KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

COE 381

PROJECT TOPIC: IoT-Based Smart Irrigation System

Group Members:

- Benjamin Darko -1821122
- Raymond Ampaabeng Kyeremeh - 1816622
- Joseph Enoch Baffoe Maison - 1824122
- Andoh David Nkuting - 1816922
- Amevenku Setor Yao - 1816022
- Rahinatu Adam - 1813222
- Ferdinand Collins Domeh - 1821822
- Abosti Benjamin Etornam - 1812622
- Emmanuella Noeline Mensah - 1824522
- Asumang Godwin Pobi - 1818822

Table of Contents

Introduction

Smart irrigation systems leverage modern technology to optimize water usage in agriculture. Traditional methods, which rely on manual watering or fixed schedules, often lead to inefficiencies like overwatering or underwatering. By integrating IoT, these systems use

sensors and automation to monitor soil moisture, temperature, and weather conditions in real time. This enables precise water distribution, reducing waste, enhancing crop yield, and lowering operational costs. The adoption of IoT in agriculture makes irrigation more efficient, sustainable, and responsive to environmental conditions.

Problem Statement

Traditional irrigation techniques often result in water wastage as they fail to consider the actual water needs of plants. Many farmers or gardeners either:

1. Overwater, which wastes water and harms plants
2. Underwater, which leads to poor plant growth and lower crop yields

Manually watering plants or setting irrigation on a fixed schedule ignores real-time factors such as soil moisture, weather, and temperature. This may lead to unnecessary watering, even after rainfall or when the soil is already sufficiently moist.

- **Solution By an IoT-Based Irrigation System**

An IOT-based smart irrigation system uses sensors to monitor soil moisture, temperature, and other environmental factors. Based on this data, it automatically controls water flow using valves or pumps. This means:

- Water is used only when needed (reducing wastage)
- Plants get optimal water levels, improving their health and yield
- The system operates automatically, reducing manual effort for farmers or gardeners.

Methodology

- **Proposed Solution**

The approach chosen leverages advanced sensor technology, a microcontroller, and wireless communication to optimize water usage for agricultural and landscape irrigation. This system integrates multiple components to monitor soil moisture levels, environmental conditions, and water availability in real time. The key elements of the proposed solution are:

1. **Sensor Development**: Sensors (moisture, temperature and humidity) are deployed strategically into the irrigation field. These sensors will continue to collect data on soil and atmospheric conditions in real time to determine the optimal watering requirements for crops or plants.

2. **Microcontroller-Based Processing Unit**: An ESP32 serves as the central processing unit. It receives data from the sensors, processes the information, and determines the irrigation schedule based on predefined thresholds and algorithms. The implementation was carried out using Wokwi, a virtual simulation platform.

3. **Automated Watering System**: Based on the sensor readings, the microcontroller activates the servo motor to regulate water flow. The system ensures that irrigation occurs only when necessary, reducing water wastage and promoting efficient resource utilization. To provide remote control and automation, the Blynk IoT platform was used, allowing users to monitor and control the system via a mobile application.

4. **Power Management**: The system can be powered by renewable energy sources, such as solar panels, to enhance sustainability and ensure continuous operation, even in remote areas with limited access to electricity.

• System Architecture

Overview of The Components Used

The project features an IoT-based irrigation system using an ESP32 microcontroller. It connects a moisture sensor, DHT22 (for temperature and humidity), an LED, a button for manual override, and a servo motor to control irrigation. The moisture sensor reads soil moisture, and if the moisture is low or manual override is triggered, the system activates irrigation. The servo motor opens a valve for water flow, and an LCD displays sensor data. The system also manages irrigation duration and manually controlled overrides.

How the System Works

The system monitors soil moisture using a sensor and reads environmental conditions from a DHT22 sensor. If the soil moisture falls below a set threshold, the ESP32 triggers a servo motor to open a valve for irrigation. The LED provides a visual status indicator, and a manual override button allows user control. An LCD displays real-time sensor readings and system status. The ESP32 processes all inputs and executes predefined control logic for efficient water management.

• Hardware Implementation

Circuit Diagram & Explanation

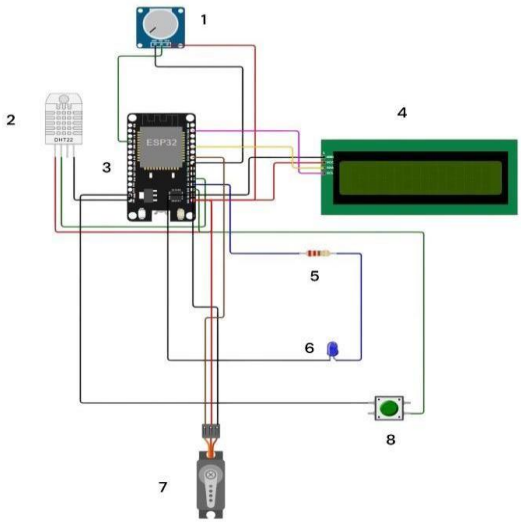


Figure 1. Circuit Diagram

1. **Potentiometer** – Used to adjust the soil moisture values
2. **DHT22 Temperature & Humidity Sensor** – Measures environmental temperature and humidity
3. **ESP32 Microcontroller** – The main control unit handling sensor data and system logic
4. **LCD Display (I2C)** – Displays system information such as temperature and humidity readings
5. **Resistor** – Used to limit current flow in the circuit
6. **LED Indicator** – Provides visual feedback on system status
7. **Servo Motor** – Controls movement for irrigation control
8. **Push Button** – Manual input for user interaction(override) Power is supplied via ESP32's 3.3V and GND pins.

Components And Their Functions

- Potentiometer

The potentiometer is used to manually adjust the soil moisture threshold value. It acts as a variable resistor, providing an analogue voltage output that the ESP32 reads to determine the desired soil moisture level for irrigation. By turning the potentiometer's knob, the resistance changes, altering the voltage sent to the ESP32. This voltage is mapped to a soil moisture percentage, allowing dynamic control over the irrigation system. The ESP32 compares this user-defined threshold with real-time soil moisture readings from a sensor. If the soil moisture falls below the set threshold, the system can trigger the servo motor to open a valve for watering. This setup enables user customization of irrigation levels without modifying the code.

Figure 2. Potentiometer

- DHT22 Temperature & Humidity Sensor

The DHT22 sensor measures both temperature and humidity levels in the environment. It uses a capacitive humidity sensor and a thermistor to detect changes in air moisture and temperature. The sensor processes the readings and sends digital signals to the ESP32 microcontroller for further analysis. It operates with a single-wire data communication protocol, making it easy to interface with the ESP32. The DHT22 provides accurate readings, making it suitable for automation projects. It has a built-in calibration coefficient, ensuring reliable measurements over time. It helps monitor environmental conditions, which could be used for smart irrigation.

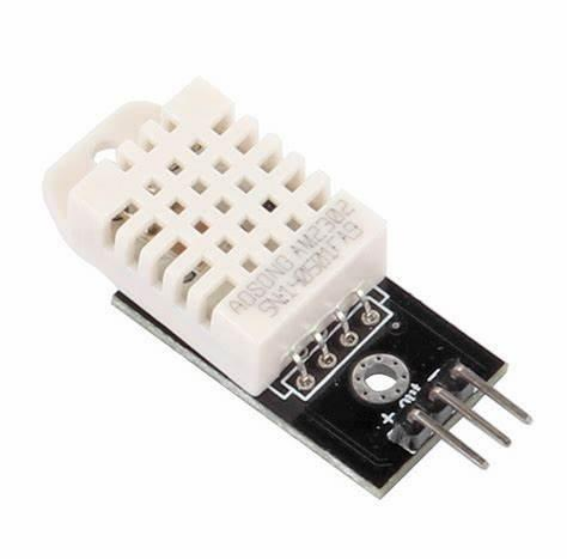


Figure 3. DHT22 Temperature and Humidity Sensor

- ESP32 Microcontroller

The ESP32 acts as the central processing unit, managing all connected components. It collects data from the DHT22 sensor, processes it, and controls the display, LED, and servo motor. The ESP32 communicates with the LCD to display temperature, humidity, and other relevant information. It also processes user input from the push button and adjusts the servo motor accordingly. It operates on a low-power, high-performance architecture, making it suitable for IoT applications. In the circuit, the ESP32 ensures seamless integration and automation of the system's functions.

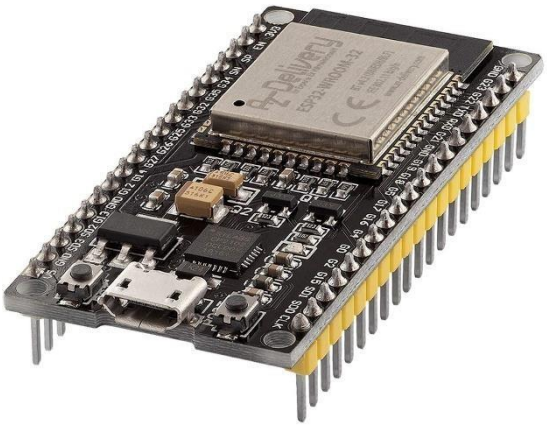


Figure 4. ESP32 Microcontroller

- LCD Display (I2C)

The LCD Display (I2C) in the circuit is used to visually present information such as temperature, humidity, and system status. It communicates with the ESP32 using the I2C protocol, reducing the number of pins required for connection. The potentiometer adjusts its contrast, ensuring readability under different lighting conditions. The display receives processed data from the ESP32 and updates in real-time. It provides a userfriendly interface, eliminating the need for additional output devices. The I2C interface allows efficient data transfer with minimal wiring complexity. In the circuit, the LCD ensures clear and convenient monitoring of system parameters.

Figure 5. LCD Display (I2C)

- Resistor

The resistor in the circuit is used to limit the current flow to protect components from damage. It is connected in series with the LED to prevent excessive current from burning it out. By providing resistance, it controls voltage distribution across the circuit elements. The resistor ensures stable and efficient operation of the LED and other connected components. Its value is chosen based on Ohm's Law to achieve the desired current flow. Without it, the LED could receive too much current, leading to failure. In the circuit, the resistor plays a crucial role in maintaining circuit safety and functionality.

Figure 6. Resistors

- LED

The LED in the circuit serves as a visual indicator to show system status or activity. It is controlled by the ESP32, which turns it on or off based on specific conditions. A resistor is connected in series with the LED to limit the current and prevent damage. The LED can signal various states, such as power-on, successful sensor readings, or an active process. It provides immediate feedback without needing a display or serial monitor. The LED operates on low power, making it efficient for embedded applications. In the circuit, it enhances user interaction by offering real-time visual cues.

Figure 7. LED

- Servo Motor

The **servo motor** in your circuit is used as a substitute for a valve to control water flow in the irrigation system. Since the simulation lacks a physical valve component, the servo motor was chosen to mimic its function. The **ESP32** controls the servo by sending PWM signals to adjust its angular position. Based on soil moisture readings, the ESP32 determines whether the soil is too dry or adequately moist. If the moisture level drops below the user-set threshold (adjusted via the potentiometer), the servo rotates to simulate opening a valve, allowing water flow. Once the moisture level reaches the desired value, the servo returns to its original position, simulating valve closure. This method enables automated irrigation control without a physical solenoid valve. It requires three connections: power, ground, and a signal wire from the ESP32.

Figure 8. Servo Motor

- Push Button

The push button in your circuit serves as a manual input device for user interaction. It is connected to the ESP32, allowing users to trigger specific actions when pressed. When the button is pressed, it completes the circuit, sending a signal to the microcontroller. The ESP32 detects this change and can respond by activating or deactivating components such as the servo motor or LED. The button allows for manual control, complementing the automated functions of the system. In the circuit, it provides an easy way for users to interact with and override automated processes when needed.

Figure 9. Push Button

- **Software Implementation**

Overview Of The Code

This section provides an in-depth explanation of the code structure, key functions, and interactions between different components.

Code Structure

The code follows a modular approach, with distinct functions handling various tasks such as sensor reading, decision-making, actuation, and display updates. The structure is as follows:

- Initialization (*setup function*)
- Main Loop (*loop function*)
- Sensor Reading *(readSensors function*)
- Manual Override Check (*checkButton function)*
- Display Update (*updateDisplay function*)
- Irrigation Decision Logic *(checkIrrigation function)*
- Irrigation Management *(manageIrrigation function)* - Serial Output *(serialOutput function)* - Actuation *(OpenWater function)*

Key Functions and Their Roles

- Initialization *(setup function)*

This function configures the system by setting up serial communication for debugging. It also configures pin modes for input (sensors, button) and output (LED, servo motor). Initializing the DHT22 sensor and LCD display, attaching the servo motor to its designated pin, displaying a startup message on the LCD are all roles it performs.

- Main Loop *(loop function)*

This function repeatedly reads sensor data, checks if the manual override button is pressed, updates the LCD display with current readings, determines whether irrigation should be activated, manages the irrigation process, outputs data to the serial monitor and introduces a delay of one second before repeating.

- Sensor Reading *(readSensors function)*

This function reads the soil moisture level using the analog pin. It reads temperature and humidity values from the DHT22 sensor. If sensor readings are invalid, logs an error and resets values to zero.

- Manual Override Check (*checkButton function)*

This function checks if the manual override button is pressed, toggles the irrigation mode between automatic and manual control and implements a short delay to prevent multiple rapid toggles.

- Display Update *(updateDisplay function)*

This function clears the LCD screen and updates it with,

  - Soil moisture level ○ Temperature ○ Humidity

It also ensures real-time monitoring of environmental conditions.

- Irrigation Decision Logic *(checkIrrigation function)*

The function determines whether irrigation should be started based on:

  - Soil moisture level exceeding a predefined threshold.
  - Manual override being activated.

And if conditions are met, starts irrigation and logs the event.

- Irrigation Management *(manageIrrigation function)*

The function controls the duration of irrigation, stops irrigation after a predefined time unless manual override is active. It manages LED indicators to reflect irrigation status and calls the OpenWater function to control the water valve.

- Serial Output *(serialOutput function)*

The function prints sensor readings and irrigation status to the serial monitor for debugging and monitoring purposes.

- Actuation *(OpenWater function)*

The Function controls the servo motor to simulate opening and closing of a water valve. It moves the servo to different positions with short delays to regulate water flow.

Library

- Blynk Library (BlynkSimpleEsp32.h) ○ Purpose: Facilitates communication between the ESP32 microcontroller and the Blynk IoT platform, enabling remote monitoring and control of the irrigation system.

- Wi-Fi Library (WiFi.h) ○ Purpose: Manages Wi-Fi connectivity for the ESP32, allowing it to connect to wireless networks and communicate over the internet.

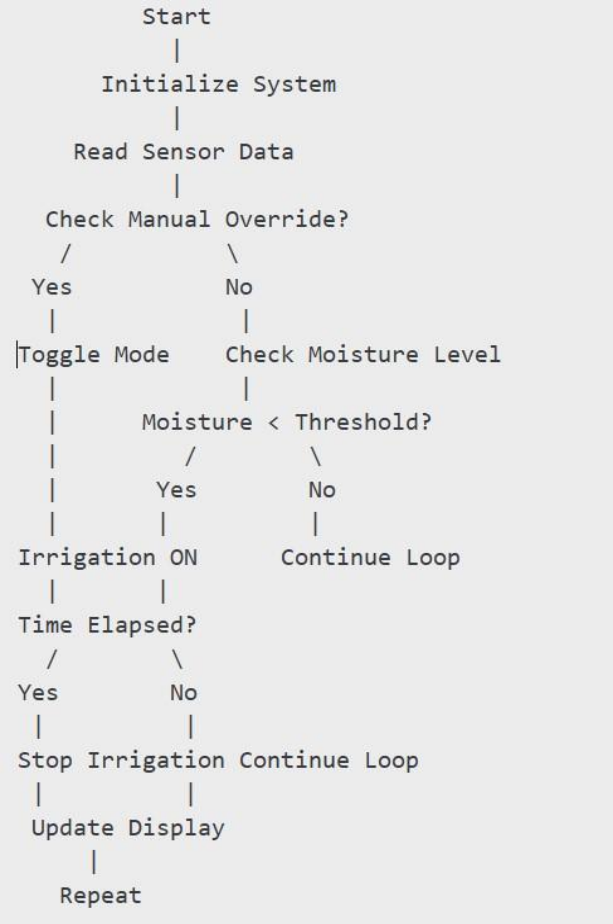
- LiquidCrystal I2C Library (LiquidCrystal_I2C.h) ○ Purpose: Controls the I2C-based LCD display, used to present real-time sensor readings and system status.

- DHT Sensor Library (DHT.h) ○ Purpose: Interfaces with the DHT22 sensor to read temperature and humidity data.

- ESP32 Servo Library (ESP32Servo.h) ○ Purpose: Enables control of servo motors using the ESP32, which can be utilized to operate valves or other mechanical components in the irrigation system.

System Flowchart

The following flowchart outlines the logical flow of the system:



EasyEDA Schematic Design

A schematic representation of the circuit was designed in EasyEDA to analyze component interactions and voltage distribution. The circuit design was reviewed before hardware implementation to ensure correct connections and expected behaviour.

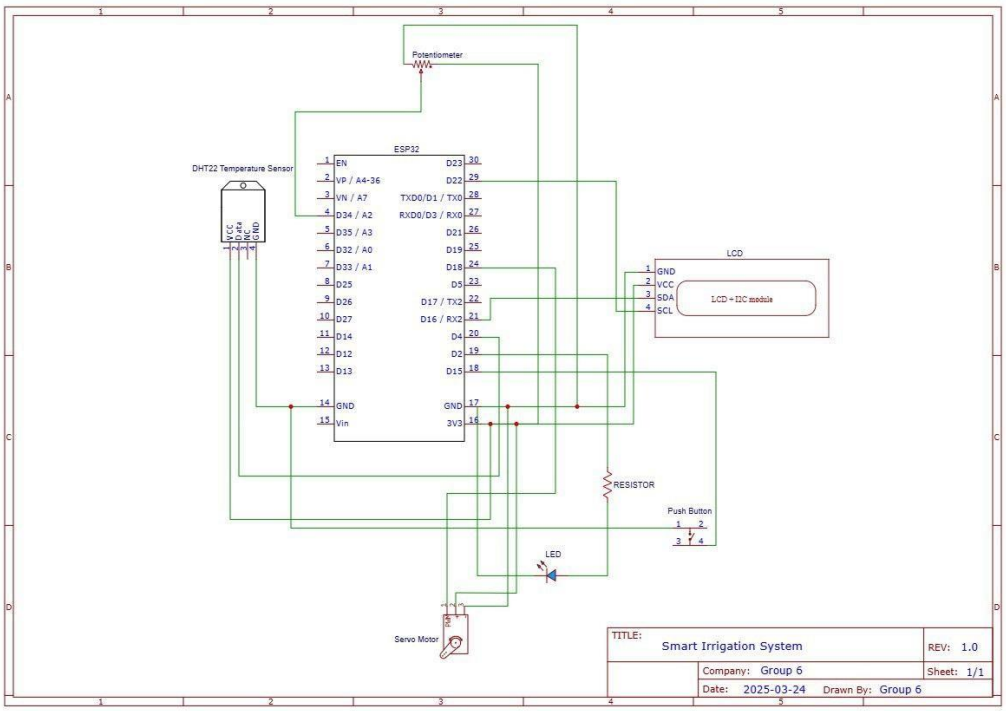


Figure 10. Circuit Schematic

Simulation/ Demonstration

- **Setting Up The Simulation Environment**

The simulation and testing of the ESP32-based system were conducted using the following tools:

1. Wokwi – A web-based platform used to simulate the circuit virtually before physical implementation.
2. Proteus – Used for designing the circuit schematic and running simulations to verify the system's functionality.
3. Arduino IDE – Utilized for writing, compiling, and uploading the code to the ESP32 microcontroller.

Wokwi Simulation

The ESP32 circuit was simulated using Wokwi, where all components, including the ESP32 microcontroller, LCD (I2C), DHT22 sensor, push button, LED, servo motor, and resistors, were virtually connected. The system was programmed and tested in the platform to ensure

proper sensor readings, display output, and actuator responses. The LCD successfully displayed temperature and humidity data from the DHT22 sensor, while the LED and servo motor responded appropriately to user inputs.

- **Running The Simulation**

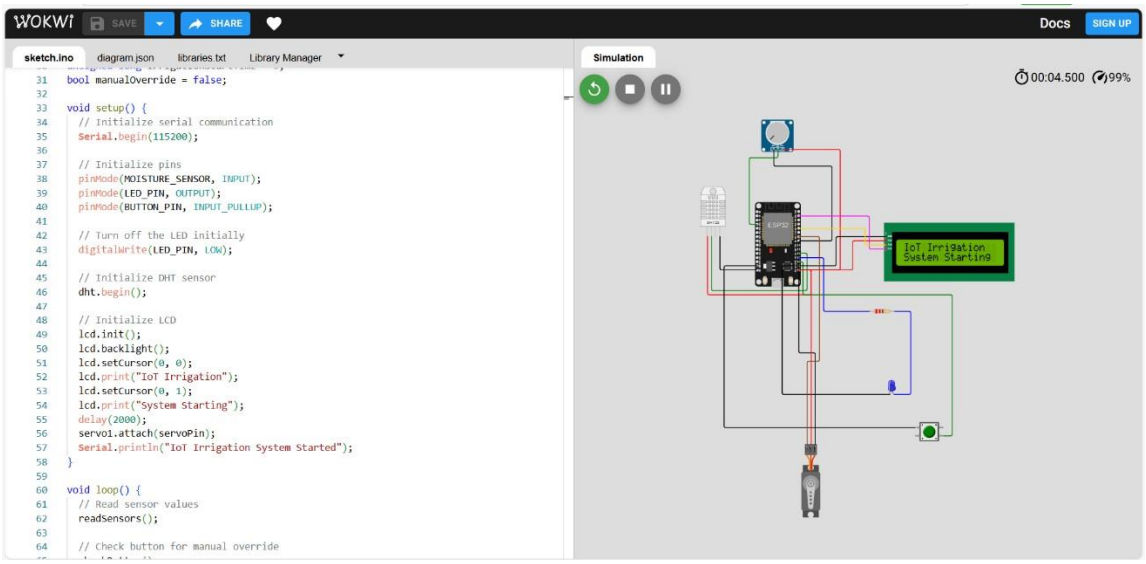


Figure 11. Starting The Simulation

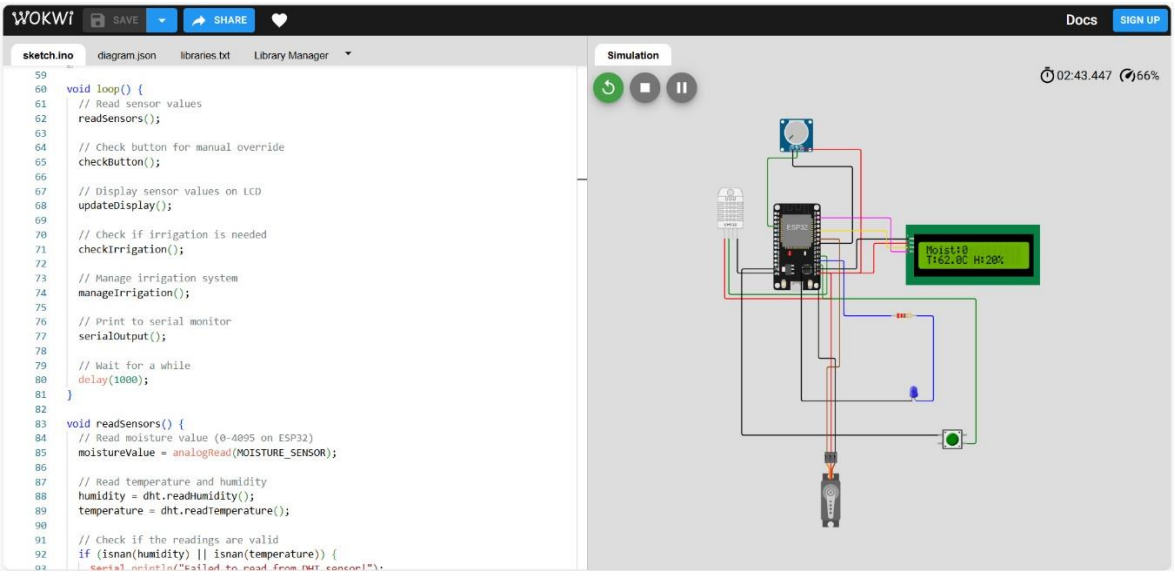


Figure 12. Simulation in progress

Outputs And Key Takeaways

1. Sensor Data Display: The DHT22 sensor successfully provided temperature and humidity readings, which were accurately displayed on the LCD.
2. Component Interaction: The push button effectively controlled the LED and servo motor, confirming proper communication between components.
3. System Stability: The circuit operated consistently without major errors, confirming that the design and code implementation were effective.
4. Improvements and Adjustments: Minor modifications were made to optimize the servo motor's response time and improve LCD readability under different conditions.

Conclusion

The simulation testing of the ESP32-based system confirmed the feasibility and reliability of the design. The system successfully integrated sensor readings, user inputs, and actuator responses, demonstrating proper component interaction. One of the key challenges encountered was a pin configuration issue, which caused initial difficulties in component connections and required reassigning GPIO pins for proper functionality. Additionally, ensuring accurate sensor data transmission and display synchronization required adjustments in code timing and signal processing. Minor issues with LCD visibility under varying lighting conditions were also addressed.

Future improvements could include implementing wireless data logging for remote monitoring, enhancing power efficiency, making the system faster by optimizing code execution and reducing response delays, and integrating additional sensors for expanded functionality. Overall, the project successfully achieved its objectives, validating both the hardware and software design.