Aluno do Embarcatech_37 no IFMA

Nome: Manoel Felipe Costa Furtado

Matrícula: 20251RSE.MTC0086

Atividade – Referente ao capítulo 04 da unidade 02

Tema do Capítulo – Sensores e Atuadores

Prazo dia 22/06/2025 as 23:59

Enunciado: Sistema de Monitoramento Ambiental com Resposta Ativa.

Desenvolver um sistema embarcado com o Raspberry Pi Pico W que integre sensores (temperatura, luminosidade e gás) com atuadores (relé, motor vibracall e LED infravermelho), criando uma solução automatizada de monitoramento e resposta a variações ambientais.

Você deverá criar um sistema embarcado utilizando o Raspberry de forma simulada com Wokwi, capaz de:

- Detectar aumento de temperatura ambiente com o sensor DHT-22;
- Verificar queda de luminosidade com o sensor LDR;
- Identificar presença de gás/fumaça com sensor MQ-2;
- Responder com atuadores específicos a cada evento monitorado, de forma individual ou combinada.

Descrição do Sistema:

- Quando a temperatura ultrapassar 30°C, um motor vibracall deve ser ativado para gerar um alerta físico. ERRATA: Mudou de Vibracall para Servo Motor.
- Quando a luz ambiente for baixa, um LED infravermelho deve acender para prover iluminação discreta (ex: segurança). ERRATA: Mudou de LED infravermelho para LED comum?
- Ao detectar presença de gás ou fumaça, um relé deve ser ativado para acionar um dispositivo externo, como um exaustor ou alarme.

Requisitos técnicos:

- Leitura do DHT-22 via entrada analógica (ADC); ERRATA: entrada Digital
- Leitura do LDR em divisor de tensão com ADC;
- Leitura do sensor de gás MQ-2 em sinal analógico;
- Uso de pinos GPIO para controle de relé, Servo Motor e LED;

- Uso adequado de resistores de pull-up/pull-down, se necessário;
- Projeto executado e testado no Wokwi com Raspberry.

Instruções de Implementação:

1) Desenvolva o projeto no Wokwi;
2) Teste o sistema conectando os sensores e atuadores conforme os exemplos propostos no eBook;
3) Implemente condições lógicas para ativar os atuadores conforme os sensores detectarem os eventos;
4) Priorize clareza, organização e boas práticas na estrutura do programa.

Links das simulações

- Nome do arquivo principal: “Atividade_Uni_02_Cap_04.c”.
- Link Wokiwi
  Link: https://wokwi.com/projects/434192157198876673
- GitHub:
  https://github.com/ManoelFelipe/Embarcatech_37/tree/main/Unidade_02/Cap_04/Atividade_04

Como entender o projeto:

1. Comece pelo main() em Atividade_Uni_02_Cap_04.c para ver a ordem em que tudo acontece.
2. Vá abrindo os headers.h para saber o que cada módulo oferece.
3. Leia os .c se quiser entender os detalhes internos.
4. Todos os arquivos estão comentados com Doxygen em português para facilitar.

- Código: Atividade_Uni_02_Cap_04.c

```
/**
 * @file Atividade_Uni_02_Cap_04.c
 * @version 1.0
 * @date 2025-06-20
 * @author Manoel Felipe Costa Furtado
 * @brief Sistema de Monitoramento Ambiental com Resposta Ativa para Raspberry Pi Pico W.
 *
 * @details Este programa robusto integra múltiplos sensores (temperatura, gás e luminosidade)
 * e atuadores (servo motor, relé e LED) para criar um sistema de monitoramento e controle
```

```
 * automatizado. O sistema é projetado para rodar continuamente no Raspberry Pi Pico W,
 * avaliando as condições do ambiente e reagindo de acordo com limiares pré-definidos.
 *
 * @section Funcionalidades Principais
 * 1.  Monitoramento de Temperatura (DHT22): Utiliza o driver customizado `dht22.h` para ler
 *     a temperatura ambiente. Se o valor ultrapassar o limiar de 30°C, um servo motor é
 *     acionado para uma posição de 180°, servindo como um alerta físico ou para acionar um mecanismo
 *     mecânico (como a abertura de uma ventilação).
 * 2.  Detecção de Gás/Fumaça (MQ-2): Mede a concentração de gás no ar através da leitura
 *     analógica do sensor MQ-2. Se a concentração estimada em PPM (Partes Por Milhão)
 *     ultrapassar um limiar de segurança, um módulo relé é ativado. Este relé pode ser
 *     conectado a dispositivos de maior potência, como um exaustor ou um alarme sonoro.
 * 3.  Monitoramento de Luminosidade (LDR): Um LDR (Resistor Dependente de Luz) é usado para
 *     medir a intensidade da luz ambiente. Em condições de baixa luminosidade (abaixo de um
 *     limiar), um LED vermelho de alto brilho é aceso, funcionando como luz de sinalização,
 *     luz de emergência ou indicador de status.
 *
 * @section Arquitetura do Software
 * O sistema utiliza o SDK do Raspberry Pi Pico (`pico-sdk`) e suas bibliotecas de hardware para
 * gerenciar ADC (Conversor Analógico-Digital), PWM (Modulação por Largura de Pulso) e GPIO (Entrada/Saída
 * de Propósito Geral), além do driver `dht22.c` para o sensor de temperatura.
 * A lógica principal opera em um loop infinito (`while(true)`), garantindo monitoramento contínuo.
 */

// --- Inclusão de Bibliotecas ---
#include <stdio.h>          // Biblioteca padrão de entrada/saída para funções como printf().
#include <math.h>           // Biblioteca matemática para funções como pow() e powf(), usadas nos cálculos dos
sensores.
#include "pico/stdlib.h"    // Biblioteca principal do SDK do Pico, inclui funções de inicialização e temporização
como sleep_ms().
#include "hardware/gpio.h" // Biblioteca para controle de pinos digitais (GPIO), usada para o LED e o Relé.
#include "hardware/pwm.h"  // Biblioteca para gerar sinais PWM, usada para controlar o servo motor.
#include "hardware/adc.h"  // Biblioteca para ler sinais analógicos, usada para os sensores MQ-2 e LDR.
#include "dht22.h"          // Inclui a interface do driver customizado para o sensor de temperatura e umidade DHT22.

// --- Definições de Pinos GPIO ---
/** @brief Pino GPIO onde o pino de dados do sensor DHT22 está conectado. */
#define DHT22_PIN 6
/** @brief Pino GPIO conectado ao catodo do LED vermelho (o anodo deve ir para 3.3V através de um resistor). */
#define LED_RED_PIN 10
/** @brief Pino GPIO que envia o sinal de controle PWM para o servo motor. */
#define SERVO_PIN 17
/** @brief Pino GPIO que envia o sinal de ativação para o módulo relé. */
#define RELAY_PIN 18
/** @brief Pino GPIO analógico (GP26) conectado à saída analógica (AO) do sensor de gás MQ-2. Corresponde ao canal 0
do ADC. */
#define MQ2_ADC_PIN 26
```

```c
/** @brief Pino GPIO analógico (GP28) conectado ao divisor de tensão com o sensor de luminosidade LDR. Corresponde ao canal 2 do ADC. */
#define LDR_ADC_PIN 28

// --- Constantes de Limite (Thresholds) para Ativação dos Atuadores ---
/** @brief Limiar de temperatura em graus Celsius. Acima deste valor, o servo motor será acionado. */
const float TEMPERATURE_THRESHOLD = 30.0f;
/** @brief Limiar de luminosidade em Lux. Abaixo deste valor, o LED será aceso. Este valor é empírico e pode precisar de calibragem. */
const float LUMINOSITY_THRESHOLD = 150.0f; // No código original a lógica está invertida (>), então o LED acende com LUZ ALTA.
/** @brief Limiar de concentração de gás em PPM. Acima deste valor, o led do relé será acionado. */
const float GAS_PPM_THRESHOLD = 6.0f; //  GÁS BAIXO.

// --- Parâmetros de Calibração para o Sensor de Gás MQ-2 ---
/** @brief Resistência do sensor em ar limpo (R0), em Ohms. Este é o valor mais importante para calibrar a precisão do sensor. */
const float MQ2_R0 = 8000.0f; // Valor obtido experimentalmente ou a partir do datasheet, ajustado para simulação.
/** @brief Valor da resistência de carga (RL) presente no módulo do sensor MQ-2, em Ohms. Geralmente 5kΩ. */
const float MQ2_RL = 5000.0f;

/**
 * @brief Constantes da curva de sensibilidade do gás de referência (ex: GLP).
 * @details Estes valores (a, b) são extraídos do gráfico log-log no datasheet do sensor e definem.
 * a relação entre a razão (Rs/R0) e a concentração de gás (PPM).
 * A fórmula é: PPM = a * (Rs/R0) ^ b
 */
const float MQ2_CURVE_A = 8.664f;   ///< Constante 'a' para a curva de gás de referência (fator de escala).
const float MQ2_CURVE_B = 0.116f;   ///< Constante 'b' para a curva de gás de referência (expoente).

// --- Parâmetros de Calibração para o Sensor de Luminosidade LDR ---
/** @brief Valor do resistor (em Ohms) que forma o divisor de tensão com o LDR. */
const float LDR_SERIES_RESISTOR = 10000.0f; // 10kΩ
/** @brief Tensão de referência do ADC do Raspberry Pi Pico. */
const float ADC_VREF = 3.3f;
/** @brief Resolução máxima do ADC de 12 bits (2^12 = 4096 valores, de 0 a 4095). */
const float ADC_MAX_RESOLUTION = 4095.0f;

/**
 * @brief Estrutura para agrupar os resultados da leitura do sensor MQ-2.
 * @details Usar uma struct torna o código mais limpo, permitindo que a função
 * `read_mq2_ppm` retorne tanto o valor bruto do ADC quanto o valor calculado em PPM.
 */
typedef struct {
    float ppm;          ///< Concentração de gás calculada em Partes Por Milhão (PPM).
    uint16_t raw_adc;   ///< Leitura bruta do conversor analógico-digital (0-4095).
} Mq2Result;
```

```c
// --- Protótipos das Funções Auxiliares ---
// Declarar as funções aqui permite que `main` as chame antes de suas implementações completas.
void setup_peripherals();
void set_servo_angle(uint pin, float angle);
float read_ldr_lux();
Mq2Result read_mq2_ppm();


/**
 * @brief Função principal do programa (ponto de entrada).
 *
 * @details Esta função é executada quando o Pico é ligado. Ela primeiro inicializa
 * todos os periféricos necessários através da função `setup_peripherals()`. Em seguida,
 * entra em um laço infinito (`while (true)`) que constitui o ciclo de vida
 * principal do sistema: ler, processar, agir e repetir.
 *
 * @return int Esta função nunca retorna sob operação normal.
 */
int main() {
    // Inicializa a comunicação serial (USB) para permitir o envio de mensagens (printf) para o computador.
    stdio_init_all();
    // Chama a função para configurar todos os pinos e periféricos de hardware.
    setup_peripherals();

    // Uma pequena pausa inicial para garantir que os sensores (especialmente o MQ-2) se estabilizem após serem
ligados.
    sleep_ms(2000);
    printf("Sistema de Monitoramento Ambiental Iniciado.\n\n");

    // Laço de execução principal. O programa permanecerá aqui para sempre.
    while (true) {
        // --- Leitura e Lógica do Sensor de Temperatura (DHT22) ---
        float temperature, humidity;
        // Chama a função do driver para ler os dados do sensor DHT22.
        int dht_result = dht22_read(&temperature, &humidity);

        // Verifica se a leitura foi bem-sucedida.
        if (dht_result == DHT22_OK) {
            printf("\nTemperatura: %.1f °C | Umidade: %.1f %%\n", temperature, humidity);

            // Compara a temperatura lida com o limiar definido.
            if (temperature > TEMPERATURE_THRESHOLD) {
                set_servo_angle(SERVO_PIN, 180); // Move o servo para a posição de "alerta" (180 graus).
                printf("ALERTA: Temperatura ALTA! Servo acionado.\n");
            } else {
                set_servo_angle(SERVO_PIN, 0); // Mantém ou retorna o servo à posição de repouso (0 graus).
            }
        } else {
            // Informa sobre a falha na leitura do DHT22, incluindo o código de erro.
```

```
            printf("Falha ao ler DHT22 (cod: %d) | ", dht_result);
        }
        // --- Fim Leitura e Lógica do Sensor de Temperatura (DHT22) ---

        // --- Leitura e Lógica do Sensor de Gás (MQ-2) ---
        Mq2Result mq2_data = read_mq2_ppm(); // Chama a função que retorna a struct com os dados do MQ-2.

        printf("Gás: Leitura bruta: %4d | PPM (estimado): %.0f\n",
               mq2_data.raw_adc, mq2_data.ppm);

        // Compara a concentração de gás com o limiar.
        if (mq2_data.ppm <= GAS_PPM_THRESHOLD) {
            gpio_put(RELAY_PIN, 1); // Ativa o relé (envia nível alto).
            printf("ALERTA: Condicao de gas para ativacao do rele atingida!\n");
        } else {
            gpio_put(RELAY_PIN, 0); // Desativa o relé (envia nível baixo).
        }
        // --- Fim Leitura e Lógica do Sensor de Gás (MQ-2) ---


        // --- Leitura e Lógica do Sensor de Luminosidade (LDR) ---
        float lux = read_ldr_lux();
        printf("Luz: %.0f Lux\n", lux);
        // Compara a luminosidade com o limiar. Acima de 150 Lux  acende o led
        if (lux > LUMINOSITY_THRESHOLD) {
            gpio_put(LED_RED_PIN, 1); // Liga o LED.
            printf("ALERTA: Condicao de luz para ativacao do LED atingida!\n");
        } else {
            gpio_put(LED_RED_PIN, 0); // Desliga o LED.
        }
        // --- Fim Leitura e Lógica do Sensor de Luminosidade (LDR) ---

        // Aguarda por 2 segundos antes de iniciar o próximo ciclo de leituras.
        // Este delay é importante para respeitar o intervalo mínimo do sensor DHT22 e para
        // evitar sobrecarregar o sistema com leituras excessivamente rápidas.
        sleep_ms(2000);
    }

    return 0; // Esta linha nunca será alcançada.
}

/**
 * @brief Inicializa e configura todos os pinos e periféricos de hardware necessários.
 *
 * @details Esta função é chamada uma única vez no início do `main`. Ela centraliza toda
 * a configuração de hardware, tornando o código mais organizado.
 */
void setup_peripherals() {
    // Inicialização do pino do LED Vermelho como saída digital.
```

```
    gpio_init(LED_RED_PIN);
    gpio_set_dir(LED_RED_PIN, GPIO_OUT);

    // Inicialização do pino do Relé como saída digital.
    gpio_init(RELAY_PIN);
    gpio_set_dir(RELAY_PIN, GPIO_OUT);

    // --- Inicialização do PWM para o Servo Motor ---
    // Atribui a função de PWM ao pino do servo.
    gpio_set_function(SERVO_PIN, GPIO_FUNC_PWM);
    // Descobre qual "fatia" (slice) de PWM está conectada ao pino. O Pico tem 8 slices de PWM.
    uint slice_num = pwm_gpio_to_slice_num(SERVO_PIN);
    // Obtém uma configuração padrão de PWM.
    pwm_config config = pwm_get_default_config();

    // Configura o clock do PWM para operar a 50Hz, que é a frequência padrão para servos.
    // Clock do sistema (125MHz) / 62.5 (clkdiv) / 40000 (wrap) = 50 Hz.
    // Período de 20ms.
    pwm_config_set_clkdiv(&config, 62.5f);
    pwm_config_set_wrap(&config, 40000);
    // Aplica a configuração à fatia de PWM e a inicia.
    pwm_init(slice_num, &config, true);
    set_servo_angle(SERVO_PIN, 0); // Define a posição inicial do servo como 0 graus.

    // --- Inicialização do Conversor Analógico-Digital (ADC) ---
    adc_init(); // Inicializa o hardware do ADC.
    adc_gpio_init(MQ2_ADC_PIN); // Habilita a função de ADC no pino GP26.
    adc_gpio_init(LDR_ADC_PIN); // Habilita a função de ADC no pino GP28.

    // --- Inicialização do driver do sensor DHT22 ---
    dht22_init(DHT22_PIN);
}

/**
 * @brief Controla o ângulo de um servo motor padrão.
 *
 * @details Converte um ângulo desejado (de 0 a 180 graus) em um valor de duty cycle de PWM.
 * Servos padrão respondem a pulsos de largura específica dentro de um período de 20ms:
 * - Pulso de ~1ms: Posição de 0 graus.
 * - Pulso de ~1.5ms: Posição de 90 graus.
 * - Pulso de ~2ms: Posição de 180 graus.
 * A função calcula o valor de contagem (`duty`) correspondente a essa largura de pulso.
 *
 * @param[in] pin O pino GPIO onde o servo está conectado (deve ser um pino com PWM).
 * @param[in] angle O ângulo desejado para o servo, de 0.0 a 180.0 graus.
 */
void set_servo_angle(uint pin, float angle) {
    // Garante que o ângulo esteja dentro da faixa permitida (0-180).
```

```
    if (angle < 0) angle = 0;
    if (angle > 180) angle = 180;

    // Mapeia o ângulo (0-180) para o valor de duty cycle (contagem do PWM).
    // Com nosso wrap de 40000 e clkdiv de 62.5, o período é de 20ms.
    // 1ms (1000us) de pulso corresponde a um duty de (1000us / 20000us) * 40000 = 2000.
    // 2ms (2000us) de pulso corresponde a um duty de (2000us / 20000us) * 40000 = 4000.
    // A fórmula abaixo mapeia linearmente o ângulo [0, 180] para o duty [2000, 4000].
    uint16_t duty = (uint16_t)(2000 + (angle / 180.0f) * 2000);

    // Define o nível do PWM para o pino especificado, movendo o servo.
    pwm_set_gpio_level(pin, duty);
}

/**
 * @brief Lê a luminosidade de um sensor LDR e a converte para uma estimativa em Lux.
 *
 * @details O LDR está em um circuito divisor de tensão com um resistor fixo de 10kΩ.
 * 1. Lê o valor bruto do ADC, que é proporcional à tensão no pino.
 * 2. Calcula a tensão real a partir do valor do ADC.
 * 3. Usa a fórmula do divisor de tensão para calcular a resistência atual do LDR.
 * 4. Converte a resistência do LDR para uma estimativa de Lux usando uma fórmula
 * de aproximação comum. A precisão depende muito do LDR específico e da fórmula usada.
 *
 * @return float O valor estimado da luminosidade em Lux. Retorna 0 para escuridão total
 * ou -1 em caso de erro de leitura.
 */
float read_ldr_lux() {
    // Seleciona o canal do ADC conectado ao LDR (ADC2 -> GP28).
    adc_select_input(2);
    uint16_t raw_adc = adc_read();

    // Caso especial: se a leitura do ADC for máxima, a resistência do LDR é próxima de zero (luz muito intensa).
    // A fórmula abaixo daria divisão por zero. O retorno correto aqui dependeria do circuito.
    if (raw_adc >= ADC_MAX_RESOLUTION) {
        return 50000; // Retorna um valor alto para indicar luz intensa.
    }

    // Converte o valor bruto do ADC (0-4095) para uma tensão (0-3.3V).
    float adc_voltage = ADC_VREF * (raw_adc / ADC_MAX_RESOLUTION);

    // Evita divisão por zero se a tensão for 0V (escuridão total).
    if (adc_voltage <= 0.0f) {
        return 0; // Retorna 0 Lux para escuridão.
    }

    // Calcula a resistência do LDR usando a fórmula do divisor de tensão:
    // V_out = V_ref * R_ldr / (R_fixo + R_ldr) => R_ldr = R_fixo * V_out / (V_ref - V_out)
```

```
    float ldr_resistance = (LDR_SERIES_RESISTOR * adc_voltage) / (ADC_VREF - adc_voltage);

    // Converte a resistência do LDR para Lux (fórmula de aproximação)
    float lux = powf((50.0f * 1000.0f * powf(10.0f, 0.7f)) / ldr_resistance, 1.0f / 0.7f);

    // Converte a resistência do LDR para Lux usando uma fórmula empírica de aproximação.
    // Esta fórmula (Lux = 500 * (10k / R_ldr)) é um exemplo comum, mas pode não ser precisa.
    // A fórmula original no código era mais complexa e também uma aproximação.
    // Usando uma versão mais simples para clareza: Lux = k / R_ldr
    // float lux = 5e6 / ldr_resistance;

    return lux;
}


/**
 * @brief Lê a concentração de gás do sensor MQ-2 e a converte para PPM.
 *
 * @details O processo envolve:
 * 1. Ler a tensão na saída analógica do sensor MQ-2.
 * 2. Calcular a resistência do sensor (Rs) com base nessa tensão e na resistência de carga (RL).
 * 3. Calcular a razão entre a resistência atual (Rs) e a resistência do sensor em ar limpo (R0).
 * 4. Usar a curva de sensibilidade do datasheet (modelada por A e B) para converter essa razão em PPM.
 *
 * @return Mq2Result Uma estrutura contendo a concentração estimada de gás em PPM e a leitura bruta do ADC.
 */
Mq2Result read_mq2_ppm() {
    Mq2Result result; // Cria uma instância da struct para armazenar os resultados.

    // Seleciona o canal do ADC conectado ao MQ-2 (ADC0 -> GP26).
    adc_select_input(0);
    // Lê o valor bruto e armazena diretamente no campo da struct.
    result.raw_adc = adc_read();

    // Converte o valor bruto do ADC (0-4095) para uma tensão (0-3.3V).
    float adc_voltage = result.raw_adc * ADC_VREF / ADC_MAX_RESOLUTION;

    // Evita divisão por zero se a tensão for 0V (o que não deve acontecer em operação normal).
    if (adc_voltage <= 0) {
        result.ppm = 0;
        return result;
    }

    // Calcula a resistência do sensor (Rs) usando a fórmula do divisor de tensão.
    // O módulo do sensor tem um circuito onde a tensão medida (V_out) está em RL.
    // V_out = V_ref * RL / (Rs + RL) => Rs = (V_ref * RL / V_out) - RL
    float rs = ((ADC_VREF * MQ2_RL) / adc_voltage) - MQ2_RL;

    // Se a resistência for negativa ou zero (pode acontecer com leituras instáveis), trate como erro.
```

```
    if (rs <= 0) {
        result.ppm = 0;
        return result;
    }

    // Calcula a razão Rs/R0, que é a base para encontrar a concentração no gráfico do datasheet.
    float ratio = rs / MQ2_R0;

    // Calcula o PPM usando a fórmula da curva de potência: PPM = A * ratio^B
    // A função pow() da biblioteca math.h é usada para a exponenciação.
    result.ppm = MQ2_CURVE_A * pow(ratio, MQ2_CURVE_B);

    // Retorna a struct inteira contendo tanto o valor bruto quanto o calculado.
    return result;
}
```