AASTMT, Course Code: ECE4206

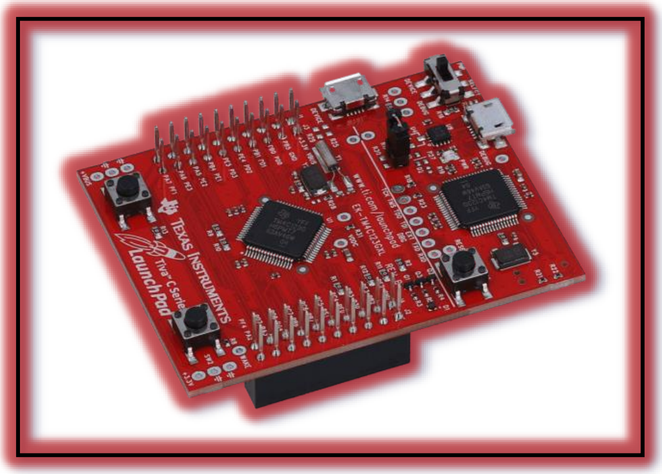
AASTMT

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		

Intro. To Microprocessor
Lecturer:
Dr. Ahmad Sayed

Teacher Assistant:
Eng. Fatma Sharawy

Table of Contents

What is Tiva C Launchpad?

The Tiva C LaunchPad 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, I^2C), 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.

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 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 the any device that can access the internet and connect to the ESP-32S's WIFI network.

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.

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.

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.

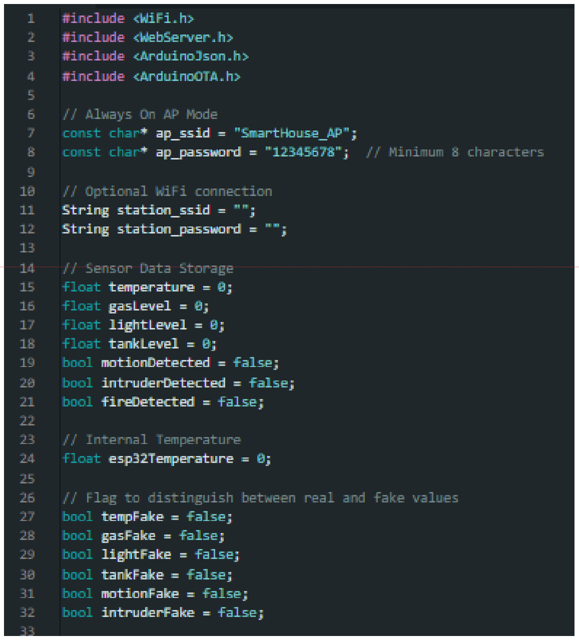
ESP-32S Coding

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.

Commented [AA1]: WIP

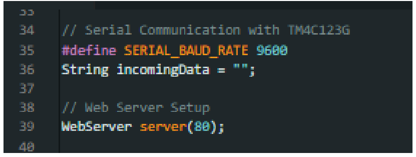


```
#include <WiFi.h>
#include <WebServer.h>
#include <ArduinoJson.h>
#include <ArduinoOTA.h>

// Always On AP Mode
const char* ap_ssid = "SmartHouse_AP";
const char* ap_password = "12345678";  // Minimum 8 characters

// Optional WiFi connection
String station_ssid = "";
String station_password = "";

// Sensor Data Storage
float temperature = 0;
float gasLevel = 0;
float lightLevel = 0;
float tankLevel = 0;
bool motionDetected = false;
bool intruderDetected = false;
bool fireDetected = false;

// Internal Temperature
float esp32Temperature = 0;

// Flag to distinguish between real and fake values
bool tempFake = false;
bool gasFake = false;
bool lightFake = false;
bool tankFake = false;
bool motionFake = false;
bool intruderFake = false;

```

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.

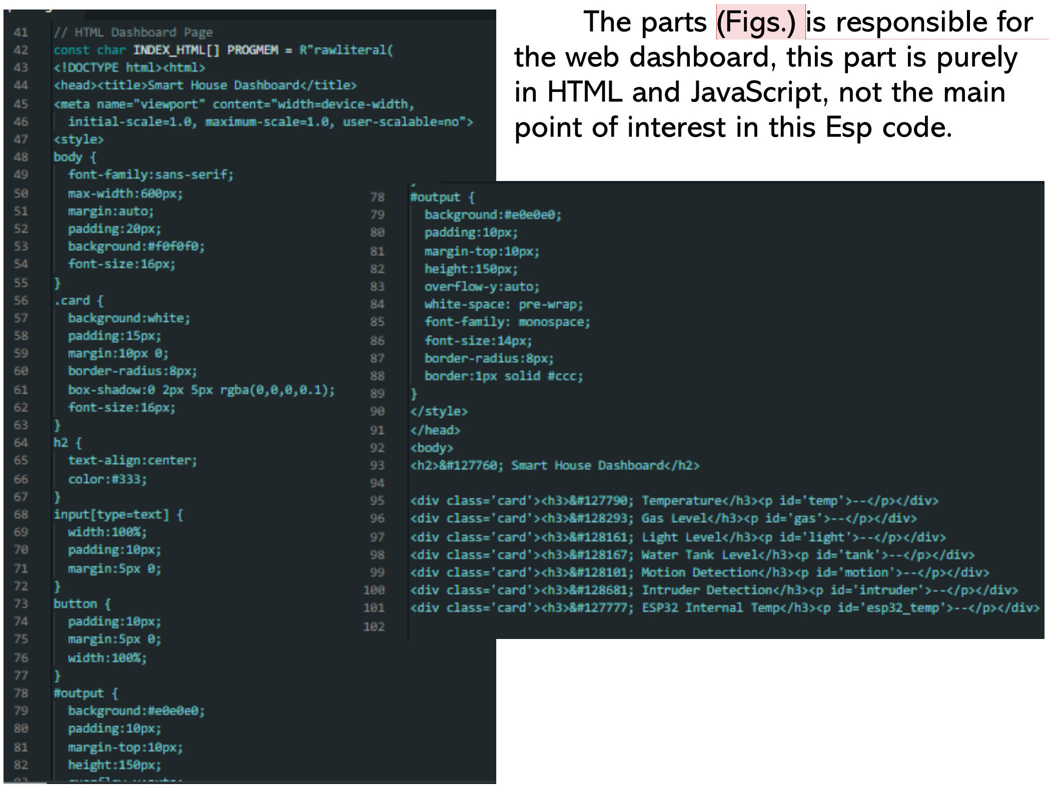
There is also the initialization of some Booleans that tracks whether the current values of the variables are fake or not, later on we will discuss why this was implemented in the first place.

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
```

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.



```
// HTML Dashboard Page
const char INDEX_HTML[] PROGMEM = R"rawliteral(
<!DOCTYPE html><html>
<head><title>Smart House Dashboard</title>
<meta name="viewport" content="width=device-width,
  initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
<style>
body {
  font-family:sans-serif;
  max-width:600px;
  margin:auto;
  padding:20px;
  background:#f0f0f0;
  font-size:16px;
}
.card {
  background:white;
  padding:15px;
  margin:10px 0;
  border-radius:8px;
  box-shadow:0 2px 5px rgba(0,0,0,0.1);
  font-size:16px;
}
h2 {
  text-align:center;
  color:#333;
}
input[type=text] {
  width:100%;
  padding:10px;
  margin:5px 0;
}
button {
  padding:10px;
  margin:5px 0;
  width:100%;
}
#output {
  background:#e0e0e0;
  padding:10px;
  margin-top:10px;
  height:150px;
  overflow-y:auto;
```

```
#output {
  background:#e0e0e0;
  padding:10px;
  margin-top:10px;
  height:150px;
  overflow-y:auto;
  white-space: pre-wrap;
  font-family: monospace;
  font-size:14px;
  border-radius:8px;
  border:1px solid #ccc;
}
</style>
</head>
<body>
<h2>&#127760; Smart House Dashboard</h2>

<div class='card'><h3>&#127790; Temperature</h3><p id='temp'>--</p></div>
<div class='card'><h3>&#128293; Gas Level</h3><p id='gas'>--</p></div>
<div class='card'><h3>&#128161; Light Level</h3><p id='light'>--</p></div>
<div class='card'><h3>&#128167; Water Tank Level</h3><p id='tank'>--</p></div>
<div class='card'><h3>&#128101; Motion Detection</h3><p id='motion'>--</p></div>
<div class='card'><h3>&#128681; Intruder Detection</h3><p id='intruder'>--</p></div>
<div class='card'><h3>&#127777; ESP32 Internal Temp</h3><p id='esp32_temp'>--</p></div>

```

Commented [AA2]: WIP

Commented [AA3]: 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:
  -