Subject: Micro-Processor

Fire And Gas Alarm System based on the ATmega32 microcontroller

Submitted to

Professor Dr: Mohamed Torad

Submitted By

	Pep		HW	Simasm			B		
1	2.5	Mostafa Ammar	2.5	2.5	0	20200216	1	✓	8.5
2	2.5	Mohamed Ahmed	2.5	2.5	2.5	20200484	1	✓	11
3	2.5	Amen Ahmed	2.5	2.5	2	20220063	1	✓	10.5
4	2.5	Salem Yasser	2.5	2.5	2	20220126	1	✓	10.5
5	2.5	Zeyad Mohamed	2.5	2.5	2	20220192	2	✓	11.5
6	2.5	Ahmed Yasser	2.5	2.5	2.5	20233055	2	✓	12

12.2025

Executive Summary

This project implements a Fire Alarm System based on the ATmega32 microcontroller intended as an educational embedded-systems exercise.

The system continuously monitors for fire-related signatures (smoke and flame) using dedicated sensors, displays system status on an LCD, and visual alarms (LEDs) when hazardous conditions are detected. The design was modeled and validated in Proteus and documented with wiring diagrams suitable for Fritzing and physical assembly.

The project teaches microcontroller programming, sensor interfacing, real-time monitoring, and system simulation.

Table of Contents

1 Introduction

Early fire detection plays a critical role in preventing loss of life and property, and modern safety engineering increasingly relies on embedded systems to achieve fast, reliable, and low-cost monitoring. This project presents the design of a microcontroller-based fire alarm system using the **ATmega32**, integrating both a **smoke sensor (MQ-2)** and an **infrared flame detector** to provide multi-criteria detection. Using more than one sensing method enhances system robustness, since smoke sensors respond to combustion gases while flame sensors detect the characteristic infrared emissions of open flames.

The ATmega32 microcontroller coordinates sensor sampling through its 10-bit ADC, processes detection logic using a state-based algorithm, and manages user interface outputs such as the LCD, LEDs.
Overall, this project demonstrates how embedded systems can be used to build an efficient and extendable fire detection platform. The design methodology, simulation, and testing presented in this report follow engineering best practices and align with general principles emphasized in modern fire safety standards.

2 Objectives

The goal of this project is to design a reliable and low-cost fire detection system using the ATmega32 microcontroller. The main objectives are:

1. Fire Detection and Monitoring
Integrate a smoke sensor (MQ-2) and an IR flame sensor to achieve multi-criteria detection.
Calibrate sensor thresholds and apply simple filtering to ensure stable and accurate readings.

2. Embedded System Design
Use the ATmega32 to acquire sensor data, process detection logic, and control all output devices.
Implement a clear state-machine structure (NORMAL, WARNING) for consistent system behavior.

3. User Interface and Alarm Signaling
Provide real-time feedback through a 16×2 LCD showing system status and alarm messages.
Use and LEDs to deliver visual alerts until reset.

4. Hardware and Circuit Development
Create a complete wiring design including power regulation, sensor interfacing, and transistor-driven alarm outputs.
Apply proper grounding and decoupling practices to ensure reliable ADC performance.

5. Simulation and Testing
Validate the system using Proteus simulation to test logic, timing, and sensor responses.
Evaluate the prototype's detection accuracy, response time, and overall reliability.

3 List of Component

Table 3-1:List of Components

Quantity	Component	Function / Notes
1	ATmega32 (40-pin DIP)	Microcontroller (core controller)
1	16×2 Character LCD (HD44780)	Display system state & messages
1	MQ-2 Smoke Gas Sensor	Smoke / combustible gas detection (analog)
1–2	IR Flame Sensor	Fast flame detection (analog output)
2	LEDs (Red/Green)	Visual alarm & system status
1	8 MHz Crystal + caps	Clock source for ATmega32
1	7805 regulator or 5V supply	Stable 5 V power supply
Several	Resistors, Capacitors, Push buttons, Diodes	Pull-ups, debouncing, power decoupling
1	16×2 & wiring for Proteus/Fritzing	For simulation & schematic diagrams

4 Methodology

The project was executed in four structured phases: design, development, implementation, and evaluation — mirroring the template workflow.

4.1 Design Phase

Requirement analysis: Define detection thresholds, alarm logic, user interface messages, and desired responses (immediate alarm, latch vs. auto-reset options).
Schematic creation: Draw schematic in Proteus showing ATmega32, sensor inputs (analog and/or digital), LCD interface (4-bit data mode), buzzer driver (transistor), and LEDs.
Pin mapping: Allocate MCU pins for sensors, LCD (RS, RW, EN, D4–D7), buzzer (through transistor), and LEDs. A clear pin map was prepared for both Proteus and Fritzing.

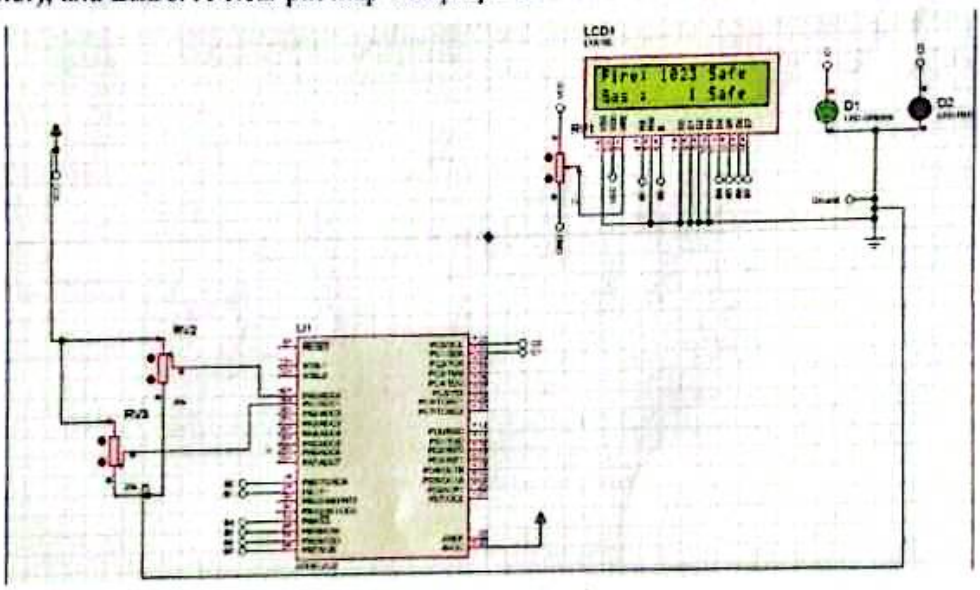

Figure 4.1-1 :Proteus Simulation

4.2 Development Phase

Firmware: Implemented in C (Atmel AVR/GCC or Microchip Studio). Key tasks:
ADC sampling for analog smoke sensor (if analog sensor used) with filtering and thresholding.
Digital interrupt or polling for flame sensor.
Debounce logic for any user buttons (e.g., silence/reset).
LCD driver routines (4-bit mode), status messages.
Alarm control (LED).
Simulation: Stepwise simulation in Proteus to validate sensor thresholds and user interactions.

4.3 Implementation Phase

Hardware assembly: Build prototype on breadboard or PCB using the prepared schematic and pin map. Use a regulated 5 V supply and decoupling capacitors near the MCU.
Programming: Flash the compiled .hex into ATmega32 via ISP (USBasp/AVRISP) and verify boot-up behavior.
Integration testing: Test each sensor channel, LCD updates, and alarm outputs

4.4 Evaluation Phase

Functional tests: Confirm that smoke/flame events reliably trigger alarms at designed thresholds.
Robustness tests: Test for false positives (ambient changes), evaluate sensor placement influence, and optimize thresholds.
Documentation: Produce wiring diagrams, pin maps, component datasheets, and simulation steps for reproducibility.

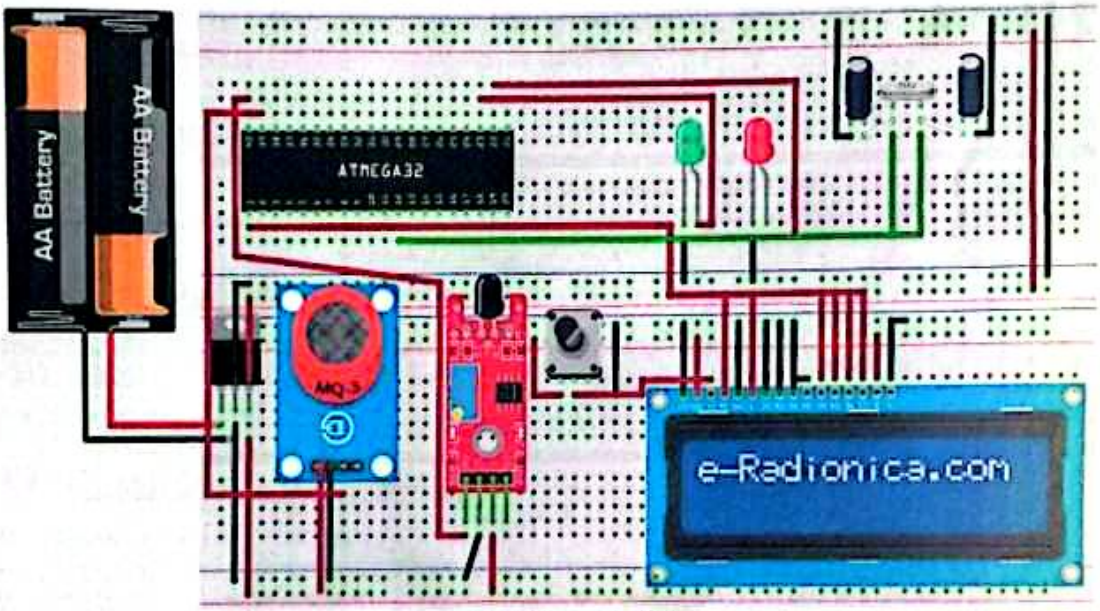

Figure 4.4-1:Fritzing Simulation

5 Results and Discussion

5.1 Sensors (Smoke & Flame)

Smoke sensor (MQ-series): Provides analog voltage proportional to gas concentration. With ADC sampling and a low-pass filter, stable readings were achieved. A threshold was chosen experimentally to balance sensitivity and false alarms.

Flame sensor (IR): Provides fast digital output on detecting specific flame frequencies. It is effective at short range and used for immediate detection.

5.2 Indicators

LEDs: Red LED indicates alarm; green LED indicates normal operation. Blink patterns implemented in firmware make alarm condition obvious even with muted buzzer.

5.3 LCD Interface

16×2character LCD used in 4-bit mode to reduce pin usage. Messages include "SYSTEM OK", "SMOKE DETECTED", "FLAME DETECTED". Contrast potentiometer included for readability.

5.4 Circuit Design & Pin Mapping

Pin allocation example (suggested):
PORTA: ADC inputs (smoke and flame sensors)
PORTB: LCD data lines (D4–D7) + control lines (RS, EN)
PORTC: Alarm outputs (LEDs)
Reset, Vcc, GND, and crystal connected per ATmega32 datasheet.
A full pin map and wiring diagram are provided in Appendix B.

5.5 Simulation (Proteus)

Proteus simulation validated:
ADC reading and threshold crossing behavior.
LCD message updates on alarm transitions.
Buzzer and LED activation sequences.
Loading compiled .hex to the microcontroller and running timed tests.

ATmega32 Microcontroller

The ATmega32 microcontroller is the core of this digital clock project, responsible for managing the time, updating the LCD, and handling user inputs from the push buttons. Using Microchip Studio, we programmed the ATmega32 to keep accurate time and respond to button presses for setting and adjusting the clock.

40 pin dip ATMEGA32 chip with 8 MHz crystal oscillator and Push button reset switch.

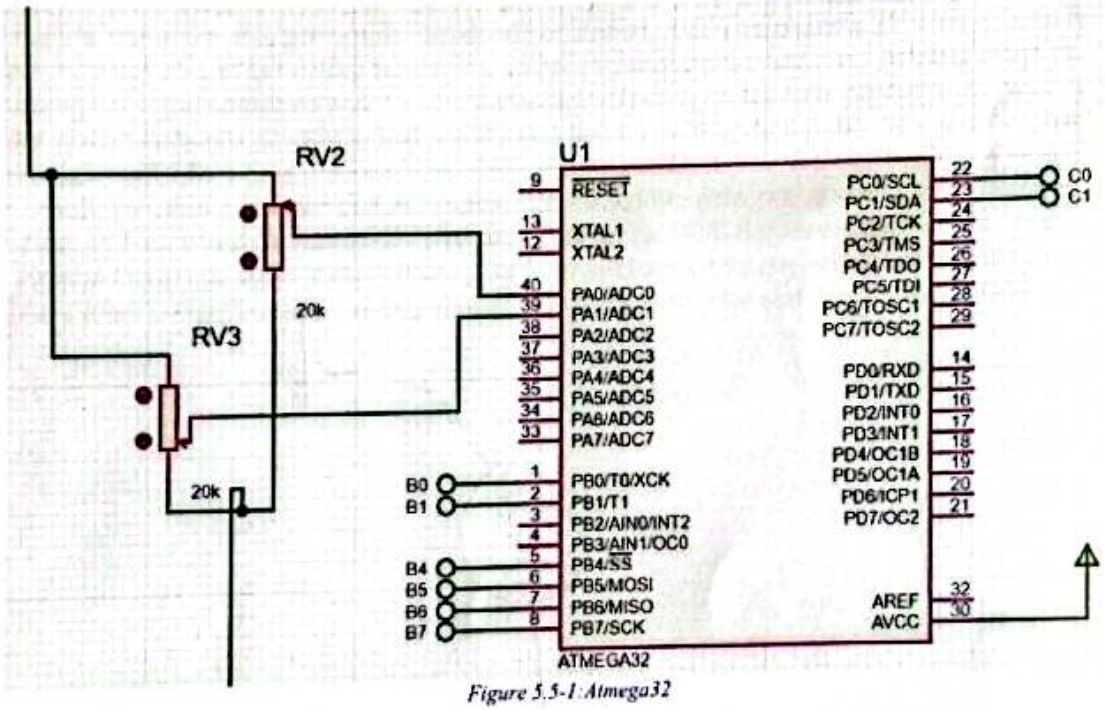

Figure 5.5-1 Atmega32

Observations: The microcontroller performed effectively, maintaining accurate time and executing programmed instructions reliably.

Challenges: The initial programming process required careful attention to timing functions to ensure the clock remained accurate. We used internal timers within the ATmega32 to achieve precise timekeeping.

6 Conclusion

This project successfully demonstrates the design and implementation of a compact, reliable, and low-cost fire detection system using the ATmega32 microcontroller. By integrating both a smoke sensor (MQ-2) and an IR flame sensor, the system benefits from multi-criteria detection, which improves accuracy and reduces the chances of false alarms compared to single-sensor solutions. The use of ADC sampling, threshold-based decision making, and a structured state machine allowed the system to operate consistently and react appropriately to different fire-related conditions.

The hardware design, including regulated power, proper decoupling, and transistor-driven outputs, ensured stable performance and safe operation of all components. Proteus simulation played a key role in validating the system before physical implementation, helping to verify timing behavior, sensor responses, and alarm logic. Testing on the final prototype showed that the system responds quickly to both smoke and flame events and provides clear user feedback through the LCD, LEDs.

Overall, the results indicate that microcontroller-based fire detection systems can provide an effective foundation for more advanced safety solutions. With further enhancements—such as wireless communication, battery backup, or improved sensor fusion—the system could evolve into a more comprehensive fire-monitoring platform suitable for real-world applications.

7 ACKNOWLDGEMENT

We would like to express our sincere appreciation to **Professor Dr. Mohamed Torad** for his continuous guidance, valuable feedback, and support throughout the development of this project. His insights into embedded system design and engineering principles greatly contributed to improving the quality and depth of our work.

We also extend our gratitude to the laboratory supervisors and technical staff for providing the necessary equipment, components, and testing environment that enabled us to build and evaluate the system effectively. Their assistance in troubleshooting hardware issues and facilitating the simulation process was essential to completing this project successfully.

Finally, we would like to acknowledge the collaborative efforts of all team members. Each member played an important role in research, coding, circuit construction, documentation, and validation. This project would not have reached its full potential without the dedication and cooperation of the entire group.

8 References

ATmega32 Datasheet — Microchip/Atmel.
MQ-series Sensor Datasheet (MQ-2 / MQ-135) — Manufacturer.
IR Flame Sensor Module Datasheet.
HD44780 Character LCD Datasheet.
Proteus Design Suite documentation (simulation and hex loading).
(Datasheets and references used during design and selection are listed above and included in Appendix C.)

9 Appendices

```
/*
 * Microcontroller: ATmega32
 * Frequency: 8 MHz
 * Project: Fire & Gas Monitor with LED Alerts
 */

#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>

// --- LCD Pin Definitions (PORTB) ---
#define LCD_Dir  DDRB
#define LCD_Port PORTB
#define RS PB1
#define EN PB0

// --- LED Pin Definitions (PORTC) ---
#define LED_Dir  DDRC
#define LED_Port PORTC
#define RED_LED  PC0
#define GREEN_LED PC1

// --- Function Prototypes ---
void ADC_Init();
uint16_t ADC_Read(uint8_t channel);
void LCD_Init();
void LCD_Command(unsigned char cmnd);
void LCD_Char(unsigned char data);
void LCD_String(char *str);
void LCD_Set_Cursor(unsigned char row, unsigned char col);
void LCD_Pulse();

int main(void)
{
    uint16_t fire_val, gas_val;
    char buffer[16];

    DDRA = 0x00;
    LCD_Dir = 0xF3;

    LED_Dir |= (1 << RED_LED) | (1 << GREEN_LED) | (1 << PC2);

    ADC_Init();
    LCD_Init();

    LCD_String("System Ready");
    _delay_ms(1000);
    LCD_Command(0x01);

    while(1)
    {
        fire_val = ADC_Read(0);
        gas_val  = ADC_Read(1);

        if (fire_val < 700 || gas_val > 40)
```

```
        {
            // --- DANGER MODE ---
            LED_Port |= (1 << RED_LED);
                    LED_Port |= (1 << PC2);
            LED_Port &= ~(1 << GREEN_LED);

            LCD_Command(0x01); // Clear Screen to show alert clearly

            // Check specifically which one triggered the alarm
            if (fire_val < 700) {
                LCD_Set_Cursor(1, 1);
                LCD_String("FIRE DETECTED!");
            }

            if (gas_val > 40) {
                LCD_Set_Cursor(2, 1);
                LCD_String("GAS DETECTED!");
            }
        }
        else
        {
            // --- SAFE MODE ---
                    LED_Port &= ~(1<<PC2);
            LED_Port &= ~(1 << RED_LED);
            LED_Port |= (1 << GREEN_LED);

            LCD_Command(0x01); // Clear Screen

            LCD_Set_Cursor(1, 1);
            sprintf(buffer, "Fire: %4u Safe", fire_val);
            LCD_String(buffer);

            LCD_Set_Cursor(2, 1);
            sprintf(buffer, "Gas : %4u Safe", gas_val);
            LCD_String(buffer);
        }

        _delay_ms(300); // Update speed
    }
}

void ADC_Init()
{
    ADMUX = (1 << REFS0); // AVCC as ref
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // Enable,
Prescaler 128
}

uint16_t ADC_Read(uint8_t channel)
{
    ADMUX = (ADMUX & 0xF8) | (channel & 0x07);
    ADCSRA |= (1 << ADSC);
    while (ADCSRA & (1 << ADSC));
    return ADC;
}

// --- LCD Functions ---
void LCD_Pulse()
{
    LCD_Port |= (1 << EN);
```

```
    _delay_us(1);
    LCD_Port &= ~(1 << EN);
    _delay_us(100);
}

void LCD_Send_Nibble(unsigned char data, unsigned char type)
{
    if(type) LCD_Port |= (1 << RS);
    else     LCD_Port &= ~(1 << RS);

    LCD_Port = (LCD_Port & 0x0F) | (data & 0xF0);
    LCD_Pulse();
}

void LCD_Command(unsigned char cmnd)
{
    LCD_Send_Nibble(cmnd, 0);
    LCD_Send_Nibble(cmnd << 4, 0);
    _delay_ms(2);
}

void LCD_Char(unsigned char data)
{
    LCD_Send_Nibble(data, 1);
    LCD_Send_Nibble(data << 4, 1);
    _delay_us(50);
}

void LCD_Init()
{
    _delay_ms(20);
    LCD_Send_Nibble(0x30, 0);
    _delay_ms(5);
    LCD_Send_Nibble(0x30, 0);
    _delay_us(200);
    LCD_Send_Nibble(0x30, 0);
    LCD_Send_Nibble(0x20, 0);
    LCD_Command(0x28);
    LCD_Command(0x0C);
    LCD_Command(0x06);
    LCD_Command(0x01);
}

void LCD_String(char *str)
{
    int i;
    for(i=0; str[i]!=0; i++) LCD_Char(str[i]);
}

void LCD_Set_Cursor(unsigned char row, unsigned char col)
{
    unsigned char cmd;
    if(row == 1) cmd = 0x80 + (col - 1);
    else         cmd = 0xC0 + (col - 1);
    LCD_Command(cmd);
}
```

10 Assembly Code

```
;---------------------------------------------------
; Fire & Gas Monitor - Simple Assembly
; Processor: ATmega32
; Frequency: 8 MHz
;---------------------------------------------------

.include "m32def.inc"

; --- Register Definitions ---
.def temp = r16
.def data = r17
.def adcl_reg = r18
.def adch_reg = r19

; --- Constants ---
.equ THRESHOLD_FIRE_LOW  = 0xBC  ; 700 (Low Byte)
.equ THRESHOLD_FIRE_HIGH = 0x02  ; 700 (High Byte)
.equ THRESHOLD_GAS_LOW   = 0x28  ; 40  (Low Byte)

; --- Code Segment ---
.cseg
.org 0x0000
   rjmp RESET

; --- Interrupt Vectors ---
.org 0x0040 ; Past interrupt table

RESET:
   ; 1. Initialize Stack Pointer (REQUIRED)
   ldi temp, low(RAMEND)
   out SPL, temp
   ldi temp, high(RAMEND)
   out SPH, temp

   ; 2. Port Setup
   ; Port A = Input (ADC)
   ldi temp, 0x00
   out DDRA, temp

   ; Port B = Output (LCD: PB0, PB1, PB4-7)
   ldi temp, 0xFF
   out DDRB, temp

   ; Port C = Output (LEDs: PC0, PC1, PC2)
   ldi temp, 0xFF
   out DDRC, temp

   ; 3. Initialize Modules
```

```
    rcall ADC_Init
    rcall LCD_Init

    ; Initial Screen
    rcall LCD_Clear
    ldi ZL, low(MSG_READY*2)
    ldi ZH, high(MSG_READY*2)
    rcall LCD_Send_String

    rcall Delay_1s

MAIN_LOOP:
    ; --- READ FIRE SENSOR (ADC 0) ---
    ldi temp, 0     ; Channel 0
    rcall ADC_Read
    ; Result is in r19:r18 (High:Low)

    ; Check if Fire < 700
    ; Compare Low Byte
    cpi r18, THRESHOLD_FIRE_LOW
    ldi temp, THRESHOLD_FIRE_HIGH
    cpc r19, temp
    brlo ALARM_FIRE  ; Branch if Lower (Fire Detected)

    ; --- READ GAS SENSOR (ADC 1) ---
    ldi temp, 1     ; Channel 1
    rcall ADC_Read
    ; Result is in r19:r18

    ; Check if Gas > 40
    ; Compare (Note: Gas uses only low byte check mostly as 40 is small)
    cpi r18, THRESHOLD_GAS_LOW
    ldi temp, 0
    cpc r19, temp
    brsh ALARM_GAS   ; Branch if Same or Higher (Gas Detected)

    ; --- SAFE MODE ---
    rcall SYSTEM_SAFE
    rjmp MAIN_LOOP

; --- ALARM ROUTINES ---

ALARM_FIRE:
    ; Red LED ON, Buzzer ON, Green OFF
    sbi PORTC, 0    ; RED ON
    sbi PORTC, 2    ; BUZZER ON
    cbi PORTC, 1    ; GREEN OFF

    rcall LCD_Clear
    ldi ZL, low(MSG_FIRE*2)
```

```
    ldi ZH, high(MSG_FIRE*2)
    rcall LCD_Send_String
    rcall Delay_Long
    rjmp MAIN_LOOP

ALARM_GAS:
    ; Red LED ON, Buzzer ON, Green OFF
    sbi PORTC, 0    ; RED ON
    cbi PORTC, 1    ; GREEN OFF

    rcall LCD_Clear
    ldi ZL, low(MSG_GAS*2)
    ldi ZH, high(MSG_GAS*2)
    rcall LCD_Send_String
    rcall Delay_Long
    rjmp MAIN_LOOP

SYSTEM_SAFE:
    ; Green LED ON, Red/Buzzer OFF
    cbi PORTC, 0    ; RED OFF
    sbi PORTC, 1    ; GREEN ON

    rcall LCD_Clear
    ldi ZL, low(MSG_SAFE*2)
    ldi ZH, high(MSG_SAFE*2)
    rcall LCD_Send_String
    rcall Delay_Long
    ret

; ---------------------------------------
; SUBROUTINES
; ---------------------------------------

; --- ADC Init ---
ADC_Init:
    ; AVCC with external capacitor at AREF pin (REFS0 = 1)
    ldi temp, (1<<REFS0)
    out ADMUX, temp
    ; ADC Enable, Prescaler 128 (ADEN=1, ADPS=111)
    ldi temp, (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
    out ADCSRA, temp
    ret

; --- ADC Read ---
; Input: temp (channel number 0-7)
; Output: r19:r18 (10-bit result)
ADC_Read:
    ; Mask channel to 0-7
    andi temp, 0x07
    ; Move current REFS0 bit to temp2
```

```
    ldi r20, (1<<REFS0)
    or temp, r20
    out ADMUX, temp

    ; Start Conversion
    sbi ADCSRA, ADSC

Wait_ADC:
    ; Wait for ADSC to clear
    sbic ADCSRA, ADSC
    rjmp Wait_ADC

    ; Read Result (Low first, then High)
    in r18, ADCL
    in r19, ADCH
    ret

; --- LCD Init (4-bit mode) ---
LCD_Init:
    rcall Delay_Long
    ldi temp, 0x02  ; Command to Init 4-bit
    out PORTB, temp
    sbi PORTB, 0    ; Enable Pulse
    cbi PORTB, 0
    rcall Delay_Short

    ldi data, 0x28  ; 2 lines, 5x7 matrix
    rcall LCD_Cmd
    ldi data, 0x0C  ; Display ON, Cursor OFF
    rcall LCD_Cmd
    ldi data, 0x06  ; Increment Cursor
    rcall LCD_Cmd
    ldi data, 0x01  ; Clear Display
    rcall LCD_Cmd
    ret

; --- LCD Send Command ---
LCD_Cmd:
    cbi PORTB, 1    ; RS = 0 (Command)
    rjmp LCD_Write

; --- LCD Send Character ---
LCD_Char:
    sbi PORTB, 1    ; RS = 1 (Data)
    rjmp LCD_Write

; --- LCD Write Logic (Handles Nibbles) ---
LCD_Write:
    mov r20, data     ; Save data
    andi data, 0xF0   ; Get upper nibble
```

```
    in r21, PORTB     ; Read current port
    andi r21, 0x0F    ; Keep control bits
    or data, r21      ; Combine
    out PORTB, data
    sbi PORTB, 0      ; Pulse EN
    nop
    cbi PORTB, 0

    rcall Delay_Short

    mov data, r20     ; Restore data
    swap data         ; Swap nibbles
    andi data, 0xF0   ; Get lower nibble (now in upper pos)
    in r21, PORTB
    andi r21, 0x0F
    or data, r21
    out PORTB, data
    sbi PORTB, 0      ; Pulse EN
    nop
    cbi PORTB, 0
    rcall Delay_Short
    ret

; --- LCD Clear ---
LCD_Clear:
    ldi data, 0x01
    rcall LCD_Cmd
    rcall Delay_Long
    ret

; --- LCD String (Data in Z Pointer) ---
LCD_Send_String:
    lpm data, Z+
    cpi data, 0
    breq String_End
    rcall LCD_Char
    rjmp LCD_Send_String
String_End:
    ret

; --- Delays ---
Delay_Short:
    ldi r20, 200
L1: dec r20
    brne L1
    ret

Delay_Long: ; Approx 20ms
    ldi r20, 255
    ldi r21, 100
```

```
L2: dec r20
    brne L2
    dec r21
    brne L2
    ret

Delay_1s:   ; Very long delay
    ldi r22, 50
L3: rcall Delay_Long
    dec r22
    brne L3
    ret

; --- Data Messages (Stored in Program Memory) ---
MSG_READY: .db "System Ready", 0
MSG_SAFE:  .db "System Safe", 0
MSG_FIRE:  .db "FIRE DETECTED!", 0
MSG_GAS:   .db "GAS DETECTED!", 0
```