Rorras neves da silva / 20251RSE.MTC0099

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.

```c
#include <stdio.h>
#include <string.h>
#include "pico/stdlib.h"
#include "hardware/adc.h"
#include "hardware/dma.h"
#include "font.h"
#include "ssd1306.h" // Use sua lib SSD1306 preferida

#define I2C_PORT i2c1 // I2C port (A i2c0 nao funcionou nesse codigo)
#define I2C_SDA 14
#define I2C_SCL 15
#define ADC_CHANNEL 4
#define SAMPLE_COUNT 1 // Número de amostras por ciclo de leitura, pois não será impresso a media
#define LARGURA_DA_TELA 128
#define ALTURADA_TELA 64

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

ssd1306_t ssd; // Instância do display OLED
int dma_chan;

void i2c_init_display()
{
    i2c_init(I2C_PORT, 400 * 1000); // I2C Initialisation. Using it at 400Khz.
    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);

    ssd1306_init(&ssd, LARGURA_DA_TELA, ALTURADA_TELA, 0X3C, I2C_PORT); // Inicializa display
    ssd1306_clear(&ssd);
    ssd1306_draw_string(&ssd, 0, 0, 1, "Sistema");
    ssd1306_draw_string(&ssd, 0, 16, 1, "Iniciado"); // Feedback inicial
    ssd1306_show(&ssd);                              // Endereço padrão do SSD1306
    sleep_ms(1000);
}
void setup_adc_dma(uint chan)
{
    adc_init();
    adc_set_temp_sensor_enabled(true);
    adc_select_input(ADC_CHANNEL);

    adc_fifo_setup(
        true,  // Envia dados para o FIFO
        true,  // Habilita DMA para o FIFO
        1,     // Gatilho a cada amostra
        false, // no ERR
        false  // no 8-bit mode
    );
    adc_fifo_drain(); // Garante que o FIFO esteja vazio antes de começar

    // Configura o canal DMA para receber dados do ADC
    int dma_chan = dma_claim_unused_channel(true);                    // escolhe um canal livre
              // 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);

    dma_channel_configure(
        dma_chan,
        &cfg,
        adc_buffer,    // Endereço de destino na RAM
        &adc_hw->fifo, // Endereço de origem (registrador FIFO do ADC)
        SAMPLE_COUNT,  // Número de transferências (amostras)
        false          // não inicia imediatamente
    );
}
float convert_to_temperature(uint16_t raw)
{
    const float conversion_factor = 3.3f / (1 << 12);
    float voltage = raw * conversion_factor;
    return 27.0f - (voltage - 0.706f) / 0.001721f;
}

bool repeating_timer_callback(struct repeating_timer *t)
{
    char buffer[32];
    // === Captura da temperatura via ADC e DMA ===
    adc_run(true);                                  // Liga ADC para começar a amostrar          //
Inicia conversão ADC
    dma_channel_start(dma_chan);                    // Inicia transferência via DMA
    dma_channel_wait_for_finish_blocking(dma_chan); // Espera terminar
    adc_run(false);                                 // Para o ADC

    float temp_c = convert_to_temperature(adc_buffer[0]); // recebe o valor da temperatura ja
convertido

    snprintf(buffer, sizeof(buffer), "= %.2f C", temp_c);
    // === Atualização do conteúdo do display OLED ===
    ssd1306_clear(&ssd);
    ssd1306_draw_string(&ssd, 0, 0, 1, "Temperatura:");
    ssd1306_draw_string(&ssd, 0, 16, 3, buffer);
    ssd1306_show(&ssd);

}
int main()
{
    stdio_init_all();
    setup_adc_dma(dma_chan);
    i2c_init_display();
    sleep_ms(2000);
    adc_run(false); // Desliga ADC temporariamente

    struct repeating_timer timer;
    // Configura um temporizador repetitivo que chama a função 'repeating_timer_callback' a cada
segundo (1000 ms).
    add_repeating_timer_ms(1000, repeating_timer_callback, NULL, &timer);
    while (true)
    {
        tight_loop_contents();
        printf("Iniciado\n");
        sleep_ms(1000);
    }

    return 0;
}
```