Nome Aluno: Manoel Felipe Costa Furtado

Matrícula: 20251RSE.MTC0086

Atividade 05 – Referente ao capítulo 5 da unidade 01

Prazo dia 18/05/2025 as 23:59

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.

Nome do arquivo principal: “Atividade_05.c” → Na pasta src.

Na pasta lib tem os arquivos para display oled.

Vídeo mostrando o seu funcionamento
Link: https://youtube.com/shorts/3R1in6k1zhA?feature=share

Foto do funcionamento, mais detalhes no vídeo.

Código: Atividade_05.c

```c
/**
 * @file    Atividade_05.c
 * @brief   Sistema de Aquisição de Temperatura com DMA e Interface I2C em Microcontrolador RP2040
 *
 *          Este progrma tem como objetivo demonstar como utilizar:
 *             * ADC interno (canal 4) para ler o sensor de temperatura
 *             * Controlador DMA para transferir automaticamente as amostras do FIFO do ADC para a RAM
 *             * Barramento I2C para atualizar um display OLED SSD1306
 *
 *          O ciclo de trabalho é:
 *             1. O ADC é configurado para amostragem contínua do canal 4.
 *             2. Um canal de DMA lê amostras(SAMPLES) do FIFO e armazena em um buffer.
 *             3. Quando o bloco de DMA termina, calculamos a média, convertemos em °C e
 *                enviamos o valor formatado para o display.
 *             4. O DMA é usado sem intervenção intensiva da CPU, permitindo que o núcleo possa
 *                ficar em espera reduzindo consumo.
 *
 * @note     Também foi esclarecida a razão de se utilizar `memset()` +
 *           `render_on_display()` em vez dos atalhos `ssd1306_clear()` e
 *           `ssd1306_show()`. O uso direto do frame-buffer permite:
 *              1. Renderizar apenas regiões específicas da tela, economizando
 *                 tráfego I²C quando necessário;
 *              2. Manter a lógica didática de *clear → draw → show* explícita;
 *              3. Suportar múltiplos frame-buffers simultâneos caso o projeto
 *                 evolua para animações ou janelas sobrepostas.
 *           Caso deseje simplificar, basta substituir o bloco marcado em
 *           @ref display_temperature() pelos atalhos da biblioteca.
 *
 *           Não foi tratado os caracteres especiais que apace no display OLED então, a temperatura médida a exemplo
 *           26 55 c --> 26,55 ºc
 *           A blibioteca atual ssd1306_font.h e ssd1306_i2c.c só reconhece acaracteres simples maiúsculas.
 *
 * @hardware     Requisitos de hardware:
 *                  • Raspberry Pi Pico / Pico W
 *                  • Display OLED SSD1306 conectado aos pinos I2C:
 *                      – SDA → GP14   (GPIO 14)
 *                      – SCL → GP55   (GPIO 15)
 *
 * @compile      Compilação (exemplo de CMake):
 *                  add_executable(Atividade_05
 *                          src/Atividade_05.c
 *                          lib/ssd1306_i2c.c
 *                      )
 *
 *          IMPORTANTE:
 *              • Todas as funções estão extensivamente comentadas, estilo Doxygen para fins didáticos.
 *              • O código assume Vref = 3,3 V (padrão do RP2040) para a conversão do ADC.
```

```
 *              • Ajuste SAMPLES e TEMP_UPDATE_MS conforme necessidade.
 *
 * @author  Manoel Furtado
 * @date    18 mai 2025
 * @copyright 2025 Manoel Furtado (MIT License) (veja LICENSE.md)
 */

#include <stdio.h>
#include <string.h>
#include "pico/stdlib.h"

#include "hardware/adc.h"
#include "hardware/dma.h"
#include "hardware/i2c.h"

#include "ssd1306_i2c.h"
#include "ssd1306.h"
#include "ssd1306_font.h"

/* ----------------------- GPIO Mapping ------------------------- */
/**
 * @name  OLED I²C Mapping
 * Pinos utilizados pelo barramento I²C dedicado ao display.
 * @{ */
#define OLED_I2C_SDA_PIN   14  /**< GPIO que carrega o sinal SDA  */
#define OLED_I2C_SCL_PIN   15  /**< GPIO que carrega o sinal SCL  */
/** @} */

/* --------------------------- Parâmetros do sistema ------------------------- */
#define SAMPLES             100         /**< Nº de amostras que compõem cada bloco de DMA */
#define TEMP_UPDATE_MS      500         /**< Intervalo (ms) entre atualizações do display */

#define I2C_PORT            i2c1        /**< Porta I²C utilizada pinos GP4/GP5 no Pico */
#define I2C_BAUDRATE        400000      /**< 400 kHz – modo Fast I²C                   */

/* ---------------------- Protótipos de funções ------------------- */
static float adc_to_celsius(uint16_t raw);
static void  init_display(void);
static void  init_adc(void);
static void  init_dma(void);
static inline void restart_dma(void);
static void  display_temperature(float temp_c);

/* --------------------------- Buffers e variáveis -------------------------- */
static uint16_t adc_buffer[SAMPLES];    /**< Destino de cada transferência DMA         */
static int dma_chan;                    /**< Canal DMA alocado                          */

/**
```

```c
 * Buffer de vídeo (frame-buffer) para o display – 1 byte por página x largura.
 * O tamanho é definido pela própria biblioteca SSD1306. */
static uint8_t fb[ssd1306_buffer_length] = {0};

/** Área que cobre toda a tela (0,0 -> 127,63) */
static struct render_area full_area = {
    .start_column = 0,
    .end_column   = ssd1306_width  - 1,
    .start_page   = 0,
    .end_page     = ssd1306_n_pages - 1,
};

/* ------------------------- Funções auxiliares ------------------------- */

/**
 * @brief Converte um valor digital do ADC (12 bits) para temperatura em °C.
 *
 * Fórmula do datasheet do RP2040:
 *      T = 27 – (Vout – 0,706) / 0,001721
 * onde Vout = ADC * Vref / 4096
 *
 * @f[ T = 27 – \frac{V_{out} – 0{,}706}{0{,}001721} @f]
 * onde @f$ V_{out} = ADC_{raw} \times V_{ref} / 4096 @f$.
 *
 * @param raw Valor bruto do ADC (0–4095)
 * @return Temperatura em graus Celsius (float)
 */
static float adc_to_celsius(uint16_t raw)
{
    const float VREF         = 3.3f;      // Tensão de referência do ADC
    const float CONVERT      = VREF / 4096.0f; /* Tensão por LSB (12 bits)            */

    float voltage    = (float)raw * CONVERT;   // Converte valor para tensão
    float temperature = 27.0f - (voltage - 0.706f) / 0.001721f;  // Fórmula do datasheet do RP2040
    return temperature;
}

/**
 * @brief Inicializa o barramento I²C e o display SSD1306.
 *
 * Nesta aplicação mantemos um frame-buffer local (@c fb) e utilizamos
 * `render_on_display()` para enviá-lo. Poderíamos, de maneira equivalente,
 * empregar `ssd1306_clear()` + `ssd1306_show()`, mas preferimos o fluxo
 * explícito para evidenciar os passos envolvidos. (Ver @ref display_temperature). */
static void init_display(void)
{
    // Configura I2C1
    i2c_init(I2C_PORT, I2C_BAUDRATE);                    // Clock = 400 kHz
```

```
    gpio_set_function(OLED_I2C_SDA_PIN, GPIO_FUNC_I2C);  // SDA
    gpio_set_function(OLED_I2C_SCL_PIN, GPIO_FUNC_I2C);  // SCL
    gpio_pull_up(OLED_I2C_SDA_PIN);
    gpio_pull_up(OLED_I2C_SCL_PIN);

    // Inicializa hardware do display
    ssd1306_init();

    // Calcula comprimento do buffer de renderização completo
    calculate_render_area_buffer_length(&full_area);

    // Limpa memória do framebuffer e exibe tela em branco
    memset(fb, 0, sizeof(fb));
    render_on_display(fb, &full_area);
    /* Alternativa simplificada:
     *     ssd1306_clear();
     *     ssd1306_show();
    */
}

/**
 * @brief Configura ADC (canal 4) e inicia conversão contínua.
 */
static void init_adc(void)
{
    adc_init();                          // Habilita bloco ADC
    adc_set_temp_sensor_enabled(true);  // Conecta sensor de temperatura ao canal 4
    adc_select_input(4);                // Seleciona canal 4

    /* FIFO configurado para amostragem contínua:
     *  - shift left = false (mantém valor de 12 bits)
     *  - dreq_en    = true  (gera solicitação DMA)
     *  - dreq_thresh= 1     (quando ≥1 amostra FIFO aciona DREQ)
     */
    adc_fifo_setup(
        true,   // enable FIFO
        true,   // enable DMA data request (DREQ)
        1,      // DREQ quando >=1
        false,  // disable ERR bit
        false   // não deslocar bits
    );

    adc_run(true);   // Inicia conversão contínua
}

/**
 * @brief Configura canal DMA para transferir SAMPLES valores do FIFO do ADC
 *        para @ref adc_buffer. Após a conclusão, o canal gera interrupção
```

```
 *        (opcional) ou podemos bloquear aguardando no laço principal.
 *
 *        A transferência é de 16 bits porque o registrador FIFO fornece os 12 bits
 *        válidos alinhados à direita. O endereço de leitura é fixo (FIFO) e o de
 *        escrita é incrementado dentro de @c adc_buffer.
 */
static void init_dma(void)
{
    /* Obtém canal livre e configuração padrão */
    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

    /* -------------------- Ajustes específicos --------------------------- */
    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) - FIFO:
endereço fixo
    channel_config_set_write_increment (&cfg, true);          // Incrementa para armazenar em adc_buffer[] - Buffer:
índice ++
    channel_config_set_dreq            (&cfg, DREQ_ADC);      // Acionado pelo ADC - Dispara automaticamente com
dados do ADC

    /* Configura sem iniciar ainda (start=false) */
    dma_channel_configure(
        dma_chan,
        &cfg,
        adc_buffer,                // Destino
        &adc_hw->fifo,             // Origem
        SAMPLES,                   // Quantidade de transferências
        false                      // Não iniciar agora
    );

    /* ---- Opcional: Pode configurar interrupção, mas para simplicidade
     *      o loop principal aguardará a conclusão com
     *          dma_channel_wait_for_finish_blocking()               ---- */
}

/**
 * @brief Configura o canal DMA para o próximo bloco de @c SAMPLES amostras.
 */
static inline void restart_dma(void)
{
    /* Reinicia contagem e recomeça imediatamente */
    dma_channel_set_read_addr  (dma_chan, &adc_hw->fifo, false);
    dma_channel_set_write_addr (dma_chan, adc_buffer,   false);
    dma_channel_set_trans_count(dma_chan, SAMPLES, true);
}

/**
```

```
 * @brief Atualiza o display com a temperatura fornecida.
 *
 * @param temp_c Temperatura em graus Celsius
 */
static void display_temperature(float temp_c) {
    char line1[] = "Manoel Ativ 05";
    char line2[] = "";
    char line3[] = "Temperatura";
    char line4[] = "Media";
    char line5[32];
    snprintf(line5, sizeof(line5), "       %.2f C", temp_c);



    // Limpa framebuffer, desenha texto nas coordenadas (0,0)
    memset(fb, 0, sizeof(fb)); // ou ssd1306_clear(); já explicado



    // Desenha cada linha em pixels (altura da fonte = 8 px)
    ssd1306_draw_string(fb, 0, 0,  line1);  // primeira linha
    ssd1306_draw_string(fb, 0, 8,  line2);  // segunda linha (8 px abaixo)
    ssd1306_draw_string(fb, 0, 16, line3);  // terceira linha (16 px abaixo)
    ssd1306_draw_string(fb, 0, 24, line4);  // quarta linha (24 px abaixo)
    ssd1306_draw_string(fb, 0, 32, line5);  // quinta linha (32 px abaixo)

    // Envia buffer para o display
    render_on_display(fb, &full_area); // ou ssd1306_show(); já explicado
}

/* ------------------------------ Função main ------------------------------ */
int main(void)
{
    stdio_init_all();    // Para debug via USB (pode ser removido em produção)

    init_display();      // I2C + OLED
    init_adc();          // ADC contínuo no canal 4 (sensor interno)
    init_dma();          // Canal DMA configurado

    /* Primeira execução de DMA */
    restart_dma();

    while (true)
    {
        /* Aguarda DMA preencher o buffer (modo bloqueante, mas CPU pode
         * ser colocada em espera se desejar).
         * Bloqueia até o DMA preencher o buffer. Em projetos mais avançados
         * pode-se colocar o núcleo em `sleep_until()` ou usar IRQs. */
        dma_channel_wait_for_finish_blocking(dma_chan);
```

```
        /* -------------------- Processa amostras ------------------------- */
        uint32_t sum = 0;
        for (size_t i = 0; i < SAMPLES; ++i)
            sum += adc_buffer[i] & 0x0FFF;   // 12 bits válidos

        uint16_t raw_avg = sum / SAMPLES;    // Média aritmética
        float temperature = adc_to_celsius(raw_avg);

        display_temperature(temperature);

        /* Prepara o próximo ciclo - para novo bloco e aguarda próximo período */
        restart_dma();
        sleep_ms(TEMP_UPDATE_MS);
    }
}
```