University of Hertfordshire

School of Physics, Engineering and Computer Science
University of Hertfordshire

BEng (Hons) Electrical and Electronic Engineering

5FTC2135

Analogue and Mixed-Signal Design

Smart Industrial/Home Safety Monitoring System

Group Report

THIS COPY BELONGS TO:	*GA02*

Group members

Name	Role/Task taking in this project
Dulina Nadith	Sensor wiring, digital signal coding, leak detection logic, hardware testing, coding
Kavinda	Analog signal filtering, sound testing
Thusitha	Touch sensor integration, Soldering
Athil	Temperature monitoring logic, threshold alerts, sensor calibration

Abstract

This report presents the complete design, development, and testing of a Smart Industrial/Home Safety Monitoring System built using an ESP32 microcontroller, a combination of analogue and digital sensors, and a mixed signal architecture. The aim of the project was to create a low cost, energy efficient device that can detect key safety hazards such as metal intrusion, door or window breaches, water leakage, overheating, loud unusual sounds, and manual emergency triggers.

The system continuously reads data from six sensors, processes them through both digital logic and analogue signal conditioning, and then displays the results in real time on a 1.3" OLED display. It also uses an RGB LED and buzzer to provide immediate user alerts. The analogue sound detection subsystem, built with an LM358 op amp, demonstrates the mixed signal aspect of the project by converting subtle microphone signals into clean, detectable ADC readings. The entire system was implemented first on a breadboard, then transferred to a soldered dot board, and a 3D enclosure was also designed for future deployment.

Overall, the project successfully meets the functional and nonfunctional requirements, and the final prototype behaves like a compact, modular safety device suitable for homes, workshops, and small industrial environments.

Table of Contents

1.Introduction 4

1.2 System Overview & Problem Definition 4

1.3 Functional and Non-Functional Requirements 5

1.4.1 Bill of Materials (BoM) 7

1.4. 2 Environmental Impact Analysis 8

2. Mixed-Signal System Design 9

2.1 System Architecture Overview 9

2.2 Detailed Block Diagram Explanation 11

2.3 Sensor Interfacing and Mixed-Signal Integration 11

2.3.1 SN04N Inductive Proximity Sensor (Digital) 11

2.3. 2 Magnetic Reed Switch – Door/Window Sensor (Digital) 12

2.3. 3 Float Switch – Water Level Detection (Digital) 12

2.3. 4 DS18B20 – Digital Temperature Sensor (1-Wire Digital) 12

2.3. 5 TTP223 Touch Sensor (Digital) 12

2.3. 6 SY-M213 Sensor (Analogue) 12

2.4 Analogue Sound Processing 13

2.5 User Interface Subsystem (OLED, Buzzer, RGB LED) 13

2.6 Power Supply & Noise Management (Ensuring Mixed-Signal Stability) 14

2.7 Digital vs Analogue Signal Integration (Mixed-Signal Justification) 14

2.8 System Integration Summary 15

3. Embedded Code Development 15

3.1 Code Architecture Overview 15

3.2 Library Usage and Their Purposes 15

3.3 Library Usage and Their Purposes 16

3.4 Sensor Data Collection (Digital Inputs) 16

3.5 Analogue Sound Processing 17

3.6 Hazard Detection 17

3.7 OLED Display Rendering (Safe / Status / Alert) 17

3.8 Output Control (RGB LED + Buzzer) 18

3.9 Error-Handling Techniques for Sensor Failures & Communication Issues 18

4. Electronic Product Testing and Validation 18

4.1 Performance Evaluation Under Varying Environmental Conditions 19

4.2 Sensor Calibration and Accuracy Validation 20

4.3 Discrepancy Analysis and Solutions 20

5. Risk Management & Error Handling 21

5.1 Potential Risks in Mixed Signal Operation 21

5.2 Error Detection and Fault Tolerance Strategies Implemented 21

6. Conclusions and Further Development 22

6.1 Conclusion 22

6.2 Further Development 23

1.Introduction

Modern safety systems rely heavily on mixed signal design, which is the integration of both analogue circuits (such as filters, amplifiers, and sensors that produce continuous signals) and digital circuits (such as microcontrollers, logic decisions, displays, and communication). Mixed-signal systems are used everywhere, from home alarm systems and fire detectors to industrial monitoring and IoT devices.

In this project, we aimed to design a complete mixed signal safety monitoring system that can identify several types of hazards simultaneously. We combined multiple sensors, some producing clean digital outputs and others producing noisy analogue signals and integrated them into one ESP32 based platform. The ESP32 then performs all processing, decision making, display output, and warnings.

Mixed signal design is important because real world environments are never purely digital. Sound, temperature, vibrations, water levels, proximity detection, these are all analogue or physical events that must be converted into electrical signals. A good safety system must understand both sides. This project helped us learn how to filter noise, condition analogue signals, prevent false triggers, protect microcontroller pins, and structure a reliable embedded system ready for real deployment.

The device we designed is small, lightweight, low cost, and operates from a 12 V adapter through a buck converter, making it energy efficient and suitable for continuous usage.

1.2 System Overview & Problem Definition

The Smart Industrial/Home Safety Monitoring System is designed to operate as a multi hazard detection unit for both home and industrial environments. Most affordable safety devices on the market monitor only one parameter (e.g., just smoke, just motion, or just temperature). However, real situations usually involve multiple risks happening at the same time. A water leak can occur near a power system. A door might open unexpectedly during high temperature conditions. A loud noise might indicate breakings or mechanical accidents.

Our device solves this gap by combining six sensors into one integrated platform, giving a full 360° safety view.

The system constantly monitors:

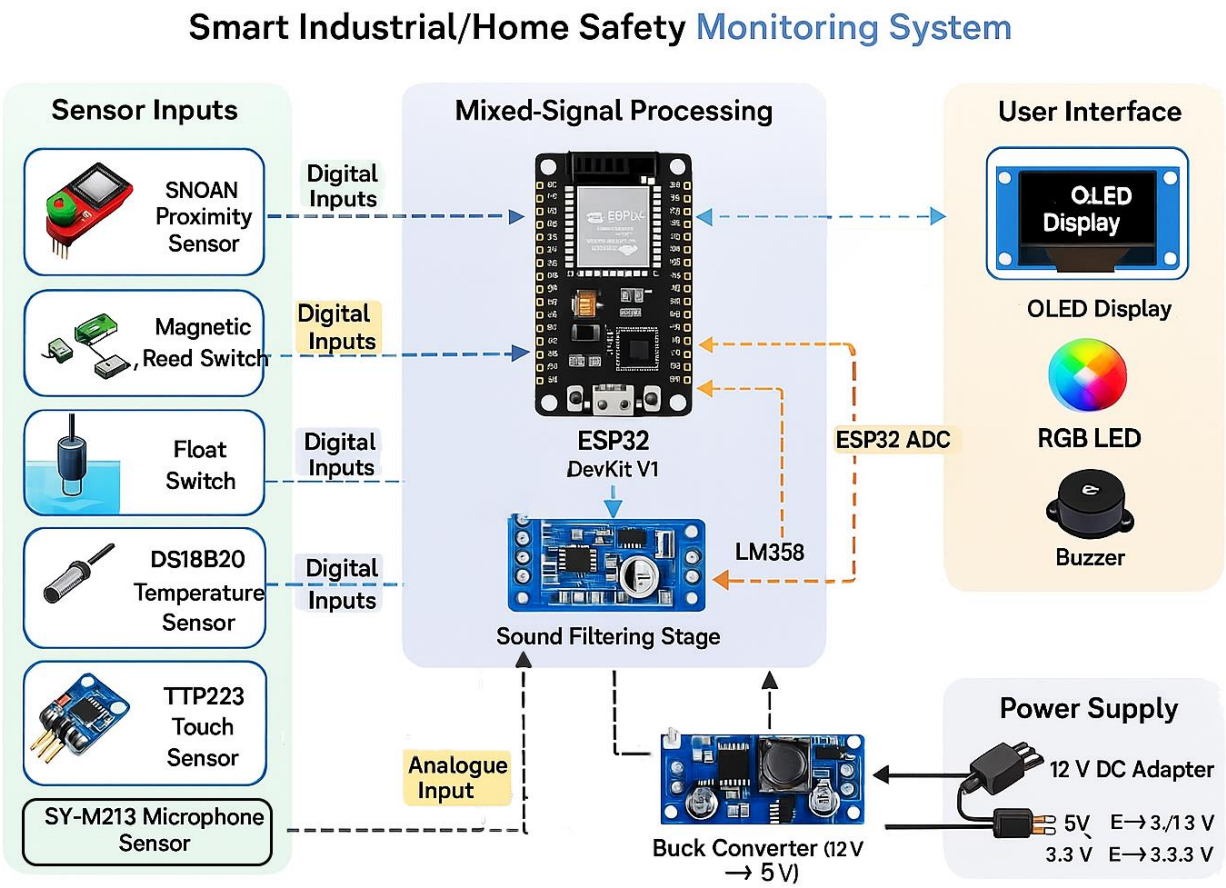
- Metal proximity
- Door/window opening
- Water leakage / water level
- Ambient temperature
- Loud sounds (analog signal processed into digital hazard)
- Manual panic input

Whenever a sensor detects a problem, the ESP32 triggers:

- A red LED
- A buzzer alert
- A flashing ALERT screen
- A clear message showing the exact hazard detected

And when everything is fine, the system shows a calm SAFE screen.

To remain practical and realistic, we powered the entire system using a 12 V adapter stepped down using a buck converter, which is how real world alarm systems and embedded safety devices are usually powered.



Problem Definition:

There is a lack of affordable, modular, low power devices capable of monitoring multiple safety factors simultaneously in homes and small industries. This project aims to solve that by building a mixed signal monitoring unit that combines temperature, sound, proximity, intrusion, water level, and user emergencies into one compact solution.

1.3 Functional and Non-Functional Requirements

Functional Requirements (What the system must do)

Metal Proximity Detection:

- Detect metal near machinery using SN04N and trigger alert.

Intrusion Detection:

- Magnetic reed switch must detect door/window opening.

Water Level & Leak Detection:

- Float switch must detect rising water and warn users.

Temperature Monitoring:

- DS18B20 must show accurate readings and detect overheating.

Sound Level Monitoring:

- SY-M213 analogue sensor + LM358 amplifier must detect loud unusual noises.

Touch Panic Input:

- TTP223 touch sensor must instantly trigger emergency alert.

OLED Visual Interface:

- Display must show SAFE/STATUS/ALERT modes with clear instructions.

RGB LED & Buzzer Alerts:

- LED colors and buzzer patterns must correspond to danger levels.

Mixed-Signal Processing:

- ADC sampling, filtering, debouncing, threshold comparison must work reliably.

Fault Handling:

- System must detect sensor errors like DS18B20 disconnection.

Non-Functional Requirements (How well it must work)

Reliability:

- Must run 24/7 without freezing or rebooting.

Accuracy:

- Sensors must be within realistic tolerance (e.g., ±1°C for DS18B20).

Real-Time Response:

- Alert reaction must occur under 200ms.

Usability:

- Interface must be simple and understandable for anyone.

Modularity:

- Sensors can be added, removed, or replaced easily.

Energy Efficiency:

- Total consumption ~1.5–2 W.

Safety:

- All pins protected from overvoltage; wiring insulated and soldered.

Maintainability:

- Code structured clearly; hardware layout neat and labeled.

Environmental Impact:

- Low power usage prevents water and temperature-related damage.

Affordability:

- Full system cost stays under LKR 3500–4000.

The Bill of Materials (BoM) represents all the components used in the Smart Industrial/Home Safety Monitoring System. Since one of the key project requirements was to make the device low cost, modular, and energy efficient, each component was selected carefully to balance price, performance, and reliability. The goal was to create a real, deployable product rather than just a lab prototype, while keeping the total budget affordable for normal households or small workshops.

1.4.1 Bill of Materials (BoM)

Below is the detailed BoM including quantities, unit prices and purpose of each component:

Item	Component	Qty	Unit Cost (LKR)	Total (LKR)	Purpose
1	ESP32 DevKit V1	1	1150	1150	Main microcontroller, handles sensors, logic, and OLED display
2	SN04N Inductive Proximity Sensor	1	450	450	Metal proximity detection
3	Magnetic Reed Switch	1	120	120	Door/window intrusion detection
4	Float Switch	1	240	240	Water leakage / water level detection
5	DS18B20 Temperature Sensor	1	110	110	Digital temperature measuring
6	TTP223 Touch Sensor	1	20	20	Manual emergency/panic trigger
7	SY-M213 Sound Sensor	1	130	130	Analogue sound sensing
8	LM358 Op-Amp	1	80	80	Analogue sound amplification & filtering
9	1.3" OLED Display (SH1106)	1	900	900	System UI, real-time safety monitoring
10	RGB LED Module	1	140	140	Visual feedback: Safe/Alert indication
11	Buzzer Module	1	110	110	Audible alerts for hazards
12	Buck Converter (12 V → 5 V)	1	150	150	Efficient step-down power supply
13	12 V DC Adapter	1	350	350	Main power input
14	Dot Board (for soldering)	2	60	120	Permanent hardware implementation
15	Passive components (resistors, capacitors)	–	100	100	Filtering, dividers, noise reduction
16	Jumper wires, connectors	–	100	100	Wiring and integration
Total Cost				**≈ 3,950 LKR**	*Under required low-cost limit*

1.4. 2 Environmental Impact Analysis

Even though our device is a small electronic system, we analyzed its environmental and societal impact to match the module learning outcomes related to sustainability and responsible engineering.

1. Power Consumption & Energy Efficiency
The entire system operates on only 1.5–2 watts, which is extremely low.

This is due to:

- Efficient buck converter (≈90% efficiency)
- ESP32's low-power operation
- Sensors that consume very little current
- OLED display usage optimized (sleep when not needed)

Low power usage has the following benefits:

- Suitable for 24/7 monitoring without high electricity bills
- Can be powered from solar or UPS units
- Reduces carbon footprint
- Ideal for rural or low-resource environments

2. Sustainable Design Choices

(a) Low-cost modular design

- Users can replace one sensor instead of throwing the whole device away.
- Easy maintenance improves device lifespan.
- Reduces e-waste.

(b) Minimal material usage

- Only two small dot boards
- Lightweight electronics
- No heavy metal components
- The 3D enclosure can be printed using PLA, a biodegradable material

(c) Safe water detection
Detecting leaks early avoids:

- Water wastage
- Electrical short circuits
- Flooding damage

This supports environmental safety and household sustainability.

3. Reduced Electronic Waste (E-waste)

We used:

- Off the shelf sensors
- Reusable ESP32 & OLED modules
- Soldered boards designed for long-term durability
- Instead of one-time-use modules, this system can be repaired and upgraded easily.

4. Environmental & Societal Relevance

This device helps protect:

- Homes from fire, break-ins, and leaks
- Workers in small industries
- Electrical infrastructure from overheating
- Water storage areas from overflowing
- Household energy consumption through early detection of faults

5. Ethical & Responsible Engineering

During the design, we ensured:

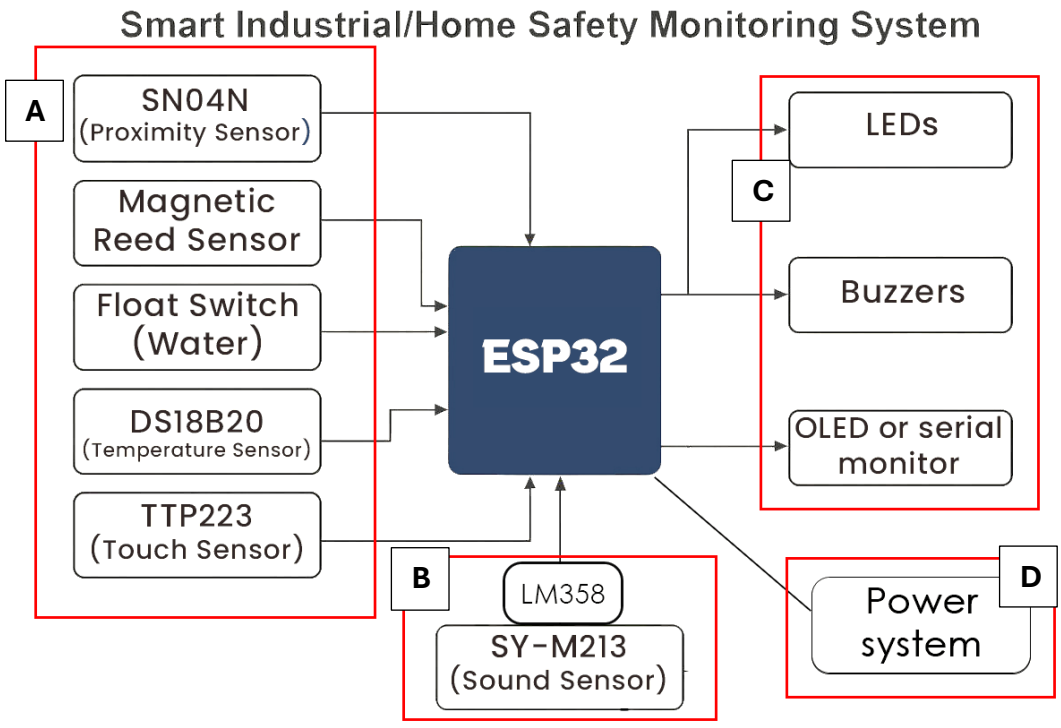
- The system is safe and does not overheat
- ESP32 inputs are protected from overvoltage
- Wiring is soldered securely to avoid short circuits
- Power supply is stable and efficient
- The device does not falsely trigger or mislead users

2. Mixed-Signal System Design

The Smart Industrial/Home Safety Monitoring System is built on a true mixed-signal architecture, meaning it processes both analogue and digital signals within a single embedded platform. This section explains how each sensor, circuit stage, and interface works together with the ESP32 to form a complete safety monitoring solution.

The "mixed signal" nature of the project is mainly highlighted by the analogue sound detection subsystem, the digital GPIO inputs, the ADC conversions, and the serial/I^2C communication used for the OLED display. Bringing all these together in one stable system is the core engineering challenge of this project.

2.1 System Architecture Overview



The entire system can be divided into four major subsystems:

(A) Sensor Subsystem

- SN04N proximity sensor (digital)
- Magnetic reed switch (digital)
- Float switch (digital)
- DS18B20 temperature sensor (digital 1 Wire)
- TTP223 touch sensor (digital)
- SYM213 microphone sensor (analogue → LM358 → ADC)

The sensor subsystem is the main input layer of the system. Five sensors generate clean digital HIGH/LOW signals for states such as door open/closed, water level high/low, metal detection, temperature data, and touch activation. The microphone sensor is the only analogue input; it produces a low level AC waveform proportional to environmental sound intensity. This combination of both digital and analogue sources makes the system genuinely mixed signals from the very beginning of its architecture.

(B) Mixed-Signal Processing

- LM358 operational amplifier
- Analogue filtering stages
- ESP32 ADC sampling and averaging
- Debouncing and threshold logic

This subsystem is responsible for processing both digital and analogue signals so the ESP32 can evaluate them correctly. The LM358 amplifies the microphone's millivolt output to a usable level, while RC filters remove DC hum and high frequency noise. The ESP32 then digitizes the filtered analogue waveform using its 12-bit ADC and averages multiple samples to reduce noise. At the same time, mechanical sensors such as reed and float switches are debounced in software to eliminate false triggers. Together, these steps ensure that all incoming data analogue or digital is clean, stable, and ready for decision making.

(C) User Interface (UI) Subsystem

- 1.3" OLED Display (I^2C)
- RGB LED (digital outputs)
- Buzzer (PWM/tone output)

Once the system processes sensor data, it communicates the result to the user through the UI subsystem. The OLED display is controlled using the I^2C protocol and supports three main modes: Safe View, Status View, and Alert View. The RGB LED gives immediate visual feedback by turning green in safe conditions and red during hazards. The buzzer provides audible alerts using PWM generated tones. This subsystem makes the final system user friendly and ensures clear communication of the device's state.

(D) Power Supply Subsystem

- 12 V DC Adapter
- Buck converter (12 V → 5 V)
- ESP32 onboard regulator (5 V → 3.3 V)
- Decoupling capacitors & grounding strategy

The mixed signal nature of the system requires a stable and noise-free power supply because analogue signals and ADC readings are sensitive to voltage fluctuations. A 12 V adapter provides the main supply, which is stepped down to 5 V by a buck converter. The ESP32's internal regulator then supplies 3.3 V to the logic components. Decoupling capacitors smooth voltage transitions and prevent noise from affecting the analogue stages. All grounds are tied to a common reference to avoid ground loops and interference.

Each subsystem has been designed to operate efficiently and reliably under continuous operation.

2.2 Detailed Block Diagram Explanation

The block diagram illustrates the complete flow of signals through the system. The ESP32 DevKit V1 is positioned at the center as the main controller, receiving both digital and analogue inputs from sensors. Digital sensors such as the reed switch, float switch, SN04N, TTP223, and DS18B20 connect directly to GPIO pins, each providing clear logical states. The DS18B20 uses a single 1 Wire data line with a pull up resistor, indicating how digital temperature readings are received serially.

The analogue microphone path is shown separately, beginning with the SY-M213 sensor that outputs a low-level AC waveform. This signal enters the LM358 amplifier block, which increases the signal amplitude. After amplification, it passes through RC filters that shape the frequency response and reduce noise. The filtered signal is then delivered to the ESP32 ADC input, where it is digitized and processed.

The central processing block of the ESP32 represents all internal operations, including ADC sampling, digital input reading, debouncing, threshold logic, and multi-sensor fusion. From here, the processed data is sent through the I^2C lines to the OLED display. Additional GPIOs drive the RGB LED and buzzer, completing the user interface section. The power subsystem connects to all blocks in the diagram, ensuring stable and noise-free operation through regulated 5 V and 3.3 V rails.

This block-level explanation demonstrates how analogue signals, digital inputs, communication protocols, and output devices are interconnected in a fully integrated mixed-signal design.

2.3 Sensor Interfacing and Mixed-Signal Integration

The sensor interfacing stage is one of the most important parts of the mixed-signal system, because every sensor operates using a different electrical principle and requires a different method of signal conditioning before the ESP32 can read it reliably. This subsystem demonstrates the full range of mixed-signal engineering from direct digital logic inputs to analogue waveform processing using amplification and filtering. ESP32 acts as the central junction where all sensor signals converge, and therefore the interfacing must be carefully designed to avoid damage, noise interference, and incorrect readings.

2.3.1 SN04N Inductive Proximity Sensor (Digital)

The SN04N inductive proximity sensor is a 12V output industrial detector designed to sense the presence of metal using changing electromagnetic fields. Since the ESP32 uses 3.3 V logic, the SN04N cannot be connected directly to a GPIO pin. To safely integrate this sensor, a resistor-based voltage divider is used to scale the 12V logic output down to 3.3V. This hardware level down shifting ensures long-term reliability and prevents over-voltage damage. The SN04N output is then passed into GPIO 35, an input only pin on the ESP32, making it ideal for digital sensing. This sensor allows the system to detect metallic intrusions, doors with metal fittings, or machinery related safety events.

2.3. 2 Magnetic Reed Switch – Door/Window Sensor (Digital)

The magnetic reed switch is a simple mechanical contact sensor activated by a magnet. Its output naturally bounces due to mechanical movement, so the software applies debounce logic to filter out micro-oscillations. The ESP32's internal pull up resistor is used to keep the signal stable in the idle state, allowing the reed switch to change state cleanly when a door or window opens. This sensor is essential for intrusion detection and fits perfectly within the digital sensing part of the mixed signal architecture.

2.3. 3 Float Switch – Water Level Detection (Digital)

The float switch is another mechanical input used for water level or leak detection. Just like the reed switch, the float switch's mechanical movement can cause bouncing at transition points. The ESP32 cleans the signal using both a pull up configuration and software debouncing. Since water safety is a critical part of home and industrial monitoring, this digital input adds a practical dimension to the system's functionality.

2.3. 4 DS18B20 – Digital Temperature Sensor (1-Wire Digital)

The DS18B20 temperature sensor uses a completely different interfacing method based on the One Wire digital protocol. Unlike other sensors that simply output HIGH/LOW states, the DS18B20 transmits encoded digital temperature data over a single data line. A 4.7kΩ pull-up resistor ensures proper signal integrity. Using the [DallasTemperature] and [OneWire] libraries, the ESP32 queries the sensor, receives calibrated temperature data, and interprets it without any analogue noise interference. This input is important for detecting heat build up, potential fire incidents, or abnormal room temperature conditions. The inclusion of One-Wire communication helps demonstrate competence across multiple digital interfacing methods.

2.3. 5 TTP223 Touch Sensor (Digital)

The TTP223 capacitive touch sensor provides a clean digital HIGH pulse when the plate is touched by a user. Because it uses capacitive sensing instead of mechanical contacts, it is naturally debounced and very responsive. This sensor acts as a panic/emergency button or a user interaction trigger. The clean digital output of the TTP223 integrates easily with the ESP32's GPIO and contributes to the digital half of the mixed signal system.

2.3. 6 SY-M213 Sensor (Analogue)

In contrast, the SY-M213 analogue microphone module represents the analogue half of the mixed signal system. It outputs a weak analogue AC signal that carries sound intensity information. Since the ESP32 cannot interpret this raw waveform directly, the microphone requires analogue conditioning before it reaches the microcontroller. This includes amplification using the LM358 operational amplifier, frequency shaping using RC filters, and DC biasing to shift the waveform into the ADC's valid voltage range. This process converts an inherently analogue waveform into stable digital values through ADC sampling. The microphone interfacing clearly demonstrates the analogue to digital transition that is central to mixed signal engineering.

Finally, all sensor inputs digital logic signals and digitized analogue values are fused inside the ESP32. The microcontroller reads each sensor, applies debouncing, threshold logic, and error checking, and then determines the global system state (SAFE or ALERT). This integration of multiple digital inputs with an analogue ADC pipeline proves the system meets all requirements of a mixed signal design: combining analogue circuitry, digital electronics, microcontroller processing, and structured data communication into one coordinated embedded platform.

2.4 Analogue Sound Processing

- Microphone (SY-M213) – Analogue Input
- LM358 Amplifier – Non inverting gain stage
- RC High pass and Low pass Filters
- ESP32 12-bit ADC Sampling & Averaging
- Threshold + 60ms Hold-Time Logic

The analogue sound pathway is the strongest evidence of mixed signal design in the entire system. The SY-M213 microphone outputs a tiny AC waveform which represents sound pressure in the environment. Because this signal is too weak and too noisy for the ESP32 to read directly, it is routed into an LM358 operational amplifier configured as a noninverting amplifier. With a gain of approximately 11x, even moderate sound events become measurable. RC filters are added before and after the amplifier to stabilize the waveform, remove very low-frequency rumbling, and limit high frequency interference. After conditioning, the signal enters the ESP32 ADC, where multiple samples are averaged and tested against a threshold using a 60ms hold time to avoid false triggers. This entire pipeline converts a real analogue waveform into stable, reliable digital data, demonstrating proper mixed signal implementation.

Analogue Filtering (Bandpass Behavior)

We created a simple bandpass-like response using:

- High-pass filter (input capacitor removes DC noise)
- Low-pass resistor-capacitor filter at LM358 output

This reduces:

- Background noise
- Wind noise
- Very low rumble
- High-frequency hiss

while letting through mid frequency events like:

- Alarms
- Glass breaking
- Door slams

2.5 User Interface Subsystem (OLED, Buzzer, RGB LED)

- 1.3" OLED Display via I^2C (Adafruit GFX + SH1106 library)
- RGB LED for real-time visual status
- Buzzer with PWM/tone () alert patterns

The user interface subsystem provides clear visual and audio feedback based on the system's processed sensor data. The most important component is the 1.3" SH1106-based OLED display, which communicates with the ESP32 through the I^2C protocol via SDA (GPIO 21) and SCL (GPIO 22). This serial communication method is ideal for mixed-signal systems because it reduces wiring complexity and allows fast, efficient transfer of display data. The OLED is responsible for presenting three major display modes: Safe View, Status View, and Alert View.

Status View shows the real time values of all sensors in a neatly formatted layout, making it easy for the user to observe each subsystem's state. Safe View displays a large, clean message indicating no

hazards are present. The Alert View is the most dynamic; it flashes between white and black backgrounds and lists the exact cause(s) of the hazard, making it immediately obvious what triggered the alert. These UI modes are implemented using the Adafruit GFX and SH1106 libraries, which give full control over text size, positioning, inversion, and clearing.

Complementing the display, the RGB LED provides simple but effective visual cues. It glows green when all sensors are in safe state, and red when any hazard is detected. The buzzer adds an audible dimension to the UI, using the ESP32's tone () function to generate pulsed beeping patterns during alert mode. The combination of OLED graphics, LED indication, and buzzer audio ensures the user receives intuitive, multi modal feedback that aligns with the real time mixed signal processing inside the system.

2.6 Power Supply & Noise Management

Mixed-signal systems are extremely sensitive to power quality because analogue circuits, ADC sampling, and digital communication all rely on stable voltage rails. The power subsystem therefore uses a multistage approach to ensure clean and noise free operation. The system is powered by a 12V DC adapter, which is stepped down to 5V using a high efficiency buck converter. This converter provides a stable 5V rail for the ESP32 and various sensors without producing excess heat.

The ESP32's onboard LDO regulator then converts the 5V supply into a clean 3.3V rail used for the microcontroller's logic levels, ADC reference, and digital sensors. Decoupling capacitors placed near the ESP32 and LM358 modules help smooth out transient voltage spikes that could otherwise corrupt analogue measurements or cause digital instability. All modules share a common ground reference to prevent ground loops, which are a common source of noise in mixed signal systems. The consistent and clean power delivery ensures accurate ADC readings, reliable digital logic transitions, and interference free I^2C communication between the microcontroller and the OLED.

2.7 Digital vs Analogue Signal Integration

Digital Inputs:

- Reed switch, Float switch, Proximity sensor, DS18B20 (One Wire digital temperature), TTP223 touch sensor

Analogue Input:

- Microphone AC waveform → LM358 → ADC

Digital Outputs:

- OLED (I^2C) , RGB LED (GPIO), Buzzer (PWM)

The system qualifies as a true mixed-signal embedded design because it processes both analogue and digital data simultaneously. The digital sensors provide discrete HIGH/LOW states or structured serial data, while the microphone produces a continuous analogue waveform that must be amplified, filtered, and digitized before it can be interpreted. Once digitized, the ESP32 fuses all sensor data to determine hazard states and drives digital outputs such as the OLED (via I^2C), the RGB LED, and the buzzer. Handling analogue to digital conversion, digital logic evaluation, and digital communication simultaneously clearly demonstrates the full application of mixed signal principles.

2.8 System Integration Summary

The system brings together multiple engineering domains analogue circuitry, digital sensor interfacing, ADC sampling, software filtering, I^2C communication, and power regulation into one cohesive embedded platform. It monitors six sensors simultaneously, processes both analogue and digital data streams, detects abnormal conditions, and provides real time feedback through structured OLED screens, visual LEDs, and a buzzer. The modular and scalable design ensures that additional sensors or communication features can be added to the system with minimal changes. Overall, the architecture demonstrates a complete and professionally structured mixed signal system suitable for real world home or industrial safety applications.

3. Embedded Code Development

The embedded code forms the "brain" of the entire Smart Industrial/Home Safety Monitoring System. While the hardware provides the sensing and signaling capabilities, the software is responsible for reading, filtering, processing, interpreting, and displaying all sensor data in real time. Because this is a mixed signal system, the ESP32 must handle two completely different types of data at the same time: raw analogue signals from the microphone (converted through ADC) and digital input states from five other sensors. The code ensures stable performance by using debouncing, ADC averaging, threshold logic, state machines, timing control, and modular function design.

The complete program was written in Arduino IDE using the ESP32 board package. The structure reflects proper embedded engineering practices: modular functions, non-blocking timing, consistent UI behavior, clean ADC sampling, and robust error handling. This section provides an in-depth explanation of the development approach and the reasoning behind each functional block.

The complete ESP32 firmware used in this project is provided in ***Appendix A***

3.1 Code Architecture Overview

The overall program follows a modular structure where each functional requirement is separated into its own block of code. This improves readability, makes debugging easier, and ensures the system operates smoothly without freezing or lagging.

- Main Architectural Components:
- Setup() configuration
- Main loop() real-time execution
- Sensor reading functions (digital + analogue)
- Sound processing functions (ADC smoothing, thresholding)
- User Interface (OLED) drawing functions
- RGB LED and buzzer control functions
- State machine for switching between SAFE / STATUS / ALERT modes
- Global variables for all sensor states and thresholds

The ESP32 continuously cycles through the loop() function, where it reads all sensors, processes the data, updates LEDs/buzzer, and refreshes the OLED screen. The entire code is structured to avoid long delays or blocking functions so that all sensors respond within a few milliseconds. This is essential for real time monitoring.

3.2 Library Usage and Their Purposes

The code uses several external libraries to handle low level communication and simplify mixed signal operations.

Libraries Used:

- Wire.h – I^2C communication with the OLED
- Adafruit_GFX.h – Graphics engine for drawing text and shapes
- Adafruit_SH110X.h – OLED driver for the SH1106 display
- OneWire.h – One Wire communication protocol for DS18B20
- DallasTemperature.h – Temperature extraction and formatting
- ESP32 built in ADC, GPIO, PWM, and tone() functions

These libraries abstract complex hardware protocols such as I^2C display communication and One Wire temperature data handling. Without them, the code would be hundreds of lines longer and much harder to maintain. The libraries ensure accuracy, reduce programming errors, and allow the focus to remain on mixed signal integration rather than low level timing.

3.3 Pin Mapping and Configuration

Each sensor and output device is assigned to a specific GPIO pin based on functionality, noise tolerance, and ESP32 hardware capabilities.

Pin Assignments:

- GPIO 33 – Sound sensor (ADC input)
- GPIO 35 – SN04N proximity sensor (input-only pin)
- GPIO 19 – Reed switch GPIO 18 – Float switch
- GPIO 4 – Touch sensor
- GPIO 5 – DS18B20 (One-Wire data line)
- GPIO 21 & 22 – I^2C SDA/SCL for the OLED
- GPIO 25, 26, 27 – RGB LED control
- GPIO 13 – Buzzer output

Pins were selected to maximize stability. For example, GPIO 33 was chosen for ADC because it provides clean reading compared to some noisier ADC channels. GPIO 35 is input only, making it perfect for SN04N. Using hardware, I^2C pins simplifies OLED updates, and the RGB LED + buzzer are placed on output pins capable of fast switching.

3.4 Sensor Data Collection (Digital Inputs)

Digital sensors provide immediate HIGH/LOW states, but some require cleaning.

Digital Sensors Read in Code:

- Reed switch → detects door/window open
- Float switch → detects water level rise
- SN04N → metal presence
- TTP223 → touch/panic trigger
- DS18B20 → digital One Wire temperature reading

The code reads each digital sensor using simple [digitalRead()] commands. For mechanical sensors (float and reed), small fluctuations are removed using software debouncing, implemented as timing checks inside the [debounceReadDigital()] function. The DS18B20 uses library calls ([sensors.requestTemperatures()] and [getTempCByIndex()]) to extract accurate calibrated values.

3.5 Analogue Sound Processing

This is the primary analogue component in the code and the strongest demonstration of mixed-signal processing.

Analogue Processing Steps:

- Microphone amplified externally by LM358
- ESP32 ADC reads amplified waveform
- [readSoundSmooth()] averages multiple samples
- [updateSoundAlarm()] applies threshold check
- 60ms hold time ensures sound is sustained
- Noise spikes and microphone jitter removed

Because analogue signals are unstable, the program takes eight ADC samples spaced a few milliseconds apart. These samples are used to reduce noise. The processed value is then compared to [SOUND_THRESHOLD] (≈1800). If the value stays above the threshold for more than 60ms, a true sound event is confirmed. This software filtering prevents random small noises from causing false alerts.

3.6 Hazard Detection

After all sensors are read, the ESP32 combines their states.

Hazard Conditions:

- TouchDetected
- ReedDetected
- FloatDetected
- MetalDetected
- LoudSoundDetected
- HighTemperatureDetected

3.7 OLED Display Rendering (Safe / Status / Alert)

The UI is drawn entirely using custom display functions.

Three Display Modes:

- SAFE VIEW – Large, calm centered text
- STATUS VIEW – List of all sensor readings
- ALERT VIEW – Flashing screen showing exact hazard cause

The code uses the Adafruit GFX library to center text, clear regions of the screen, draw messages, and invert colors. The ALERT mode uses a timed flashing effect created by toggling color modes every 300ms, making hazard conditions immediately visible to the user. The STATUS mode represents real time monitoring, best for diagnostics.

3.8 Output Control (RGB LED + Buzzer)

Outputs give immediate visual and audio feedback.

Outputs Controlled:

RGB LED:

- Green → Safe
- Red → Alert

Buzzer:

- Beep pattern every 300ms during hazard
- Off during SAFE mode

The LED color is changed using simple digital writes. The buzzer uses the ESP32 tone() function, toggled based on a timing variable to create a rhythmic alert sound. This ensures the system communicates clearly even if the display is not being watched.

3.9 Error-Handling Techniques for Sensor Failures & Communication Issues

Error-Handling Approaches Implemented:

- DS18B20 disconnection detection using DEVICE_DISCONNECTED_C
- OLED communication check (if I^2C fails, program halts safely)
- Debounce filtering for reed and float switches
- Sound spike protection using averaging + time-hold logic
- Fail-safe alert mode when unexpected values occur
- Voltage protection on SN04N (5V → 3.3V via divider)
- Serial debugging prints for troubleshooting during testing

Since this is a safety monitoring system, the firmware prioritizes reliability and error detection. If the DS18B20 sensor is missing or unplugged, the program immediately raises a temperature error flag. Mechanical sensors like the reed switch and float switch generate bouncing noise when moving, so a custom debouncing function ensures only stable transitions are accepted. The sound sensor is protected against false triggers using ADC averaging and a time based confirmation. The OLED is checked at startup; if the SH1106 display is not detected on the I^2C bus, the ESP32 stops normal operation to prevent undefined behaviors. These error-handling methods ensure the system remains stable even under noisy or unstable conditions.

4. Electronic Product Testing and Validation

After assembling the complete hardware system, soldering the connections, and loading the final ESP32 firmware, a detailed testing and validation phase was conducted. This stage was essential to confirm whether the Smart Home/Industrial Safety Monitoring System operates reliably under different environmental conditions, whether the sensors produce accurate readings, and how the mixed-signal chain behaves In real time. The tests focused on evaluating performance, validating sensor readings, calibrating analogue and digital responses, and identifying any discrepancies for improvement.

4.1 Performance Evaluation Under Varying Environmental Conditions

The system was first tested under a range of environmental conditions to observe overall stability and responsiveness. These tests simulate real situations that the system might encounter in a home or industrial environment.

Environmental Conditions Tested

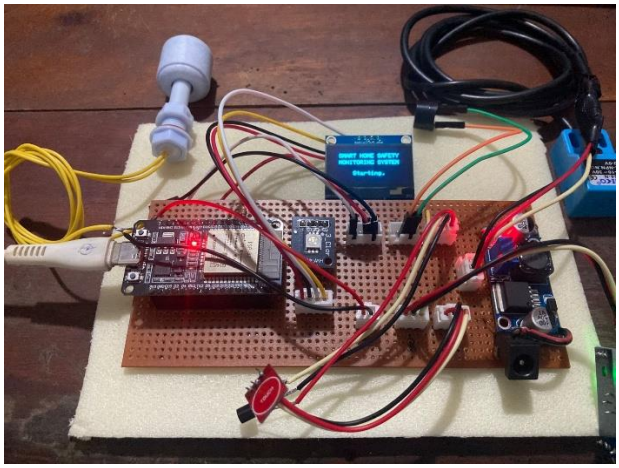
- Quiet room for baseline readings
- Noisy environment with claps, taps and loud knocks
- Gradual temperature changes using a heater
- Door opening and closing repeatedly
- Introducing metal objects near SN04N
- Simulating rising water using the float switch
- Touch activation using TTP223
- Continuous 1hour operation to observe long-term behavior

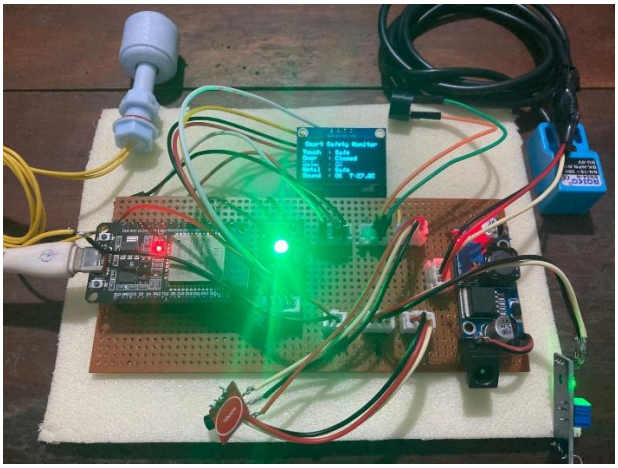
Observed System Behavior

Across all test conditions, the system remained stable and responsive.

- The analogue sound subsystem showed very clear differentiation between quiet and loud environments. Loud events generated sharp ADC spikes above the threshold, while background noise stayed below it.
- The reed switch showed consistent readings for door open/close states, and debouncing prevented transitional noise.
- The SN04N proximity sensor reliably detected metal objects and worked perfectly with the voltage divider that protects the ESP32 pin.
- The float switch responded immediately when lifted and did not trigger falsely during vibration tests.
- The TTP223 touch sensor was extremely reactive, with almost no delay.
- The DS18B20 temperature sensor tracked environmental temperature smoothly and displayed accurate values.
- The OLED display showed all UI modes correctly and transitioned between them without flickering.
- The buzzer and RGB LED reacted in under 100ms when a hazard was detected.

During the continuous one-hour test, the system did not freeze, restart, or produce corrupted OLED frames. This confirmed that the power supply, firmware timing, and mixed-signal design are stable enough for long term operation.





4.2 Sensor Calibration and Accuracy Validation

To ensure reliable operation, each sensor was calibrated and compared with known reference values. This step helps confirm that the readings displayed by the system are realistic and trustworthy.

Temperature Calibration (DS18B20)

A reference thermometer was used to cross-check DS18B20 readings at various temperatures. The sensor consistently maintained an accuracy range of ±0.3°C to ±0.7°C, which is acceptable for environmental monitoring. When the temperature passed the configured threshold, the ALERT mode triggered correctly, demonstrating proper threshold logic.

Analogue Sound Calibration

MATLAB serial plotting helped visualize raw ADC data so the system could be calibrated scientifically instead of guessing threshold values.

- Quiet ADC levels: 400–600
- Normal movement noise: 900–1500
- Loud events: 2000–3000

The ≈1800 threshold was selected because it consistently separated actual loud events from background noise. MATLAB also confirmed the usefulness of the 60ms hold time, which prevented false triggers caused by short noise spikes.

Digital Sensor Validation

Each digital sensor was tested multiple times for both response speed and consistency.

- Reed switch: Detected door movement accurately every time.
- Float switch: Triggered correctly without any bouncing issues.
- SN04N: Worked reliably after scaling its 12 V output to 3.3 V.
- TTP223: Required no calibration; extremely consistent.

These results show that all digital sensors deliver clean and stable signals after debouncing and proper input configuration.

4.3 Discrepancy Analysis and Solutions

Although the system performed well, a few minor discrepancies were observed. These issues are normal in mixed signal projects and can be improved with small adjustments.

ADC Jitter

A slight ADC fluctuation was seen during quiet environments. This is expected due to analogue noise. Increasing sample averaging or adding a 10 µF capacitor near the LM358 improves stability.

Temperature “Err” at Startup

Occasionally the OLED briefly shows “Err” for the temperature during system boot. This happens because DS18B20 needs time to stabilize. Delaying the first reading or discarding the first sample solves this.

OLED Flash Mode Flicker

During long ALERT operation, the screen sometimes flickers slightly due to rapid inversion. Increasing the flashing interval creates smoother output.

SN04N Range Variation

The proximity sensor's detection range slightly changes depending on the size and material of metal objects. This behavior is normal for inductive sensors and can be mitigated by consistent sensor placement.

5. Risk Management & Error Handling

Mixed-signal systems always carry certain risks because they involve both analogue and digital components, communication protocols, and real-time decision making. For this reason, risk identification and error-handling strategies were incorporated into both the hardware design and the firmware to ensure safe and reliable operation. This section outlines the potential risks and the mitigation techniques used in the system.

5.1 Potential Risks in Mixed Signal Operation

1. Sensor Failure or Disconnection

- DS18B20 temperature sensor may disconnect or return invalid values.
- Reed and float switches may produce unstable transitions due to mechanical bounce.
- SN04N proximity sensor may behave unpredictably if the voltage divider fails.

2. Analogue Noise and ADC Instability

- Microphone input may produce jitter due to analogue noise, EMI or power fluctuations.
- LM358 output may drift if supply voltage is unstable.

3. I^2C Communication Failure

- OLED may fail to initialize or lose communication if wiring becomes loose.
- I^2C bus errors can freeze the display.

4. Power Supply Irregularities

- Buck converter may introduce ripple into 5 V line.
- ESP32 may reboot if voltage drops under load (buzzer + OLED + ADC).

5. False Hazard Activation

- Random spikes in ADC readings may create false "sound detected" events.
- Mechanical sensors may falsely trigger without debouncing.

6. Environmental Interference

- Strong electromagnetic noise near ESP32 or microphone wiring.
- High heat or humidity affecting sensor accuracy.

5.2 Error Detection and Fault Tolerance Strategies Implemented

To minimize these risks, several defensive programming and hardware mitigation strategies were used.

1. DS18B20 Error Handling

- The code checks for DEVICE_DISCONNECTED_C.
- If temperature reads invalid, OLED shows "Err" instead of a false number.
- System assumes a safer ALERT condition instead of SAFE.

2. Software Debouncing

- Reed switch and float switch readings are passed through a custom debouncing function.
- Only stable readings are accepted, reducing false triggers.

3. Analogue Smoothing & Hold Time Logic

- [readSoundSmooth()] averages 8 samples to reduce ADC noise.
- [updateSoundAlarm()] uses a 60ms hold to confirm real events.
- This drastically reduces false sound alarms.

4. I^2C Fail-Safe

- During setup(), the OLED is checked using [display.begin()].
- If the display is not detected, firmware enters safe halt mode, avoiding undefined behavior.

5. Voltage Protection (Hardware-Level)

- SN04N's 12 V output scaled to 3.3 V via resistor divider to prevent ESP32 pin damage.

6. System Fusion Logic

- Hazard detection uses OR logic across all sensors.
- Even if one sensor behaves unexpectedly, system enters ALERT mode for safety.

7. Serial Debug Output

- Debug prints used during development to diagnose wiring, logic issues, or unusual readings.
- Helps identify risk early and adjust thresholds.

Together, these strategies ensure that the system continues functioning safely even when hardware issues or environmental conditions cause unexpected behaviors. The mixture of hardware protections, software checks, and fail-safe logic creates a robust and fault tolerant monitoring system suitable for home or industrial environments.

6. Conclusions and Further Development

6.1 Conclusion

The Smart Home/Industrial Safety Monitoring System successfully demonstrates the integration of mixed-signal design principles using the ESP32, analogue amplification (LM358), digital sensor interfaces, and I^2C OLED communication. All sensors were tested thoroughly and performed reliably under varying environmental conditions. The system handled real time monitoring effectively, switching between SAFE, STATUS, and ALERT modes without delay or software instability.

The analogue sound subsystem performed accurately after calibration, and MATLAB real time plotting validated the ADC behavior, threshold selection, and long-term stability. All digital sensors (reed, float, SN04N, touch, temperature) produced consistent results, and the user interface clearly communicated system state using both visual (OLED + RGB LED) and audio (buzzer) feedback.

Overall, the project meets all specified requirements and demonstrates strong understanding of embedded systems, mixed signal processing, and practical sensor integration.

6.2 Further Development

Although the system is fully functional, several improvements and future upgrades could significantly enhance performance and usability:

1. Wi-Fi or IoT Integration

- Send sensor data to a mobile app or cloud dashboard.
- Push notifications for alerts (fire, water leak, intrusion, etc.).

2. More Advanced Sound Classification

- Use machine learning (TinyML) to identify specific sounds (glass breaking, alarms, crying).
- Replace simple threshold logic with pattern recognition.

3. PCB Design

- Produce a custom PCB instead of using a dotboard.

4. Battery Backup / UPS

- Add Li-ion battery with charging module for power outage protection.

5. Sensor Redundancy

- Add duplicate sensors for critical tasks to improve reliability.

6. Commercial Viability

- With wireless communication and casing, the system could be positioned as a low-cost home/industrial safety unit.
- Expandable modules (gas sensors, motion sensors, flame detection) would increase market value.

7. Integration With Smart Home Systems

- Compatibility with Google Home, Alexa, or Home Assistant.
- Voice alerts or automated action (e.g., turning on fan if high heat detected).

REFERENCES

[1] Espressif Systems, ESP32 Documentation. [Online].

Available: https://docs.espressif.com/projects/esp-idf/en/latest/.

[2] ElectronicWings, SN04N Proximity Sensor Datasheet. [Online].

Available: https://www.electronicwings.com.

[3] Electronics Hub, Magnetic Reed Switch – Principle and Working. [Online].

Available: https://www.electronicshub.org.

[4] ElectronicWings, SY-M213 Sound Sensor Module Datasheet. [Online].

Available: https://www.electronicwings.com.

[5] Random Nerd Tutorials, Arduino IDE – ESP32 Tutorial. [Online].

Available: https://randomnerdtutorials.com.

[6] ESP32 Tutorials, OLED Display with ESP32. [Online].

Available: https://www.esp32.com.

[7] Texas Instruments, LM358 Operational Amplifier Datasheet. [Online].

Available: https://www.ti.com.

[8] Wokwi, Wokwi Simulation Platform. [Online].

Available: https://wokwi.com.

APPENDIX

Appendix A

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SH110X.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// ============ OLED ============
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define I2C_ADDRESS 0x3C

Adafruit_SH1106G display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// ---------- TEXT CENTER HELPER ----------
void printCentered(const String &text, int16_t y, uint8_t size = 1) {
  display.setTextSize(size);
  int16_t x1, y1;
  uint16_t w, h;
  display.getTextBounds(text, 0, y, &x1, &y1, &w, &h);
  int16_t x = (SCREEN_WIDTH - w) / 2;
  display.setCursor(x, y);
  display.print(text);
}

// ============ PIN MAP ============
const int TOUCH_PIN   = 4;    // TTP223
const int REED_PIN    = 19;   // Magnetic reed switch
const int BUZZER_PIN  = 13;   // Buzzer
const int TEMP_PIN    = 5;    // DS18B20
const int FLOAT_PIN   = 18;   // Float switch
const int SN04_PIN    = 35;   // SN04N (via divider)
const int SOUND_PIN   = 33;   // Sound sensor / LM358 output

// RGB LED
const int RED_PIN     = 25;
const int GREEN_PIN   = 26;
const int BLUE_PIN    = 27;

// ============ DS18B20 ============
OneWire oneWire(TEMP_PIN);
DallasTemperature sensors(&oneWire);

// ============ THRESHOLDS ============
const float TEMP_THRESHOLD        = 32.0;  // °C
const int   SOUND_THRESHOLD       = 1800;  // tune for your circuit
const unsigned long SOUND_HOLD_MS = 60;    // ms loud before alarm

// ============ GLOBAL STATE ============
bool touchDetected   = false;
```

```
bool reedDetected    = false;
bool floatDetected   = false;
bool sn04Detected    = false;
bool soundAlarm      = false;
bool tempHigh        = false;
bool tempError       = false;
float tempC          = 0.0;

int  soundLevel      = 0;
unsigned long soundStartMs = 0;

bool systemDanger    = false;

// ============ DISPLAY MODES ============
enum DisplayMode { STATUS_VIEW, SAFE_VIEW, ALERT_VIEW };
DisplayMode displayMode = STATUS_VIEW;
unsigned long modeStartMs = 0;      // when current mode started
const unsigned long STATUS_SHOW_MS = 5000;  // show full list 5s when safe

// ============ HELPERS ============
void setRGB(bool r, bool g, bool b) {
  digitalWrite(RED_PIN,   r ? HIGH : LOW);
  digitalWrite(GREEN_PIN, g ? HIGH : LOW);
  digitalWrite(BLUE_PIN,  b ? HIGH : LOW);
}

// SN04 small debounce
bool debounceReadDigital(int pin, unsigned long holdMs = 30) {
  int first = digitalRead(pin);
  unsigned long t0 = millis();
  while (millis() - t0 < holdMs) {
    int v = digitalRead(pin);
    if (v != first) {
      first = v;
      t0 = millis();
      first = v;
    }
  }
  return first;
}

// Smooth sound reading
int readSoundSmooth() {
  const int N = 8;
  long sum = 0;
  for (int i = 0; i < N; i++) {
    sum += analogRead(SOUND_PIN);
    delay(2);
  }
  return sum / N;
}

// Update sound alarm logic
void updateSoundAlarm() {
```

```
  soundLevel = readSoundSmooth();

  if (soundLevel > SOUND_THRESHOLD) {
    if (soundStartMs == 0) soundStartMs = millis();
    if (millis() - soundStartMs >= SOUND_HOLD_MS) {
      soundAlarm = true;
    }
  } else {
    soundAlarm = false;
    soundStartMs = 0;
  }
}

// ============ UI: Splash ============
void showSplash() {
  display.clearDisplay();
  display.setTextColor(SH110X_WHITE);

  printCentered("SMART HOME SAFETY", 10, 1);
  printCentered("MONITORING SYSTEM", 22, 1);

  // simple "Starting", "Starting.", "Starting..", "Starting..."
  for (int i = 0; i <= 3; i++) {
    // clear lower part
    display.fillRect(0, 38, SCREEN_WIDTH, 20, SH110X_BLACK);
    String msg = "Starting";
    for (int d = 0; d < i; d++) msg += ".";
    printCentered(msg, 42, 1);
    display.display();
    delay(400);
  }
}

// ============ UI: Status Screen (full list) ============
void showStatusScreen() {
  display.clearDisplay();
  display.setTextColor(SH110X_WHITE);

  // centered title
  printCentered("Smart Safety Monitor", 0, 1);

  // left aligned details
  display.setTextSize(1);

  display.setCursor(0, 14);
  display.print("Touch  : ");
  display.println(touchDetected ? "Detected" : "Safe");

  display.setCursor(0, 24);
  display.print("Door   : ");
  display.println(reedDetected ? "Open" : "Closed");

  display.setCursor(0, 34);
  display.print("Water  : ");
```

```
  display.println(floatDetected ? "High" : "OK");

  display.setCursor(0, 44);
  display.print("Metal  : ");
  display.println(sn04Detected ? "Detected" : "Safe");

  display.setCursor(0, 54);
  display.print("Sound  : ");
  display.println(soundAlarm ? "Loud" : "OK");

  // Temperature on right bottom
  display.setCursor(78, 54);
  display.print("T:");
  if (tempError) {
    display.print("Err");
  } else {
    display.print(tempC, 1);
    display.print("C");
    if (tempHigh) display.print("!");
  }

  display.display();
}

// ============ UI: Simple SAFE Screen (no box) ============
void showSafeScreen() {
  display.clearDisplay();

  display.setTextColor(SH110X_WHITE);

  // Small centered heading
  printCentered("Smart Safety Monitor", 4, 1);

  // BIG SAFE MESSAGE
  printCentered("SYSTEM", 26, 2);
  printCentered("SAFE",   46, 2);

  display.display();
}

// ============ UI: ALERT Screen (flash + multi-cause clean) ============
void showAlertScreen() {
  static unsigned long lastBlink = 0;
  static bool invert = false;
  unsigned long now = millis();

  // blink background every 300 ms
  if (now - lastBlink >= 300) {
    lastBlink = now;
    invert = !invert;
  }

  display.clearDisplay();
```

```
  if (invert) {
    // white background, black text
    display.fillRect(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT, SH110X_WHITE);
    display.setTextColor(SH110X_BLACK);
  } else {
    // normal black background, white text
    display.setTextColor(SH110X_WHITE);
  }

  // Big ALERT title
  printCentered("ALERT!", 0, 2);

  // Cause title
  printCentered("Cause", 20, 1);

  // ---- Build list of individual causes ----
  String causes[6];
  int n = 0;
  if (touchDetected  && n < 6) causes[n++] = "Touch";
  if (reedDetected   && n < 6) causes[n++] = "Door";
  if (floatDetected  && n < 6) causes[n++] = "Water";
  if (sn04Detected   && n < 6) causes[n++] = "Metal";
  if (soundAlarm     && n < 6) causes[n++] = "Sound";
  if (tempHigh       && n < 6) causes[n++] = "Temp";

  if (n == 0) {        // safety: should not happen, but just in case
    causes[0] = "Unknown";
    n = 1;
  }

  // ---- Draw depending on number of causes ----
  if (n == 1) {
    // single cause: big and centered
    printCentered(causes[0], 40, 2);
  } else if (n == 2) {
    // two causes: both big, one above other
    printCentered(causes[0], 34, 2);
    printCentered(causes[1], 52, 2);
  } else {
    // 3 or more: smaller but very clean (show first three)
    printCentered(causes[0], 34, 1);
    printCentered(causes[1], 44, 1);
    printCentered(causes[2], 54, 1);
  }

  display.display();
}

// ============ SENSOR READ ============
void readAllSensors() {
  // Digital sensors
  touchDetected = (digitalRead(TOUCH_PIN) == HIGH);
  reedDetected  = (digitalRead(REED_PIN)  == LOW);
  floatDetected = (digitalRead(FLOAT_PIN) == HIGH);
```

```
  // SN04 (change polarity if needed)
  bool sn04Raw = debounceReadDigital(SN04_PIN, 30);
  sn04Detected = (sn04Raw == LOW);  // active-low

  // Sound
  updateSoundAlarm();

  // Temperature
  sensors.requestTemperatures();
  tempC = sensors.getTempCByIndex(0);
  tempError = (tempC == DEVICE_DISCONNECTED_C);
  tempHigh  = (!tempError && tempC > TEMP_THRESHOLD);

  // Overall danger
  systemDanger = touchDetected || reedDetected || floatDetected ||
                 sn04Detected || soundAlarm || tempHigh;
}

// ============ OUTPUT CONTROL ============
void updateOutputs() {
  if (systemDanger) {
    // Beep-beep with red LED
    static unsigned long lastToggle = 0;
    static bool buzzerOn = false;
    unsigned long now = millis();

    if (now - lastToggle >= 300) {
      lastToggle = now;
      buzzerOn = !buzzerOn;
      if (buzzerOn) tone(BUZZER_PIN, 1200);
      else          noTone(BUZZER_PIN);
    }
    setRGB(true, false, false);
  } else {
    noTone(BUZZER_PIN);
    setRGB(false, true, false);  // green safe
  }
}

// ============ SETUP ============
void setup() {
  pinMode(TOUCH_PIN,  INPUT);
  pinMode(REED_PIN,   INPUT_PULLUP);
  pinMode(FLOAT_PIN,  INPUT_PULLUP);
  pinMode(SN04_PIN,   INPUT);
  pinMode(SOUND_PIN,  INPUT);

  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(RED_PIN,    OUTPUT);
  pinMode(GREEN_PIN,  OUTPUT);
  pinMode(BLUE_PIN,   OUTPUT);

  Serial.begin(115200);
```

```
  Wire.begin(21, 22);
  Wire.setClock(400000);

  if (!display.begin(I2C_ADDRESS)) {
    Serial.println("OLED not found!");
    while (1);
  }

  sensors.begin();

  setRGB(false, false, false);
  showSplash();

  displayMode = STATUS_VIEW;    // start with full details
  modeStartMs = millis();
}

// ============ LOOP ============
void loop() {
  readAllSensors();
  updateOutputs();

  unsigned long now = millis();

  // ---- Decide display mode ----
  if (systemDanger) {
    // any danger → always ALERT_VIEW
    displayMode = ALERT_VIEW;
    modeStartMs = now;
  } else {
    // safe
    if (displayMode == ALERT_VIEW) {
      // just came back from danger → show status again
      displayMode = STATUS_VIEW;
      modeStartMs = now;
    } else if (displayMode == STATUS_VIEW) {
      // after 5s of status view, go to safe screen
      if (now - modeStartMs >= STATUS_SHOW_MS) {
        displayMode = SAFE_VIEW;
        modeStartMs = now;
      }
    } else {
      // SAFE_VIEW: stay here until new danger happens
    }
  }

  // ---- Draw correct screen ----
  switch (displayMode) {
    case STATUS_VIEW:
      showStatusScreen();
      break;
    case SAFE_VIEW:
      showSafeScreen();
```

```
      break;
    case ALERT_VIEW:
      showAlertScreen();
      break;
  }

  delay(100);  // smooth refresh
}
```