FIRE DETECTION AND ALARM NOTIFICATION SYSTEM ESP-32

Team Members:

BALAJI N M - 312323106028

HAFEEZUR RAHIMAN - 312323106052

DILLAN ROMAULD VITUS - 312323106044

ASHWIN SUDHAKARAN - 312323106026

Aim

The aim of the "Fire Detection and Alarm Notification System ESP-32" project is to develop an intelligent fire detection system that can detect fire threats and activate alarms to alert users in real-time. This system utilizes sensors, including the HW 036A infrared (IR) flame sensor and the HW 484 ultraviolet (UV) flame sensor, to detect the presence of fire.

Tools/Hardware Required

- ESP-32
- HW 036A sensor
- HW 484 sensor
- Breadboard and Jumper Wires

Theory

ESP-32 - The ESP32 is a powerful, low-cost microcontroller with integrated Wi-Fi and Bluetooth. It acts as the central control unit for this system, reading data from the sensors and executing the logic for the fire alarm. Its built-in Wi-Fi can be used to send alerts or notifications to a user's phone or a cloud service when a fire is detected.

HW 036A - The HW 036A is an infrared (IR) flame sensor. It works by detecting the infrared light spectrum emitted by a flame. The sensor is highly sensitive to the IR wavelength emitted by fire. The module typically has both an analog and a digital output. The digital output can be used for a simple on/off detection of a

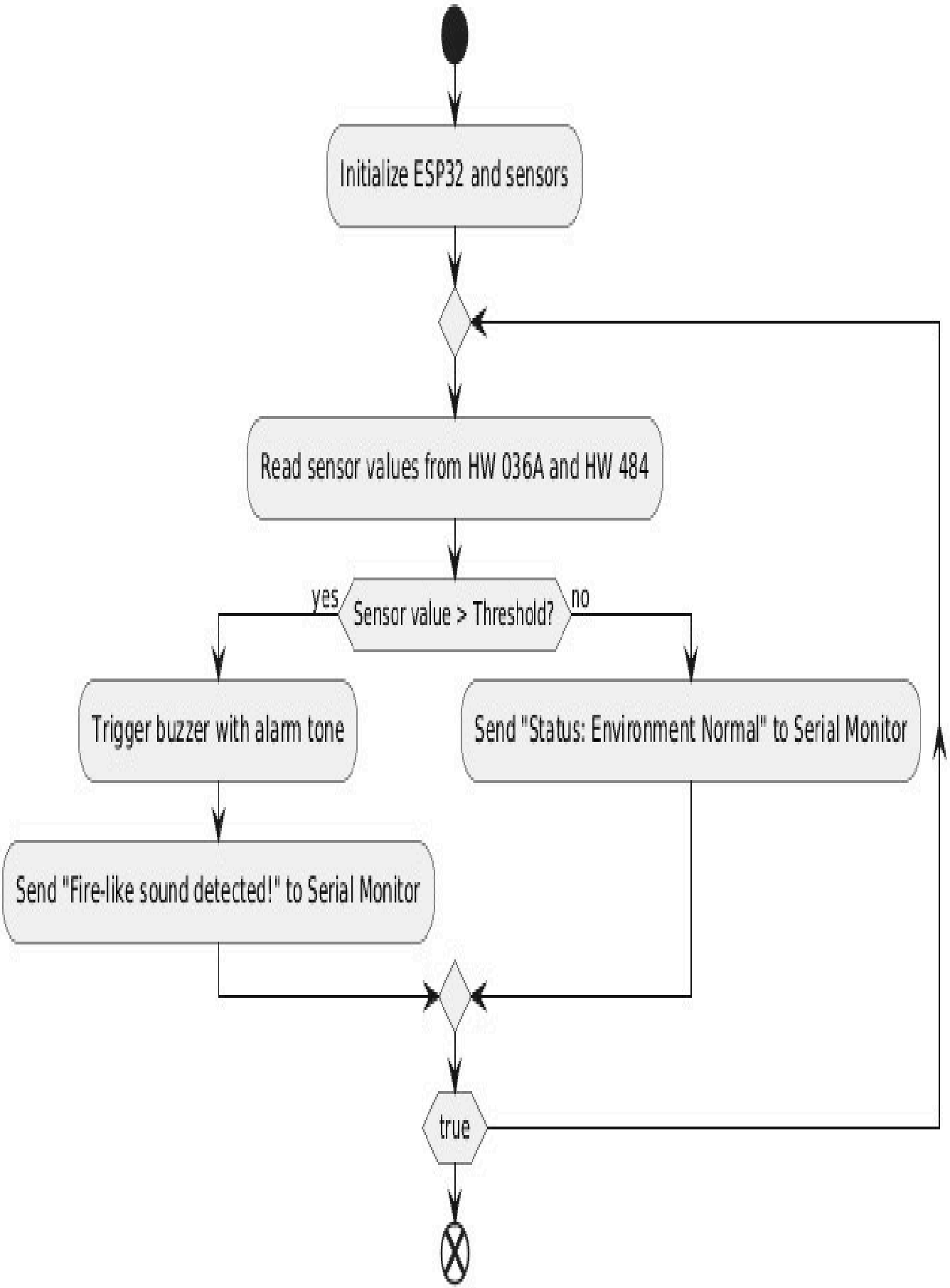
flame, while the analog output provides a value proportional to the intensity of the IR light, allowing for more precise fire detection.

HW 484 - The HW 484 is an ultraviolet (UV) flame sensor. It detects the ultraviolet light emitted by a flame, which is a different spectrum from the IR light detected by the HW 036A. Using both an IR and a UV sensor together significantly improves the system's reliability. A common fire detection strategy is to combine these two sensor types to reduce false alarms. For example, a strong heat source (like a hot plate) might trigger an IR sensor, but it won't emit UV light, preventing a false alarm. A real flame, however, will trigger both sensors.

Pin table

Components	**ESP32**
HW-484 OUT	GPI0 21
HW-484 AOUT	GPIO 34
HW-036A Signal	GPIO 18
HW-036A VCC	3.3 V
HW-036A GND	GND

Flowchart

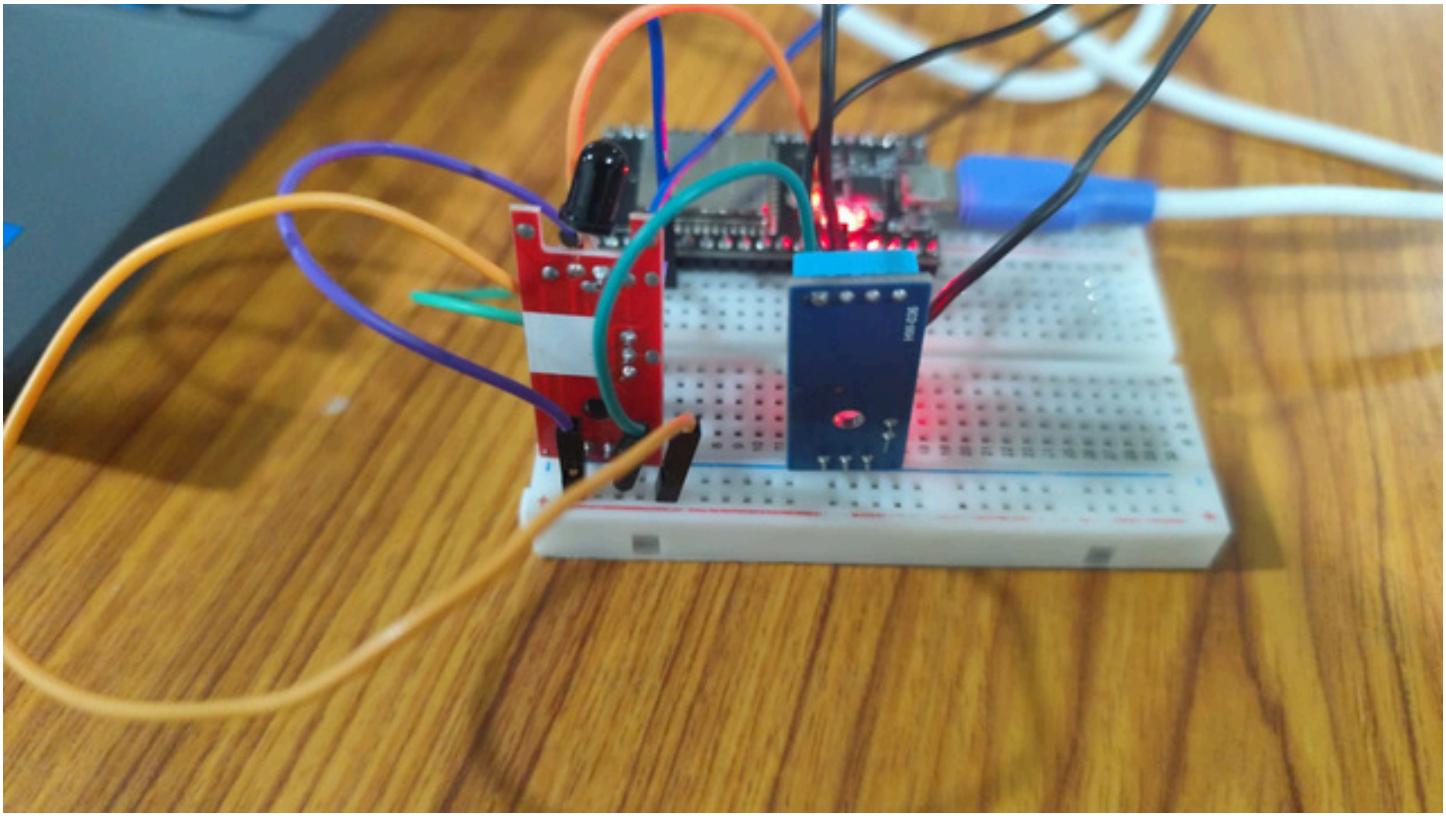

Initialize ESP32 and sensors
Read sensor values from HW 036A and HW 484
Sensor value > Threshold?
yes
no
Trigger buzzer with alarm tone
Send "Fire-like sound detected!" to Serial Monitor
Send "Status: Environment Normal" to Serial Monitor
true

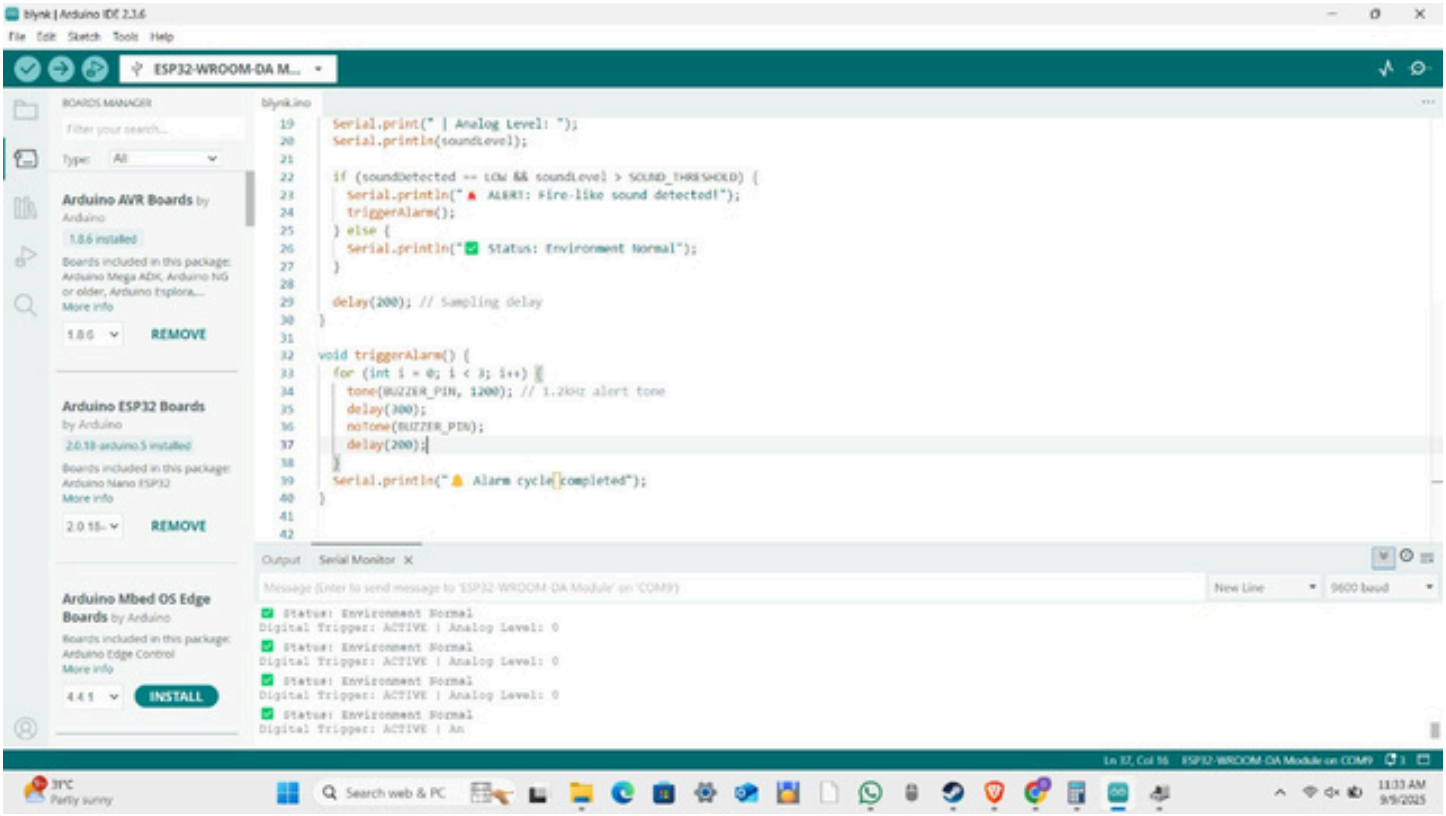
Code

```
#define SOUND_DIGITAL_PIN 21   // Digital output from HW-484
#define SOUND_ANALOG_PIN 34    // Analog output from HW-484
#define BUZZER_PIN 18          // Signal pin to HW-036A buzzer
#define SOUND_THRESHOLD 500    // Adjust based on environment
void setup() {
  pinMode(SOUND_DIGITAL_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  Serial.begin(9600);
  Serial.println("Fire Detection System Initialized");
}
void loop() {
  int soundDetected = digitalRead(SOUND_DIGITAL_PIN);
  int soundLevel = analogRead(SOUND_ANALOG_PIN);
  Serial.print("Digital Trigger: ");
  Serial.print(soundDetected == LOW ? "ACTIVE" : "INACTIVE");
  Serial.print(" | Analog Level: ");
  Serial.println(sound Level);
  if (soundDetected == LOW && soundLevel > SOUND_THRESHOLD) {
    Serial.println("ALERT: Fire-like sound detected!");
    triggerAlarm();
  } else {
    Serial.println("Status: Environment Normal");
  }
  delay(200); // Sampling delay
}
void triggerAlarm() {
  for (int i = 0; i < 3; i++) {
    tone(BUZZER_PIN, 1200); // 1.2kHz alert tone
    delay(300);
```

```
    noTone(BUZZER_PIN);
    delay(200);
  }
  Serial.println("Alarm cycle completed");
}
```

Demonstration:





Execution

Hardware Setup

The ESP32 development board (HW-036A/HW-484) was connected to the system using a data USB cable.

A sound/fire detection sensor was interfaced with the ESP32 on analog pin GPIO34.

A buzzer was connected to digital pin GPIO25 for alarm indication.

The circuit was powered via the ESP32 USB port.

Software Setup

The Arduino IDE was installed and configured with ESP32 board support.

The board package for ESP32 was added via the URL:

https://dl.espressif.com/dl/package_esp32_index.json

From Tools → Board Manager, the ESP32 Dev Module was selected.

The correct COM port was identified and chosen under Tools → Port.

Driver Installation

To enable communication between the ESP32 and the computer, the appropriate USB-to-Serial drivers (CH340 or CP210x) were installed.

Once installed, the ESP32 COM port became available in the Arduino IDE.

Program Upload

A calibration code was first uploaded to test sensor readings and on-board LED blink.

After verifying hardware functionality, the fire detection program was uploaded.

The program initializes serial communication at 115200 baud rate, continuously monitors the sensor, and compares values with a threshold (500).

Execution Process

In a normal environment, sensor readings were below the threshold, and the Serial Monitor displayed:

Status: Environment Normal

When a fire-like event (sound/smoke/light) was simulated, the sensor value exceeded the threshold.

The ESP32 triggered the buzzer with a 1.2 kHz alarm tone in three quick cycles.

The Serial Monitor displayed:

ALERT: Fire-like sound detected!

Alarm cycle completed

Calibration

Multiple tests were conducted to note sensor readings under both normal and abnormal (fire-like) conditions.

The threshold value was fine-tuned to 500, ensuring reliable detection while minimizing false alarms.

Final Outcome

The ESP32 successfully detected fire-like conditions.

Immediate buzzer alerts and real-time serial notifications confirmed the system's proper working.

This forms the base implementation, which can be extended to include cloud notifications (Blynk, Firebase, SMS/WhatsApp alerts) for remote fire monitoring