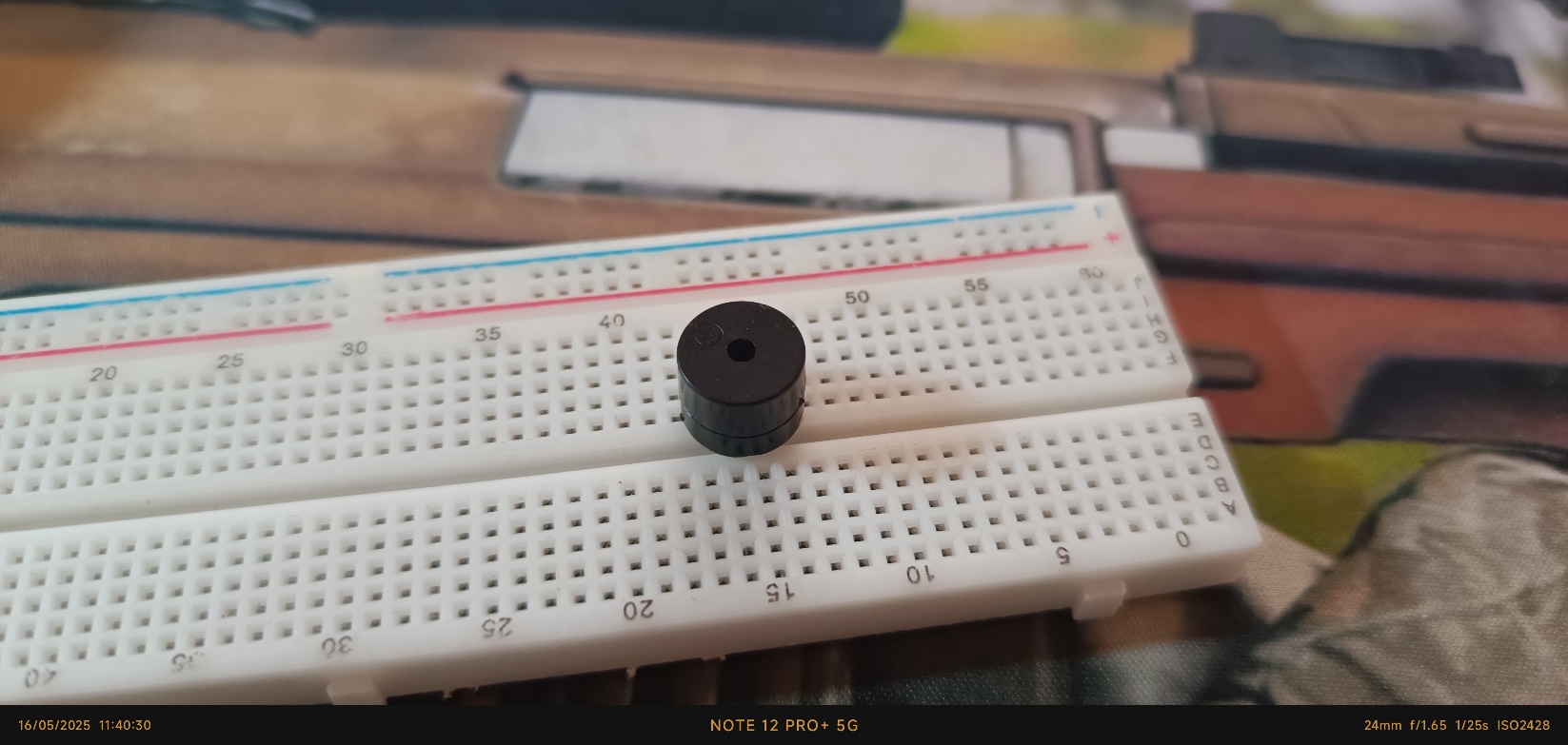
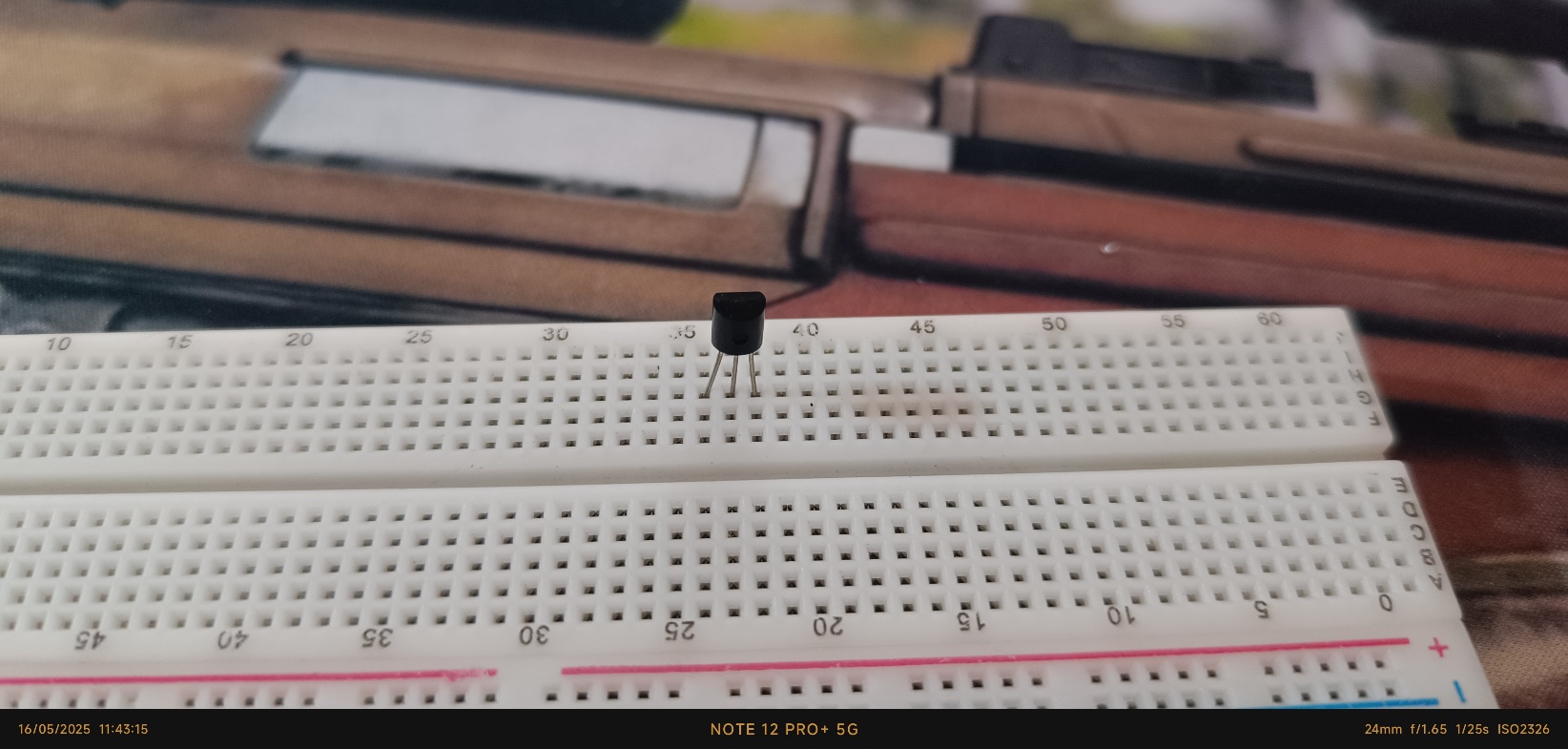
**Buzzer**

Figure 9 - Buzzer IRL

A buzzer **(Fig.9)** is used as an output device to produce sound or alerts. It can be triggered in response to specific conditions such as detecting motion, fire, or low moisture levels, providing immediate audible feedback to users.

**Temperature Sensor (LM35)**

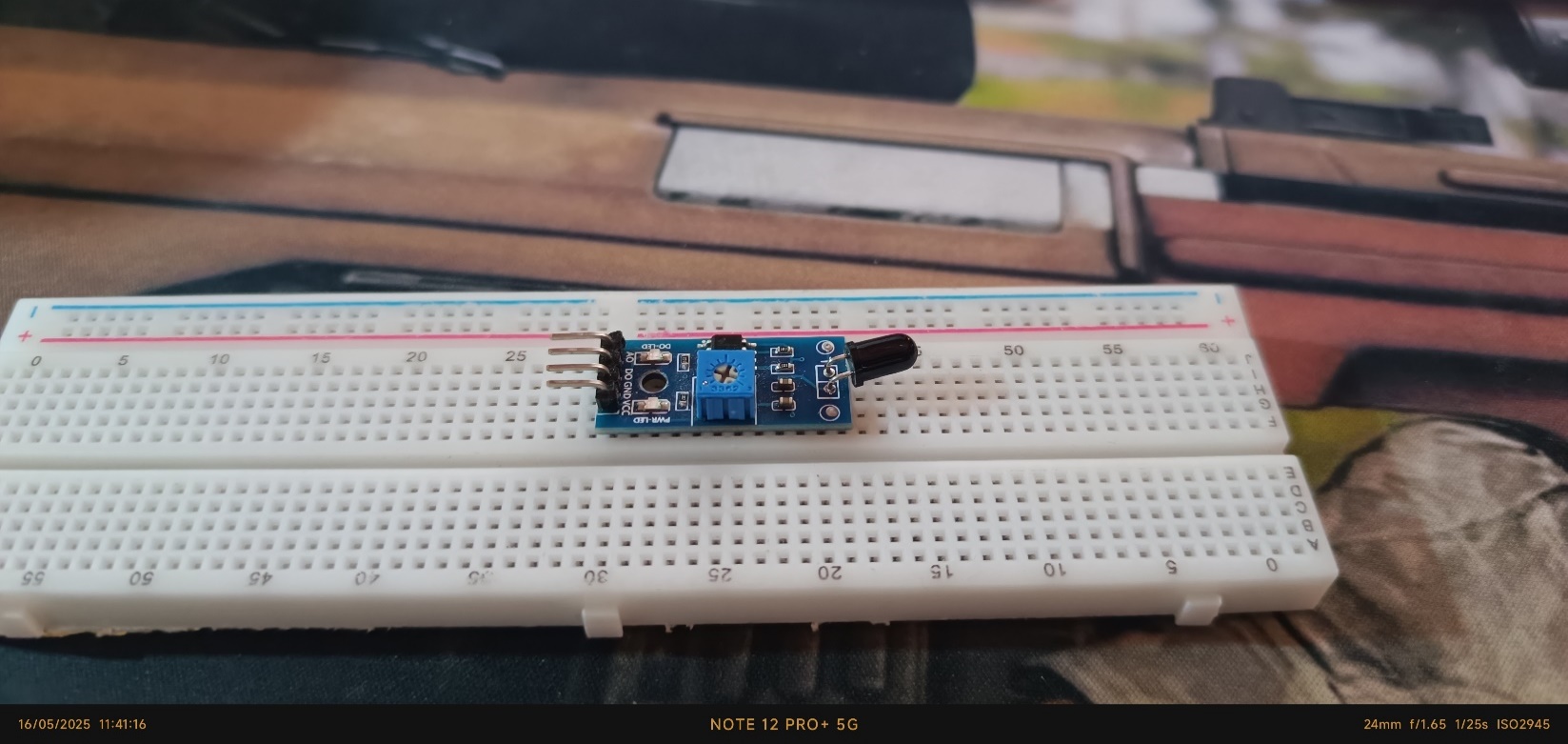
The LM35 **(Fig.10)** is an analog temperature sensor that outputs a voltage directly proportional to the ambient temperature. It is used for monitoring environmental conditions and triggering actions when temperature thresholds are exceeded.

Figure 11 - Fire Sensor IRL

Figure 10 - LM35 IRL

**Fire Sensor**

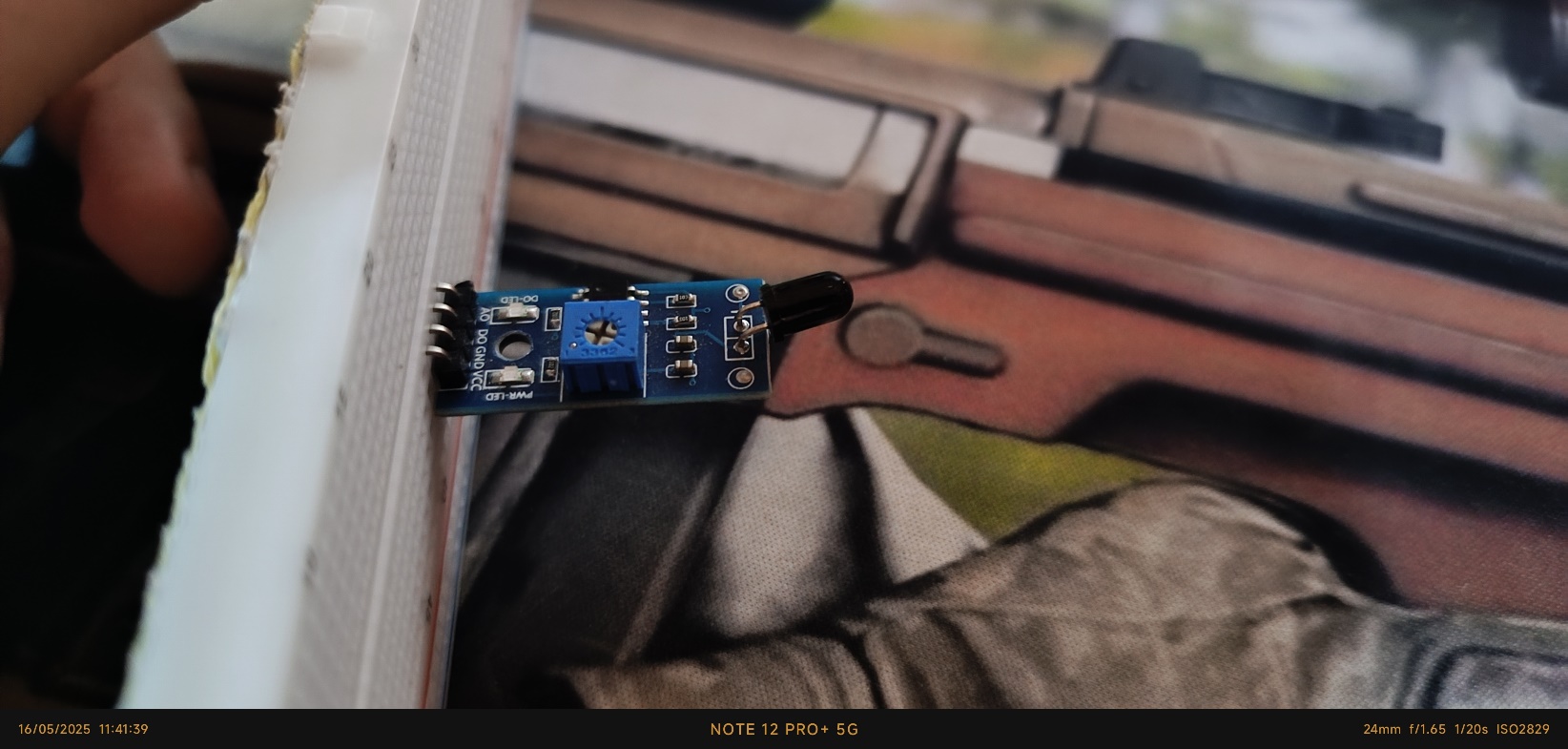
The fire sensor **(Figs.11,12)** detects the presence of flame using infrared or ultraviolet light emitted by fire. It is a critical component for fire detection systems, providing an early warning signal to prevent hazards.

Figure 12 - Fire Sensor IRL

### **Bi-Directional Logic Level Shifter**

A bi-directional logic level shifter **(Fig.13)** is used to safely interface devices operating at different voltage levels. In embedded systems, it’s common to have components like sensors, microcontrollers, or communication modules that work on different logic voltages—such as 5V and 3.3V. Directly connecting them could damage the lower-voltage components or lead to unreliable operation.

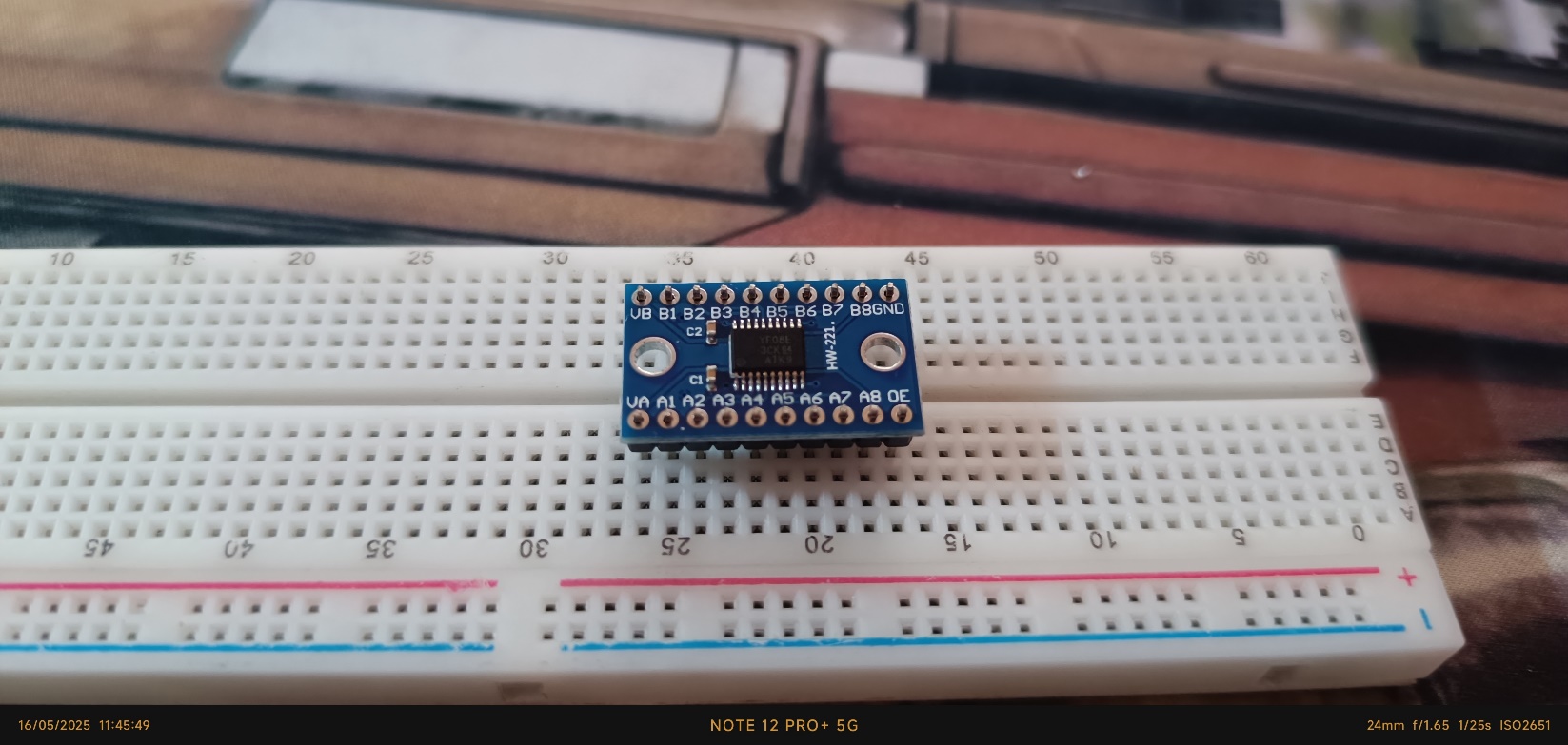
****The logic level shifter acts as a translator, converting signals from one voltage level to another in both directions. This is especially useful in I²C or SPI communication, where devices need to send and receive data across mixed-voltage systems.

Figure 13 - Bi-Directional Logic Level Shifter IRL

**LCD 16x2 with I2C Module**

The 16x2 Liquid Crystal Display **(Fig.14)** is used to visually present real-time data such as sensor readings, status messages, or system alerts. The I2C module simplifies wiring and communication with microcontrollers like the ESP32.

**Mini Submersible Water Pump**

This component **(Fig.15)** is responsible for pumping water in small-scale applications, such as plant irrigation. It can be controlled via a microcontroller to automate watering based on soil moisture sensor readings.

**2N2222 NPN Transistor**

The 2N2222 transistor (**Fig.16)** is used as an electronic switch or amplifier in the tank system. It enables the control of higher current devices like the water pump using low current signals from the microcontroller.

**Soil Moisture Sensor**

This sensor **(Fig.17)** is used to measure the water content in soil. It helps monitor the soil’s moisture level, which is essential for automatic irrigation systems or agricultural monitoring. By comparing the sensor’s readings to a preset threshold, the system can decide when to activate a water pump.

**Ultrasonic Sensor**

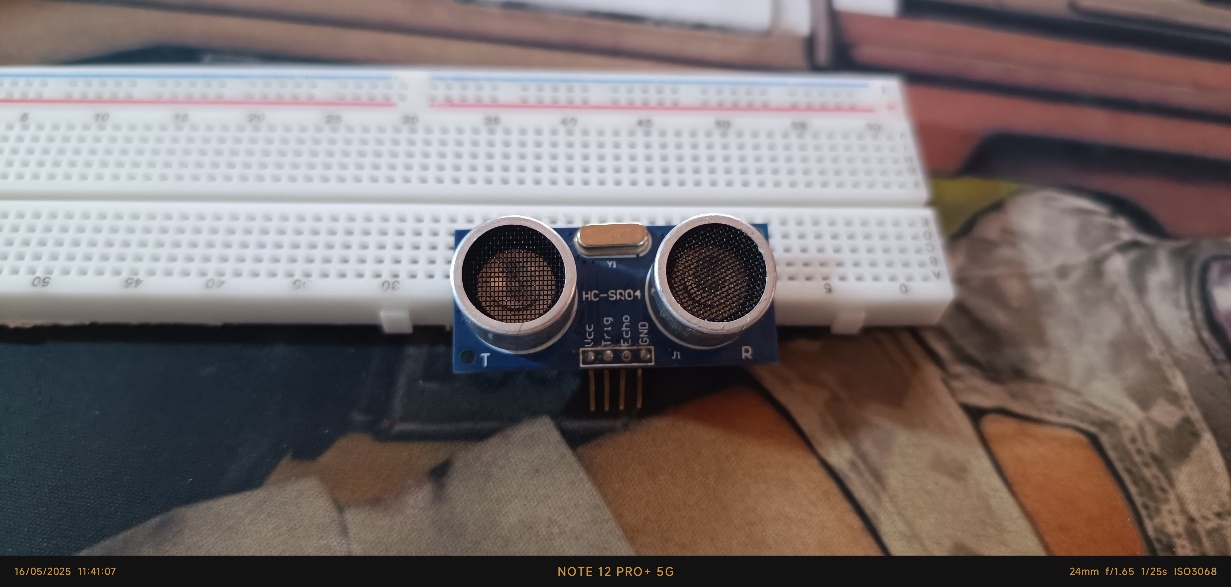
The ultrasonic sensor **(Figs.18,19)** measures distance by emitting sound waves and calculating the time it takes for the echo to return. It is useful for obstacle detection, object distance measurement, or liquid level sensing.

Figure 19 - Ultrasonic Sensor IRL

Figure 18 - Ultrasonic Sensor IRL

**ESP-32S**

The ESP-32S is a versatile microcontroller known for its built-in Wi-Fi and Bluetooth capabilities. It is commonly used in IoT and smart automation projects due to its low power consumption, high processing speed, and wireless communication support. In general, it allows embedded systems to connect to networks, interact with cloud services or local interfaces, and exchange data with other devices wirelessly, making it ideal for remote monitoring and control applications.

Figure 21 - ESP-32S

Figure 20 - ESP-32S IRL

**Project Design Vision**

The vision behind this smart house system **(Optional REVIT)** is to build a modular, low-cost, and efficient home automation solution that enhances everyday living through improved safety, environmental monitoring, and user convenience. The core idea is to use widely available components and microcontrollers to create a prototype that is not only functional but also scalable for future development.

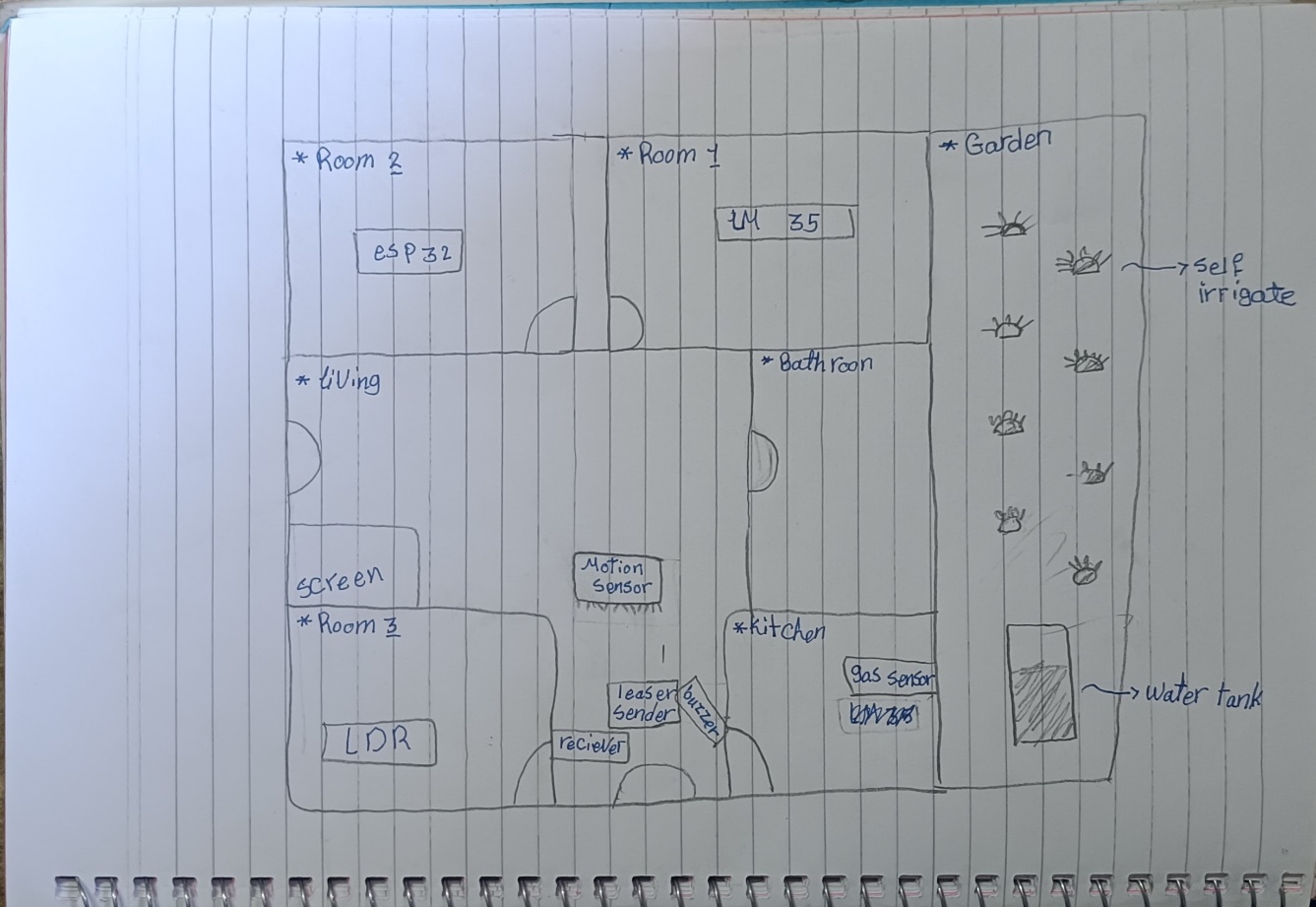
At its heart, the system is divided into two primary roles: the Tiva-C microcontroller acts as the master controller, interfacing directly with sensors and actuators to manage real-time home conditions and trigger automated responses. Meanwhile, the ESP32 microcontroller serves as a wireless communication and user interface layer, hosting a local web dashboard accessible from any Wi-Fi-enabled device. This separation of responsibilities allows for smoother task distribution, improved performance, and easier debugging.

The goal is to offer users a reliable method to monitor and control home conditions locally, while still having the foundation to expand into remote access and advanced AI-driven features in the future. This approach emphasizes user-friendliness, hardware reusability, and flexibility, making it adaptable to various home environments and user needs.

**Design Philosophy and Approach:**

* **Modularity by Design:** The system is intentionally divided into distinct modules—sensing, control, communication, and user interface. This modular architecture allows each part to be developed, tested, or replaced independently. If any sensor or output component fails or needs upgrading, the rest of the system remains operational.
* **Dual Microcontroller Setup (Tiva + ESP32):** By using two microcontrollers with different responsibilities, the system avoids overloading a single unit with all tasks. The Tiva-C handles critical real-time tasks such as sensor data processing and actuator control. The ESP32 focuses solely on communication and interface, enabling smoother user interactions without interfering with time-sensitive operations.
* **Web Dashboard Interface:** A simple web-based interface hosted on the ESP32 provides users with real-time status updates and control options without the need for cloud-based servers or mobile apps. This choice increases privacy, reduces dependency on third-party platforms, and ensures quick local responsiveness.
* **Design Constraints and Considerations:** Given the limited time, budget, and experience of the team, the design had to stay simple yet effective. Components were selected based on affordability and availability, and the implementation favored clarity over complexity to ensure that the team could both understand and maintain the system. The web interface was designed to be lightweight to avoid overloading the ESP32, and fake data injection was introduced temporarily to allow testing while the Tiva-C setup was being finalized.

**Drafts**

The 1st draft **(Fig.22)** that was drawn by Abdallah, was the initial vision and plan on how the house should look like, later that plan changed, and some sensors were replaced by others, some sensors were put with each other and so on.

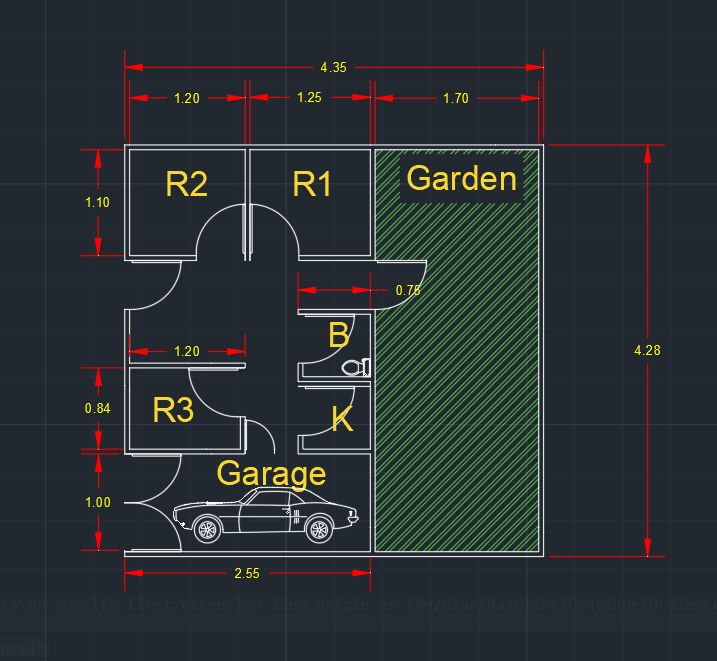
The 2nd draft **(Fig.23)** that was made by Eslam using AutoCAD, was almost like the current plan that we wanted to stick with, but due to limited time and budget we decided to tweak it a little bit, so we made the house less stylish and focused on its functionality more, that led to removing the indoor garage concept, the living room, the doors in general, the toilet and the kitchen room. Although there is no room specifically for the kitchen, we did stick the kitchen sensors in the same room later. We also tried sticking to the written dimensions, but it didn’t work out well.

Figure 23 – 2nd Draft by Eslam

Figure 22 – 1st Draft by Abdallah

**Final Design**

**House Electronics Infrastructure**

This section outlines the electronics infrastructure of the smart house, encompassing both interior and exterior systems. It distinguishes between the theoretical full implementation and the practical demo setup, explaining the rationale behind the distribution of components and the chosen connectivity methods. Additionally, it provides an overview of the communication architecture between the TM4C123G Tiva C Launchpad and the ESP32, which manages data visualization and the user interface.  
To ensure clarity and depth, this section is further divided into several focused sub-sections, each addressing a specific aspect of the system's design and implementation.

* **Theoretical vs Practical Distribution:**

Theoretically, each room inside the house should be equipped with the full set of environment monitoring and safety sensors for maximum coverage, including:

* PIR motion detection
* Light intensity control (LDR)
* Temperature and flame detection (LM35 + flame sensor)
* Sound and visual alerting system (buzzer and RGB LED)

However, due to hardware constraints, component availability, and demo clarity, the sensors were distributed across four different rooms in the demo **(Table.1)**, where each room showcases a different subsystem. This allows the system to demonstrate full functionality without unnecessary component duplication.

|  |  |  |
| --- | --- | --- |
| **Room** | **Sensors/Actuators** | **Purpose** |
| **1** | PIR Motion Sensor, LED 1 | Auto-Lighting based on motion   |  | | --- | |  | |
| **2** | LDR, LED 2 | |  | | --- | |  |   Light dimming based on ambient light |
| **3** | |  | | --- | |  |   LM35, Flame Sensor | |  | | --- | |  |   Temperature/Fire detection |
| **4** | |  | | --- | |  |   LED 3 (RGB), Buzzer, ESP32, Tiva C | Central control + alerts |

Table 1 - Sensor and Actuator Distribution Across Rooms

The entrance corridor includes an LDR + laser module that acts as an intrusion detection system.   
In the exterior, the system is designed around:

* Soil Moisture Sensor to determine if the garden requires watering.
* Submersible Water Pump, controlled via 2N2222 transistor, to draw water from the tank.
* Water level indicators (25%, 50%, 75%, 100%) to monitor tank fullness.
* Ultrasonic sensor in the parking garage to detect proximity of a car, dynamically controlling buzzer frequency and RGB LED color.
* **Electrical Connectivity & Communication:**
  + **Sensor Integration:** All sensors and actuators are directly connected to the TM4C123G Tiva C Launchpad using digital and analog GPIO pins. These inputs are sampled at regular intervals, processed, and converted into a structured JSON object, which includes sensor status and values.
  + **Tiva C to ESP32 Communication:** Communication between the Tiva C Launchpad and the ESP32 is achieved through UART serial communication using the RX/TX pins, allowing real-time transfer of the sensor data in JSON format.
  + **ESP32 Web Dashboard:** The ESP32 hosts a local Wi-Fi Access Point, allowing users to connect and access a dynamic web dashboard. This dashboard visualizes real-time sensor data and system status (e.g., garden soil moisture, room conditions, fire/intrusion alerts, etc.).

In future iterations, this dashboard could be transformed into a mobile application capable of pushing notifications in case of emergencies.

* **Alert System Mapping:**

The alerting system is based on buzzer frequency and RGB LED color, which are coordinated depending on the type of incident in **(Table.2,3,4)**:

|  |  |  |
| --- | --- | --- |
| **Condition** | **RGB LED Color** | **Buzzer Behavior** |
| **High Temp (LM35)** | Orange | Constant tone |
| **Fire Detected** | Crimson Moon Night | Fast pulsing |
| **Intrusion Detected**  **(Laser + LDR)** | Purple | Fast pulsing |
| **Car Approaching**  **(Ultrasonic)** | Yellow to Red | Gradual increase in buzzer frequency |

Table 2 – Buzzer and RGB LED Behavior

|  |  |  |
| --- | --- | --- |
| **Color** | **RGB** | **HEX** |
| **🔶** Orange | 255, 165, 0 | #FFA500 |
| **🔴 Crimson Moon Night** | 138, 12, 33 | #8A0C21 |
| **🟣** Purple | 128, 0, 128 | #800080 |

Table 3 - 🔶 🔴 🟣 Colors RGB & HEX

|  |  |  |  |
| --- | --- | --- | --- |
| **Gradient Steps** | **Event** | **RGB** | **HEX** |
| 🟡 Yellow | Far (Safe Distance) | 255, 255, 0 | #FFFF00 |
| 🟠 Orange - Yellow | Mid-Far | 255, 200, 0 | #FFC800 |
| 🔶 Orange | Mid | 255, 165, 0 | #FFA500 |
| 🔴 Red - Orange | Near | 255, 69, 0 | #FF4500 |
| 🟥 Red | Very Near (Too close) | 255, 0, 0 | #FF0000 |

Table 4 - 🟡🔴 Yellow to Red Gradient (Based on car distance from ultrasonic sensor)

A toggle switch can a be added in future versions to control what type of alert the RGB LED is currently reflecting.

* **Power Supply and Monitoring:**

The system is powered via two USB connections from a laptop:

* One powers the Tiva C Launchpad.
* The other powers the ESP32.

This design also enables real-time serial monitoring for both microcontrollers, simplifying debugging, data logging, and demonstration.

* **Summary:**

This infrastructure strikes a balance between practical constraints and a theoretically scalable smart home system. Each subsystem is modular and independently demonstrable while also integrating into a centralized architecture for real-time monitoring, actuation, and alerts — forming the foundation for a truly intelligent living space.

**Demo vs Theoretical**

In the demo version of the smart house project, each room contains one specific sensor subsystem, carefully chosen to showcase the system's key functionalities individually. This approach simplifies hardware complexity, conserves resources, and provides clearer demonstrations during testing and presentation. For example, the motion sensor is in Room 1, while temperature and fire detection are both implemented together in Room 3 due to their closely related safety functionality.

In contrast, the theoretical implementation envisions a fully equipped smart house where every room is outfitted with a complete suite of sensors and actuators, including:

* Motion detection (PIR sensor)
* Light sensing and dimming (LDR + LED)
* Environmental monitoring (LM35 temperature sensor)
* Safety detection (Flame sensor)
* Alert system (RGB LED + Buzzer)

This comprehensive deployment would ensure consistent automation, monitoring, and alert capabilities throughout the house.

The ESP32 web dashboard in the demo provides basic real-time monitoring of the house status through a local Wi-Fi access point. The theoretical system expands this into a dedicated mobile application, capable of sending push notifications for critical alerts such as fire detection, temperature anomalies, or intrusions, ensuring remote accessibility and user responsiveness.

Regarding visual feedback, the LCD was canceled in the final demo, as the web interface was deemed more versatile and scalable there were many issues with the current LCD due to lack of information and limited time for further testing its capabilities. However, in the full implementation, a small LCD or OLED screen near the main entrance could serve as a quick outdoor status display.

The garden watering system, in the demo, operates based on a soil moisture sensor and a submersible pump drawing from a simulated tank with discrete level indicators (25%, 50%, 75%, 100%). While in practice this setup controls only garden irrigation, the theoretical model assumes the tank is also connected to indoor plumbing systems and has a better way of detecting water levels.

Finally, the parking garage in the demo is placed outside the main house model for practical reasons but still inside the house maquette. Theoretically, it would be integrated into the house structure, as originally planned in the AutoCAD drafts, and its ultrasonic sensor would continuously monitor car distance, feeding data into the alerting system (buzzer frequency and RGB LED color gradient.

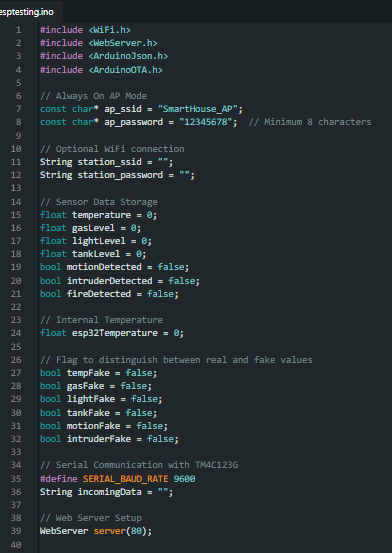
**TM4C123GXL & ESP-32S Coding**

This part of the report will be explained in detail in the project GitHub repository. Scan the following QR or click on the GitHub Logo.



[A black background with a black square

Description automatically generated with medium confidence](https://github.com/A7mad-Alone/smart-house-using-tiva-c-launchpad-tm4c123gxl)

As Before, this coding section will be divided into several parts to explain the whole code.

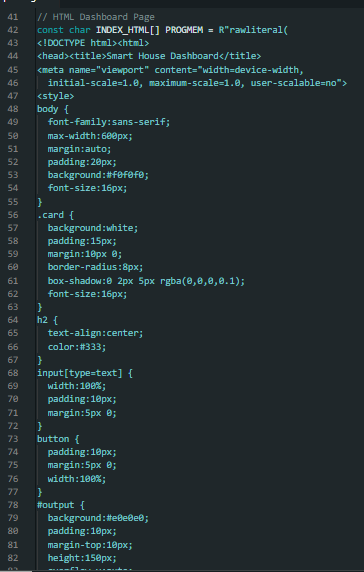
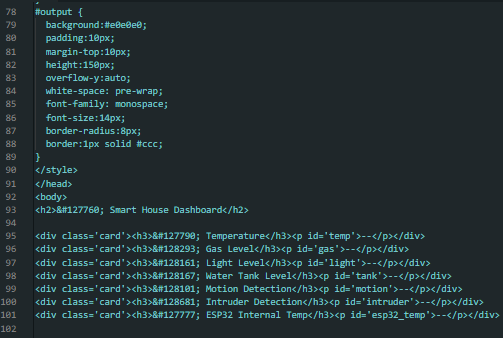
**Code Initialization:**

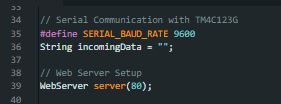
This part of the code (Fig.) contains the libraries used in the Esp32 code, the configuration of the Access Point for the Esp32 which is named “SmartHouse\_AP” with a basic password of “12345678”, there is also the station part which is optional to connect the Esp to your current home network to access the internet or connect the system to the internet.

There is also the part that is responsible for initializing the variables “temperature, gasLevel, lightLevel, tankLevel, motionDetected, intruderDetected, fireDetected” , each of those variables mostly represent a reading for their corresponding sensor, also there is the “esp32Temperature” which is used to monitor the approximate internal temperature.

There is also the initialization of some Booleans that tracks whether the current values of the variables are fake or not, later on we will discuss why this was implemented in the first place.

This part (Fig.) is setting the baud rate for the communication that happens between the Launchpad and the Esp, and initializing a string variable named “incomingData” which will be used later to process the data (parsing) that the Esp received. Also the WebServer line (line 39) is used for the following (Note: this explanation is a networking wise explanation), This line creates an HTTP web server object names “server”, this “server” will listen on port “80” which is the default port for HTTP protocol, this line is actually why we used the esp32 in combination with the launchpad in the first place, to make the web dashboard.

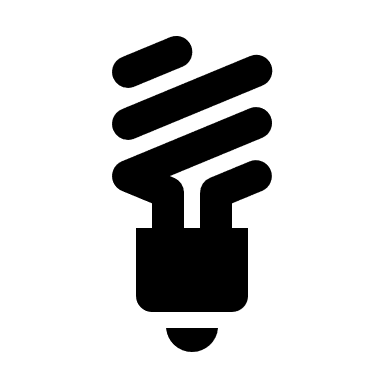
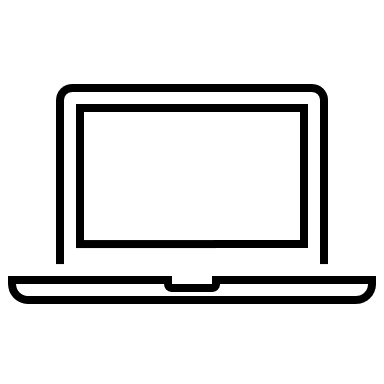
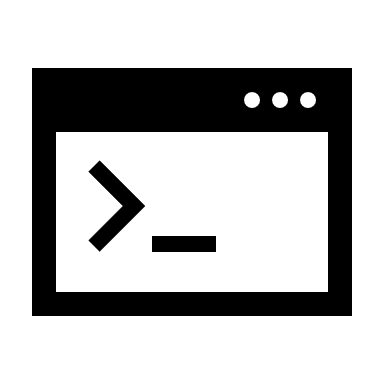
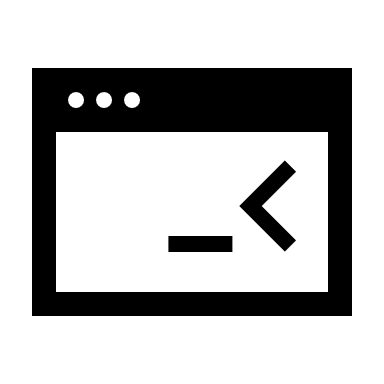
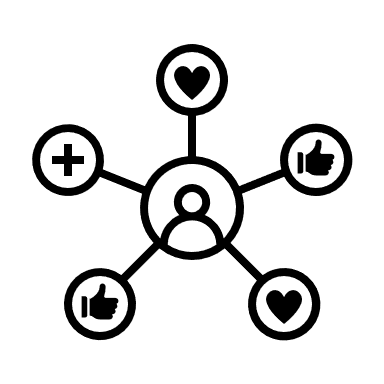
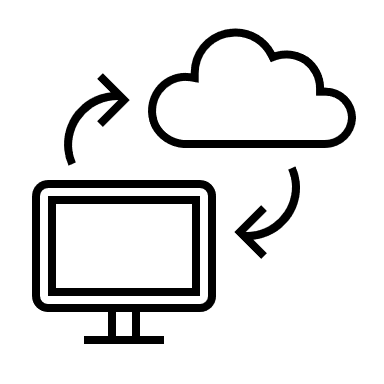
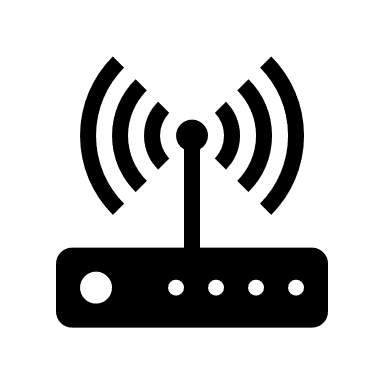
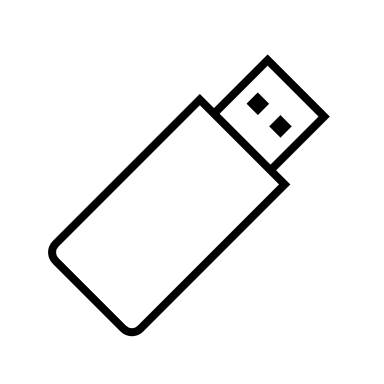
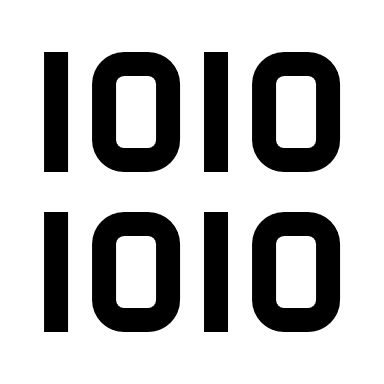
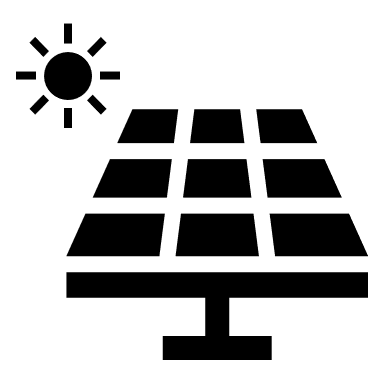
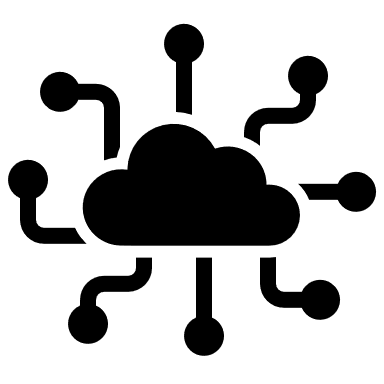
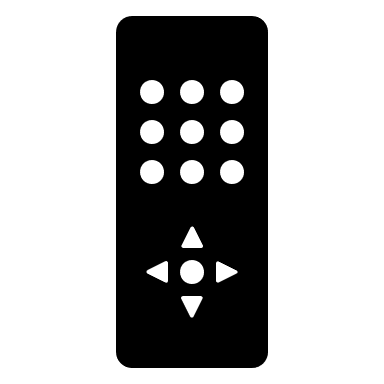
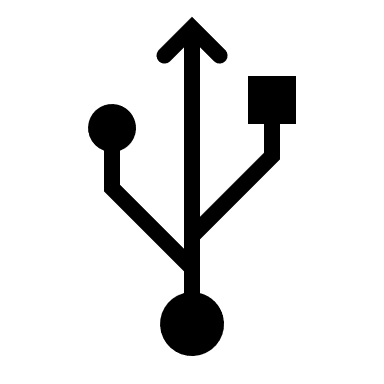
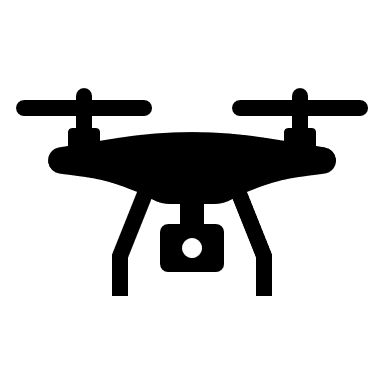
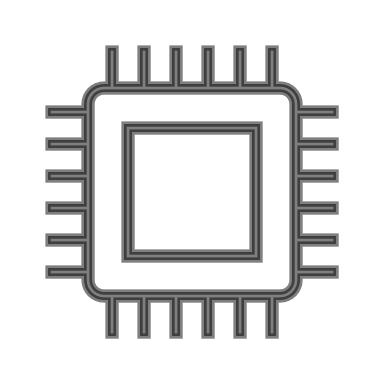
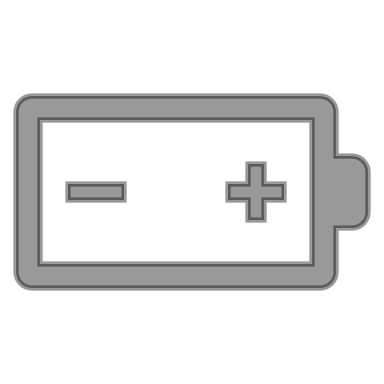
The parts (Figs.) is responsible for the web dashboard, this part is purely in HTML and JavaScript, not the main point of interest in this Esp code.



**Challenges & Solutions**

**Coding challenges will be divided into two parts, Tiva C Launchpad Coding Challenges, and Esp32 Coding Challenges.**

* **Tiva C Launchpad Coding Challenges:**
  + A
  + B
  + C
* **Esp32 Coding Challenges:**
  + Web Sockets (Initial Approach):
    - Purpose: Enable real-time updates from the ESP32 to the web page.
    - Issue: The ESP32 frequently crashed and rebooted during AP mode initialization.
    - Cause: WebSocket handling overloaded the ESP32 (likely due to RAM constraints or poorly managed asynchronous callbacks).
    - Outcome: Abandoned due to instability.
  + MQTT (Alternative Attempt):
    - Purpose: Use MQTT for lightweight, publish-subscribe communication to update the web page.
    - Issue: Mobile MQTT clients failed to connect or receive data.
    - Cause: Potential misconfiguration of the MQTT broker, port issues, or network isolation (ESP32 in AP mode).
    - Outcome: Not used due to complexity and setup issues.
  + Auto-Refreshing the Page:
    - Purpose: Periodically reload the web page to show updated data.
    - Method: Used <meta http-equiv="refresh"> or location.reload() in JavaScript.
    - Issue: Disrupted user input in the command textbox (typed text would be erased).
    - Outcome: Dropped due to poor user experience.
  + Final Implementation – AJAX Polling with fetch ():
    - Method: JavaScript on the page periodically sends fetch("/data") requests to get sensor data in JSON format and updates the DOM.
    - Command Input: Separate fetch("/command”, method: "POST", body: ... }) used to send user commands without refreshing.
    - Advantages:

****