Gas and Smoke Detector

AUTOR: Buduran Cătălina-Andreea, 333CA

Introducere

Prezentarea pe scurt a proiectului:

- Ce face?

Proiectul este un detector de gaz/fum și flacără care monitorizează calitatea aerului într-un spațiu închis și declanșează avertizări sonore și vizuale în cazul detectării unor niveluri periculoase de fum sau gaze inflamabile, transmițând starea atmosferei spre un database hostat pe PC.

- Care este scopul lui?

Scopul proiectului este prevenirea accidentelor casnice prin detectarea timpurie a scurgerilor de gaz sau a începuturilor de incendii, oferind un sistem de alertă rapid și accesibil.

- Care a fost ideea de la care aţi pornit?

Ideea de la care am pornit este necesitatea de a avea un sistem de siguranță accesibil pentru locuințe, în special în apartamente sau case care nu sunt dotate cu senzori de fum/gaz integrați, reducând pericolele.

- De ce credeţi că este util pentru alţii şi pentru voi?

Este util pentru alții, deoarece crește siguranța în locuințe, oferind un mijloc ieftin de prevenire a tragediilor cauzate de scurgeri de gaz sau incendii. Pentru noi, proiectul este o oportunitate de a pune în practică cunoștințele hardware și software dobândite, aplicate într-un context real și relevant.

Descriere generală

Componentele detectorului:

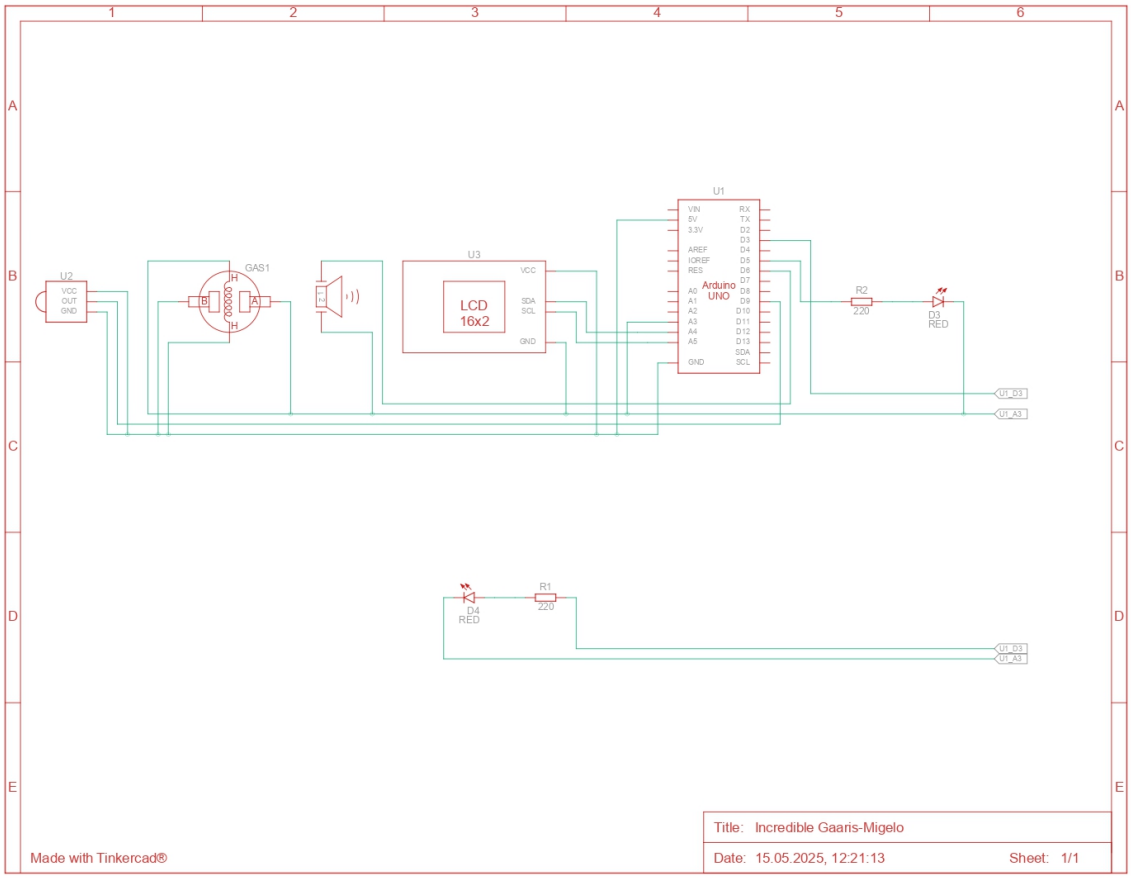
- Senzor analogic de gaz/fum MQ-2
- Senzor de flacără
- Buzzer – emite avertizări sonore când e detectat gaz/fum/flacără
- LCD display – afișează mesaje relevante: „Normal”, „Fum detectat”, „Gaz detectat”
- Cablu USB – pentru trimiterea datelor în laptop (UART)
- LED-uri de culoare roșu și verde – roșu se aprinde la detecție, verde când aerul este curat
- 2 Rezistențe 220 kOhm pentru cele 2 Led-uri

Schemă Bloc:

Modul în care interacționează componentele:

- Senzor MQ-2 (gaz/fum) - trimite către Arduino un semnal analogic proporțional cu concentrația de gaz/fum;
- Senzor de flacără (cu 4 pini) - trimite semnal analogic sau digital dacă detectează flacără în apropiere;
- Arduino UNO - centrul de control, primește semnale de la senzori, decide în funcție de praguri și trimite comenzi către buzzer, LED-uri și LCD. Comunică și cu laptopul;
- LCD 16×2 cu I2C - afișează mesaje relevante, în funcție de senzorii activați;
- Buzzer pasiv - activat de Arduino prin semnal PWM dacă este detectat gaz/fum/flacără, pentru a emite un semnal sonor de avertizare;
- LED roșu / LED verde - LED-ul roșu se aprinde când e detectat pericol, cel verde când nu este detectat nimic;
- Laptop - primește date de la Arduino prin UART (prin cablu USB)

Schemă electrică:



Descriere detaliată și justificată a pinilor:

- MQ-2 (senzor de gaz și fum)

1. **Pin utilizat:** A1
2. **Tip pin:** Analog IN

3. **Justificare:** Modulul MQ-2 are ieșire AO (analog out), oferă tensiune proporțională cu concentrația gazului sau fumului.

- Senzor de flacără

1. **Pin utilizat:** D9
2. **Tip pin:** Digital IN
3. **Justificare:** Senzorul are ieșire digitală (DO), care devine LOW când detectează flacără. Se citește cu digitalRead(D9).

- Buzzer pasiv

1. **Pin utilizat:** D6
2. **Tip pin:** Digital OUT
3. **Justificare:** Fiind un buzzer pasiv, trebuie controlat cu semnal de frecvență(PWM).

- LED roșu (pericol)

1. **Pin utilizat:** D8
2. **Tip pin:** Digital OUT
3. **Justificare:** Aprins când se detectează flacără, gaz sau fum.

- LED verde (stare normală)

1. **Pin utilizat:** D11
2. **Tip pin:** Digital OUT (are și PWM, dar nu e necesar aici)\
3. **Justificare:** Se aprinde doar când nu e niciun pericol. Pentru a folosi conceptul de PWM o să afișez cu diferite intensități culorile.

- LCD 16×2 cu I2C

1. **Pini utilizați:** A4 (SDA), A5 (SCL)
2. **Tip pini:** Interfață I2C dedicată
3. **Justificare:** Pe Arduino UNO, pinii A4 și A5 sunt cei dedicați comunicației I2C. M-am documentat și acestea sunt recomandate pentru a nu avea un Wire.begin(sda, scl) forțat.

- Alimentare

1. **5V:** toate componentele sunt compatibile cu 5V și pot fi alimentate direct de la Arduino.
2. **GND:** fiecare componentă e conectată la masă comună, foarte important pentru funcționarea stabilă.

Hardware Design

- Listă de piese:

Componentă	Funcție	Observații	Link achiziție	Datasheet
Senzor MQ-2	Detectare gaz/fum	Ieșire analogică	Optimus MQ-2	MQ-2 PDF
Senzor flacără	Detectare flacără	Ieșire analogică/digitală	Optimus Flacără	Senzor flacără
LCD 16×2 I2C	Afișare mesaje	Comunicare I2C	LCD I2C	LCD
Buzzer pasiv	Alertă sonoră	Controlat prin PWM	Buzzer pasiv	Buzzer
LED roșu	Aprins la detecție	Pin digital	LED roșu	LED Roșu

LED verde	Aprins când aerul e curat	Pin digital	LED verde	LED Verde
Cablu USB	Transmitere date către laptop (UART)	Comunicare serială	Cablu USB	Cablu USB
Arduino UNO	Microcontroler – control principal	Comunică cu toate componentele (senzori, buzzer, LED-uri, LCD) prin pini digitali, analogici și I2C. Rulează codul principal.	Arduino UNO	Arduino UNO
Breadboard	Platformă de conectare fără lipire	Permite conectarea rapidă a componentelor. Distribuie 5V și GND către toată rețeaua	Breadboard	Breadboard
Fire tată-tată	Conectare între componente și Arduino	Asigură conexiuni electrice fără lipire. Leagă pinii Arduino de senzori, buzzer, LED-uri	Fire tată-tată	-

Software Design

Proiectul este funcțional și implementează următoarele:

- Detectarea flăcării (senzor digital pe D9)
- Detectarea gazului/fumului (senzor analogic MQ-2 pe A1)
- Reacție vizuală și sonoră: LED-uri și buzzer
- Transmiterea datelor senzorilor prin UART (Serial) către laptop
- Toate funcționalitățile sunt scrise cu acces direct la registre (PORTx, DDRx, ADC, UART, PWM)

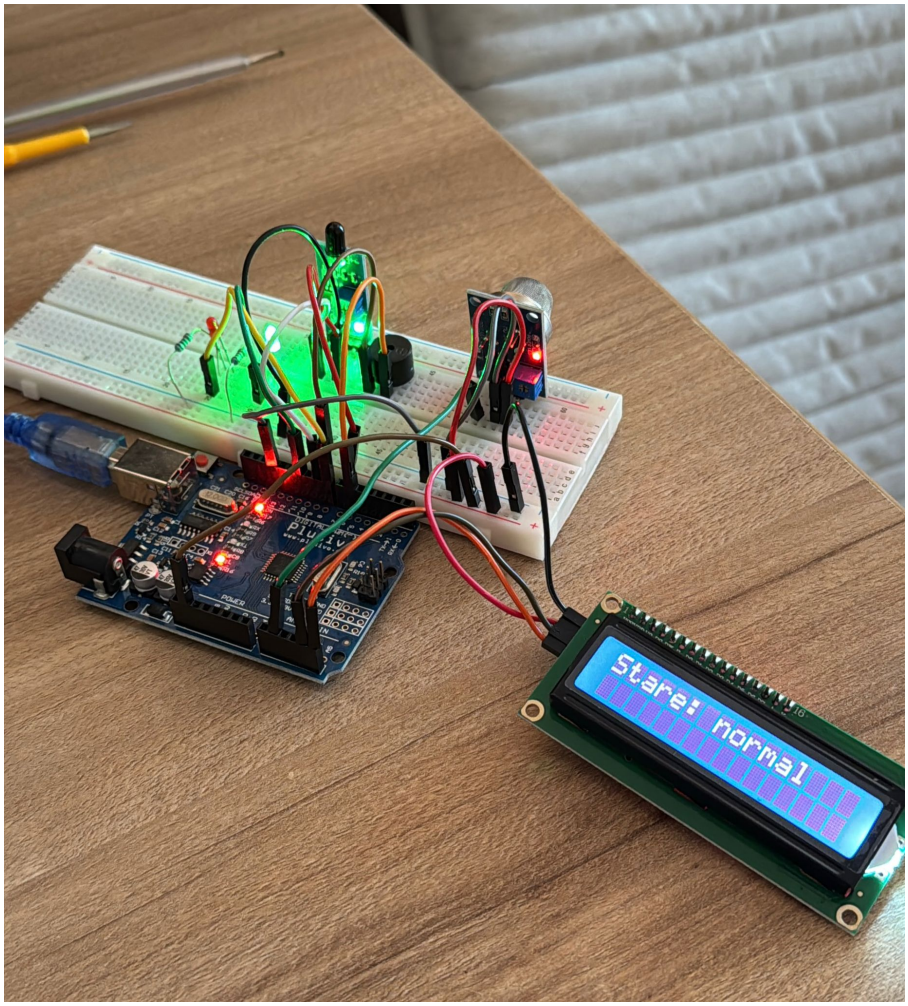
Proiectul meu nu folosește biblioteci externe Arduino. Toate funcționalitățile (PWM, UART, ADC, GPIO) au fost implementate folosind registre directe ale microcontrolerului ATmega328P, deoarece reflectă înțelegerea profundă a hardware-ului, dar și corelare directă cu conceptele din laborator. Proiectul este scris exclusiv cu registre AVR, fără funcții Arduino. Toate componentele hardware sunt controlate la nivel de low-level, iar proiectul este calibrat și funcțional pe o placă compatibilă Arduino.

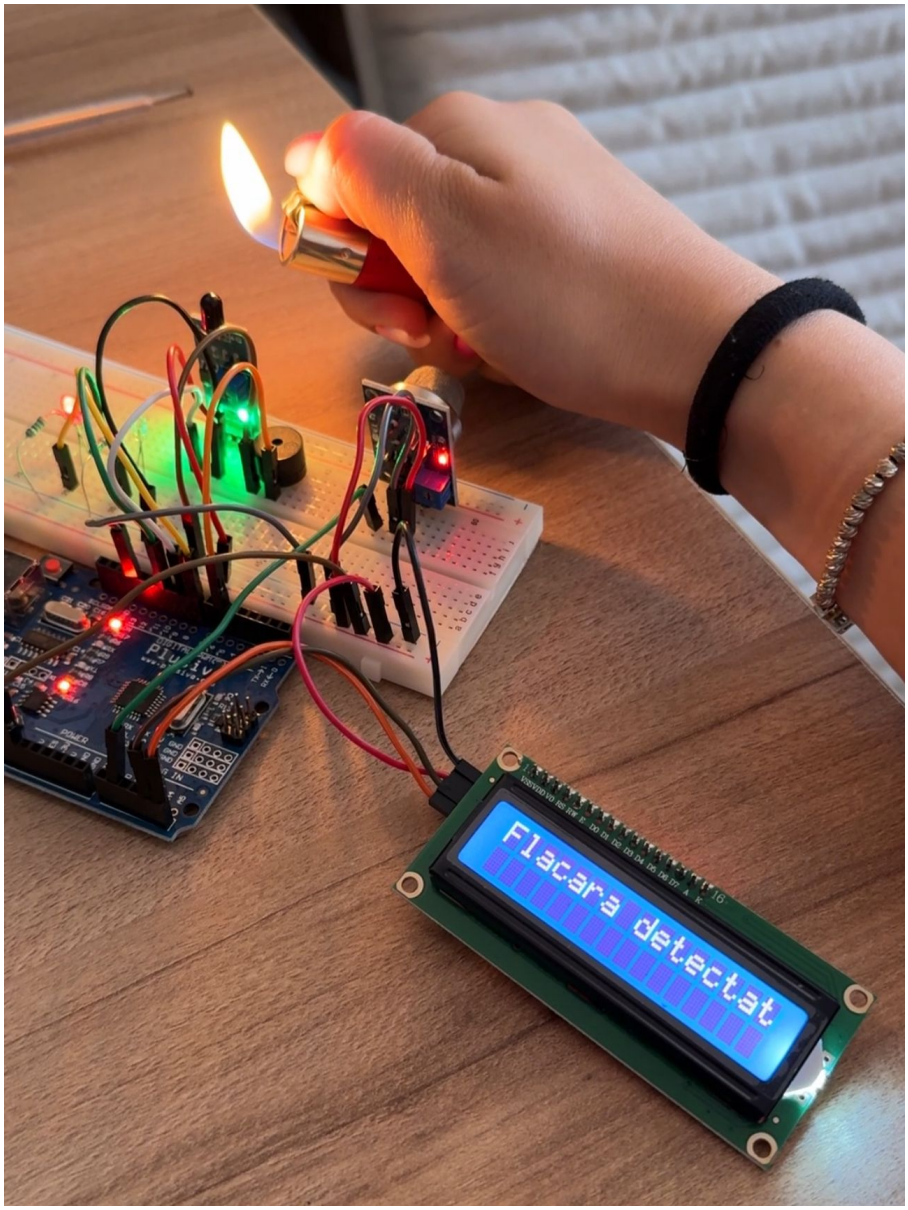
Calibrarea senzorilor

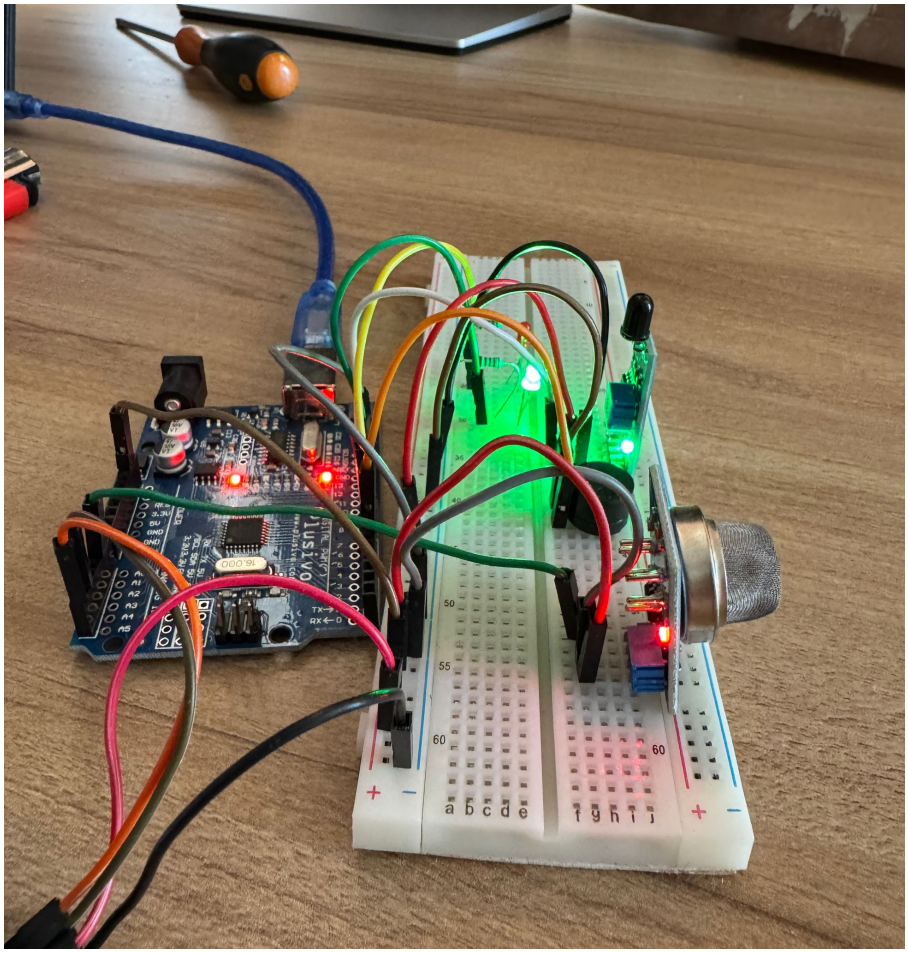
- Pentru gaz: am stabilit empiric un prag de detecție la > 300 (valoare ADC), testând cu bricheta sau fum de chibrit.
- Pentru flacără: senzorul returnează LOW la detecție, iar pragul este binar (nu necesită calibrare).
- Calibrarea s-a realizat comparând valorile UART în condiții de aer curat vs. poluat.

Optimizări realizate

- Folosirea directă a registrelor elimină funcții costisitoare (ex: analogRead(), tone()), reducând latențele
- PWM activat doar la nevoie, apoi complet dezactivat pentru a evita zgomot rezidual
- UART rulează asincron, neblocant pentru restul logicii
- Bucla principală are un delay controlat pentru a nu satura UART-ul și pentru a permite actualizări corecte


Stare: normal


Flacara detectat



Laborator	Temă	Funcționalitate folosită în proiect
Laboratorul 0	GPIO	Control LED roșu (D8) și LED verde (D11), citire senzor flacără (D9) prin `PINB`
Laboratorul 1	UART	Trimitere valori citite de la senzorii de gaz și flacără către laptop prin UART
Laboratorul 3	Timere și PWM	Generare sunet buzzer prin **Timer0** în modul **Fast PWM** pe pinul **PD6** (`TCCR0A`, `OCR0A`)
Laboratorul 4	ADC	Citire analogică de la senzorul MQ-2 (gaz/fum) prin canalul
Laboratorul 6	I2C	LCD I2C

Explicarea codului:

Componentă	Funcție din cod	
Buzzer (PWM)	`pwm_start()` și `pwm_stop()`	Pornește/oprește semnalul sonor generat pe pinul D6 prin Timer0 (Fast PWM)
Senzor gaz (ADC)	`adc_init()` și `citeste_adc()`	Inițializează convertorul analog-digital și citește tensiunea de la MQ-2

Serial (UART)	`uart_init()`,`uart_transmit()`,`uart_print()`,`uart_print_number()`	Trimite mesaje către laptop prin USB pentru a vizualiza starea senzorilor
LED-uri și senzori	în `main()`	LED verde/roșu indică stare normală/pericol. Senzor flacără (digital pe D9)

Interacțiunea dintre funcționalități

- Senzorul de flacără (D9 - PB1): citit ca digital [1] == 0) → dacă flacără detectată, devine true.
- Senzorul de gaz (A1 - ADC1): se citește analog cu funcția citeste_adc() → dacă valoarea depășește pragul 300, se consideră pericol.
- LED-urile: LED roșu (D8 - PB0) aprins dacă există pericol (gaz sau flacără), LED verde (D11 - PB3) aprins în mod normal
- Buzzerul: e pornește doar dacă e detectat pericol → se activează PWM prin pwm_start(), care setează Timer0 în Fast PWM cu frecvență de 2kHz
- UART: trimite în fiecare ciclu starea citită: “Gaz: 364 | Flacara: 1” către laptop

Validarea funcționării

- LED-uri: verificate vizual în funcție de condiții (roșu la pericol, verde normal)
- Buzzer: testat auditiv că pornește doar când e gaz sau flacără, și se oprește complet altfel
- UART: testat în laptop, dacă laptopul primește date
- Senzor gaz: testat cu o brichetă – valorile cresc în UART, buzzer și LED roșu se activează
- Senzor flacără: testat apropiind flacără - detectare instantanee

#include <avr/io.h> #include <util/delay.h> #include “twi.h”

#define LCD_ADDR 0x27 #define LCD_BACKLIGHT 0x08 #define LCD_ENABLE 0x04 #define LCD_RS 0x01

void delay_ms_custom(uint16_t ms) {

```
  for (uint16_t i = 0; i < ms; i++) {
      TCNT2 = 0;
      TIFR2 |= (1 << OCF2A);
```

```
      while (!(TIFR2 & (1 << OCF2A))) {
          //astept (1ms)
      }
  }
```

}

void lcd_send_nibble(uint8_t nibble, uint8_t rs) {

```
  uint8_t data = (nibble & 0xF0) | LCD_BACKLIGHT;
  if (rs)
```

```
    data |= LCD_RS;
```

```
  twi_start();
  twi_write(LCD_ADDR << 1);
  twi_write(data | LCD_ENABLE);
  _delay_us(1);
  twi_write(data & ~LCD_ENABLE);
  twi_stop();
```

```
}
```

void lcd_send_byte(uint8_t data, uint8_t rs) {

```
  lcd_send_nibble(data & 0xF0, rs);
  lcd_send_nibble((data << 4) & 0xF0, rs);
  delay_ms_custom(1);
```

```
}
```

void lcd_command(uint8_t cmd) {

```
  lcd_send_byte(cmd, 0);
```

```
}
```

void lcd_write_char(char c) {

```
  lcd_send_byte(c, 1);
```

```
}
```

void lcd_print(const char* str) {

```
  while (*str) lcd_write_char(*str++);
```

```
}
```

void lcd_clear() {

```
  lcd_command(0x01);
  delay_ms_custom(2);
```

```
}
```

void lcd_set_cursor(uint8_t col, uint8_t row) {

```
  uint8_t row_offsets[] = {0x00, 0x40};
  lcd_command(0x80 | (col + row_offsets[row]));
```

```
}
```

void lcd_print_number(uint16_t val) {

```
  char buf[6];
  uint8_t i = 0;
```

```
  if (val == 0) {
      lcd_write_char('0');
      return;
  }
```

```
  while (val > 0 && i < 5) {
      buf[i++] = '0' + (val % 10);
      val /= 10;
  }
```

```
  while (i > 0) {
      lcd_write_char(buf[--i]);
  }
```

}

void lcd_init() {

```
  delay_ms_custom(50);
  lcd_send_nibble(0x30, 0); delay_ms_custom(5);
  lcd_send_nibble(0x30, 0); delay_ms_custom(150);
  lcd_send_nibble(0x30, 0); delay_ms_custom(150);
  lcd_send_nibble(0x20, 0); delay_ms_custom(150);
```

```
  lcd_command(0x28);
  lcd_command(0x0C);
  lcd_command(0x06);
  lcd_command(0x01);
  delay_ms_custom(2);
```

}

void pwm_start(uint8_t level) {

```
  DDRD |= (1 << PD6);
  TCCR0A = (1 << COM0A1) | (1 << WGM01) | (1 << WGM00);
  TCCR0B = (1 << CS01) | (1 << CS00);
  OCR0A = level;
```

}

void pwm_stop() {

```
  TCCR0A = 0;
  TCCR0B = 0;
  OCR0A = 0;
```

```
  PORTD &= ~(1 << PD6);

}

void adc_init() {

  ADMUX = (1 << REFS0) | (1 << MUX0);
  ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1);

}

uint16_t citeste_adc() {

  ADCSRA |= (1 << ADSC);
  while (ADCSRA & (1 << ADSC));
  return ADC;

}

void uart_init() {

  UBRR0 = 103;
  UCSR0B = (1 << TXEN0);
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);

}

void uart_transmit(char c) {

  while (!(UCSR0A & (1 << UDRE0)));
  UDR0 = c;

}

void uart_print(const char* s) {

  while (*s) uart_transmit(*s++);

}

void uart_print_number(uint16_t n) {

  char buffer[6];
  uint8_t i = 0;
  if (n == 0) {
      uart_transmit('0');
      return;
  }
  while (n > 0 && i < 5) {
      buffer[i++] = '0' + (n % 10);
      n /= 10;
```

```
  }
  while (i > 0) {
      uart_transmit(buffer[--i]);
  }

}

void timer2_init_ctc(void) {

  TCCR2A = (1 << WGM21);
  TCCR2B = (1 << CS22);
  OCR2A = 188;

}

int main(void) {

  timer2_init_ctc();
  twi_init();
  lcd_init();
  adc_init();
  uart_init();

  // LED roșu (D8), verde (D11), flacara (D9), buzzer (D6)
  DDRB |= (1 << PB0) | (1 << PB3);
  DDRB &= ~(1 << PB1);
  DDRD |= (1 << PD6);

  //buton (D3)
  DDRD &= ~(1 << PD3);
  PORTD |= (1 << PD3);

  lcd_set_cursor(0, 0);
  lcd_print("Incalzire senzor...");
  delay_ms_custom(20000);
  lcd_clear();

  uint8_t pwm_level = 0;
  uint8_t alarma_oprita = 0;
  uint8_t buton_apasat_ultima = 0;

  while (1) {
      uint8_t flacara = !(PINB & (1 << PB1));
      uint16_t gaz_fum_val = citeste_adc();
      uint8_t fum_detectat = gaz_fum_val > 300;
      uint8_t gaz_detectat = gaz_fum_val > 400;
      uint8_t pericol = flacara || gaz_detectat || fum_detectat;

      uint8_t buton = !(PIND & (1 << PD3));
      if (pericol && buton && !buton_apasat_ultima) {
```

```
        alarma_oprita = 1;
    }
    buton_apasat_ultima = buton;
```

```
    if (!pericol) {
        alarma_oprita = 0;
    }
```

```
    if (pericol) {
        PORTB |= (1 << PB0);
        PORTB &= ~(1 << PB3);
    } else {
        PORTB |= (1 << PB3);
        PORTB &= ~(1 << PB0);
    }
```

```
    if (pericol && !alarma_oprita) {
        if (pwm_level < 255) pwm_level += 5;
        pwm_start(pwm_level);
    } else {
        pwm_level = 0;
        pwm_stop();
    }
```

```
    uart_print("Gaz: ");
    uart_print_number(gaz_fum_val);
    uart_print(" | Flacara: ");
    uart_print_number(flacara);
    uart_print(" | Alarma oprita: ");
    uart_print_number(alarma_oprita);
    uart_print("\r\n");
```

```
    lcd_clear();
    lcd_set_cursor(0, 0);
    if (!pericol) {
        lcd_print("Stare normala");
    } else {
        if (flacara) {
            lcd_print("Pericol: FLACARA");
        } else if (gaz_detectat) {
            lcd_print("Pericol: GAZ");
        } else if (fum_detectat) {
            lcd_print("Pericol: FUM");
        }
    }
```

```
    lcd_set_cursor(0, 1);
    lcd_print("Alarma:");
    lcd_print_number(alarma_oprita);
```

```
      delay_ms_custom(300);
  }
```

```
  return 0;
```

```
}
```

Export to PDF

[1] PINB & (1 « PB1