الجامعة المصرية اليابانية للعلوم والتكنولوجيا

E-JUST

Egypt - Japan University of Science and Technology

エジプト日本科学技術大学

CSE324: Embedded Systems

Gas Leakage Detector Project

Abdelrahman Amr Abokhalil 120210335

Salma Amin Noureddin 120210389

Presented to

Dr Rami Zewail

Eng. Abdallah Elbarkokey

Eng. Habib Eltabakh

I. Abstract

Gas leakage detection is critical for ensuring safety in residential, commercial, and industrial environments. This project presents the design and implementation of a gas leakage detector using the MQ-5 gas sensor and an Arduino UNO microcontroller.

The primary objective of the project is to create a dependable system that can identify flammable gases such as LPG and CH_4, triggering a buzzer to prevent potential hazards.

The system comprises the MQ-5 gas sensor, which detects gas concentrations in the air, outputting an analog signal. The signal is then processed by the Arduino UNO, which we have programmed to evaluate the sensor readings and activate a buzzer system when gas levels exceed a predetermined threshold. Additionally, the hardware setup includes two LEDs: a green one to ensure that the system is activated, and a red one that is triggered by detection of gas.

The project involved extensive testing and calibration to ensure the sensor's accuracy and reliability. The code implementation was carried out using Microchip Studio, ISIS Proteus, and Arduino IDE, with code developed to handle sensor input, perform data analysis, and control output devices.

II. Introduction

Gas leakage detection is an essential safety measurement in residential, commercial, and industrial settings. The inadvertent release of combustible gases, such as liquefied petroleum gas (LPG) and natural gas, poses significant risks, including fires, explosions, and health hazards. These dangers necessitate the implementation of effective gas detection systems to prevent accidents and ensure the safety of occupants and property. Common applications of gas leakage detectors include homes, manufacturing plants, storage facilities, and any other environment where gas is used or stored.

The primary objective of this project is to design and implement a gas leakage detector that can effectively sense the presence of undesirable gases and provide timely alerts to mitigate the potential hazards. The system aims to combine the sensitivity of the MQ-5 sensor with the processing capability of the Arduino UNO microcontroller to create a reliable and efficient gas detection solution.

The project encompasses the development of a hardware and software system capable of detecting gas leaks, processing sensor data, and activating alert mechanisms such as LEDs and buzzers. The detector will measure gas concentrations in the air and trigger the buzzer and red LED when these concentrations exceed a predefined safety threshold. The expected outcomes include a fully functional gas leakage detector that can be tested and calibrated for accuracy, ensuring it meets safety standards and performs reliably in real-world situations.

III. System Design

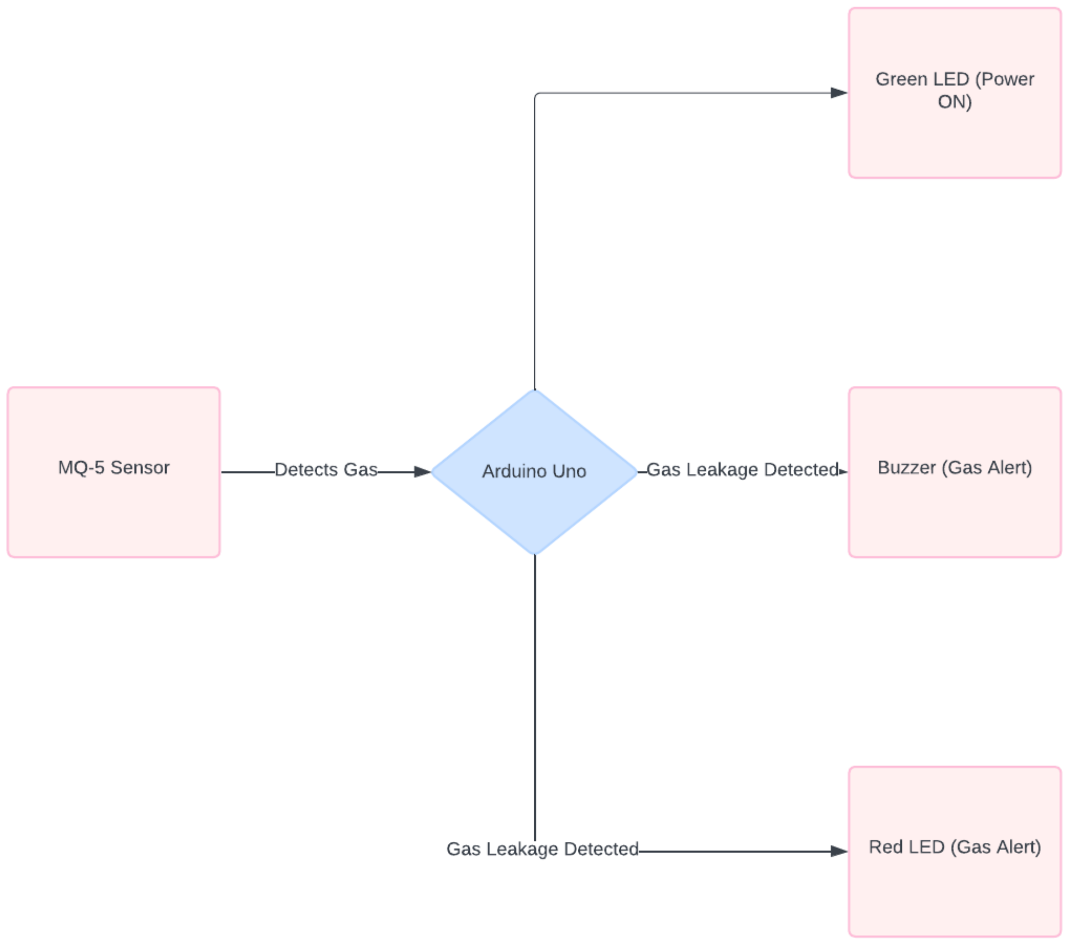

Figure 1 – Block diagram of the system

The MQ-5 makes use of the sensitive semiconductor material tin(IV) oxide, also known as stannic oxide (SnO_2), which has very low conductivity in clean air. When exposed to combustible gases, the sensor's conductivity gets higher, increasing as the gas concentration rises.

The sensor has high sensitivity to butane, propane, methane, and LPG. It can also detect both methane and propane simultaneously. The features of this sensor include its high sensitivity, low cost, and simple operating circuit.

The Arduino UNO is a microcontroller based on the ATmega328P. It has 14 digital input/output pins, 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button.

Our projects also utilized breadboards, LEDs, two 1 KΩ resistors, and jumper cables.

IV. Hardware Implementation

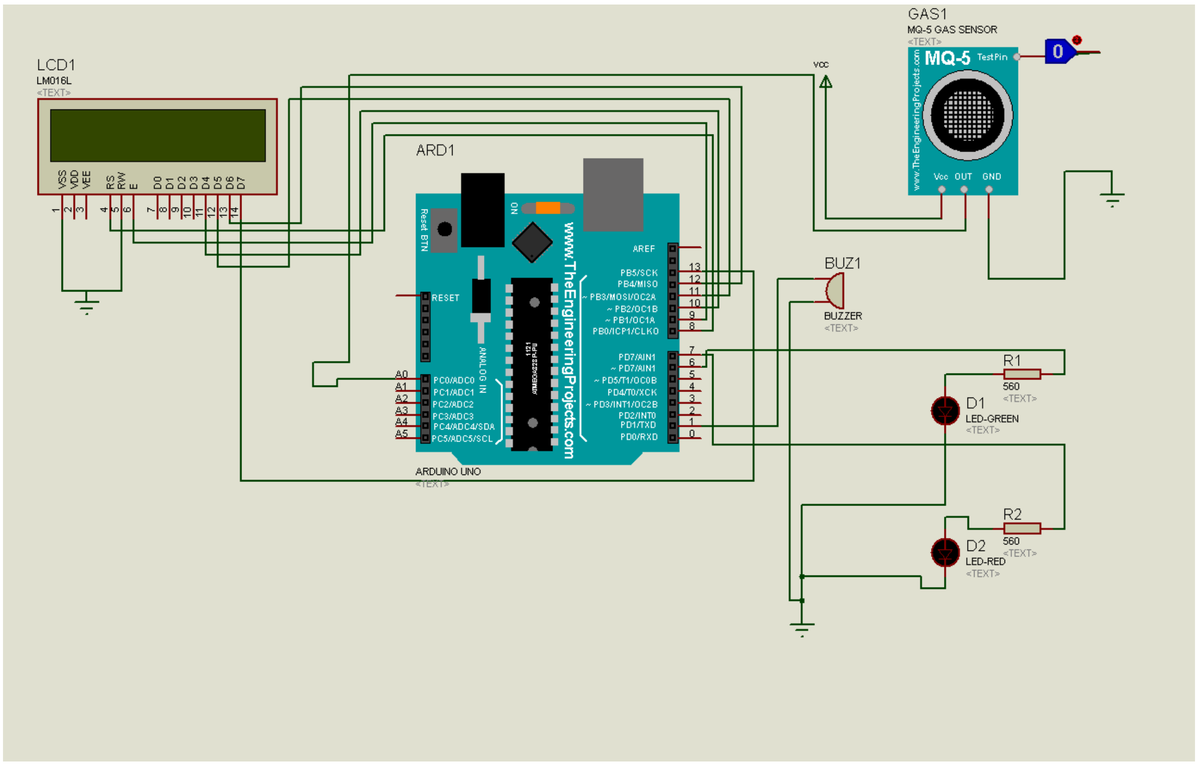

Figure 2 – Circuit diagram of the system`

Figure 2 shows the circuit diagram of the system. The MQ-5 sensor uses the following pin connections: `GND` → Ground; `Vcc` → Power source (5V), `OUT`: Output.

The GND and Vcc pins are connected accordingly, and the output pin is connected to our Arduino UNO's Port C Bit 0 (PB0) pin, which receives input and acts as an analog-to-digital converter (ADC), which converts the output analogue signals from the sensor into digital ones.

The buzzer has one pin connected to the ground, and the other pin to the Port D Bit 1 (PD1) pin. According to the ATmega328P datasheet, the PD1 acts as a general-purpose digital I/O pin.

The LEDs are connected to the ground on one side. On the other side, the green LED is connected to the Port D Bit 6 (PD6) pin on the Arduino UNO, and the red LED is connected to the Port D Bit 7 (PD7) pin.

Since DDRD is set (one) for both PB6 & PB7, both LEDs will act as output. Each LED is also connected to a 1 K Ω resistor.

Finally, the LCD display has pins VSS and R/W connected to the ground. When the R/W is set to low, as it is in our case, the data is written from the microcontroller, instead of read from.

The LCD's Register Select (RS) pin is connected to the PB0 pin of our Arduino board, the value is clear (0) or set (1) depending on our needs. When it is clear (low), it allows for input of instruction code. When high, it allows for input of data.

Enable pin (EN) is connected to PB1, enabling signal (pulse) when it is set (high), and disabling it when low.

The LCD has pins D4:7 connected to the Arduino's Port B bits 2:5 to send data to the LCD in 4-bit mode.

V. Software Implementation

```
#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>

#define GasSensor_pin 0
#define red_pin 7
#define green_pin 6
#define buzzer 1

#define LCD_DDR DDRB
#define LCD_PORT PORTB
// Define LCD control pins
#define RS 0
#define EN 1
```

```
// Define LCD data pins
#define D4 2
#define D5 3
#define D6 4
#define D7 5
```

These lines provide the necessary initialization for each register in our components. The `GasSensor_pin` is attached to the `A0` port, the `red_pin` and `green_pin` for the positive terminals of the LEDs are attached to `PD7` and `PD6` respectively, and the buzzer's positive terminal is attached to `PD1`.

The LCD Data Direction Register and Port are connected to `DDRB` and `PORTB`, respectively, with the control pins and data pins connected to each pin in `PORTB` as shown in the code.

```
void lcd_send_nibble(uint8_t nibble) {
      LCD_PORT &= ~(1<<D7 | 1<<D6 | 1<<D5 | 1<<D4); // Clear the data pin section of the port
      if(nibble & 1<<4) LCD_PORT |= (1<<D4);
      if(nibble & 1<<5) LCD_PORT |= (1<<D5);
      if(nibble & 1<<6) LCD_PORT |= (1<<D6);
      if(nibble & 1<<7) LCD_PORT |= (1<<D7);
      LCD_PORT |= (1<<EN); // Enable pulse
      _delay_us(1); // Provide a delay
      LCD_PORT &= ~(1<<EN); // Disable pulse
}
```

The `lcd_send_nibble` function first clears the data pins, then sends a 4-bit nibble to the LCD. It sets the data pins according to the bits in the nibble, pulses the enable pin (`EN`) to send the data, and finally disables the enable pin. It adds a delay of 1 microsecond to allow the instructions to be executed.

```
void lcd_send_byte(uint8_t data) {
    lcd_send_nibble(data); // send upper nibble
    lcd_send_nibble(data << 4); // send lower nibble
    _delay_us(50);
}
```

The `lcd_send_byte` function sends a full byte to the LCD by splitting it into two nibbles. The upper nibble is sent first, followed by the lower nibble (shifted by 4 bits).

```
void lcd_command(uint8_t command) {
    LCD_PORT &= ~(1<<RS); // RS = 0 for command
    lcd_send_byte(command);
}
```

The `lcd_command` function sends a command byte to the LCD. The `RS` pin is cleared (set to 0) to indicate that the byte is a command.

```c
void lcd_char(char character) {
    LCD_PORT |= (1<<RS); // RS = 1 for data
    lcd_send_byte(character);
}
```

The `lcd_char` function sends a character to the LCD. The `RS` pin is set (1) to indicate that the byte is data, in our case a character.

```c
void lcd_clear() {
    lcd_command(0x01); // clear display command
    _delay_ms(2);
}
```

The `lcd_clear` function clears the lcd display by sending the `0x01` command and waits for 2 milliseconds to ensure that the command is executed.

```c
void lcd_setCursor(uint8_t row, uint8_t col) {
    uint8_t offsets[] = {0x00, 0x40}; // base addresses for rows
    lcd_command(0x80 | (offsets[row] + col)); // set DDRAM address
}
```

The `lcd_setCursor` function sets the cursor position on the LCD. The `0x80` command sets the DDRAM (display data RAM) address, and the `offset` array provides the base addresses for the rows.

```c
void lcd_print(char *str) {
    while (*str) {
        lcd_char(*str++);
    }
}
```

The `lcd_print function` prints a string to the LCD by sending each character one at a time using `lcd_char.`

```c
void lcd_begin() {
    LCD_DDR = 0xFF; // set LCD port as output
    _delay_ms(15); // wait for power up

    LCD_PORT &= ~(1<<RS); // set RS to command mode

    // Initialization sequence
    lcd_send_nibble(0x30); // initialization
    _delay_ms(5);
    lcd_send_nibble(0x30); // initialization
    _delay_us(100);
    lcd_send_nibble(0x30); // initialization
    _delay_us(40);

    lcd_send_nibble(0x20); // set 4-bit mode
    _delay_us(40);

    lcd_command(0x28); // function set: 4-bit, 2-line, 5x8 dots
    lcd_command(0x08); // display off
    lcd_clear(); // clear screen
    lcd_command(0x06); // entry mode: increment automatically after each character, no display shift
    lcd_command(0x0C); // display on, cursor off, no blinking
}
```

The `lcd_begin` function initializes the LCD. It sets the data direction register, performs a special initialization sequence required to wake the LCD from its reset state, sets the LCD to 4-bit mode, configures the display settings (number of lines and character font), clears the display, and sets the entry mode and display control.

```
void GasSensor_Init(){
      DDRC &= ~(1 << GasSensor_pin);
      ADCSRA = 0x87; // Enable ADC and set prescaler to 128
      ADMUX = 0x40; // ADC0, AVCC, Right adjust
      _delay_us(500);
}
```

The `GasSensor_Init` function clears the `DDRC` to set it as input. This configures the pin connected to the gas sensor as an input pin to read the sensor's analog output.

`ADCSRA` (ADC Control and Status Register A) controls the ADC operation. The `0x87` command enables the ADC and simultaneously sets the prescaler value to 128.

`ADMUX` (ADC Multiplexer Selection Register) configures the reference voltage and input channel with the command `0x40`. This sets the reference voltage to `AVCC` and selects `ADC0` as the analog input channel. The delay allows the settings to stabilize before starting the ADC operations.

```
int AnalogRead(){
      ADCSRA |= (1 << ADSC); // start conversion
      while((ADCSRA & (1 << ADIF)) == 0); // wait for conversion
      ADCSRA |= 1 << ADIF; // conversion complete (Clear interrupt flag)
      _delay_ms(1);

      return ADC;
}
```

The `AnalogRead` function starts by setting the `ADCSRA` (ADC Control and Status Register A) to start the ADC conversion on the `ADSC` bit, which is bit `6.`

The `while` loop checks if the conversion is complete before clearing the interrupt flag. The `AnalogRead` then returns the converted digital value.

```
void LED(){
	int val = AnalogRead();
	lcd_clear();
	if(val > 60){
			PORTD |= (1 << red_pin) | (1 << buzzer);
			PORTD &= ~(1 << green_pin);
			lcd_print("GAS detected.");
	}
	else{
			PORTD |= 1 << green_pin;
			PORTD &= ~((1 << red_pin) | (1 << buzzer));
			lcd_print("SAFE.");
	}
}
```

The `LED` function integrates the gas detection with the LCD display. It reads the value from the gas sensor, clears the LCD, and updates the display with "`GAS detected.`" or "`SAFE.`" based on the sensor value. It also controls the LEDs and buzzer accordingly.

```
int main(){
	lcd_begin();
	DDRD |= (1 << green_pin) | (1 << red_pin) | (1 << buzzer);
	GasSensor_Init();
	lcd_clear();
	lcd_print("ABDERAHMAN&SALMA");
	_delay_ms(2000);
	lcd_clear();
	lcd_print("GAS SENSOR");
	_delay_ms(2000);

	lcd_clear();
	lcd_print("Welcome");
	_delay_ms(1000);
	lcd_clear();

	while(1){
			LED();
	}
	lcd_clear();
}
```

Finally, the `main` function of our code begins by initializing the LCD display, setting it up for 4-bit mode, configuring the display parameters, and making it ready to receive commands and display data. It then configures output pins, and sets bits `6`, `7`, and `1` of `DDRD` to 1, configuring them as output pins for the green LED, red LED, and buzzer.

It initializes the gas sensor by configuring the appropriate pin as an input and setting up the ADC.

It starts displaying initial messages on the LCD, by first clearing the display, then printing “`ABDERAHMAN&SALMA`” (our team members’ names). The 2-second delay allows the displayed message to be read. The process repeats for the messages `"GAS SENSOR"` and `"Welcome"` with respective delays of 2 seconds and 1 second. This sequence of messages provides initial information and a welcome message to the user.

The `while(1)` creates an infinite loop, ensuring the code inside the loop runs continuously. `LED()` is called repeatedly within this loop. This function reads the gas sensor value and updates the LED and buzzer status based on the gas concentration.