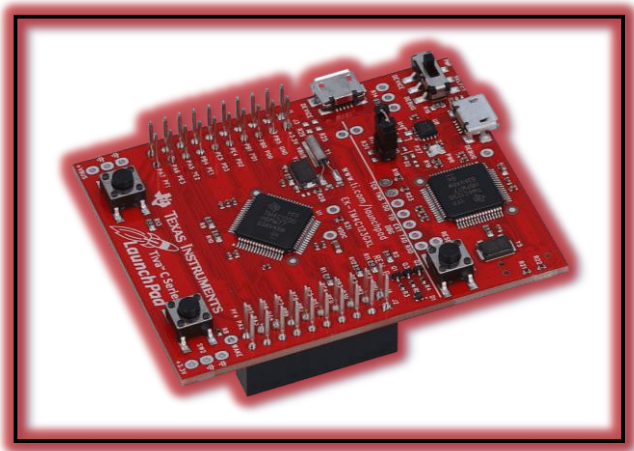


Intro. To Microprocessor

Smart House



(Via the TM4C123G Launchpad)

Participants Names:

Team A:

- | | |
|--------------------------|---------------------|
| 1. Ahmad Adham Badawy | Reg. No. /231018156 |
| 2. Ali Abd El Nasser Ali | Reg. No. /231018210 |
| 3. Abdallah Fahmy Rabea | Reg. No. /231008522 |
| 4. Abdelrahman Mostafa | Reg. No. /231008579 |
| 5. Eslam Mohammed | Reg. No. /231018107 |

Team B:

- | | |
|-----------------------|---------------------|
| 6. Mohamed Sayed | Reg. No. /231008761 |
| 7. Abdelrahman Hamdy | Reg. No. /231018182 |
| 8. Mohammed Ehab Badr | Reg. No. /231008607 |
| 9. Mostafa Roshdy | Reg. No. /222008507 |

Intro. To Microprocessor

Lecturer:

Dr. Ahmad Sayed

Teacher Assistant:

Eng. Fatma Sharawy

Table of Contents



1. Introduction	3
1.1. What is Tiva C Launchpad?	
1.2. Why It Is Useful for Our Project?	
1.3. Why did we use ESP-32S?	
1.4. Brief Summary on the project Idea	
1.5. Project Vision	
2. Tools & Components Used	4
2.1. Motion Sensor (PIR)	
2.2. LDR	
2.3. Laser Transmitter	
2.4. Buzzer	
2.5. Temperature Sensor (LM35)	
2.6. Fire Sensor	
2.7. Bi-Directional Logic Level Shifter	
2.8. LCD 16x2 (I ² C)	
2.9. Mini Submersible Water Pump	
2.10. 2N2222 – NPN Transistor	
2.11. Soil Moisture Sensor	
2.12. UltraSonic Sensor	
2.13. ESP-32S	
3. Smart House Concept/Idea	6
3.1. Project Design Vision	
3.2. Drafts	
3.3. Final Design	
4. Planning Wise	X
4.1. House Electronics Infrastructure (Interior & Exterior)	
4.2. Demo vs Theoretical	
4.3. Challenges & Solutions	
5. Coding Wise	X
5.1. TM4C123G (Launchpad)	
5.2. ESP-32S	
5.3. Challenges & Solutions	
6. Conclusion	X
6.1. A	
6.2. B	
7. References	X

What is Tiva C Launchpad?



The Tiva C LaunchPad (**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 IRL



Figure 2 - Tiva C Launchpad IRL

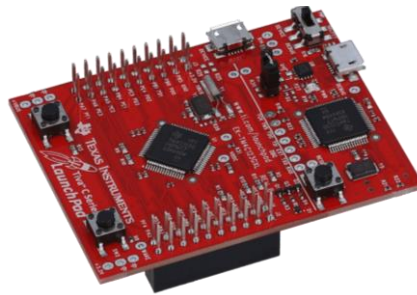


Figure 3 - Tiva C Launchpad

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

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.



Figure 4 - ESP-32S

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.

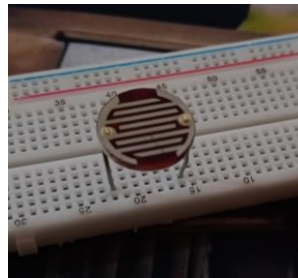


Figure 6 - LDR IRL



Figure 7 - Laser Module 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.

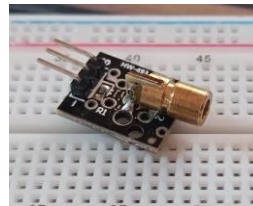


Figure 8 - Laser Module IRL

Buzzer

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.

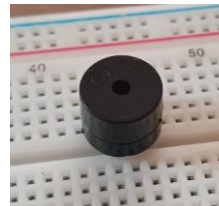


Figure 9 - Buzzer IRL

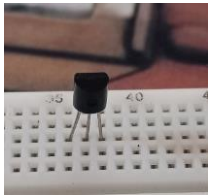


Figure 10 - LM35 IRL

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.

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.

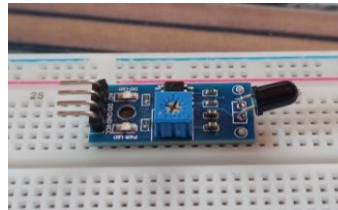


Figure 11 - Fire Sensor IRL

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.



Figure 12 - Fire Sensor IRL



Figure 13 - Bi-Directional Logic Level Shifter IRL

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.

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.

Commented [AA1]: WIP

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.

Commented [AA2]: WIP

2N2222 NPN Transistor

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

Commented [AA3]: WIP

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.

Commented [AA4]: WIP

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.

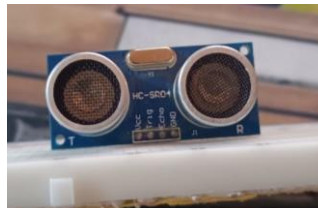


Figure 18 – Ultrasonic Sensor IRL



Figure 19 – Ultrasonic Sensor IRL

ESP-32S



The ESP-32S is a powerful microcontroller with built-in Wi-Fi and Bluetooth capabilities. It serves as the brain of the system, handling sensor data collection, device control, and hosting the local web dashboard interface for user interaction.





House Electronics Infrastructure

A

Demo vs Theoretical

In the demo version of the project, each room will contain only one sensor to simplify the hardware setup and focus on showcasing individual features.

Theoretically, every room in the smart house should include the full sensor system (motion detection, temperature monitoring, gas detection, and light sensing).

The LCD screen is placed outside the model in the demo for ease of display, but in the theoretical design, it would be installed next to the main door inside the house.

The web dashboard provided by the ESP32 offers basic monitoring and control in the demo, while a full version would include a dedicated mobile application capable of sending real-time alerts and notifications about fire detection, intrusions, or other emergencies.

Additionally, the tank system is simulated for watering the garden but would theoretically also supply water for internal house use.

TM4C123G (Launchpad) Coding



A

ESP-32S Coding

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.

```

1  #include <WiFi.h>
2  #include <WebServer.h>
3  #include <ArduinoJson.h>
4  #include <ArduinoOTA.h>
5
6  // Always On AP Mode
7  const char* ap_ssid = "SmartHouse_AP";
8  const char* ap_password = "12345678"; // Minimum 8 characters
9
10 // Optional Wifi connection
11 String station_ssid = "";
12 String station_password = "";
13
14 // Sensor Data Storage
15 float temperature = 0;
16 float gasLevel = 0;
17 float lightLevel = 0;
18 float tankLevel = 0;
19 bool motionDetected = false;
20 bool intruderDetected = false;
21 bool fireDetected = false;
22
23 // Internal Temperature
24 float esp32Temperature = 0;
25
26 // Flag to distinguish between real and fake values
27 bool tempFake = false;
28 bool gasFake = false;
29 bool lightFake = false;
30 bool tankFake = false;
31 bool motionFake = false;
32 bool intruderFake = false;
33

```

Commented [AA5]: WIP

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.

```

34 // Serial Communication with TM4C123G
35 #define SERIAL_BAUD_RATE 9600
36 String incomingData = "";
37
38 // Web Server Setup
39 WebServer server(80);
40

```

Commented [AA6]: WIP

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.

```

41 // HTML Dashboard Page
42 const char INDEX_HTML[] PROGMEM = R"rawliteral(
43 <!DOCTYPE html><html>
44 <head><title>Smart House Dashboard</title>
45 <meta name="viewport" content="width=device-width,
46   initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
47 <style>
48 body {
49   font-family:sans-serif;
50   max-width:600px;
51   margin:auto;
52   padding:20px;
53   background:#f0f0f0;
54   font-size:16px;
55 }
56 .card {
57   background:white;
58   padding:15px;
59   margin:10px 0;
60   border-radius:8px;
61   box-shadow:0 2px 5px rgba(0,0,0,0.1);
62   font-size:16px;
63 }
64 h2 {
65   text-align:center;
66   color:#333;
67 }
68 input[type=text] {
69   width:100%;
70   padding:10px;
71   margin:5px 0;
72 }
73 button {
74   padding:10px;
75   margin:5px 0;
76   width:100%;
77 }
78 #output {
79   background:#e0e0e0;
80   padding:10px;
81   margin-top:10px;
82   height:150px;
83 }
84 #output {
85   background:#e0e0e0;
86   padding:10px;
87   margin-top:10px;
88   height:150px;
89 }
90 </style>
91 </head>
92 <body>
93 <h2>&#127760; Smart House Dashboard</h2>
94
95 <div class='card'><h3>&#127790; Temperature</h3><p id='temp'>---</p></div>
96 <div class='card'><h3>&#128293; Gas Level</h3><p id='gas'>---</p></div>
97 <div class='card'><h3>&#128161; Light Level</h3><p id='light'>---</p></div>
98 <div class='card'><h3>&#128167; Water Tank Level</h3><p id='tank'>---</p></div>
99 <div class='card'><h3>&#128101; Motion Detection</h3><p id='motion'>---</p></div>
100 <div class='card'><h3>&#128681; Intruder Detection</h3><p id='intruder'>---</p></div>
101 <div class='card'><h3>&#127777; ESP32 Internal Temp</h3><p id='esp32_temp'>---</p></div>
102

```

Commented [AA7]: WIP



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:
 -

AASTMT, Course Code: ECE4206

