Nome completo: Marcony Henrique Bento Souza
Matrícula: 20251RSE.MTC0089
Email: marconyhenrique321@gmail.com

Enunciado: Sistema de Aquisição de Temperatura com DMA e Interface I2C em Microcontrolador RP2040 Desenvolver um sistema embarcado que utilize o controlador DMA do RP2040 para capturar automaticamente as amostras do sensor de temperatura interno (canal ADC4) e exibir os valores em um display OLED SSD1306, utilizando comunicação I2C.

Código c usado na BitDogLab:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include "pico/stdlib.h"     // Funções básicas do SDK Pico
#include "hardware/adc.h"    // Controle do ADC interno
#include "hardware/dma.h"    // Controle do DMA
#include "hardware/i2c.h"
#include "hardware/timer.h"
#include "pico/binary_info.h"
#include "inc/ssd1306.h"

#define I2C_SDA 14
#define I2C_SCL 15

#define NUM_SAMPLES 100      // Número de amostras por ciclo de leitura

uint16_t adc_buffer[NUM_SAMPLES]; // Buffer para armazenar as amostras
do ADC

struct repeating_timer timer; // Timer para controle de tempo

float avg_temp = 0.0f;
volatile bool dma_complete = false;
int dma_chan;

uint8_t ssd[ssd1306_buffer_length];

// Preparar área de renderização para o display (ssd1306_width pixels
por ssd1306_n_pages páginas)
struct render_area frame_area = {
    start_column : 0,
```

```
    end_column : ssd1306_width - 1,
    start_page : 0,
    end_page : ssd1306_n_pages - 1
};

void dma_handler() {
    dma_hw->ints0 = 1u << dma_chan;
    dma_complete = true;

    // Reinicia o DMA
    dma_channel_set_write_addr(dma_chan, adc_buffer, true);
}

// Converte o valor bruto do ADC (12 bits) para temperatura em graus Celsius
float convert_to_celsius(uint16_t raw) {
    const float conversion_factor = 3.3f / (1 << 12); // Fator de conversão para 3.3V e 12 bits
    float voltage = raw * conversion_factor;          // Converte valor para tensão
    return 27.0f - (voltage - 0.706f) / 0.001721f;     // Fórmula do datasheet do RP2040
}

// Função de callback para o timer
bool timer_callback(struct repeating_timer *t) {
    if (dma_complete) {
        float sum = 0.0f;
        for (int i = 0; i < NUM_SAMPLES; i++) {
            sum += convert_to_celsius(adc_buffer[i]);
        }
        avg_temp = sum / NUM_SAMPLES;
        dma_complete = false;

        char temp_str[16];
        sprintf(temp_str, "%.2f C", avg_temp);

        char *text[] = {"Temp Atual:", temp_str};
        int y = 0;
        for (uint i = 0; i < 2; i++) {
            ssd1306_draw_string(ssd, 5, y, text[i]);
            y += 10;
        }
```

```
        render_on_display(ssd, &frame_area);
    }
    return true;
}

int main() {
    stdio_init_all();   // Inicializa a saída padrão (USB serial)
    sleep_ms(2000);     // Aguarda 2 segundos para estabilizar o
terminal serial

    // Processo de inicialização completo do OLED SSD1306
    printf("Iniciando o display OLED SSD1306...\n");
    // Inicialização do i2c
    i2c_init(i2c1, ssd1306_i2c_clock * 1000);
    gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
    gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
    gpio_pull_up(I2C_SDA);
    gpio_pull_up(I2C_SCL);

    // Processo de inicialização completo do OLED SSD1306
    ssd1306_init();

    calculate_render_area_buffer_length(&frame_area);

    // zera o display inteiro
    memset(ssd, 0, ssd1306_buffer_length);
    render_on_display(ssd, &frame_area);
    printf("Display OLED SSD1306 inicializado com sucesso!\n");

    printf("Iniciando o ADC...\n");

    // Inicializa o ADC e habilita o sensor de temperatura interno
    adc_init();
    adc_set_temp_sensor_enabled(true);
    adc_select_input(4); // Canal 4 é o sensor de temperatura interna
do RP2040

    adc_fifo_setup(
            true,   // Envia dados para o FIFO
            true,   // Habilita DMA para o FIFO
            1,      // Gatilho a cada amostra
            false,
            false
```

```
    );

    printf("ADC inicializado com sucesso!\n");
    printf("Iniciando DMA...\n");

    // Configura o canal DMA para receber dados do ADC
    dma_chan =  dma_claim_unused_channel(true); // Requisita um canal
DMA disponível
    dma_channel_config cfg = dma_channel_get_default_config(dma_chan);
// Obtem configuração padrão

    // Configurações do canal DMA
    channel_config_set_transfer_data_size(&cfg, DMA_SIZE_16);  // Cada
leitura é de 16 bits
    channel_config_set_read_increment(&cfg, false);            //
Endereço fixo (registrador ADC FIFO)
    channel_config_set_write_increment(&cfg, true);            //
Incrementa para armazenar em adc_buffer[]
    channel_config_set_dreq(&cfg, DREQ_ADC);                   //
Dispara automaticamente com dados do ADC

    dma_channel_configure(dma_chan, &cfg, adc_buffer, &adc_hw->fifo,
NUM_SAMPLES, true);
    dma_channel_set_irq0_enabled(dma_chan, true);

    irq_set_exclusive_handler(DMA_IRQ_0, dma_handler);
    irq_set_enabled(DMA_IRQ_0, true);

    adc_run(true);

    printf("DMA configurado com sucesso!\n");

    add_repeating_timer_ms(500, timer_callback, NULL, &timer); //
Inicia o timer

    while (true) {}
}
```