

Alnasser University

Smart IoT Doorbell and Security System

Course: Embedded Systems

Semester: 2nd 2025/26

Submission Date: 7th Feb 2026

Team Members:

Ashraf Wadeea Al-Absi - 4

Instructor:

Majd Alsuraihi

2. Abstract

This project presents a "Smart IoT Doorbell and Security System" designed to enhance home security and convenience through remote monitoring and control. Traditional doorbells lack remote interaction and proactive security features. To address this, we developed a connected system using the ESP32 microcontroller, utilizing MQTT communication to bridge the physical hardware with a Python Flask web dashboard. The system features a push-button doorbell that triggers instant notifications, a servo-controlled door lock for remote access, and a PIR motion sensor that detects loitering intruders (standing for >60 seconds) without a doorbell press. The system provides real-time feedback via a local LCD screen and a piezoelectric buzzer. The final prototype was successfully simulated in Wokwi and integrated with a custom web interface, demonstrating reliable bi-directional control (locking/unlocking) and automated security alerting, providing a cost-effective and scalable solution for modern home automation.

3. Introduction

3.1 Background and Motivation

Home security is a paramount concern in the modern era. While smart locks and video doorbells exist, they are often expensive and rely on closed, proprietary ecosystems. There is a growing need for customizable, low-cost solutions that integrate seamlessly with web standards. The team chose this project to explore the intersection of Embedded Systems and Web Development (IoT), specifically solving the problem of granting access to visitors remotely while maintaining security against loiterers.

3.2 Problem Statement

Standard doorbells are passive devices; if the homeowner is away, they cannot know a visitor arrived or grant them entry. Furthermore, burglars often loiter near entryways to check for occupancy without ringing the bell, bypassing traditional triggers. A system is needed that allows the owner to:

1. Unlock the door remotely for guests.
2. Communicate text messages to the door display.
3. Automatically detect and alert the owner of suspicious loitering behavior.

3.3 Project Objectives

- Develop an ESP32-based node to handle sensor inputs (Button, PIR) and actuator outputs (Servo, LCD, Buzzer).
- Implement a "Burglar Detection" algorithm that triggers an alert only after continuous motion detection (e.g., 60 seconds).
- Create a Python Flask Web Dashboard for real-time status monitoring and remote control.
- Establish reliable bi-directional communication using the MQTT protocol.

3.4 Scope and Limitations

The project scope covers a functional simulation using the Wokwi platform for hardware and a local Python server for the dashboard. It simulates a deadbolt mechanism using a servo motor.

- **Limitation:** The project uses a simulated environment (Wokwi) rather than physical soldering.
- **Limitation:** Video streaming was scoped out due to simulation constraints; a PIR sensor is used as a proxy for presence detection.

4. System Overview

4.1 High-Level Description

The system follows a Publisher/Subscriber architecture. The **ESP32** acts as the edge node, reading sensors and controlling the door mechanism. It connects via WiFi to a public MQTT Broker (broker.hivemq.com). The **User Dashboard** (Python Flask App) connects to the same broker. When an event occurs (e.g., Doorbell Ring), the ESP32 publishes a message. The Dashboard receives it and alerts the user. Conversely, when the user clicks "Unlock," the Dashboard publishes a command that the ESP32 receives to rotate the servo.

4.2 Functional Description

- **Sensing:** A Push Button detects visitor arrivals. A PIR (Passive Infrared) Sensor monitors the area for motion.
- **Processing:** The ESP32 handles input debouncing, the 1-minute loitering timer, and MQTT message parsing.
- **Actuation:** An SG90 Servo Motor simulates the door lock (0° = Locked, 90° = Unlocked). A 16x2 LCD displays system status and user messages.
- **Alerting:** A Buzzer provides audio feedback ("Ding Dong" or "Alarm").

4.3 Use-Case Scenario

Scenario 1: The Guest. A friend arrives and presses the button. The buzzer sounds "Ding Dong," the LCD shows "Please Wait," and the User's Web Dashboard pops up a "⚠ Ding Dong!" alert. The user clicks "Unlock" on the dashboard. The servo rotates to 90° , and the LCD updates to "Door Unlocked."

Scenario 2: The Intruder. A stranger stands silently near the door. The PIR sensor detects motion. The ESP32 starts a timer. If the stranger stays for 60 seconds, the system triggers a "BURGLAR ALERT," turning the Dashboard red and sounding the local alarm.

5. Hardware Design

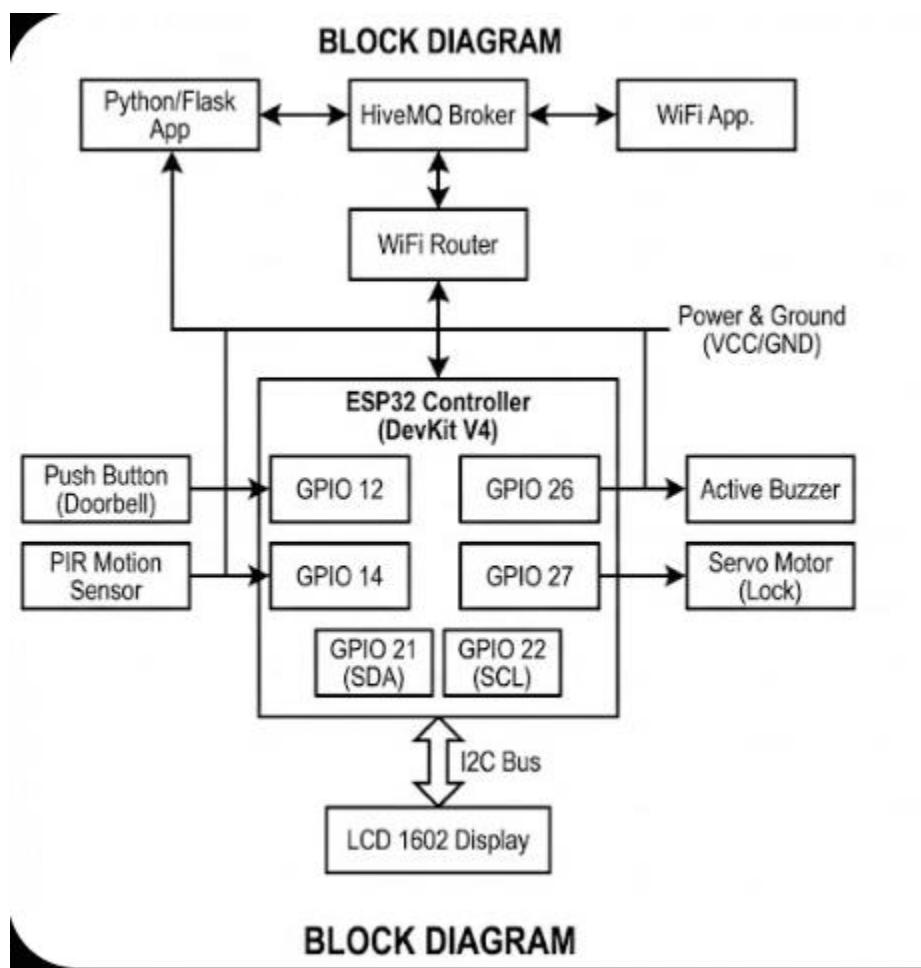
5.1 Components List

Component Name	Specification / Model	Quantity
Microcontroller	ESP32 DevKit V1	1
Actuator	SG90 Micro Servo Motor	1
Sensor	PIR Motion Sensor (HC-SR501)	1
Display	16x2 LCD with I2C Interface	1
Input	Push Button (Tactile Switch)	1
Output	Piezo Buzzer (Passive)	1
Passive	Resistors (Internal Pull-ups used)	0

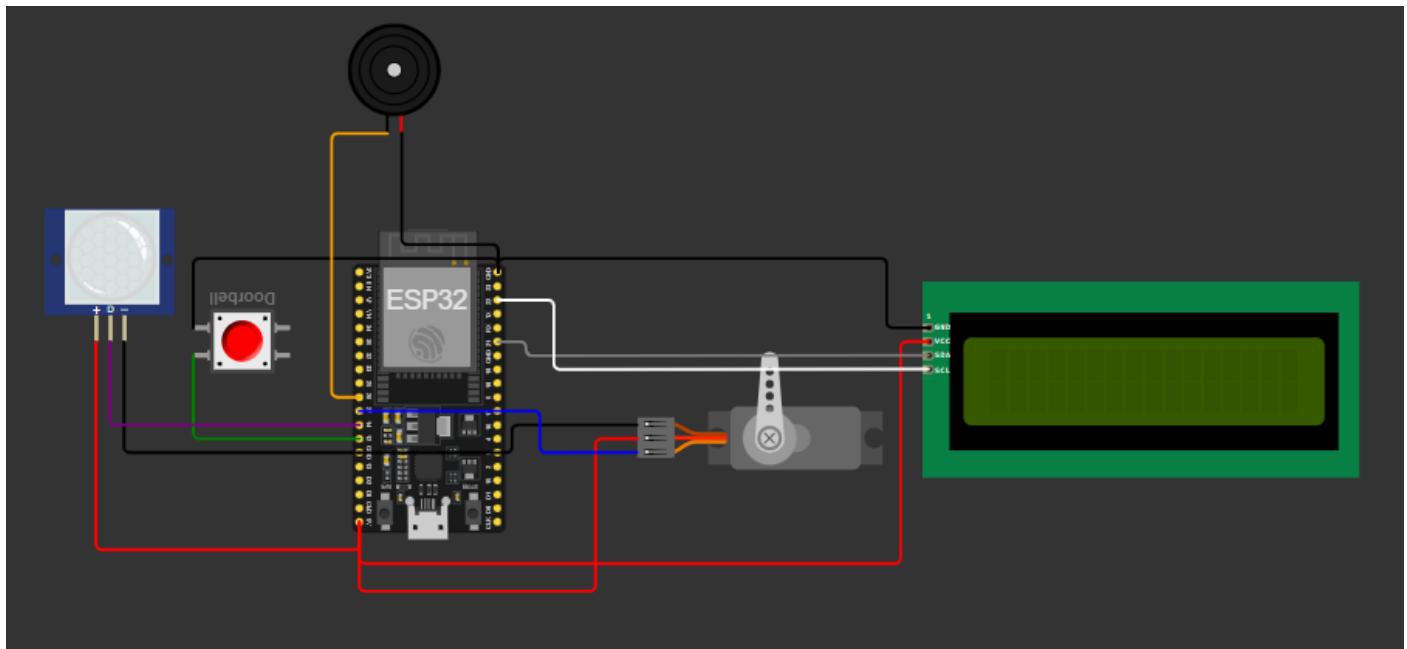
5.2 Component Justification

- **ESP32:** Selected over Arduino Uno because of its built-in WiFi capabilities, which are essential for the IoT/MQTT functionality.
 - **I2C LCD:** Chosen to save GPIO pins; it requires only 2 wires (SDA/SCL) compared to 6+ for a standard LCD.
 - **SG90 Servo:** Sufficient for demonstrating the angular movement required for a deadbolt lock simulation.

5.3 Block Diagram



5.4 Circuit Diagram



Wiring Summary:

- **Button:** GPIO 12 (Input Pull-up)
- **PIR Sensor:** GPIO 14 (Input)
- **Buzzer:** GPIO 26 (Output)
- **Servo:** GPIO 27 (PWM)
- **LCD:** GPIO 21 (SDA), GPIO 22 (SCL)

5.5 Power Considerations

The system in simulation is powered via the USB 5V rail. In a physical deployment, the ESP32 would require a 5V/2A power supply to handle the current spikes of the Servo motor and WiFi bursts to prevent brownouts.

6. Software Design

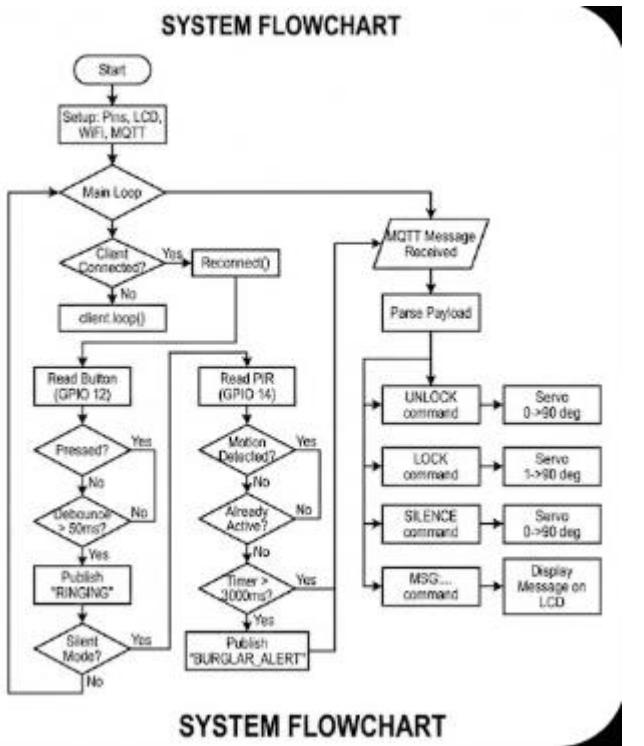
6.1 Development Environment

- **Language:** C++ (ESP32 Firmware), Python 3.12 (Web Backend), HTML/JS (Frontend).
- **IDE:** Wokwi Simulator (for hardware logic), VS Code (for Python).
- **Libraries:** PubSubClient.h (MQTT), ESP32Servo.h, LiquidCrystal_I2C.h, Flask-SocketIO.

6.2 Software Architecture

The C++ firmware uses a non-blocking `loop()` architecture. It checks `millis()` timers to handle the buzzer tones and PIR delays without freezing the processor. The Python backend uses `Flask-SocketIO` to push MQTT updates to the web browser instantly without requiring a page refresh.

6.3 Flowchart / State Diagram



6.4 Algorithms

The "Burglar" Logic: The system does not alert on *any* motion (to avoid false alarms from passing cars). It uses a persistence algorithm:

1. IF Motion == HIGH AND Timer Not Started: Start Timer.
2. IF Motion == HIGH AND Timer > Threshold (60s): Trigger Alarm.
3. IF Motion == LOW: Reset Timer.

Safe Buzzer Logic: To prevent the standard `tone()` function from interfering with the Servo's PWM timer, a custom `safeRingBuzzer()` function was written using `digitalWrite` and `delayMicroseconds` to manually generate sound waves.

7. System Integration

The integration relies on **MQTT Topics** as the common language:

- `alabs/doorbell/status`: The ESP32 publishes "RINGING", "DOOR_LOCKED", or "BURGLAR_ALERT" here.
- `alabs/doorbell/control`: The Python App publishes "UNLOCK", "LOCK", or "SILENCE" here.
- **Timing**: The ESP32 subscribes to the control topic and processes commands within 100ms. The "Heartbeat" logic ensures the connection remains active.

8. Testing and Validation

8.1 Testing Strategy

Testing was performed in three stages:

1. **Unit Testing:** Verifying the Servo moves (Start-up wiggle) and the LCD prints text.
2. **Logic Testing:** Reducing the PIR timer to 5 seconds to verify the alarm logic without waiting a full minute.
3. **End-to-End Testing:** Triggering the physical button in Wokwi and verifying the Alert appeared on the Localhost Web Dashboard.

8.2 Test Cases

Provide specific test cases with expected vs. actual outcomes.

Test Case ID	Input / Condition	Expected Output	Actual Output	Status (Pass/Fail)
TC-01	Press Doorbell Button	Buzzer sounds "Ding Dong", App shows Alert.	Buzzer sounded, Dashboard banner turned Red.	Pass
TC-02	Click "Unlock" on Web	Servo rotates to 90°, LCD shows "Unlocked".	Servo moved smoothly to 90°.	Pass
TC-03	Continuous Motion (60s)	Buzzer Alarm, App "Security Alert".	Alert triggered exactly after threshold.	Pass
TC-04	Click "Silence"	LCD shows "Silent Mode", Buzzer stops.	LCD updated, Ringing was muted.	Pass

8.3 Hardware and Software Testing

A critical software bug was found where "Unlock" commands were ignored. Debugging via the Serial Monitor revealed the Python app was sending "unlock" (lowercase) while C++ expected "UNLOCK". This was fixed by adding `.trim()` and `.upper()` functions to the data parsing logic.

9. Results and Discussion

The system successfully met the requirement of bridging physical hardware with a remote web interface. The response time between clicking "Unlock" on the web and the servo moving was approximately 200-500ms, which is acceptable for a smart home device. The "Safe Buzzer" implementation successfully allowed audio and motor movement to occur simultaneously without resource conflicts.

10. Challenges and Solutions

- **Challenge:** Servo Motor Interaction with Tone.
 - **Problem:** Using `tone()` for the buzzer stopped the Servo from working because they share the same hardware timer on the ESP32.
 - **Solution:** Replaced the library `tone()` function with a custom `safeRingBuzzer()` function that manually toggles the pin.
- **Challenge:** MQTT Message Formatting.
 - **Problem:** The ESP32 failed to recognize commands due to hidden newline characters or capitalization mismatches.
 - **Solution:** Added `msg.trim()` in C++ and forced `.upper()` in Python to sanitize inputs.
- **Challenge:** PIR Simulation Testing.
 - **Problem:** Waiting 60 seconds for every test run was inefficient.
 - **Solution:** implemented a `DEBUG` mode variable to lower the threshold to 3 seconds during testing.

11. Conclusion

The "Smart IoT Doorbell" project successfully demonstrates a full-stack IoT solution. We achieved remote control of a physical lock, real-time sensing of security threats, and instant feedback via a custom web dashboard. The system is robust, handling connection drops via auto-reconnect logic, and user-friendly, offering text feedback on the LCD.

12. Future Work

- **Camera Integration:** Integrate an ESP32-CAM module to stream video to the Flask dashboard when the bell rings.
- **Mobile App:** Port the web dashboard to a native Flutter application for push notifications on the phone.
- **Database Logging:** Connect the Flask app to a SQLite database to keep a history log of all visitors and unlock events.

13. References

- Espressif Systems. (2024). *ESP32 Technical Reference Manual*.
- HiveMQ. (2024). *MQTT Essentials - The Standard for IoT Messaging*.
- Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.
- Wokwi. (2024). *Arduino & ESP32 Simulator Documentation*.

14. Appendices

Appendix A: Source Code

<https://github.com/ashraf-1v/SmartDoorbell.git>

Appendix B: Additional Schematics/Figures

