Fire Detector System

USING ARDUINO UNO

**System Features:**

* **Fire Detection** via Flame Sensor
* **Visual Alert** through LED
* **Audible Alert** via Buzzer
* **Real-time Feedback** on LCD
* **System Reset** via Push Button
* **Threshold Sensitivity** adjustment for sensor calibration
* **Continuous Monitoring** of the flame sensor
* **Low Power Consumption**
* **Expandable System** with additional sensors
* **User Interface** for easy interaction

**Components Needed:**

* Arduino Uno
* Flame sensor module (used to detect fire)
* LED
* Push button
* Buzzer
* 16x2 LCD screen
* Resistors (220Ω for LED, 10kΩ for pull-down resistor on the push button)
* Breadboard and jumper wires

**Circuit Diagram:**

1. **Flame Sensor**:

Connect the flame sensor’s output pin to an available analog input pin (e.g., A0) on the Arduino.

1. **LED**:

Connect the LED’s anode (long leg) to a digital output pin (e.g., pin 2), and the cathode (short leg) to the ground through a 220Ω resistor.

1. **Push Button**:

Connect one terminal of the push button to a digital input pin (e.g., pin 3) and the other terminal to ground. Also, use a 10kΩ pull-down resistor between the input pin and ground to ensure the button state is read correctly.

1. **Buzzer**:

Connect the buzzer's positive terminal to a digital output pin (e.g., pin 4) and the negative terminal to ground.

1. **LCD**:

Connect the LCD screen to the Arduino Uno (using the standard 4-wire mode for simplicity) according to the pins you select (e.g., RS to pin 12, E to pin 11, D4 to pin 5, D5 to pin 4, D6 to pin 3, D7 to pin 2, and VSS/GND to ground).

**Software Development**

The software for this fire detection system is developed by programming the Arduino Uno using the Arduino IDE (Integrated Development Environment).

**1. Set Up Libraries**

To interact with the LCD, the program first needs to include the appropriate library for handling the LCD screen. In this case, we use the LiquidCrystal library, which allows for easy manipulation of the LCD screen.

**2. Define Pin Connections**

The next step is to define the input and output pins for each of the components (flame sensor, LED, buzzer, push button, and LCD). This is where you set up which Arduino pins will control or read each component.

**3. Initialize LCD**

The LCD is initialized in the setup() function. You define which pins are used for the LCD's control and data signals. The lcd.begin(16, 2) command initializes the 16x2 LCD (16 columns, 2 rows).

**4. Setup Function**

The setup() function is run once when the system starts. It’s used to initialize hardware components such as setting pin modes and initializing the LCD. Here, the push button is set to INPUT\_PULLUP mode, which means the button will be detected as LOW when pressed and will use the internal pull-up resistor.

**5. Loop Function**

The loop() function continuously executes the main logic of the program. This is where the fire detection and system control take place:

* **Read Flame Sensor**: The flame sensor is read using analogRead(), which converts the sensor’s analog voltage to a value between 0 and 1023.
* **Flame Detection**: If the sensor value exceeds a certain threshold (e.g., flameValue > 500), it indicates that fire is detected. In this case, the LED is turned on, and the buzzer is triggered. The LCD displays "Fire Detected!".
* **No Fire Detection**: If the sensor value is below the threshold, it indicates no fire, and both the LED and buzzer are turned off. The LCD displays "No Fire".
* **Button Reset**: If the push button is pressed, the system is reset. The LCD is cleared, and the message "System Reset" is displayed for 1 second.

**6. Threshold Calibration**

* The threshold value for fire detection (flameValue > 500) can be adjusted depending on the sensitivity of the flame sensor. This can be fine-tuned to suit different fire detection scenarios or environments.
* The sensor may provide different values based on ambient light conditions, so calibration is essential for accurate detection.

**Possible Enhancements:**

1. Multiple Sensors: Add more flame sensors for broader coverage.
2. Gas and Smoke Detection: Integrate gas or smoke sensors to detect hazardous conditions.
3. Temperature Sensing: Add temperature monitoring for an additional layer of detection.
4. Wireless Alerts: Use Wi-Fi or Bluetooth for remote alerts and notifications.
5. Data Logging: Track sensor data over time using an SD card.
6. Enhanced UI: Upgrade to a touchscreen or advanced display for better interaction.
7. Cloud Integration: Monitor system status remotely via cloud services like ThingSpeak or Blynk.
8. Voice Alerts: Use a sound module to provide voice alerts for more informative warnings.
9. Battery Backup: Ensure continuous operation during power outages with a battery backup.
10. Automatic Suppression: Integrate automatic fire suppression systems for advanced applications.
11. Automatic Calibration: Implement dynamic calibration for sensors.
12. Mobile App: Develop a mobile app for real-time monitoring and alerts.

Project Link:- https://wokwi.com/projects/417728180462924801