**Iot Alarm System**

**Simone Incerti, Emanuele Castagno**

Version: 0.0

Date: 18 July 2025

In partial fulfilment of the requirements of the Cyberphysical Systems exam, 1st year of the Electronic Engineering MSc programme of the University of Genova

[Abstract 3](#_Toc203843487)

[1 Introduction 4](#_Toc203843488)

[**Problem Statement** 4](#_Toc203843489)

[**Objectives** 4](#_Toc203843490)

[2 Background and related work 5](#_Toc203843491)

[3 Methodology 5](#_Toc203843492)

[**Tools and Development Environment** 6](#_Toc203843493)

[3.1 Architecture definition 6](#_Toc203843494)

[3.2 System modelling with UPPAAL 7](#_Toc203843495)

[3.3 Wiring and connection 9](#_Toc203843496)

[3.4 Firmware development 10](#_Toc203843497)

[3.5 Custom shield 11](#_Toc203843498)

[4 Experimental results 11](#_Toc203843499)

[5 Conclusions and future work 12](#_Toc203843500)

[References 13](#_Toc203843501)

Index of figures

[Figure 1: System Architecture Diagram 6](#_Toc203843502)

[Figure 2: Core Alarm Logic 8](#_Toc203843503)

[Figure 3: Burglar State 8](#_Toc203843504)

[Figure 4: Pin Checker 8](#_Toc203843505)

[Figure 5: User Interface 8](#_Toc203843506)

[Figure 6: Main controller automaton 8](#_Toc203843507)

[Figure 7: wiring diagram 9](#_Toc203843508)

[Figure 8: Firmware functionality flowchart 10](#_Toc203843509)

[Figure 9: back of the custom shield 11](#_Toc203843510)

[Figure 10: Front of the custom shield 11](#_Toc203843511)

# Abstract

This project details the design and implementation of a modular IoT-based intrusion detection system. The primary objective was to create a standalone device capable of real-time threat detection and instant user notification.

The system's core features a **dual-microcontroller architecture** that intelligently distributes tasks. An **STM32** microcontroller serves as the main controller, dedicated to managing the alarm logic, polling a PIR sensor and a reed switch for intrusions, and handling all user interactions through a keypad and an OLED display. A secondary **ESP32** microcontroller operates as a dedicated communication coprocessor, managing Wi-Fi connectivity and executing API calls to the CallMeBot service for delivering immediate alerts to a smartphone.

This distributed architecture is fundamental to the system's robustness. By assigning real-time sensor management to the **STM32** and delegating all network operations to the **ESP32**, the core security functions are effectively isolated from network-related latency. The system’s logic was formally modeled and verified to guarantee reliability. The result is a robust and effective prototype that validates this architecture for modern security applications.

# Introduction

### **Problem Statement**

Modern smart homes increasingly rely on affordable and reliable security systems to monitor potential intrusions. However, existing commercial solutions are often expensive, proprietary, or difficult to customize for specific needs. The challenge lies in developing a low‑cost yet flexible IoT‑based alarm system that can effectively monitor doors and windows, notify users in real time, and remain easy to install and operate even by non‑technical users.

During the design process, several technical challenges were addressed:

* **Component selection**: finding low‑cost, widely available, and well‑supported hardware components that are mutually compatible.
* **Real‑time constraints**: ensuring correct timing behavior for sensor polling, user input handling, and alarm triggering.
* **Persistent data storage**: securely storing the PIN code on the STM32 microcontroller while maintaining reliability across system resets and power cycles.

### **Objectives**

The main objective of this work was to design and implement a functional, standalone IoT alarm system capable of:

* Performing **real‑time monitoring** of multiple sensors to track the security status of doors and windows.
* Providing a **functional human interface** for both daily operation (arming/disarming) and initial system setup, using a keypad and OLED display.
* Establishing **communication between multiple microcontrollers** to separate alarm logic from network and notification handling.
* Sending **Wi‑Fi notifications** to the user in case of an intrusion, ensuring timely awareness of security breaches.

By focusing on simplicity, modularity, and cost‑effectiveness, the project aims to demonstrate that a reliable security solution can be achieved with open‑source tools and affordable hardware while still offering the essential functionalities expected from a home alarm system.

# Background and related work

IoT-based alarm systems have been widely studied in the context of smart home automation and low-cost security solutions. Many existing implementations leverage microcontrollers, wireless connectivity, and a variety of sensors to detect unauthorized access and provide timely alerts to users. Commercial solutions typically integrate motion detectors, magnetic reed switches, and keypads with cloud-based notification services, but they are often proprietary and lack flexibility for customization.

In the academic and maker communities, open-source projects have demonstrated how affordable components can be combined to create reliable security devices. Previous work highlights the use of **PIR sensors for motion detection**, **reed switches for door/window monitoring**, and **microcontroller-based processing** for low-power, real-time operation. Communication protocols like **I²C, SPI, or UART** are commonly employed to connect multiple microcontrollers or peripherals, while cloud services and messaging APIs (such as CallMeBot or MQTT brokers) enable remote notifications.

For the development of such systems, a variety of open-source libraries are used to simplify hardware integration, including libraries for **keypad management**, **OLED display control**, and **network communication**. Popular development tools include **STM32CubeIDE** for STM32-based firmware, **Visual Studio Code** for ESP32 development, and modeling tools like **UPPAAL** to verify timing behavior and system correctness prior to implementation.

Typical methodologies in similar projects involve:

* **System modeling and formal verification**, to ensure correct timing and state transitions.
* **Incremental firmware development**, adding and validating one feature at a time.
* **Simulation and testing**, using prototyping tools such as **Fritzing** for circuit visualization and breadboarding before final PCB design.

This project follows a similar approach by integrating two cooperating microcontrollers—STM32 for the alarm logic and ESP32 for connectivity—while leveraging open-source libraries and APIs to ensure modularity and ease of future expansion.

# Methodology

The development of the IoT alarm system followed a structured, iterative methodology that combined **system modeling**, **hardware design**, **firmware development**, and **functional testing**. The process ensured that both the logical behavior and the physical implementation of the system were reliable and met the intended security requirements.

### **Tools and Development Environment**

A variety of software tools were employed to support each stage of the project:

* **UPPAAL** – used for formal modeling and verification of the system’s real-time behavior, ensuring correct timing and state transitions.
* **STM32CubeIDE** – for firmware development and debugging on the STM32 microcontroller.
* **Visual Studio Code** – for ESP32 firmware programming and Wi‑Fi connectivity management.
* **Fritzing** – for designing and visualizing the wiring diagrams and breadboard layouts.
* **Draw.io** – for creating system architecture and workflow diagrams.
* **CallMeBot API** – for sending instant WhatsApp notifications to the user in case of intrusion.

## Architecture definition

This document outlines the architecture of an integrated security alarm system, a common approach in modern IoT applications **[1]** The system is architected around an **STM32 F401 NUCLEO development board**, a powerful and versatile microcontroller widely used in embedded applications **[4].** It serves as the central processing unit, managing all peripheral components and executing the core logic. Intrusion detection is accomplished through a combination of an **HC-SR501 Passive Infrared (PIR) sensor** to detect motion and a **reed switch** to monitor the state of doors and windows, both of which are connected to the STM32's GPIO pins.

For user interaction, the system incorporates a **4x4 matrix keypad** that allows for arming and disarming the system via a PIN code, and a **128x64 I2C OLED display** to provide visual feedback, display system status, and guide the user. In the event of a detected intrusion, the STM32 triggers a local **buzzer** for an immediate audible alert. Simultaneously, it communicates with a **Wemos S2 Mini (ESP32)** module via the I2C interface. The ESP32 module is dedicated to handling network connectivity, utilizing its built-in Wi-Fi to send a remote notification to the user through a specified API, thereby ensuring a comprehensive and responsive security solution.

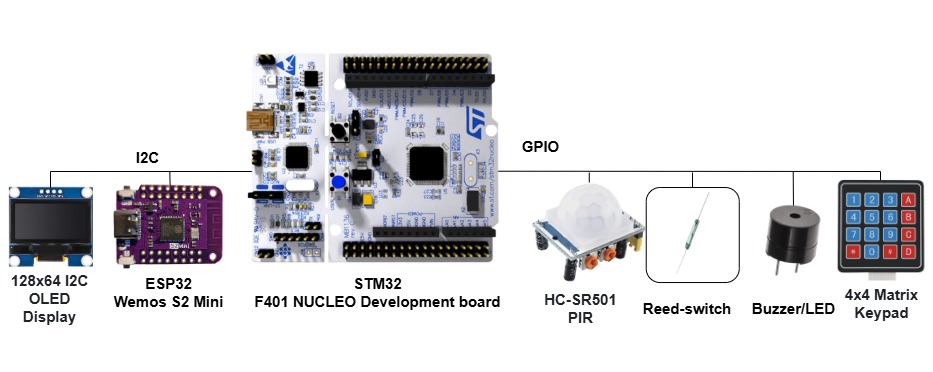


Figure 1: System Architecture Diagram

## System modelling with UPPAAL

To ensure the system's reliability and functional correctness, we employed the **UPPAAL model-checking tool [2]**. This approach allowed us to create a formal model of the system's real-time behaviour and verify its properties before implementation.

We designed a network of timed automata to represent the different components and their interactions, as shown in the diagrams. Specifically, we modelled:

* A simple automaton to represent the **burglar state** (Figure 3), transitioning from Idle to Entered and setting the burgler\_in variable.
* A dedicated automaton for the **PIN entry and verification logic (**Figure 4**)**, which includes states for handling failed attempts and a timed PIN\_suspended state to lock the system after too many incorrect entries.
* A **main controller automaton** (Figure 6) that integrates user inputs (buttons A and B), the alarm status (alarm\_active), and the PIN check results (PIN\_ok, PIN\_checked) to manage the overall system state (e.g., transitioning between Off and On).
* An automaton for the **core alarm logic** (Figure 2) that manages the primary On and Off states. Upon receiving an activate\_alarm signal, it moves to the On state; if an intrusion is detected (burgler\_in), it proceeds to trigger a SendWhatsapp notification and then a Sound alert, before being reset via a deactivate\_alarm signal.
* A simple **user interface** automaton (Figure 5) for handling button inputs. It emits btn\_A! or btn\_B! to signal a user action, then synchronizes with the insert\_PIN and PIN\_checked channels before returning to the initial state to await the next input.

By using shared variables and synchronization channels (like activate\_alarm!), these automata interact to simulate the complete system workflow. This formal model enabled us to verify critical properties, such as ensuring the alarm correctly activates upon an intrusion when armed and that the user interface logic behaves as expected under all conditions.

A diagram of a network

AI-generated content may be incorrect.

A diagram of a triangle

AI-generated content may be incorrect.

Figure 2: Core Alarm Logic

A diagram of a diagram

AI-generated content may be incorrect.Figure 3: Burglar State

A diagram of a pin

AI-generated content may be incorrect.

Figure 4: Pin Checker

Figure 5: User Interface

A diagram of a computer code

AI-generated content may be incorrect.

Figure 6: Main controller automaton

## Wiring and connection

For the visual representation of the circuit, **Fritzing** software was utilized. This electronic prototyping tool allows for the creation of clear and intuitive wiring diagrams, which are ideal for documenting and replicating hardware projects[3].

As illustrated in the diagram (Figure 7), the **STM32 NUCLEO-F401RE** development board serves as the central component in the breadboard assembly. The various peripherals are connected to it as follows: the **4x4 matrix keypad** is interfaced directly with a set of digital pins for input reading. The **HC-SR501 PIR sensor**, the **reed switch** (which is configured with 10KΩ pull-down resistors), and the **buzzer** are also connected to GPIO pins for detection and signaling purposes. Communication with more complex devices is handled via the I2C bus; both the **128x64 pixel OLED display** and the **Wemos S2 Mini** Wi-Fi module are wired to the STM32's I2C pins, enabling efficient data exchange with minimal wiring. This diagram provides a precise and comprehensive wiring map, essential for the correct assembly of the physical prototype

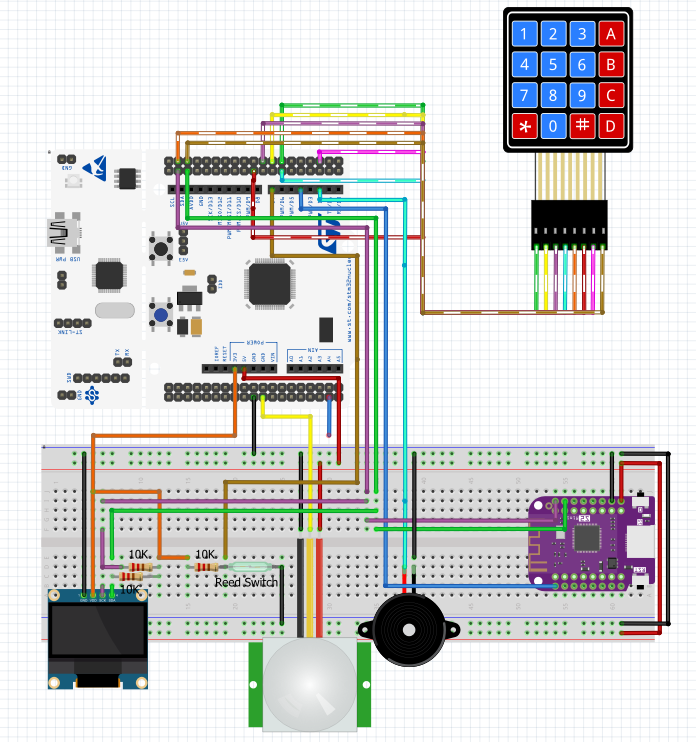


Figure 7: wiring diagram

## Firmware development

The firmware was developed following an incremental, step‑by‑step approach. Each functionality was implemented one at a time and thoroughly tested before proceeding to the next stage. This method allowed us to ensure that every module worked correctly in isolation as well as in combination with previously developed components. During the process, various challenges and issues were encountered, but they were systematically analyzed and resolved as part of the iterative development cycle. As a result, the final firmware proved to be stable, reliable, and fully aligned with the system requirements.

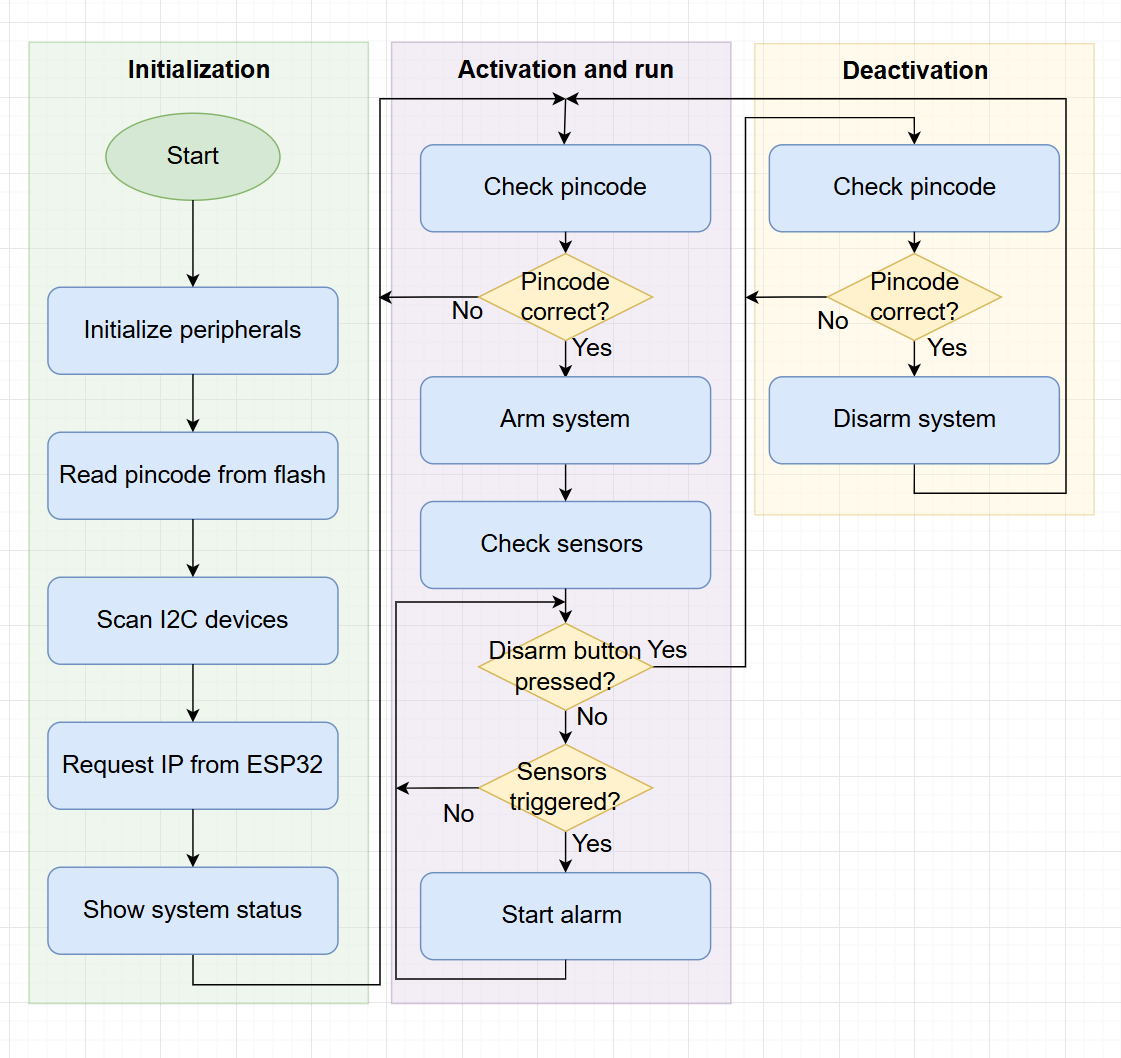
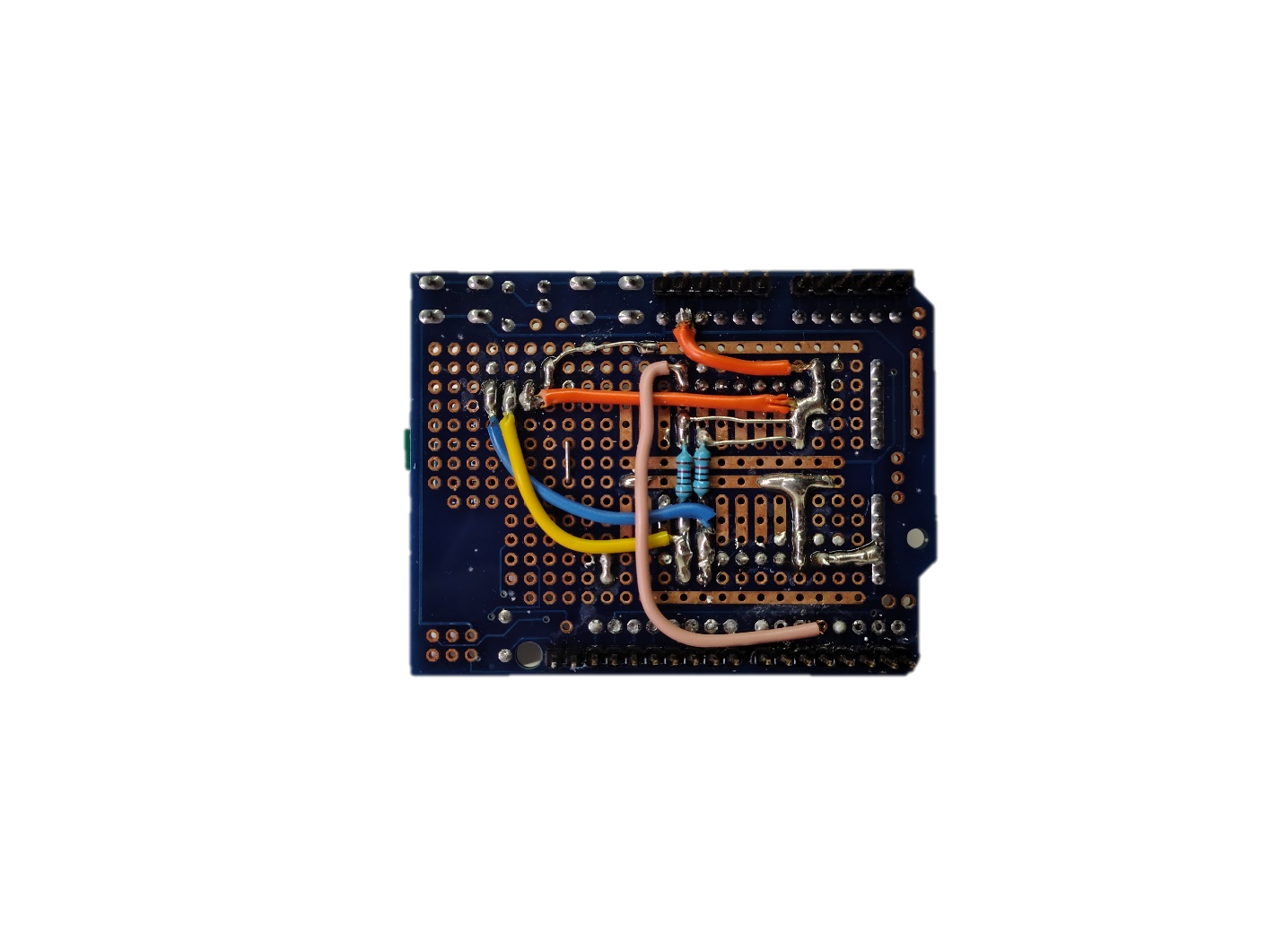


Figure 8: Firmware functionality flowchart

## Custom shield

To ensure a stable and robust connection for the I2C peripherals, we developed a custom prototype shield. This shield, designed to mount directly onto the main controller board (the STM32), integrates both the ESP32-S2 module and the OLED display onto a single protoboard. As shown in the image, the underside of the shield features carefully routed wiring and includes the necessary pull-up resistors on the I2C communication lines (SDA and SCL). This design minimizes the risk of loose connections that can occur with breadboards and jumper wires, providing a reliable hardware foundation for the system.

A close-up of a circuit board

AI-generated content may be incorrect.

Figure 9: back of the custom shield

Figure 10: Front of the custom shield

# Experimental results

The performance of the system was evaluated under standard operating conditions. Table 4.1 summarizes the key electrical and timing characteristics measured during our experiments:

|  |  |
| --- | --- |
| **Specification** | **Measured Value** |
| Voltage | 5.0 |
| Average Idle Consumption | 1.2 |
| Peak Power Consumption | 1.5 |
| Sensor Polling Rate | 4 |
| Sensor Polling Cycle Time | 8 |

**Table 4.1** Measured electrical and timing specifications of the system.

Beyond these quantitative measurements, we also conducted a functional validation to ensure the system operated correctly in a realistic environment. During testing:

* The sensor reliably polled data at 4 Hz, with consistent 8  polling cycle times.
* Power consumption remained within expected limits both in idle (≈1.2 W) and peak (≈1.5 W) conditions.
* Data acquisition and system responses were stable and error‑free throughout the testing period.

All key subsystems functioned as intended, and the system demonstrated robust and reliable performance under the described operational parameters.

# Conclusions and future work

This work has demonstrated the successful development of a functional alarm system designed for simple door and window intrusion monitoring. The system proved to be reliable in detecting unauthorized access attempts and provided real‑time Wi‑Fi notifications to the user. Additionally, the setup process was designed to be intuitive, ensuring that even non‑experienced users can easily install and configure the device.

Despite these achievements, the current prototype has some limitations. The system is primarily suited for small‑scale security scenarios and does not yet support advanced user management or large‑area monitoring. Furthermore, while the hardware and firmware perform reliably, there is still room for optimization in terms of power consumption, hardware robustness, and scalability.

Possible directions for future development include:

* **User PIN management**: allowing users to easily modify their saved PIN codes for improved security.
* **Integration of additional sensors**: for example, RFID authentication.
* **Improved hardware design**: developing a more robust and compact PCB suitable for long‑term use and mass production.
* **Optimized power consumption**: enabling longer battery life or alternative low‑power modes.
* **Enhanced connectivity**: implementing a web application or cloud‑based interface for remote access, system status monitoring, and event history tracking.
* **Modular expansion**: enabling the system to support multiple doors, windows, or even full‑house security coverage.
* **Fail‑safe mechanisms**: such as backup power sources or offline operation modes in case of network failure.

By addressing these areas, the system can evolve into a more comprehensive and scalable alarm solution suitable for both residential and small commercial environments.

# References

[1] S. Kaadan and M. Oussalah, "Automatic Generation of Attack Trees from Textual Requirements," in Proceedings of the 16th International Conference on Software Engineering and Applications (ICSEA 2021), New York, NY, USA: Association for Computing Machinery, 2021, pp. 1–8. doi: 10.1145/3480571.3480587.

[2] G. Behrmann, A. David, and K. G. Larsen, "A Tutorial on UPPAAL," in *Formal Methods for the Design of Real-Time Systems*, 2004, pp. 200–236

[3] R. Knörig, J. V. O. I. S. T. E. I. D. U. I. P., and R. Wettach, "Fritzing: a tool for advancing electronic prototyping for designers," in Proceedings of the 3rd International Conference on Tangible and Embedded Interaction, 2009, pp. 351-358.

[4] C. Noviello, *Mastering STM32*, 2nd ed. Leanpub, 2022.