# PROJECT REPORT

# ON

# Safety Alert System

**Carried Out at**

**CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING**

**Electronic city Phase-1, Bengaluru.**

**UNDER THE SUPERVISION OF**

Mr. Sheikh Asif Ali

**Submitted By:**

Drashti Babariya [230950130009]
Gaganvishal Ahire [230950130004]
Pooja Singh [230950130022]
Sairaj More [230950130027]
Sneha Salunkhe [230950130029]

# Candidate's Declaration

We hereby certify that the work being presented in the report entitled Safety Alert System, in partial fulfillment of the requirements for the award of PG Diploma Certificateand submitted in the department of PG-DESD of the C-DAC Bangalore, is an authentic record of our work carried out during the period, 19th January to 29th February 2024 under the supervision of Mr. Sheikh Asif Ali Project Engineer at C-DAC Bangalore.

**(Name of Candidate's)**

| **Student Name** | **PRN No.** |
|---|---|
| Drashti Babariya | 230950130009 |
| Gaganvishal Ahire | 230950130004 |
| Pooja Singh | 230950130022 |
| Sairaj More | 230950130027 |
| Sneha Salunkhe | 230950130029 |

Signed By:

[illegible]

--------------

Sheikh Asif Ali

# ACKNOWLEDGEMENT

We would like to express our appreciation to each member of our project group for their dedication, collaboration, and contribution to this endeavour. The combined effort and diverse perspectives of played a crucial role in the success of this group project.

We are grateful to Mr. Sheikh Asif Ali for their guidance, feedback, and support throughout the duration of this project. Their expertise and encouragement were invaluable in helping us navigate challenges and achieve our goals.Special.

Lastly, we extend our heartfelt gratitude to our families and friends for their understanding, encouragement, and unwavering support throughout this collaborative effort.

## ABSTRACT

The project involves developing a safety-based data acquisition system using the STM32 Blue Pill board. This system integrates multiple sensors to monitor environmental conditions and detect potential safety hazards in real-time. The sensors include a DHT11 sensor for temperature and humidity, a flame sensor to detect open flames, an MQ2 sensor for harmful gases and smoke, and an LDR sensor to prevent electrical fires. Alerts are triggered using LED indicators, a buzzer, and an LCD display. The system aims to enhance safety measures by providing proactive alerts for potential hazards, thereby reducing the risk of accidents or damage.

# Table of Contents

## List of Figures

## List of Tables

# INTRODUCTION

## 1.1 Purpose

The purpose of this document is to specify the requirements for a safety-based data acquisition system that uses the STM32F103C8T6. This system will integrate multiple sensors to monitor environmental conditions and detect potential safety hazards in real-time. The sensors will include a DHT11 sensor for temperature and humidity, a flame sensor to detect open flames, an MQ2 sensor for harmful gases and smoke, and an LDR sensor to prevent electrical fires. Alerts will be triggered using LED indicators, a buzzer, and an LCD display. The system aims to enhance safety measures by providing proactive alerts for potential hazards, thereby reducing the risk of accidents or damage.

## 1.2 Objectives

- Enhance safety measures by providing proactive alerts for potential hazards.
- Utilize sensors to monitor temperature, humidity, flame, harmful gases, smoke, and prevent electrical fires.
- Reduce the risk of accidents or damage through timely notifications and alerts.

## 1.3 Intended Audience and Reading Suggestion

This document is intended for the following audience:

- System designer
- Software developers
- Hardware developers
- Testers
- Project managers

It is recommended that the reader read this document in its entirety to understand the complete requirements for the safety-based data acquisition system.

## 1.4 Product Scope

The Safety Alert System is designed to monitor environmental conditions in real-time using a variety of sensors, including temperature, humidity, flame, gas, smoke, and light sensors. By continuously analyzing sensor data, the system can detect potential safety hazards such as temperature fluctuations, open flames, harmful gases, smoke, and electrical fires. When a hazard is detected, the system triggers alerts through LED indicators, a buzzer, and an LCD display, providing timely notifications to users.

## LITERATURE SURVEY

The literature survey for the Safety Alert System project involved an in-depth review of existing research, papers, articles, and publications related to safety monitoring systems, sensor integration, hazard detection techniques, and alert mechanisms. Several key findings and insights were identified during this survey:

- **Sensor Integration:** Explored various types of sensors commonly used in safety monitoring systems, including temperature, humidity, flame, gas, smoke, and light sensors. Reviewed literature on the integration of sensors with microcontroller platforms for real-time data acquisition and analysis.
- **Hazard Detection Techniques:** Investigated different methodologies and algorithms for hazard detection based on sensor data analysis. Explored approaches for detecting temperature fluctuations, open flames, harmful gases, smoke, and electrical fires.

- **Alert Mechanisms:** Examined existing alert mechanisms used in safety monitoring systems, such as LED indicators, buzzers, LCD displays, and wireless communication modules. Reviewed literature on user interface design considerations for effective alert presentation and user interaction.

- **Real-time Monitoring and Response:** Studied strategies for real-time monitoring of environmental conditions and rapid response to potential safety hazards.

- **Power Management and Efficiency:** Investigated methods for optimizing power consumption in safety monitoring systems, especially in battery-operated or energy-constrained environments. Explored techniques for efficient sensor data transmission and processing to prolong system operation.

- **Case Studies and Applications:** Reviewed case studies and practical implementations of safety monitoring systems in various domains, including industrial, residential, and commercial settings. Analysed the effectiveness and performance of existing systems in mitigating safety risks and preventing accidents or damage.

Overall, the literature survey provided valuable insights into the state-of-the-art technologies, methodologies, and challenges in safety monitoring systems, guiding the design and development of the Safety Alert System.

# SOFTWARE REQUIREMENTS SPECIFICATION

The Software Requirements Specification (SRS) for the Safety Alert System outlines the functional and non-functional requirements of the software component. Below is a detailed breakdown of the SRS:

## 3.1 Functional Requirements

### 3.1.1 Sensor Integration

The system shall integrate the following sensors:

- DHT11 sensor for temperature and humidity monitoring.
- Flame sensor for detecting open flames.
- MQ2 sensor for identifying harmful gases and smoke.
- LDR sensor for preventing electrical fires.

The system shall read data from sensors at regular intervals.

### 3.1.2 Data Processing

The system shall analyse sensor data in real-time to detect potential safety hazards. It shall implement algorithms for hazard detection based on sensor readings. Hazard detection algorithms shall include thresholds for triggering alerts.

### 3.1.3 Alert Mechanisms

The system shall trigger alerts when potential hazards are detected. Alert mechanisms shall include:

- LED indicators for visual alerts.
- Buzzer for audible alerts.
- LCD display for displaying hazard information and instructions.

Alerts shall be configurable based on severity and user preferences.

## 3.2 Non-functional Requirements

In addition to the functional features and external interfaces, the Safety-Based Data Acquisition System must adhere to various non-functional requirements to ensure its performance, safety, security, and overall quality. These requirements encompass aspects such as performance, safety, security, and software quality attributes. Below are the detailed specifications for each category:

### 3.2.1 Performance Requirements

- **Response Time:** Alerts must be triggered within seconds of detecting a potential safety hazard to facilitate rapid mitigation actions.
- **System Resource Utilization:** The system should efficiently utilize CPU, memory, and other resources to ensure optimal performance and responsiveness.
- **Scalability:** The system should be designed to accommodate future enhancements or expansions without significant degradation in performance.

### 3.2.2 Safety Requirements

- **Reliability:** The system must accurately detect and alert for all predefined safety hazards to minimize the risk of missed alerts or false alarms.
- **Fail-Safe Operation:** In the event of a system failure or malfunction, the system should default to a safe state or provide alternative means of alerting users to ensure continued safety monitoring.
- **Compliance:** The system must comply with relevant safety standards and regulations applicable to the intended deployment environment.

### 3.2.3 Security Requirements

- **Data Privacy:** Any personally identifiable information or sensitive data collected by the system must be securely stored and transmitted to prevent unauthorized access or disclosure.
- **Access Control:** Access to system configuration settings and administrative functions should be restricted to authorized users only, with appropriate authentication mechanisms in place.
- **Firmware Integrity:** Measures should be implemented to ensure the integrity of firmware updates and prevent unauthorized modifications or tampering with the system software.

By addressing these non-functional requirements, the Safety-Based Data Acquisition System will not only meet the functional objectives but also deliver a reliable, secure, and high-quality solution for enhancing safety measures and mitigating potential hazards effectively.

## ARCHITECTURE

The system Architecture consists of sensor modules of sensor modules interfaced with the STM32F103C8T6 (Blue Pill Board) and alert mechanisms.

### 4.1 Block Diagram

The below diagram demonstrates simple Architecture/Block Diagram of project:

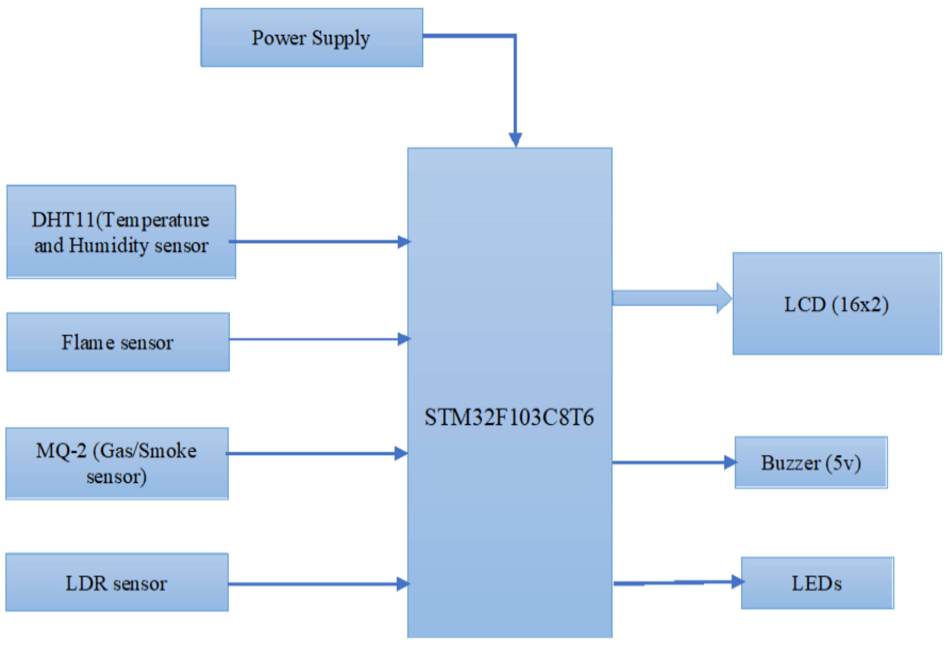


*Figure 4.1 Architecture of Safety Alert System*

This Block Diagram involves interfacing STM32f103c8t6 with different sensors such as DH11, Flame Sensor, LDR sensor, and MQ2 and alert mechanisms LCD (16x2), LEDs and Buzzer to make a successful safety alert system.

### 4.2 Hardware Requirements

The hardware components are integral to the system's capability to monitor environmental conditions and detect hazards. These include:

### 4.2.1 STM32F103C8T6 (Blue Pill Board)

- The STM32F103C8T6 (also known as 'STM32' or 'Blue Pill") is a cheap development board based on the ARM Cortex M3 microprocessor.
- It Serves as the central processing unit of the system. It is responsible for coordinating the data collection from sensors, processing this data, and controlling the output to the user interface components.

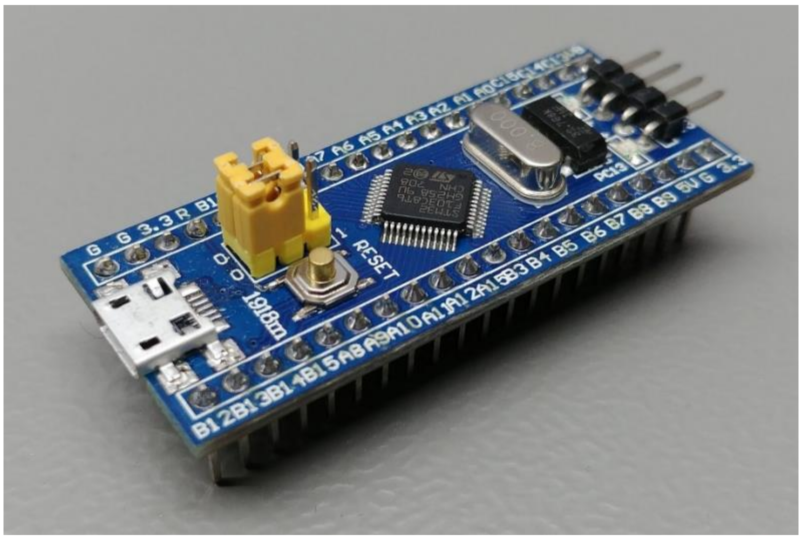


*Figure 4.2 STM32F103C8T6 (Blue Pill Board)*

**Table 4.1 Naming convention of STM32F103C8T6 (Blue Pill Board)**

| Parameter | Meaning |
|---|---|
| STM | name of the manufacturer (STMicroelectronics) |
| 32 | 32-bit ARM architecture |
| F | Foundation |
| 1 | Core (ARM Cortex M3) |
| 03 | Line (describes peripherals and speed) |
| C | 48 pins |
| 8 | 64 KB flash memory |
| T | LQFP package (Low Profile Quad Flat Pack) |
| 6 | Operating Temperature Range (-40 °C to 85 °C) |

**Table 4.2 Technical Specification of STM32F103C8T6 (Blue Pill Board)**

| **Parameter** | **Meaning** |
|---|---|
| Architecture | 32-bit ARM Cortex M3 |
| Operating Voltage | 2.7V to 3.6V |
| CPU Frequency | 72 MHz |
| Number of GPIO pins | 37 |
| Number of PWM pins | 12 |
| Analog Input Pins | 10 (12-bit resolution) |
| I2C Peripherals | 2 |
| SPI Peripherals | 2 |
| CAN 2.0 Peripheral | 1 |
| Timers | 3(16-bit), 1 |
| Flash Memory | 64KB |
| RAM | 20kB |

### 4.2.2 DHT11(Temperature and Humidity Sensor)

- The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). Its fairly simple to use, but requires careful timing to grab data.

- Measures ambient temperature and humidity. It provides data necessary for assessing environmental conditions that might affect safety.

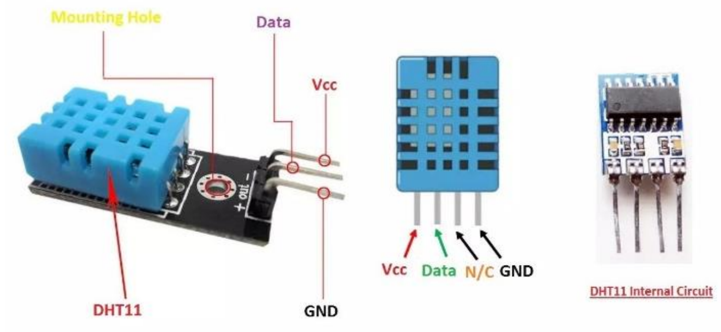


*Figure 4.2 DHT11 Sensor*

- **Technical Specifications:**

  Operating Voltage: 3.5V to 5.5V
  Operating current: 0.3mA (measuring) 60uA (standby)
  Output: Serial data
  Temperature Range: 0°C to 50°C
  Humidity Range: 20% to 90%
  Resolution: Temperature and Humidity both are 16-bit
  Accuracy: ±1°C and ±1%

- **Working Principle:**

  Temperature Measurement: The DHT11 sensor contains a thermistor that changes resistance with temperature. A microcontroller reads this resistance and converts it into a temperature value.
  Humidity Measurement: The sensor has a humidity sensing component that changes resistance with humidity. The microcontroller measures this resistance to determine the humidity level.

- **Functionality:**

  The DHT11 sensor provides real-time temperature and humidity readings. It communicates with the microcontroller using a single-wire digital protocol. The sensor is calibrated and provides accurate measurements within specified ranges.

### 4.2.3 Flame Sensor

- The flame sensor is an optical sensor used to detect the presence of open flames.

- The Fire or Flame Sensor Module can detect flames in the 760 – 1100 nanometre wavelength range. Small flames like a lighter flame can be detected at roughly 0.8m. The detection angle is roughly 60 degrees and the sensor is particularly sensitive to the flame spectrum.

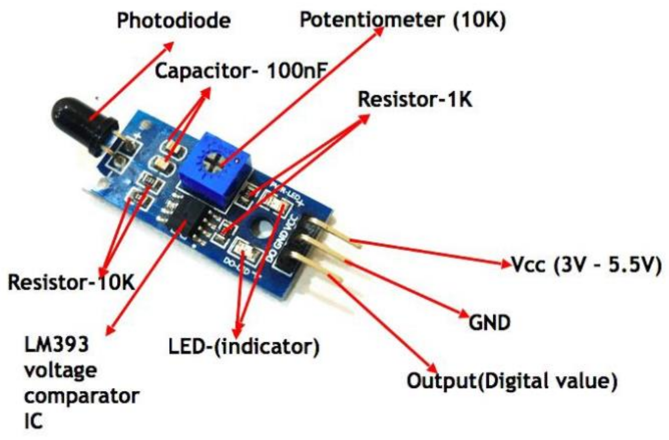


*Figure 4.3 Flame Sensor*

- **Technical Specifications**

  Operating Voltage: 3.3V to 5V DC
  Operating Current: 15ma
  Output Digital – 0V to 5V, Adjustable trigger level from pre-set
  Output Analog – 0V to 5V based on infrared radiation from fire flame falling on the sensor
  LEDs indicating output and power
  PCB Size: 3.2cm x 1.4cm
  LM393 based design

- **Working Principle**

  Detection Method: The flame sensor detects infrared radiation emitted by flames.
  Photodiode: When exposed to infrared radiation, the photodiode within the sensor generates a current proportional to the intensity of the radiation.
  Threshold Detection: The microcontroller monitors the current generated by the photodiode and triggers an alert if it exceeds a predefined threshold, indicating the presence of a flame.

- **Functionality**

  The flame sensor is capable of detecting various types of flames, including those produced by gas stoves, candles, and lighters. It provides a rapid response to flame detection, enabling timely alerts to prevent fire hazards.

### 4.2.4 MQ2 Sensor

- MQ2 sensor is a popularly used gas detecting or smoke detecting sensor. This sensor is capable of sensing LPG, Alcohol, Smoke, Hydrogen, Methane, Propane and Carbon Monoxide concentrations in the Air. Hence, we can use this sensor to detect the LPG gas leakage and program Arduino to buzz an alarm if it crosses some threshold.

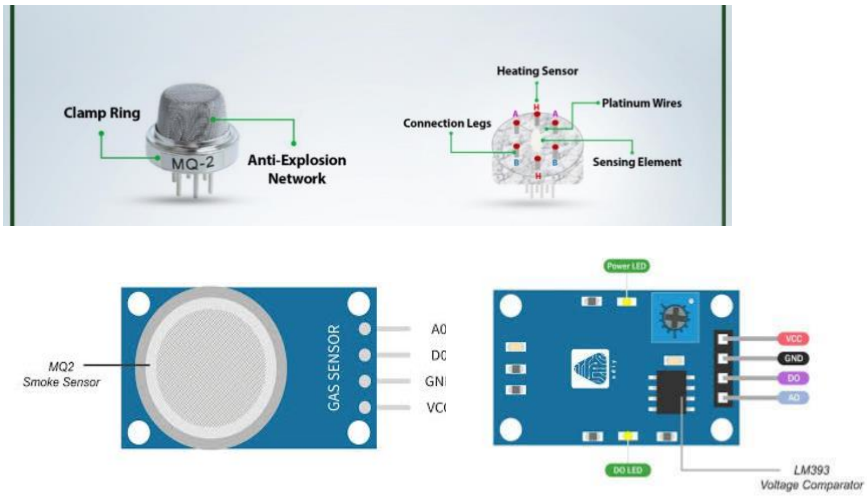


*Figure 4.4 MQ2 Gas Sensor*

- **Technical Specifications**

  Operating voltage: 5V
  Load resistance: 20 KΩ
  Heater resistance: 33Ω ± 5%
  Heating consumption: <800mw
  Sensing Resistance: 10 KΩ – 60 KΩ
  Concentration Range: 200 – 10000ppm
  Preheat Time: Over 24 hours

- **Working Principle**

  Gas Sensing Element: The MQ2 sensor contains a tin dioxide ($SnO_2$) semiconductor gas sensing element.
  Resistance Change: When exposed to target gases, the resistance of the sensing element changes.
  Voltage Output: The microcontroller measures the voltage across the sensing element and converts it into a gas concentration value using calibration curves.

- **Functionality**

  The MQ2 sensor provides real-time monitoring of gas concentrations in the environment. It is sensitive to a wide range of gases, making it suitable for detecting multiple types of hazards, including smoke and combustible gases.

### 4.2.5 LDR Sensor

- The LDR (Light Dependent Resistor) sensor is a light sensor used to detect changes in ambient light intensity.

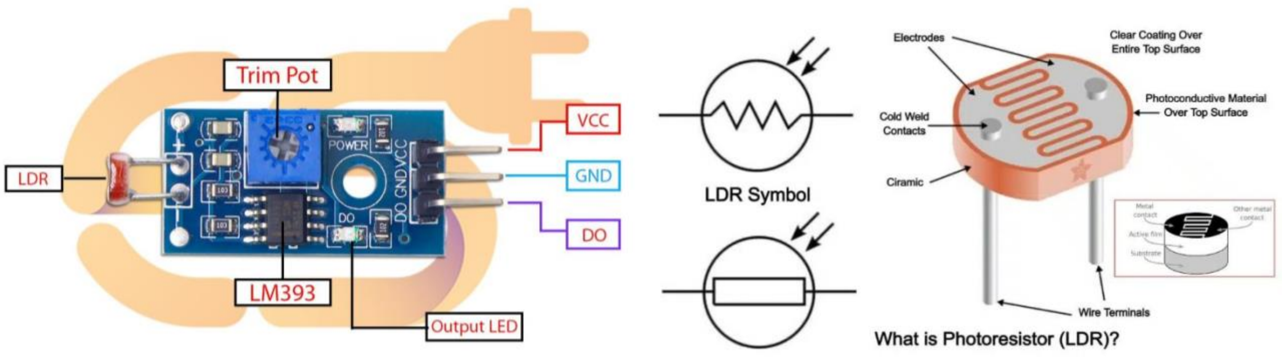

Trim Pot
LDR
POWER
VCC
GND
DO
LM393
Output LED
LDR Symbol
Electrodes
Clear Coating Over Entire Top Surface
Photoconductive Material Over Top Surface
Cold Weld Contacts
Ciramic
Wire Terminals
What is Photoresistor (LDR)?

*Figure 4.4 LDR sensor*

- **Technical Specifications**

  Operating Voltage: 3.3V to 5V DC
  Operating Current: 15ma
  Output Digital – 0V to 5V, Adjustable trigger level from preset
  Output Analog – 0V to 5V based on light falling on the LDR
  LEDs indicating output and power
  PCB Size: 3.2cm x 1.4cm
  LM393 based design

- **Working Principle**

  Photoconductivity: The resistance of an LDR varies inversely with the intensity of light incident upon it.
  - Voltage Output: The microcontroller measures the voltage across the LDR and converts it into a light intensity values.

  Threshold Detection: Based on predefined thresholds, the microcontroller triggers alerts when significant changes in light intensity occur, indicating potential electrical fires or other hazards.

- **Functionality**

  The LDR sensor continuously monitors ambient light levels, providing changes in lighting conditions. It is used in the Safety Alert System to detect anomalies in light intensity that may indicate electrical fires or other safety hazards.

### 4.2.6 LCD (16x2) Display

- The LCD (Liquid Crystal Display) display is an essential component of the Safety Alert System, providing a visual interface for users to view sensor readings, alert notifications, and system status.

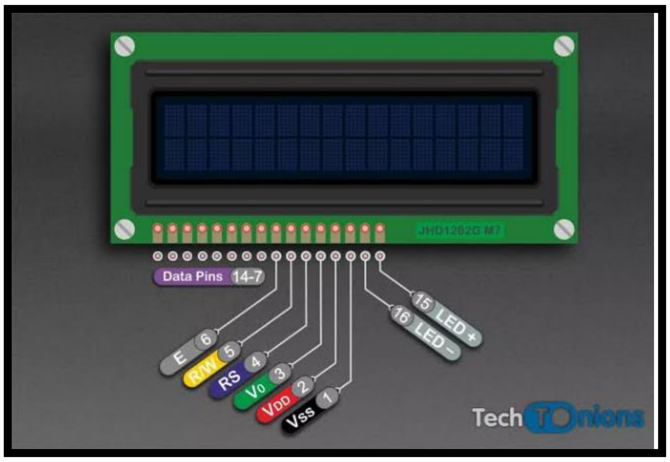


*Figure 4.5 LCD 16x2 Display*

- **Technical Specifications**

  Operating Voltage: 4.7V to 5.3V
  Operating Current 1mA (without backlight)
  Can display (16x2) 32 Alphanumeric Characters
  Custom Characters Support
  Works in both 8-bit and 4-bit Mode

- **Working Principle**

  Liquid Crystal Technology: LCD displays utilize liquid crystal molecules that can change orientation in response to an electric field.
  Segmented Display: Most LCD displays consist of a grid of pixels or segments that can be individually controlled to display alphanumeric characters, symbols, or custom graphics.
  Backlighting: Many LCD displays incorporate a backlight to improve visibility in low-light conditions. The backlight is typically controlled separately from the display segments.

- **Functionality**

  Display Output: The LCD display presents real-time sensor readings, alert notifications, and system status information in a user-friendly format.
  Customization: The display can be customized to show specific information relevant to the user's needs, such as temperature, humidity, gas concentration levels, and alert status.

### 4.2.7 I2C Module To connect LCD Display

- The I2C (Inter-Integrated Circuit) module facilitates communication between the microcontroller and external devices, such as an LCD (Liquid Crystal Display) display, using the I2C protocol.

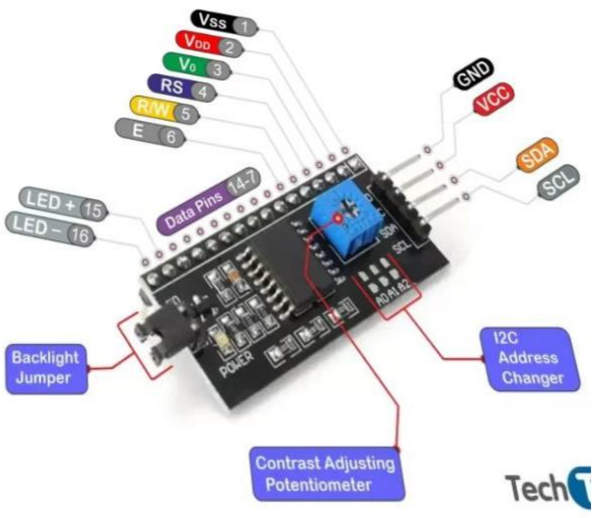


*Figure 4.6 I2C Module*

- **Working Principle**

  I2C is a serial communication protocol that allows multiple devices to communicate with each other using only two wires – SDA (Serial Data) and SCL (Serial Clock) Addressing: Each device connected to the I2C bus has a unique address, allowing the microcontroller to identify and communicate with specific devices.
  Data Transmission: The microcontroller sends data to the LCD display by first sending the device address, followed by the data bytes to be displayed. Control Signals: Along with data transmission, the microcontroller can also send control signals, such as commands to set cursor position or clear the display.

- **Functionality**

  The I2C module enables the microcontroller to interface with an LCD display using only two wires, conserving GPIO pins and simplifying hardware connections. It supports bidirectional communication, allowing both data transmission from the microcontroller to the LCD display and acknowledgment signals from the display to the microcontroller.

### 4.2.8 ST-LINK V2 Debugger

- Implements Programming and Debugging interface for full range of STM8 (SWIM) and STM32 (SWD) series ARM microcontrollers.

- The ST-Link debugger is a tool that helps developers fix problems and load software onto STM32 microcontrollers. It lets them check how the software runs in real-time, like stopping it at specific points or looking at data while it's running. The debugger connects to the microcontroller using special wires and works with software like STM32CubeIDE.

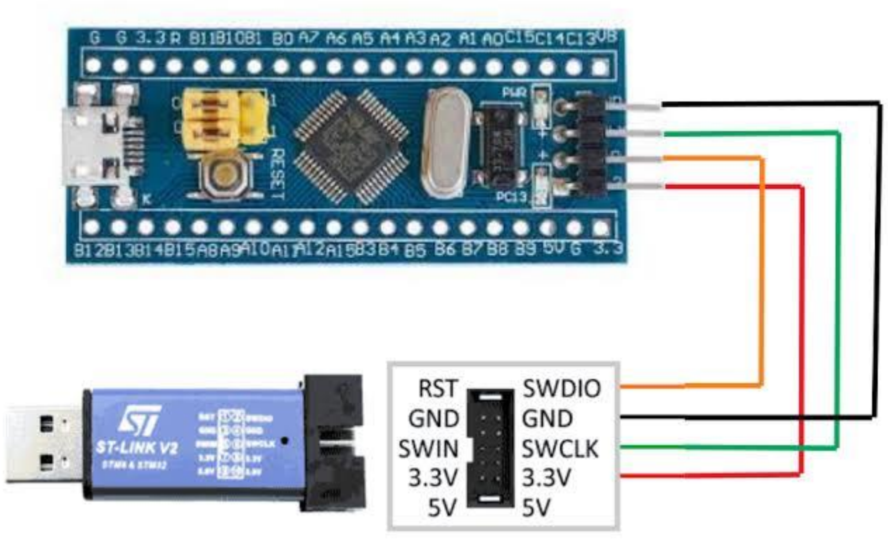

G G 3.3 R B11 B10 B1 B0 A7 A6 A5 A4 A3 A2 A1 A0 C15 C14 C13 VB
RESET
PWR
PC13
B12 B13 B14 B15 A8 A9 A10 A11 A12 A15 B3 B4 B5 B6 B7 B8 B9 5V G 3.3
ST-LINK V2
STM8 & STM32
RST
GND
SWIN
3.3V
5V
SWDIO
GND
SWCLK
3.3V
5V

*Figure: 4.7 ST-LINK V2 Debugger for programming*

## 4.3 Software Requirements

### 4.3.1 STM32CUBEIDE

- STM32CubeIDE is an integrated development environment (IDE) for STM32 microcontrollers, providing a comprehensive set of tools for software development, debugging, and project management.

- Following are the features of STM32CubeIDE:

  **Code Development:**STM32CubeIDE offers a user-friendly environment for writing, editing, and compiling code for STM32 microcontrollers. It supports multiple programming languages, including C and C++, and integrates seamlessly with STM32CubeMX for code generation and configuration.

  **Project Management:** The IDE allows users to create, manage, and organize projects for STM32 microcontroller applications. It provides project templates and wizards for quick setup and configuration, streamlining the development process.

  **Debugging Tools:** STM32CubeIDE includes advanced debugging tools for code analysis, debugging, and optimization. It supports real-time debugging using hardware debuggers, such as ST-LINK, and provides features like breakpoints, watch points, and variable inspection.

  **Peripheral Configuration:** STM32CubeIDE integrates with STM32CubeMX, a graphical tool for configuring STM32 microcontroller peripherals and generating initialization code. Users can easily configure GPIOs, timers, UARTs, SPIs, I2C interfaces, and other peripherals using a graphical interface.

# SYSTEM DESIGN & IMPLEMENTATION

The system design and Implementation of the Safety Alert System encompasses the hardware interactions that enable its functionality and implementation phase involves translating the system design into actual code and hardware configuration. It includes:

### 5.1 Hardware implementation

- The hardware implementation section provides details about the physical components used in the Safety Alert System, including sensors, microcontroller boards, display modules, and alert mechanisms. This section outlines the hardware setup, connections, and integration required to build the system.

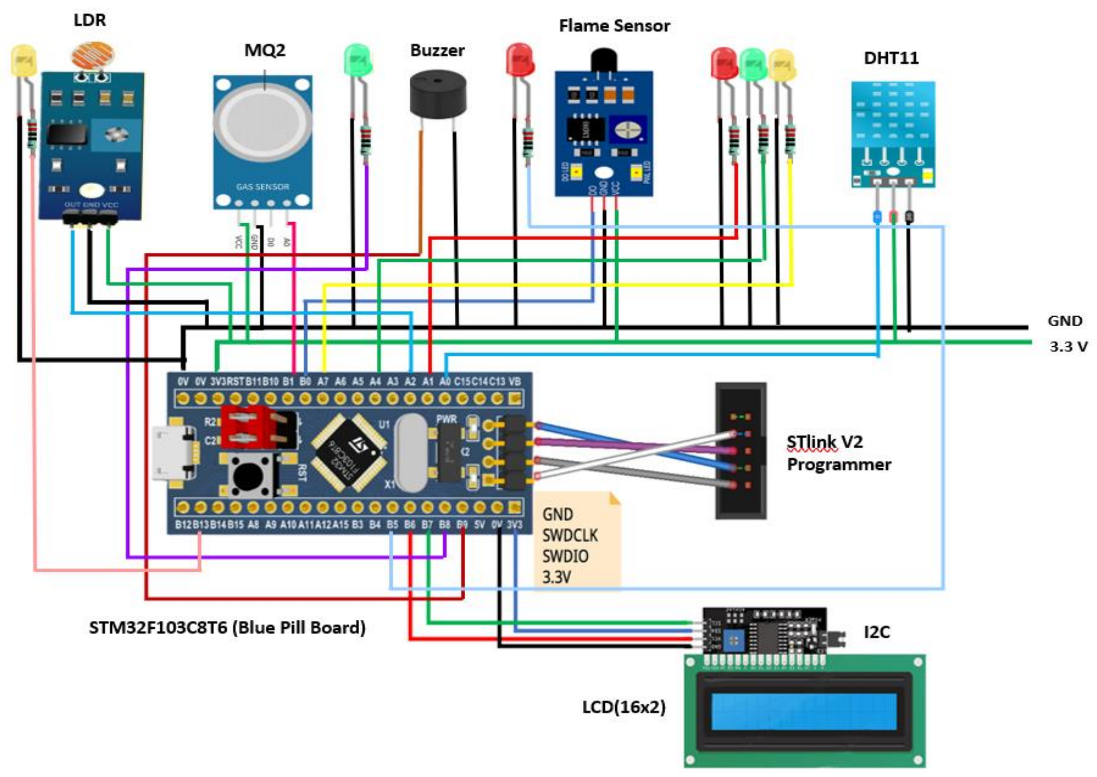


*Figure 5.1 Hardware implementation of Safety Alert System*

### 5.2 Code Implementation

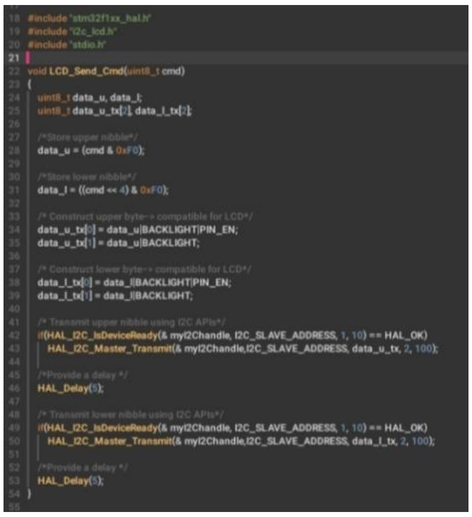
#### 5.2.1 Code Structure

- The code for the Safety Alert System is organized into several modules, each handling specific functionalities such as sensor data acquisition, hazard detection, alert triggering, and user interface interactions. The main file coordinates the execution of these modules and manages the overall system operation.

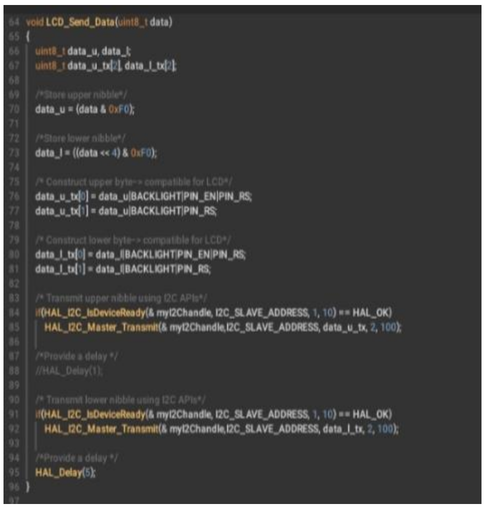
### 5.2.2 Code Snippets

#### 5.2.2.1 Code Implementation of LCD via I2C

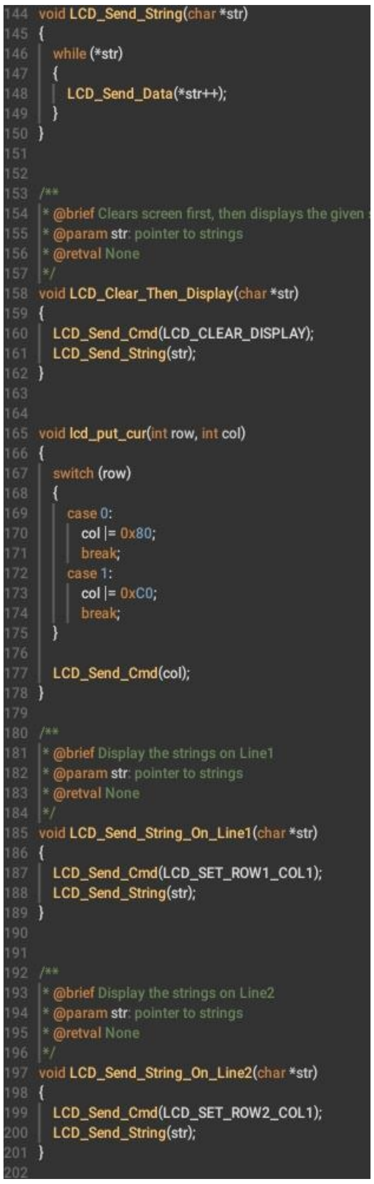
- The below code snippets demonstrate code implementation of lcd via external 12c module. In this code we implement logic for lcd send command, data, string, and display readings and messages.



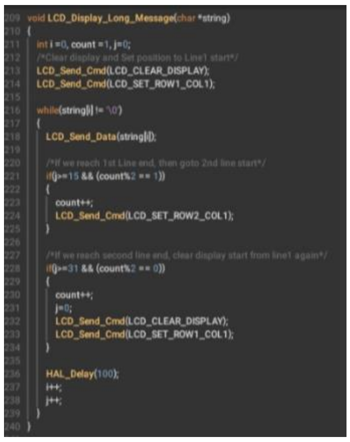
```c
#include "stm32f1xx_hal.h"
#include "i2c_lcd.h"
#include "stdio.h"

void LCD_Send_Cmd(uint8_t cmd)
{
    uint8_t data_u, data_l;
    uint8_t data_u_tx[2], data_l_tx[2];

    /*Store upper nibble*/
    data_u = (cmd & 0xF0);

    /*Store lower nibble*/
    data_l = ((cmd << 4) & 0xF0);

    /* Construct upper byte-> compatible for LCD*/
    data_u_tx[0] = data_u|BACKLIGHT|PIN_EN;
    data_u_tx[1] = data_u|BACKLIGHT;

    /* Construct lower byte-> compatible for LCD*/
    data_l_tx[0] = data_l|BACKLIGHT|PIN_EN;
    data_l_tx[1] = data_l|BACKLIGHT;

    /* Transmit upper nibble using I2C APIs*/
    if(HAL_I2C_IsDeviceReady(& myI2Chandle, I2C_SLAVE_ADDRESS, 1, 10) == HAL_OK)
        HAL_I2C_Master_Transmit(& myI2Chandle,I2C_SLAVE_ADDRESS, data_u_tx, 2, 100);

    /*Provide a delay */
    HAL_Delay(5);

    /* Transmit lower nibble using I2C APIs*/
    if(HAL_I2C_IsDeviceReady(& myI2Chandle, I2C_SLAVE_ADDRESS, 1, 10) == HAL_OK)
        HAL_I2C_Master_Transmit(& myI2Chandle,I2C_SLAVE_ADDRESS, data_l_tx, 2, 100);

    /*Provide a delay */
    HAL_Delay(5);
}
```



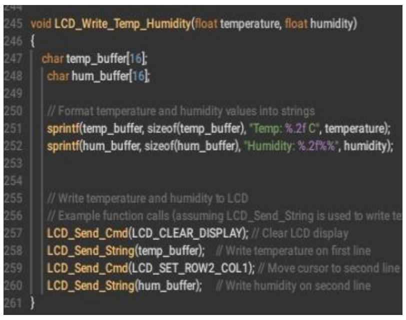
```c
void LCD_Send_Data(uint8_t data)
{
    uint8_t data_u, data_l;
    uint8_t data_u_tx[2], data_l_tx[2];

    /*Store upper nibble*/
    data_u = (data & 0xF0);

    /*Store lower nibble*/
    data_l = ((data << 4) & 0xF0);

    /* Construct upper byte-> compatible for LCD*/
    data_u_tx[0] = data_u|BACKLIGHT|PIN_EN|PIN_RS;
    data_u_tx[1] = data_u|BACKLIGHT|PIN_RS;

    /* Construct lower byte-> compatible for LCD*/
    data_l_tx[0] = data_l|BACKLIGHT|PIN_EN|PIN_RS;
    data_l_tx[1] = data_l|BACKLIGHT|PIN_RS;

    /* Transmit upper nibble using I2C APIs*/
    if(HAL_I2C_IsDeviceReady(& myI2Chandle, I2C_SLAVE_ADDRESS, 1, 10) == HAL_OK)
        HAL_I2C_Master_Transmit(& myI2Chandle,I2C_SLAVE_ADDRESS, data_u_tx, 2, 100);

    /*Provide a delay */
    //HAL_Delay(1);

    /* Transmit lower nibble using I2C APIs*/
    if(HAL_I2C_IsDeviceReady(& myI2Chandle, I2C_SLAVE_ADDRESS, 1, 10) == HAL_OK)
        HAL_I2C_Master_Transmit(& myI2Chandle,I2C_SLAVE_ADDRESS, data_l_tx, 2, 100);

    /*Provide a delay */
    HAL_Delay(5);
}
```



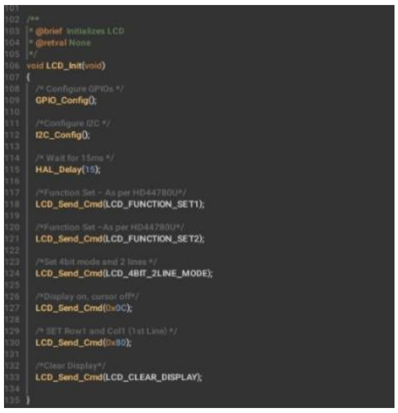
```c
void LCD_Send_String(char *str)
{
    while (*str)
    {
        LCD_Send_Data(*str++);
    }
}


/**
 * @brief Clears screen first, then displays the given
 * @param str: pointer to strings
 * @retval None
 */
void LCD_Clear_Then_Display(char *str)
{
    LCD_Send_Cmd(LCD_CLEAR_DISPLAY);
    LCD_Send_String(str);
}


void lcd_put_cur(int row, int col)
{
    switch (row)
    {
        case 0:
            col |= 0x80;
            break;
        case 1:
            col |= 0xC0;
            break;
    }

    LCD_Send_Cmd(col);
}

/**
 * @brief Display the strings on Line1
 * @param str: pointer to strings
 * @retval None
 */
void LCD_Send_String_On_Line1(char *str)
{
    LCD_Send_Cmd(LCD_SET_ROW1_COL1);
    LCD_Send_String(str);
}


/**
 * @brief Display the strings on Line2
 * @param str: pointer to strings
 * @retval None
 */
void LCD_Send_String_On_Line2(char *str)
{
    LCD_Send_Cmd(LCD_SET_ROW2_COL1);
    LCD_Send_String(str);
}
```



```c
void LCD_Display_Long_Message(char *string)
{
    int i =0, count =1, j=0;
    /*Clear display and Set position to Line1 start*/
    LCD_Send_Cmd(LCD_CLEAR_DISPLAY);
    LCD_Send_Cmd(LCD_SET_ROW1_COL1);

    while(string[i] != '\0')
    {
        LCD_Send_Data(string[i]);

        /*If we reach 1st Line end, then goto 2nd line start*/
        if(j>=15 && (count%2 == 1))
        {
            count++;
            LCD_Send_Cmd(LCD_SET_ROW2_COL1);
        }

        /*If we reach second line end, clear display start from line1 again*/
        if(j>=31 && (count%2 == 0))
        {
            count++;
            j=0;
            LCD_Send_Cmd(LCD_CLEAR_DISPLAY);
            LCD_Send_Cmd(LCD_SET_ROW1_COL1);
        }

        HAL_Delay(100);
        i++;
        j++;
    }
}
```



```c
void LCD_Write_Temp_Humidity(float temperature, float humidity)
{
    char temp_buffer[16];
     char hum_buffer[16];

     // Format temperature and humidity values into strings
     sprintf(temp_buffer, sizeof(temp_buffer), "Temp: %.2f C", temperature);
     sprintf(hum_buffer, sizeof(hum_buffer), "Humidity: %.2f%%", humidity);


     // Write temperature and humidity to LCD
     // Example function calls (assuming LCD_Send_String is used to write te
     LCD_Send_Cmd(LCD_CLEAR_DISPLAY); // Clear LCD display
     LCD_Send_String(temp_buffer);    // Write temperature on first line
     LCD_Send_Cmd(LCD_SET_ROW2_COL1); // Move cursor to second line
     LCD_Send_String(hum_buffer);     // Write humidity on second line
}
```



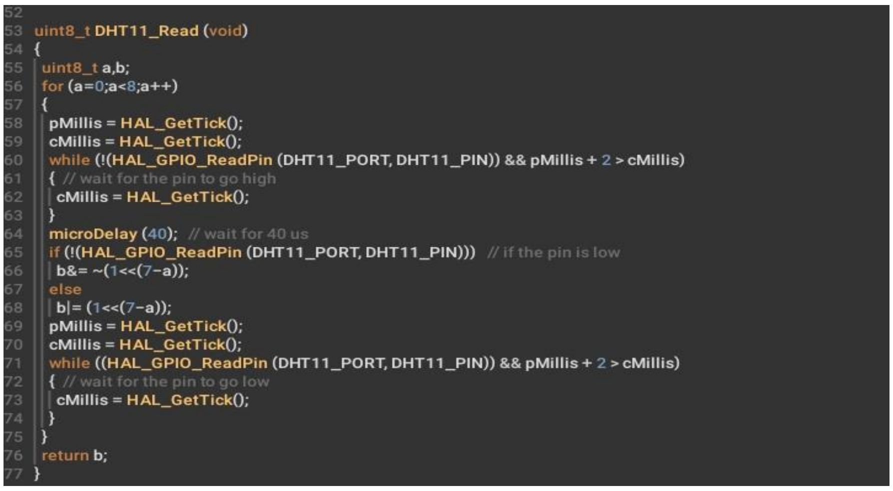
```c
/**
 * @brief Initializes LCD
 * @retval None
 */
void LCD_Init(void)
{
    /* Configure GPIOs */
    GPIO_Config();

    /*Configure I2C */
    I2C_Config();

    /* Wait for 15ms */
    HAL_Delay(15);

    /*Function Set - As per HD44780U*/
    LCD_Send_Cmd(LCD_FUNCTION_SET1);

    /*Function Set -As per HD44780U*/
    LCD_Send_Cmd(LCD_FUNCTION_SET2);

    /*Set 4bit mode and 2 lines */
    LCD_Send_Cmd(LCD_4BIT_2LINE_MODE);

    /*Display on, cursor off*/
    LCD_Send_Cmd(0x0C);

    /* SET Row1 and Col1 (1st Line) */
    LCD_Send_Cmd(0x80);

    /*Clear Display*/
    LCD_Send_Cmd(LCD_CLEAR_DISPLAY);

}
```

*Figure: 5.2 Code Implementation of LCD (16x2) with I2C*

#### 5.2.2.2 Code Implementation of DHT11

- The system continuously collects temperature and humidity data from the DHT11 sensor.
- Sensor readings are sampled at regular intervals and processed to ensure accuracy and reliability.
- Calibration routines may be implemented to account for sensor drift or environmental variations.

- The code snippet below demonstrates how temperature and humidity data are collected using the DHT11 sensor.

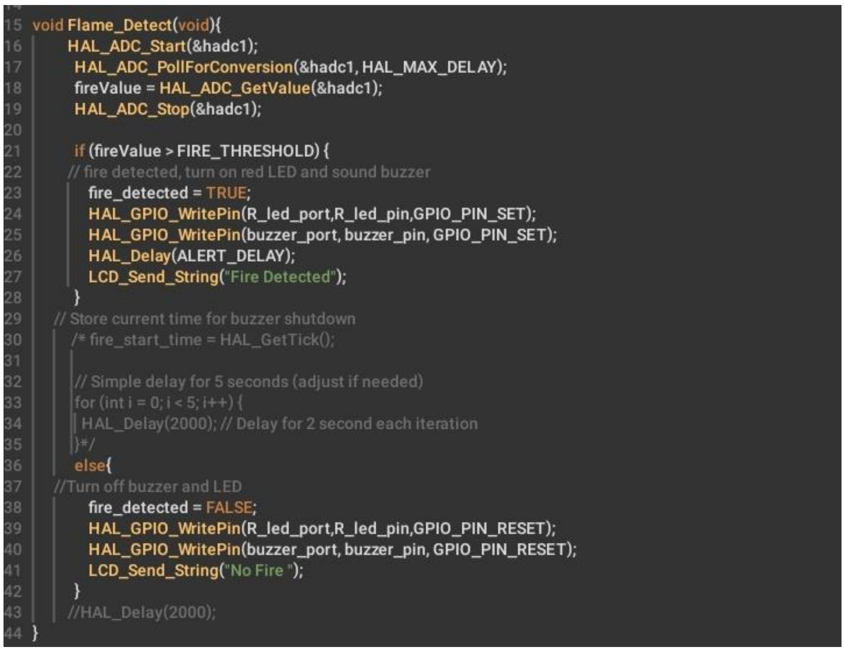


```c

uint8_t DHT11_Read (void)
{
 uint8_t a,b;
 for (a=0;a<8;a++)
 {
  pMillis = HAL_GetTick();
  cMillis = HAL_GetTick();
  while (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
  { // wait for the pin to go high
   cMillis = HAL_GetTick();
  }
  microDelay (40);  // wait for 40 us
  if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))   // if the pin is low
   b&= ~(1<<(7-a));
  else
   b|= (1<<(7-a));
  pMillis = HAL_GetTick();
  cMillis = HAL_GetTick();
  while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
  { // wait for the pin to go low
   cMillis = HAL_GetTick();
  }
 }
 return b;
}
```

*Figure: 5.3 Code Implementation of DHT11(Temperature and Humidity Sensor)*

### 5.2.2.3 Code Implementation of Flame Sensor

- The flame sensor detects the presence of open flames within its detection range.
- Sensor readings are monitored in real-time, and any significant deviation indicating the presence of a flame triggers an immediate alert.
- To prevent false positives, data filtering and thresholding algorithms may be employed to distinguish between genuine flames and other sources of heat.
- The code snippet below demonstrates how open flame detect data are collected using the Flame sensor.

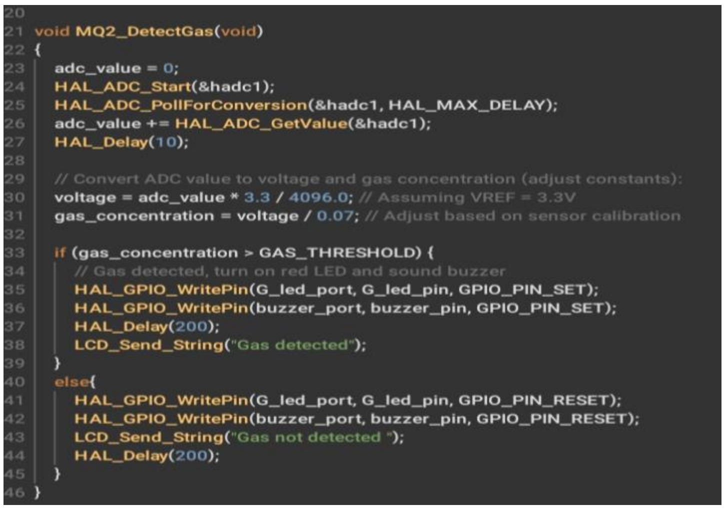


```c
void Flame_Detect(void){
     HAL_ADC_Start(&hadc1);
      HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
      fireValue = HAL_ADC_GetValue(&hadc1);
      HAL_ADC_Stop(&hadc1);

      if (fireValue > FIRE_THRESHOLD) {
     // fire detected, turn on red LED and sound buzzer
        fire_detected = TRUE;
        HAL_GPIO_WritePin(R_led_port,R_led_pin,GPIO_PIN_SET);
        HAL_GPIO_WritePin(buzzer_port, buzzer_pin, GPIO_PIN_SET);
        HAL_Delay(ALERT_DELAY);
        LCD_Send_String("Fire Detected");
      }
   // Store current time for buzzer shutdown
     /* fire_start_time = HAL_GetTick();

      // Simple delay for 5 seconds (adjust if needed)
      for (int i = 0; i < 5; i++) {
       HAL_Delay(2000); // Delay for 2 second each iteration
      }*/
      else{
   //Turn off buzzer and LED
        fire_detected = FALSE;
        HAL_GPIO_WritePin(R_led_port,R_led_pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(buzzer_port, buzzer_pin, GPIO_PIN_RESET);
        LCD_Send_String("No Fire ");
      }
     //HAL_Delay(2000);
}
```

*Figure: 5.4 Code Implementation of Flame Sensor*

### 5.2.2.4 Code Implementation of MQ2

- The MQ2 sensor monitors levels of various harmful gases and smoke particles in the environment.
- Sensor readings are analysed to detect abnormal concentrations of gases or smoke that may pose a safety risk.
- Threshold levels for different gases are configurable, allowing customization based on specific safety requirements.
- The code snippet below demonstrates how gas detect data are collected using the MQ-2 sensor.

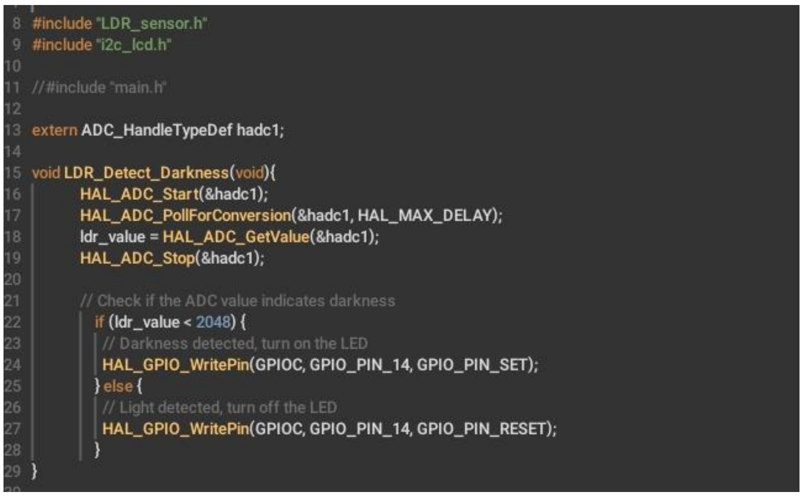


```c

void MQ2_DetectGas(void)
{
    adc_value = 0;
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
    adc_value += HAL_ADC_GetValue(&hadc1);
    HAL_Delay(10);

    // Convert ADC value to voltage and gas concentration (adjust constants):
    voltage = adc_value * 3.3 / 4096.0; // Assuming VREF = 3.3V
    gas_concentration = voltage / 0.07; // Adjust based on sensor calibration

    if (gas_concentration > GAS_THRESHOLD) {
        // Gas detected, turn on red LED and sound buzzer
        HAL_GPIO_WritePin(G_led_port, G_led_pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(buzzer_port, buzzer_pin, GPIO_PIN_SET);
        HAL_Delay(200);
        LCD_Send_String("Gas detected");
    }
    else{
        HAL_GPIO_WritePin(G_led_port, G_led_pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(buzzer_port, buzzer_pin, GPIO_PIN_RESET);
        LCD_Send_String("Gas not detected ");
        HAL_Delay(200);
    }
}
```

*Figure: 5.5 Code Implementation of MQ2(Gas Sensor)*

### 5.2.2.5 Code Implementation of LDR

- The LDR sensor continuously measures ambient light intensity.
- Variations in light intensity beyond predefined thresholds may indicate potential electrical malfunctions or fire hazards.
- Threshold levels are adjustable to accommodate different lighting conditions and environments.
- The code snippet below demonstrates how ambient light detect data are collected using the LDR sensor.

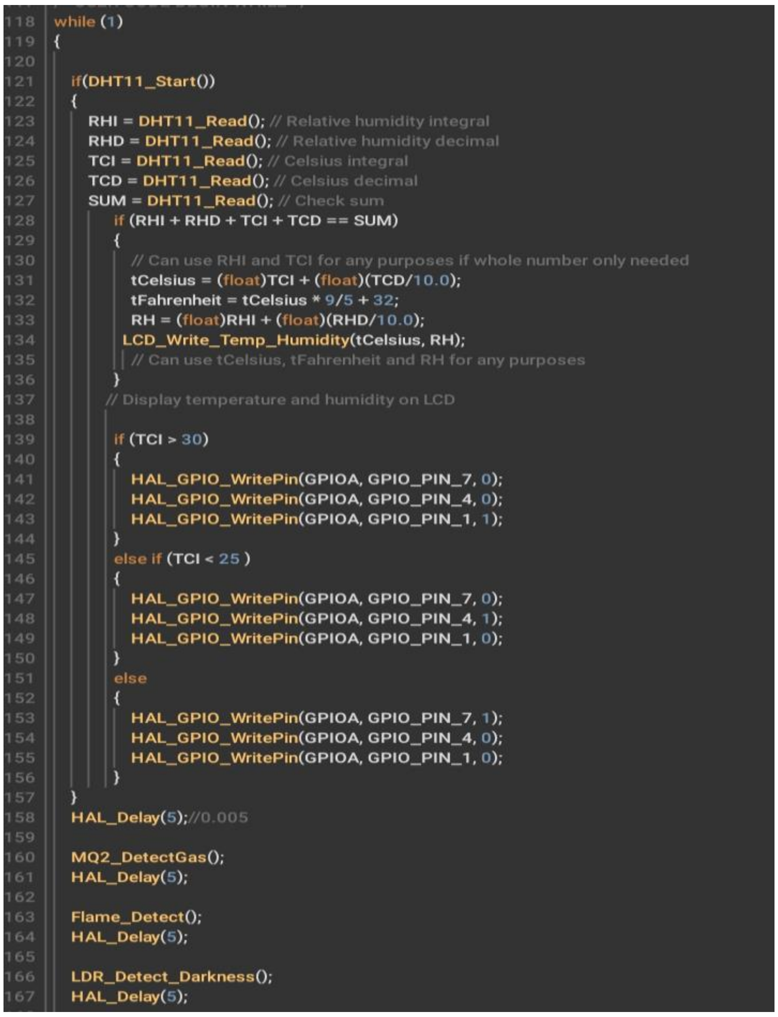


```c
#include "LDR_sensor.h"
#include "i2c_lcd.h"

//#include "main.h"

extern ADC_HandleTypeDef hadc1;

void LDR_Detect_Darkness(void){
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
        ldr_value = HAL_ADC_GetValue(&hadc1);
        HAL_ADC_Stop(&hadc1);

        // Check if the ADC value indicates darkness
          if (ldr_value < 2048) {
           // Darkness detected, turn on the LED
           HAL_GPIO_WritePin(GPIOC, GPIO_PIN_14, GPIO_PIN_SET);
          } else {
           // Light detected, turn off the LED
           HAL_GPIO_WritePin(GPIOC, GPIO_PIN_14, GPIO_PIN_RESET);
          }
}
```

*Figure: 5.6 Code Implementation of LDR (light Dependent Resistor)*

#### 5.2.2.6 Code Implementation of Main.c

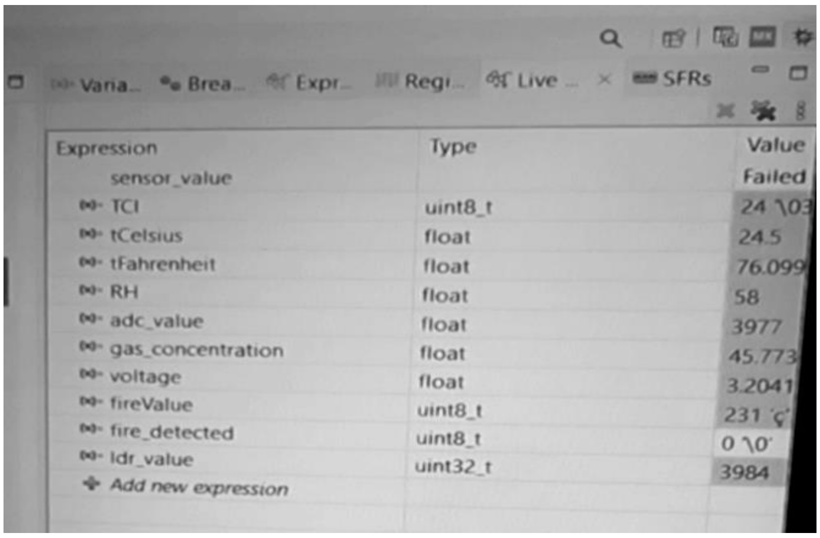
- The code snippet below demonstrates the main program coordinates of various above modules dht11.c, flame_sensor.c, MQ2.c, LDR.c, and lcd_i2c.c and the overall system operation.



```
while (1)
{

  if(DHT11_Start())
  {
    RHI = DHT11_Read(); // Relative humidity integral
    RHD = DHT11_Read(); // Relative humidity decimal
    TCI = DHT11_Read(); // Celsius integral
    TCD = DHT11_Read(); // Celsius decimal
    SUM = DHT11_Read(); // Check sum
      if (RHI + RHD + TCI + TCD == SUM)
      {
        // Can use RHI and TCI for any purposes if whole number only needed
        tCelsius = (float)TCI + (float)(TCD/10.0);
        tFahrenheit = tCelsius * 9/5 + 32;
        RH = (float)RHI + (float)(RHD/10.0);
       LCD_Write_Temp_Humidity(tCelsius, RH);
        // Can use tCelsius, tFahrenheit and RH for any purposes
      }
     // Display temperature and humidity on LCD

      if (TCI > 30)
      {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, 0);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, 0);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, 1);
      }
      else if (TCI < 25 )
      {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, 0);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, 1);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, 0);
      }
      else
      {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, 1);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, 0);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, 0);
      }
  }
  HAL_Delay(5);//0.005

  MQ2_DetectGas();
  HAL_Delay(5);

  Flame_Detect();
  HAL_Delay(5);

  LDR_Detect_Darkness();
  HAL_Delay(5);
```

*Figure: 5.7 Code Implementation of Main.c*

**Output/Readings:**



| Expression | Type | Value |
|---|---|---|
| sensor_value | | Failed |
| TCI | uint8_t | 24 '\03 |
| tCelsius | float | 24.5 |
| tFahrenheit | float | 76.099 |
| RH | float | 58 |
| adc_value | float | 3977 |
| gas_concentration | float | 45.773 |
| voltage | float | 3.2041 |
| fireValue | uint8_t | 231 'ç' |
| fire_detected | uint8_t | 0 '\0' |
| ldr_value | uint32_t | 3984 |
| Add new expression | | |

## CONCLUSION

- In summary, the Safety Alert System project has successfully created a tool to improve safety. By using various sensors and a microcontroller called STM32 Blue Pill, the system can detect dangers like temperature changes, fires, harmful gases, and electrical issues. It shows alerts using LED lights, a buzzer, and a screen.

- The project's software was made with STM32CubeIDE, which helped write and manage the code. By connecting an LCD screen with an I2C module and using the ST-Link debugger, the system became more efficient and user-friendly.

- Overall, the Safety Alert System offers a proactive way to prevent accidents and keep people safe by providing timely warnings about potential dangers. It's a testament to how technology can make a big difference in protecting lives.

## REFERENCES

- STM32CubeIDE User Guide https://STM32CubeIDE-user_guide.pdf
- Stm32 Projects  https://how2electronics.com
- STM32F103C8T Datasheet: https://pdf1.alldatasheet.com/STM32F103C8T6.html
- DHT11 sensor datasheet: https://dht11-datasheet.pdf
- Flame sensor datasheet: https://flamesensor-datasheet.pdf
- MQ2 sensor datasheet: https://Mq2-datasheet.pdf