**Sistemas Microcontrolados:** Sensor de gás com simulação de exaustão integrada

Brenda Amanda da Silva Garcez, Eduardo Hartleib Galvão.

Universidade Tecnológica Federal do Paraná (UTFPR), campus Ponta Grossa

e-mails: brendagarcez@alunos.utfpr.edu.br, eduardohartleibgalvao@alunos.utfpr.edu.br

**RESUMO**

Este artigo apresenta o desenvolvimento de um sistema embarcado de detecção de gás utilizando o microcontrolador ATmega328P, presente na plataforma Arduino UNO. O projeto visa promover segurança doméstica por meio de sensores e atuadores de baixo custo, acionados automaticamente quando níveis elevados de gás inflamável são detectados. A simulação e validação foram realizadas no Tinkercad e com prototipagem física, confirmando a funcionalidade do sistema**.**

1. **INTRODUÇÃO**

A popularização dos microcontroladores em ambientes acadêmicos e comerciais possibilitou o desenvolvimento de sistemas embarcados acessíveis. O ATmega328P, um microcontrolador de 8 bits da família AVR da Microchip, destaca-se por sua robustez, baixo consumo e grande adoção na comunidade maker por meio da plataforma Arduino UNO.

Este projeto visa explorar tanto a simulação via Tinkercad quanto a aplicação física por meio do ATmega328P em um sistema de segurança residencial, utilizando o sensor MQ-9 para detectar vazamentos de gás e acionar sinais visuais, sonoros e mecânicos.

1. **REVISÃO BIBLIOGRÁFICA**

Segundo Banzi & Shiloh (2015), o Arduino UNO é uma das plataformas mais utilizadas para prototipagem eletrônica, e é baseado no microcontrolador ATmega328P. Este microcontrolador apresenta 32 registradores de propósito geral, barramento de dados de 8 bits, conversores A/D de 10 bits e suporte a interrupções, tornando-o ideal para sistemas de monitoramento ambiental.

Souza et al. (2011) destacam que o ATmega328P utiliza uma arquitetura Harvard modificada com conjunto de instruções RISC, o que facilita o controle eficiente de sensores e atuadores em sistemas embarcados.

1. **METODOLOGIA**

A programação do microcontrolador ATmega328P foi realizada por meio da IDE Arduino, utilizando linguagem C/C++. O código realiza leituras analógicas no pino A0, onde o sensor MQ-9 está conectado, e compara os valores com um limiar definido (ex. 300). Caso esse valor seja superado, o sistema executa múltiplas ações por meio de suas saídas, programadas diretamente nos registradores via bibliotecas próprias desenvolvidas.

3.1 Proposta do Sistema

Este projeto visa implementar um sistema embarcado de monitoramento de gás inflamável utilizando o microcontrolador ATmega328P (por meio da plataforma Arduino UNO), com resposta multimodal em caso de detecção. O sensor de gás MQ-9 é responsável por capturar níveis de concentração de gases como butano, propano e fumaça. Quando detectado um nível perigoso, o sistema:

* Aciona o LED vermelho, sinalizando alerta.
* Move um servo motor, simulando a ativação de um exaustor.
* Aciona um atuador sonoro (buzzer) com som pausado.
* Em situações normais, mantém o LED verde aceso e o servo em posição de repouso.

A lógica do sistema foi implementada em linguagem C/C++, utilizando a IDE Arduino e bibliotecas desenvolvidas para controle direto dos registradores, ampliando a modularidade e o desempenho do código.

3.2 Arquitetura do Sistema

O sistema foi estruturado com os componentes presentes na Tabela 1 e na Figura 1.

| **Componente** | **Função** |
| --- | --- |
| Sensor MQ-9 | Detecção de gás/fumaça |
| ATmega328P | Microcontrolador principal (via Arduino Uno) |
| LED vermelho/verde | Indicação visual de alerta/normalidade |
| Servo motor SG90 | Simula exaustor (abertura de janela) |
| Buzzer | Alarme sonoro |
| Resistores (1kΩ) | Limitar a corrente elétrica que passa pelo LED |

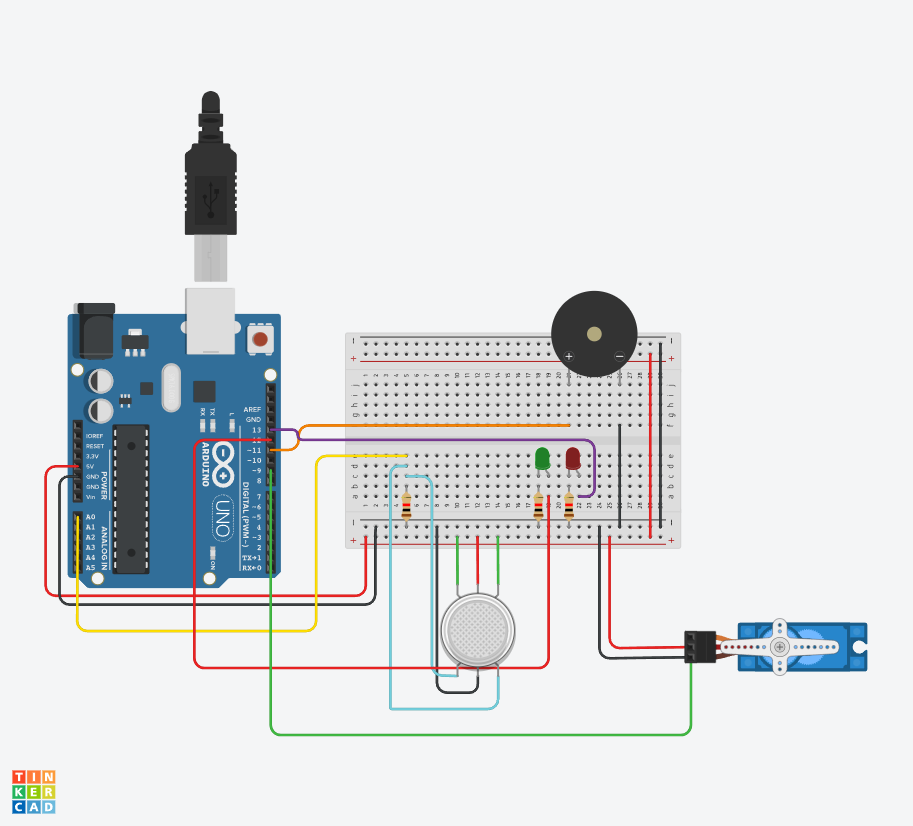
Tabela 1. Componentes 

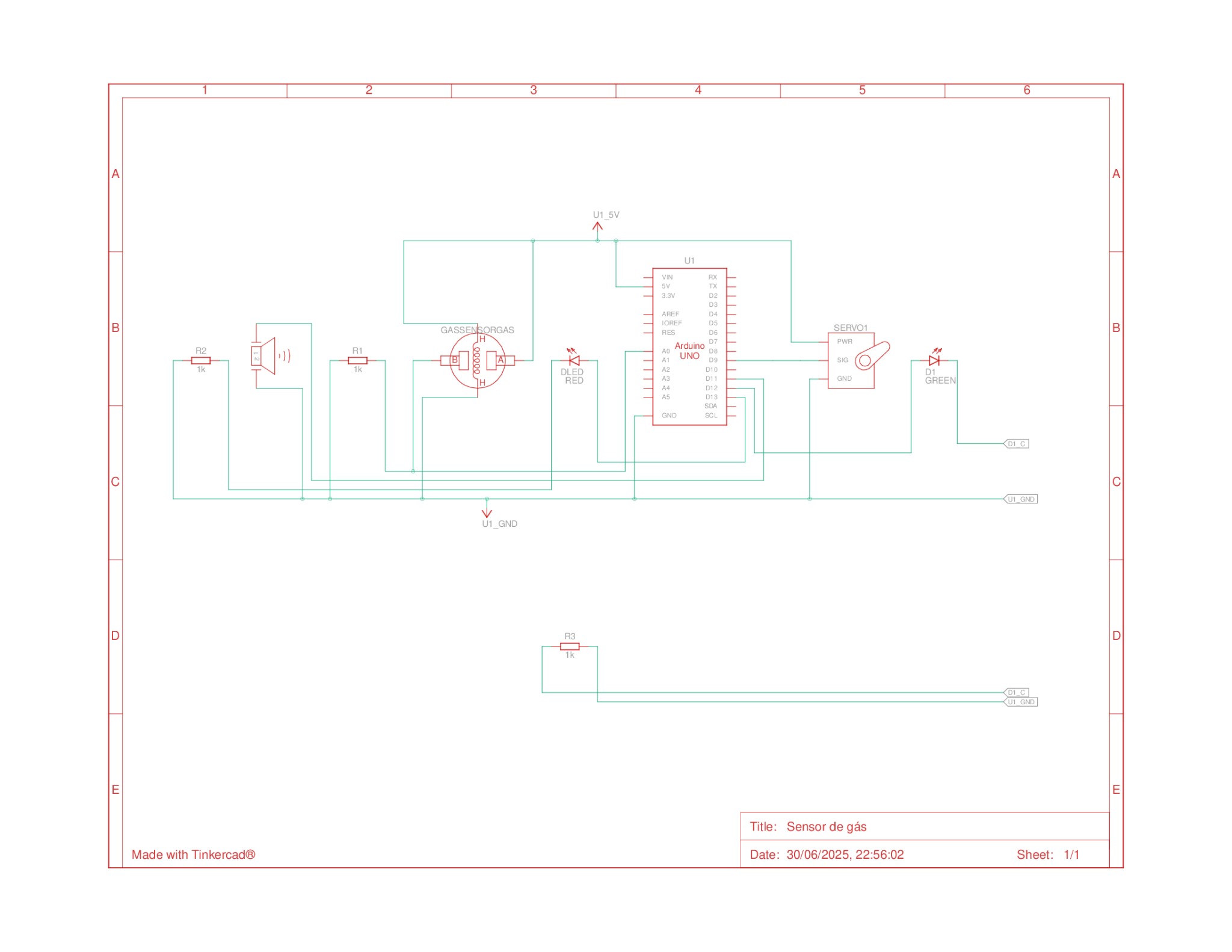
Figura 1. Montagem do circuito no Tinkercad simulando o sistema completo.

Figura 2 – Esquema elétrico do circuito utilizado.

3.3 Estrutura de Código

O código principal (main.ino) é estruturado em duas funções básicas: setup() para inicialização e loop() para a execução contínua da lógica do sistema.

#include "MeuADC.h"

#include "MeuServo.h"

#include "MeuBuzzer.h"

#include "MeuGPIO.h"

#define PinSensor 0

MeuADC adc;

MeuServo exaustor;

MeuBuzzer buzzer;

MeuGPIO ledVermelho;

MeuGPIO ledVerde;

void setup() {

Serial.begin(9600);

// LEDs como saída usando MeuGPIO

ledVermelho.configurarSaida('B', 5); // PB5 → pino 13

ledVerde.configurarSaida('B', 4); // PB4 → pino 12

// Inicializações

adc.iniciarADC();

exaustor.attach('B', 1); // PB1 = pino 9

exaustor.write(90); // posição neutra

buzzer.iniciar('B', 3); // PB3 = pino 11

buzzer.desligar();

}

void loop() {

uint16\_t valorSensor = adc.lerCanal(PinSensor);

Serial.print("Valor do sensor: ");

Serial.println(valorSensor);

if (valorSensor >= 200) {

ledVerde.escrever(false);

ledVermelho.escrever(true);

buzzer.tocarPor(300);

exaustor.write(120);

}

else {

ledVerde.escrever(true);

ledVermelho.escrever(false);

buzzer.desligar();

exaustor.write(90);

}

\_delay\_ms(500); // Aguarda 500ms entre leituras

}

Código 1. Trecho principal do sistema embarcado (Arduino IDE).

3.3.1 Bibliotecas Desenvolvidas

As bibliotecas permitem controle direto sobre o hardware do ATmega328P. Cada uma manipula registradores específicos:

* MeuADC: Configuração e leitura do conversor analógico-digital (ADC);
* MeuGPIO: Gerenciamento dos pinos de entrada e saída (GPIO) via registradores DDRx, PORTx e PINx;
* MeuBuzzer: Controle de som, permitindo acionamento temporizado do buzzer;
* MeuServo: Geração manual de pulsos PWM para o controle de servo motor SG90, sem dependência das bibliotecas padrão do Arduino.

3.3.1.1 MeuADC.h e MeuADC.cpp

Esta biblioteca realiza a configuração e leitura do conversor analógico-digital do ATmega328P, utilizando os registradores **ADMUX** e **ADCSRA**. Ela define a referência de tensão (AVcc), o prescaler e executa a conversão analógica-digital.

// Trecho de MeuADC.cpp

void MeuADC::iniciarADC() {

ADMUX = (1 << REFS0); // REFS0 = 1 → AVcc como referência

ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1); // Prescaler 64

}

uint16\_t MeuADC::lerCanal(uint8\_t canal) {

ADMUX = (ADMUX & 0xF0) | (canal & 0x0F); // Seleciona canal

ADCSRA |= (1 << ADSC); // Inicia conversão

while (ADCSRA & (1 << ADSC)); // Espera terminar

return ADC;

}

**Código 2** - A função iniciarADC() configura o ADC para usar AVcc como referência de tensão e define o prescaler para ajustar a velocidade de conversão. A função lerCanal() realiza a leitura do canal analógico selecionado.

3.3.1.2 MeuGPIO.h e MeuGPIO.cpp

A biblioteca MeuGPIO gerencia os pinos de entrada/saída, configurando sua direção e controlando os níveis lógicos. Manipula diretamente os registradores **DDRx**, **PORTx** e **PINx**.

// Trecho de MeuGPIO.cpp

void MeuGPIO::configurarSaida(char porta, uint8\_t pino) {

bit = pino;

switch (porta) {

case 'B': ddr = &DDRB; port = &PORTB; pin = &PINB; break;

// ... casos para C e D

}

\*ddr |= (1 << bit); // Define o pino como saída

}

void MeuGPIO::escrever(bool valor) {

if (valor)

\*port |= (1 << bit); // Escreve nível lógico alto

else

\*port &= ~(1 << bit); // Escreve nível lógico baixo

}

**Código 3 -** As funções permitem configurar a direção dos pinos e escrever valores lógicos diretamente, sem o uso de funções de alto nível do Arduino.

3.3.1.3 MeuBuzzer.h e MeuBuzzer.cpp

A biblioteca MeuBuzzer permite acionar o buzzer de forma simples, utilizando comandos diretos e temporização via \_delay\_ms().

// Trecho de MeuBuzzer.cpp

void MeuBuzzer::iniciar(char porta, uint8\_t pino) {

bit = pino;

// ... switch para selecionar a porta (B, C, D) e configurar DDRx

desligar();

}

void MeuBuzzer::tocarPor(uint16\_t ms) {

ligar();

while (ms--) \_delay\_ms(1); // Espera o tempo especificado

desligar();

}

**Código 4 -** A função iniciar() prepara o pino como saída. tocarPor() aciona o buzzer por um intervalo definido, utilizando um laço de delay.

3.3.1.4 MeuServo.h e MeuServo.cpp

A biblioteca MeuServo controla diretamente o servo motor SG90, gerando sinais PWM manualmente, sem depender da biblioteca Servo.h padrão do Arduino.

// Trecho de MeuServo.cpp

void MeuServo::write(int angulo) {

if (!ligado) return;

// Converte o ângulo (0-180) para largura de pulso (em microssegundos)

int pulso = map(angulo, 0, 180, 544, 2400);

// Gera o pulso manualmente (PWM de 20ms com HIGH entre 544-2400µs)

digitalWrite(pin, HIGH);

delayMicroseconds(pulso);

digitalWrite(pin, LOW);

// Aguarda o restante do ciclo PWM (20ms no total)

delay(20);

}

**Código 5** - A função write() converte o ângulo desejado em microsegundos de pulso e realiza a geração manual do sinal PWM, garantindo controle total do tempo de ativação.

3.4 Justificativas Técnicas

* **Microcontrolador ATmega328P**: escolhido devido à sua ampla documentação, suporte educacional e presença no Arduino UNO, sendo ideal para projetos didáticos e embarcados.
* **Sensor MQ-9**: O sensor MQ-9 foi escolhido por apresentar baixo custo, boa sensibilidade e a capacidade de detectar diferentes tipos de gases, como monóxido de carbono (CO), metano (CH₄) e gás liquefeito de petróleo (LPG/propano). Além disso, possui sensibilidade ajustável por meio da resistência de carga (RL), o que o torna versátil para aplicações em diversos ambientes. Abaixo, são apresentadas as curvas características de sensibilidade do sensor, demonstrando a relação entre a concentração dos gases e a razão Rs/Ro, com base nos dados fornecidos pelo fabricante:

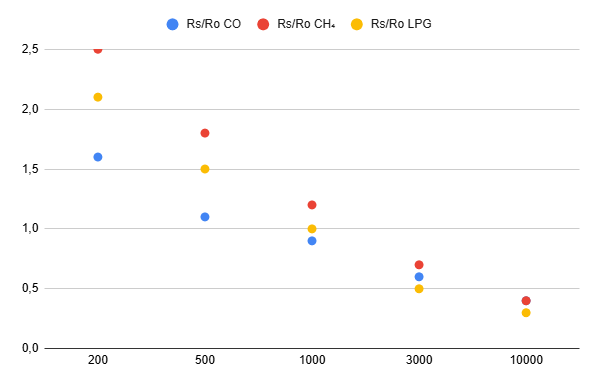


Figura 3 - Curvas de sensibilidade do sensor MQ-9. Relação Rs/Ro em função da concentração de gases (ppm), para CO, CH₄ e LPG. *Fonte: Hanwei Electronics Co., Ltd. (2025)*.

* **Uso de biblioteca própria**:desenvolvido pois permite domínio completo do hardware, otimização de código, melhor desempenho e aprendizado aprofundado sobre a arquitetura do microcontrolador.
* **Servo motor**: escolhido para simular uma ação mecânica real (como abertura de janela ou exaustor).
* **Simulação no Tinkercad**: utilizada nas fases iniciais para validação do circuito e lógica antes da prototipagem física.

**RESULTADOS E ANÁLISE**

Os testes no Tinkercad e na prototipagem física confirmaram o funcionamento do sistema. O sistema respondeu corretamente ativando os atuadores (LED, servo motor, buzzer) quando o nível de gás simulado ultrapassou o limiar de 300.

Controle e Desempenho: As bibliotecas próprias desenvolvidas para o ATmega328P (MeuGPIO, MeuBuzzer, MeuADC e MeuServo) permitiram acesso direto aos registradores do microcontrolador. Isso resultou em um controle mais preciso do hardware e maior otimização do código, garantindo respostas eficientes dos atuadores em tempo real.

Modularidade: O uso dessas bibliotecas simplificou o código principal, tornando-o mais modular e fácil de entender e expandir.

**CONCLUSÃO**

O projeto desenvolvido demonstrou viabilidade técnica e funcional do sistema para detecção de gás utilizando o microcontrolador ATmega328P. O sistema apresentou respostas rápidas e previsíveis, tanto em simulação quanto em testes reais. A modularização do código através das bibliotecas próprias de baixo nível torna a aplicação escalável e adaptável para outras situações de monitoramento. Para trabalhos futuros, sugere-se a adição de conectividade sem fio para alertas remotos e melhorias no algoritmo de análise do sensor.

**REFERÊNCIAS BIBLIOGRÁFICAS**

[1] MICROCHIP. *ATmega328P Datasheet*. Disponível em: <https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega328P-DataSheet-DS40002061B.pdf>

[2] BANZI, M.; SHILOH, M. Primeiros passos com Arduino. Novatec, 2015.

[3] HIPÓLITO, J. G. et al. Automação residencial com Arduino. Revista Inovação, v.1, n.1, 2018.

[4] TINKERCAD. Autodesk. Disponível em: <https://www.tinkercad.com>

[5] Hanwei Electronics Co., Ltd. MQ-9 Semiconductor Sensor for CO/Combustible Gas – Datasheet. Henan Hanwei Electronics Co., Ltd. Disponível em: [www.hwsensor.com](http://www.hwsensor.com). Acesso em: 02 jul. 2025.