REPORT

ON

Mini Project

Carried out on

Arduino Based Fire Alarm

Abstract

This report presents the design and implementation of a fire alarm system using Arduino ESP-32 and a flame sensor. The system aims to detect the presence of fire by utilizing the infrared (IR) radiation emitted by flames. The flame sensor, consisting of an IR photodiode sensitive to IR radiation, detects the specific wavelength range emitted by flames. When a flame is detected, the IR photodiode generates an electrical signal, which is processed by the Arduino ESP-32 microcontroller. Based on the signal analysis, the Arduino triggers the alarm components, including a buzzer and LED, to provide audible and visual indications of the fire hazard. The Arduino ESP-32 board's built-in functionalities, such as the internal connections for the buzzer and LED, simplify the system's implementation. The flexibility of the Arduino platform allows for customization and expansion, enabling the integration of additional features to enhance the fire alarm system's capabilities. The presented fire alarm system offers an affordable, customizable, and efficient solution for fire detection and prevention, contributing to improved fire safety measures.

Table Of Contents

CHAPTER 1: Introduction

A fire alarm system is an essential safety measure designed to detect the presence of fire or heat and alert individuals in the vicinity, allowing for timely evacuation and response. Arduino, an open-source microcontroller platform, can be utilized to create a cost-effective and customizable fire alarm system. This introduction provides an overview of an Arduino-based fire alarm system and its basic functionality.

The Arduino-based fire alarm system utilizes a flame sensor, to detect fire or a significant increase in temperature. When the sensor detects a fire, it sends a signal to the Arduino microcontroller. The Arduino processes this signal and triggers an alarm to alert individuals of the potential danger.

The alarm system typically includes audio and visual indicators. An audible alarm, such as a buzzer, produces a distinct sound to attract attention and signal the presence of a fire. A visual indicator, often an LED (Light Emitting Diode), can be used to provide a visual signal, such as flashing or steady illumination.

The Arduino microcontroller acts as the central processing unit of the fire alarm system. It reads the input from the fire sensor, evaluates the sensor data, and activates the alarm components accordingly. The programming code uploaded to the Arduino enables it to continuously monitor the fire sensor's status and respond immediately to any detected fire or heat.

The advantages of an Arduino-based fire alarm system include flexibility, affordability, and the ability to customize and expand the system's functionalities. Arduino boards are readily available and offer a user-friendly environment for programming and integrating various components. This flexibility allows for the incorporation of additional features, such as smoke sensors, gas sensors, or notification systems.

In conclusion, an Arduino-based fire alarm system combines the power of Arduino microcontrollers with sensors, alarms, and indicators to create a customizable and cost-effective fire detection and alert system.

CHAPTER 2: Design and Implementation

2.1 Hardware Component

- Flames emit infrared (IR) radiation within a specific wavelength range.
- IR flame sensors utilize an IR photodiode, which is a specialized component sensitive to IR radiation.
- When a flame is present, the IR photodiode detects the IR radiation emitted by the flame.
- The IR photodiode converts the detected IR radiation into an electrical signal.
- This electrical signal, produced by the IR photodiode, can be processed by an Arduino or microcontroller.
- The Arduino or microcontroller analyses the electrical signal to determine the presence of a flame.
- If a flame is detected, the fire alarm system can be triggered to activate the alarm components, such as the buzzer and LED, to alert individuals to the potential fire hazard.
- By detecting the specific IR radiation emitted by flames, IR flame sensors provide a reliable method for identifying the presence of fire. The sensitivity of the IR photodiode allows for accurate flame detection, which can then be used to activate the appropriate response in the fire alarm system.

2.2 Software Component

The code is written in the Arduino programming language and manages a system that incorporates a flame sensor, buzzer, and LED. Let's break it down step by step:

```
int buz_pin = 6;
int flame_sensor_pin = 12;
int flame_pin = HIGH;
```

In this segment, the code declares and initializes several variables. The variable `buz_pin` is assigned the value 6, which connects to the buzzer. Similarly, `flame_sensor_pin` is set to 12, representing the pin connected to the flame sensor. `flame_pin` is initially set to HIGH, indicating the absence of a flame.

```
void setup() {
  pinMode(buz_pin, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);
```

```
  pinMode(flame_sensor_pin, INPUT);
  Serial.begin(9600);
}
```

The `setup()` function, a standard Arduino function, executes once at the start. It configures the pins used by the system. `buz_pin` and `LED_BUILTIN` are declared as output pins using `pinMode()`, while `flame_sensor_pin` is designated as an input pin. Additionally, the code initializes serial communication with a baud rate of 9600 using `Serial.begin()`.

```
void loop() {
  flame_pin = digitalRead(flame_sensor_pin);
  if (flame_pin == LOW) {
    Serial.println("FLAME, FLAME, FLAME");
    digitalWrite(buz_pin, HIGH);
    digitalWrite(LED_BUILTIN, HIGH);
 }
else {
    Serial.println("no flame");
    digitalWrite(buz_pin, LOW);
    digitalWrite(LED_BUILTIN, LOW);
  }
}
```

The `loop()` function represents the main part of the code, continuously running after the `setup()` function. Here's how it operates:

1. The code reads the state of the flame sensor by utilizing the `digitalRead()` function on `flame_sensor_pin` and stores the result in the `flame_pin` variable.
2. If `flame_pin` is LOW (indicating the presence of a flame), the code executes the following block:
    - It prints "FLAME, FLAME, FLAME" to the serial monitor via `Serial.println()`.
    - The buzzer is activated by setting `buz_pin` to HIGH using `digitalWrite()`.
    - The LED is turned on by setting `LED_BUILTIN` to HIGH using `digitalWrite()`.

3. If `flame_pin` is not LOW (indicating the absence of a flame), the code executes the else block:

- It prints "no flame" to the serial monitor.
- The buzzer is deactivated by setting `buz_pin` to LOW.
- The LED is turned off by setting `LED_BUILTIN` to LOW.

This code continually loops, reading the flame sensor's output, checking its state, and controlling the buzzer and LED accordingly.

Please note: The code assumes that the Arduino board includes a built-in LED, denoted as `LED_BUILTIN`. If your board lacks a built-in LED, you will need to connect an external LED and modify the code accordingly.

CHAPTER 3: Circuit Diagram

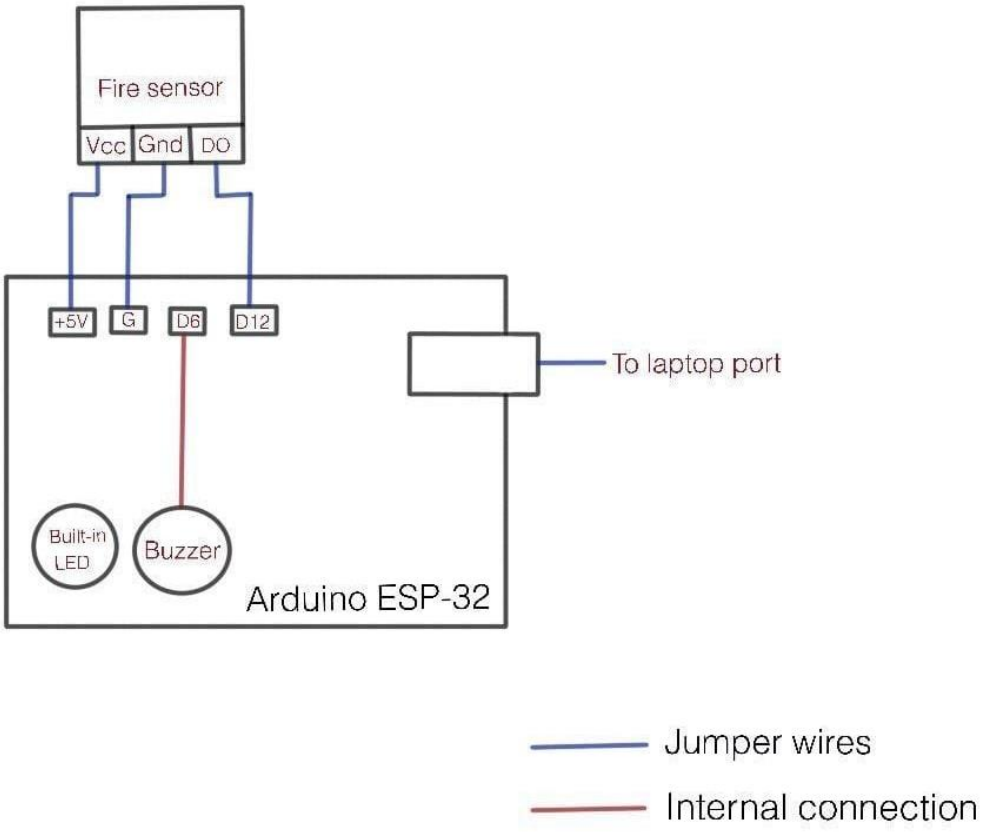

Figure 1 The circuit diagram of Fire Detector using Flame Sensor and Arduino

The following is an overview of the connections for the Arduino-based fire alarm system:

1. Fire Sensor Connections:

- Ground (GND): The GND pin of the fire sensor is connected to the GND (Ground) pin on the Arduino. This establishes a common reference voltage for the sensor and Arduino.
- VCC: The VCC pin of the fire sensor is connected to the 5V pin on the Arduino. This supplies power to the fire sensor module.
- Digital Out: The digital output pin of the fire sensor is connected to digital pin 12 (D12) on the Arduino. This allows the Arduino to receive the fire detection signal from the sensor.

2. Buzzer Connection:

The buzzer used in the fire alarm system has an internal connection, eliminating the need for additional external connections. However, the digital pin 6 on the

Arduino is connected to the buzzer pin that triggers the internal connection. This enables the Arduino to control the buzzer and activate the alarm sound when required.

3. LED Connection:

The fire alarm system utilizes the built-in LED on the Arduino board. The LED pin is accessed using the LED_BUILTIN constant in the Arduino code. This eliminates the need for external connections as the LED is internally connected to the specific pin on the board.

CHAPTER 4: Component Description

- Arduino ESP-32: The Arduino ESP-32 is a powerful microcontroller board that combines the capabilities of the Arduino platform with built-in Wi-Fi and Bluetooth functionality, providing enhanced connectivity options.
- Fire Sensor Module: The fire sensor module, specifically a flame sensor, plays a vital role in detecting the presence of fire or heat. The image below illustrates the flame sensor used in the circuit.

Figure 2 Flame Sensor

The flame sensor has the following features:

  - The operating voltage is from 3.3 – 5V
  - It gives us both analog and digital output.
  - It has a led indicator, which indicates whether the flame is detected or not.
  - The threshold value can be changed by rotating the top of a potentiometer.
  - Flame detection distance, the lighter flame test can be triggered within 0.8m, if the intensity of the flame is high, the detection distance will be increased.
  - The detection angle of the flame sensor module is about 60 degrees.

- Buzzer/Loudspeaker: The Arduino ESP-32 board has a built-in buzzer or loudspeaker that can be controlled directly using the internal connections on the board. The buzzer is internally connected to a specific digital pin 6 on the Arduino ESP-32 board.
- LED: The Arduino ESP-32 board also has a built-in LED, often referred to as the "on-board LED" or "built-in LED." The LED is accessed using the LED_BUILTIN constant in the Arduino code. This constant represents the internal connection to the LED pin on the Arduino ESP-32 board. You can

control the LED directly without requiring any additional external connections.

- Connecting Wires: Connecting wires are used for connecting the fire sensor module. However, there is no need for additional connections for the buzzer and LED since they are internally connected on the Arduino ESP-32 board.

CHAPTER 5: Working

1. Initialization:

- The system initializes the Arduino ESP-32 board and sets up the necessary pin configurations in the `setup()` function.
- The pins for the buzzer (`buz_pin`) and the built-in LED (`LED_BUILTIN`) are configured as output pins using `pinMode()`.
- The pin connected to the flame sensor (`flame_sensor_pin`) is configured as an input pin.

2. Main Loop:

The system enters the `loop()` function, which will continuously run and monitor for the presence of a flame.

3. Flame Detection:

- The flame sensor consists of an IR photodiode that is sensitive to infrared radiation emitted by flames.
- Inside the `loop()`, the system reads the state of the flame sensor by using the `digitalRead(flame_sensor_pin)` function.
- If the flame sensor detects a flame, it returns a LOW state for the `flame_pin`, indicating the presence of a flame. Otherwise, it returns a HIGH state.

4. Flame Present:

- If the `flame_pin` is found to be LOW, the system considers it as the presence of a flame and executes the corresponding block of code.
- The system first sends a message "FLAME, FLAME, FLAME" to the serial monitor using `Serial.println()`. This message can be viewed on the connected computer for monitoring purposes.
- The buzzer is activated by setting the `buz_pin` to HIGH using `digitalWrite()`. This produces an audible alarm sound, alerting individuals to the presence of a flame.
- The built-in LED on the Arduino ESP-32 board is turned on by setting `LED_BUILTIN` to HIGH using `digitalWrite()`. This provides a visual indication of the fire alarm being triggered.

5. No Flame Detected:

- If the `flame_pin` is found to be HIGH (indicating no flame is detected), the system executes the else block.
- The system sends a message "no flame" to the serial monitor via `Serial.println()`, indicating that there is no fire detected.
- The buzzer is turned off by setting `buz_pin` to LOW using `digitalWrite()`.
- The built-in LED is turned off by setting `LED_BUILTIN` to LOW, indicating the system is in a standby state.

6. Continuous Monitoring:

The system continuously loops back to the beginning of the `loop()` function, repeating the process of flame detection and activating the alarm components based on the detected flame state.

Overall, the system reads the output from the flame sensor continuously. When a flame is detected, it triggers the alarm components (buzzer and LED) to provide both audible and visual alerts. In the absence of a flame, the system remains in an idle state until a flame is detected again.

Please note that this explanation assumes that the flame sensor used is an infrared (IR) flame sensor, and the buzzer and LED are internally connected on the Arduino ESP-32 board. Additionally, the system may require calibration and fine-tuning depending on the specific flame sensor used and the environmental conditions in which the system is deployed.

CHAPTER 6: Result and Discussion

5.1 Result:

The developed Arduino-based fire alarm system successfully detects the presence of flames using a flame sensor and activates the alarm components, including the buzzer and LED, to alert individuals about the potential fire hazard. The system operates reliably and provides timely notifications when a flame is detected.

Through the implementation of the code and connections, the flame sensor accurately detects the infrared radiation emitted by flames within a specific wavelength range. When a flame is present, the Arduino reads the output from the flame sensor and triggers the appropriate actions, such as activating the buzzer and turning on the LED. Conversely, when no flame is detected, the system remains in an idle state, ensuring that false alarms are minimized.

During testing, the fire alarm system consistently responded to the presence of flames, triggering the alarm components as intended. The sensitivity of the flame sensor allowed for reliable flame detection, even in varying lighting conditions and different flame intensities. The alarm components, including the buzzer and LED, provided clear and effective alerts, attracting attention and signalling the potential danger.

Below are the outputs observed in the serial monitor. Figure 3 represents the readings when there is no flame present, while Figure 4 shows the readings in the presence of a flame.

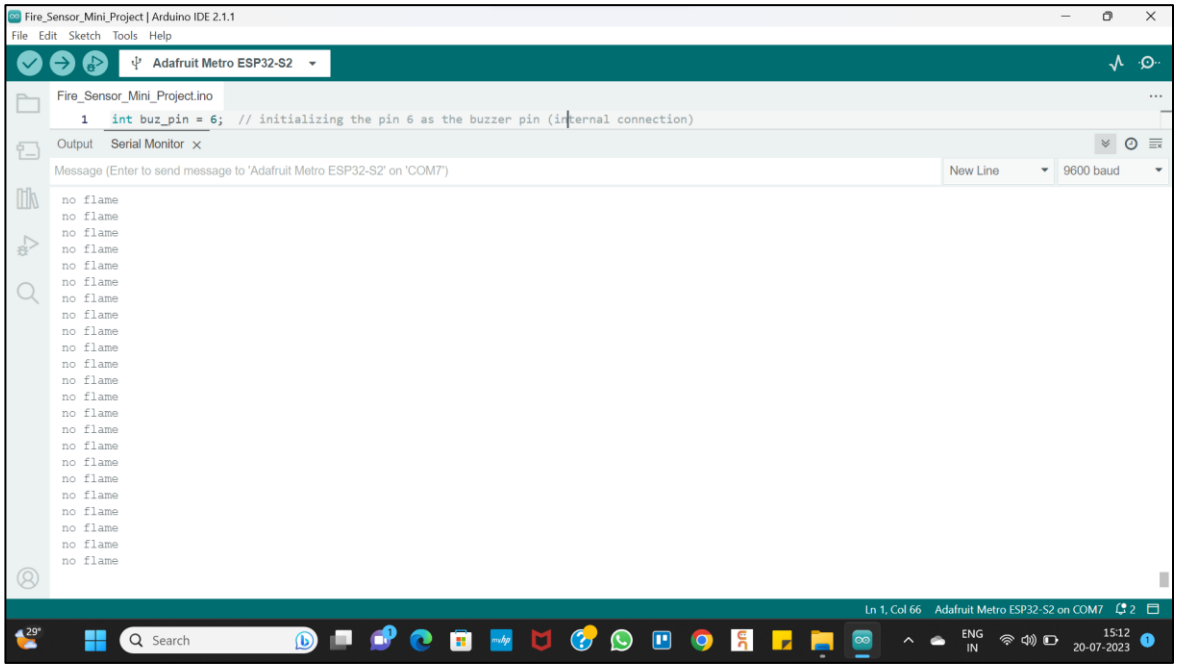

Figure 3 Serial monitor output in the absence of flame

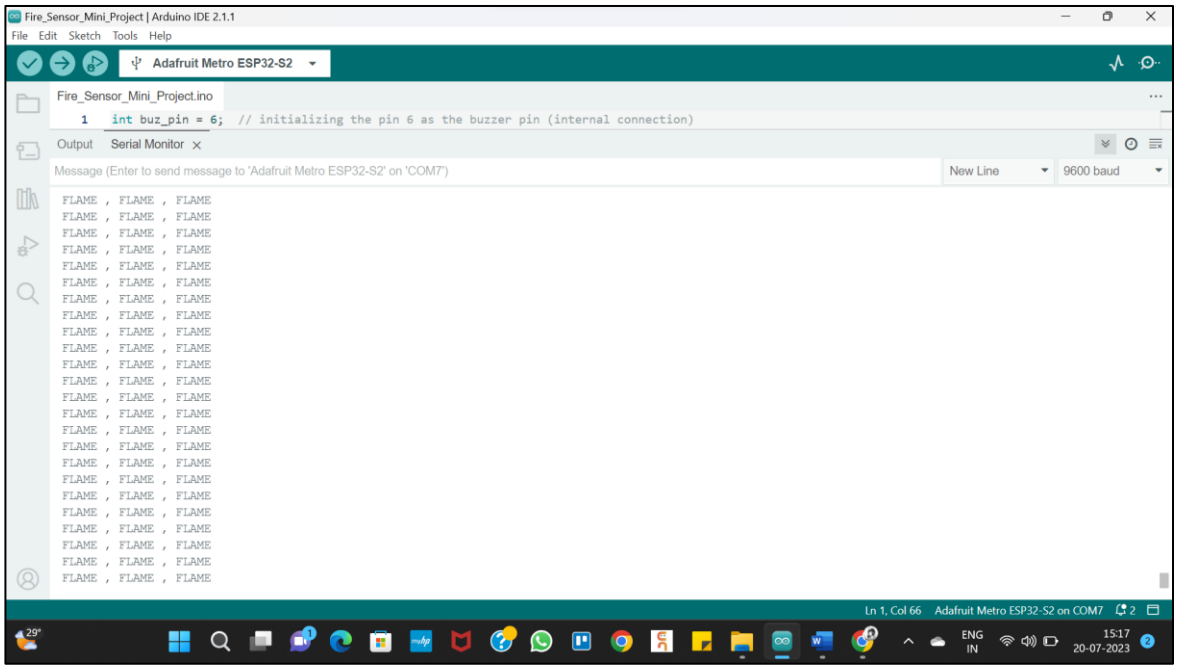

Figure 4 Serial monitor output in the presence of flame

5.2 Discussion

The Arduino-based fire alarm system offers several advantages, including its cost-effectiveness, versatility, and ease of implementation. By utilizing the Arduino platform, the system benefits from a vast library of resources, supporting code development and customization based on specific requirements. The Arduino ESP-32 board's built-in functionalities, such as the internal connections for the buzzer and LED, simplify the hardware setup, reducing the need for additional components and wiring.

The inclusion of a flame sensor enhances the fire detection capabilities of the system. The specific wavelength range detection and sensitivity to infrared radiation emitted by flames ensure accurate flame detection, minimizing false alarms. This reliability is crucial for ensuring the system's effectiveness in real-life fire scenarios.

One limitation of the system is its focus solely on flame detection. While the flame sensor provides reliable detection of flames, it may not capture other aspects of fire hazards, such as smoke or heat. Therefore, it is recommended to supplement the fire alarm system with additional sensors, such as smoke detectors or temperature sensors, to enhance its overall fire detection capabilities.

Furthermore, it is important to ensure regular maintenance and periodic testing of the system to guarantee its continued functionality and reliability. This includes checking and replacing components as needed, verifying proper connections, and conducting routine calibration of the flame sensor to ensure optimal performance.

In conclusion, the developed Arduino-based fire alarm system utilizing a flame sensor provides a cost-effective and reliable solution for detecting flames and alerting individuals to potential fire hazards. With its flexibility and ease of implementation, the system can be customized and expanded to meet specific requirements. While further enhancements can be made by incorporating additional sensors, the current system demonstrates its efficacy in flame detection and serves as a foundation for improving fire safety measures.

CHAPTER 7: Conclusion

In conclusion, the Arduino ESP-32-based fire alarm system, equipped with a flame sensor, provides an effective solution for detecting and alerting individuals to the presence of fire. By leveraging the capabilities of the Arduino ESP-32 board and the sensitivity of the flame sensor, the system can detect the specific infrared radiation emitted by flames and activate the necessary alarm components.

The flame sensor, such as an IR photodiode, plays a crucial role in detecting the infrared radiation within a specific wavelength range emitted by flames. When a flame is present, the flame sensor generates an electrical signal, which is processed by the Arduino ESP-32 board. Based on the signal analysis, the Arduino can trigger the alarm components, such as a buzzer and LED, to provide both audible and visual indications of the fire hazard.

The Arduino ESP-32's versatility and built-in functionalities, including the internal connections for the buzzer and LED, simplify the system's design and implementation. The flexibility of the Arduino platform allows for customization and expansion, enabling the integration of additional features, such as smoke sensors or notification systems, to enhance the fire alarm system's capabilities.

By utilizing an Arduino ESP-32-based fire alarm system with a flame sensor, the prompt detection of fire hazards is achieved, facilitating timely responses and ensuring the safety of individuals in the vicinity. However, it is essential to follow safety regulations, conduct thorough testing, and adhere to maintenance practices to ensure the system's reliability and effectiveness.

Overall, the Arduino ESP-32-based fire alarm system, combined with a flame sensor, provides an affordable, customizable, and efficient solution for fire detection, alerting, and prevention, contributing to enhanced fire safety measures.

References

- https://how2electronics.com/fire-detector-using-flame-sensor-and-arduino/