Aluno do Embarcatech_37 no IFMA

Nome: Manoel Felipe Costa Furtado

Matrícula: 20251RSE.MTC0086

Atividade 09 – Referente ao capítulo 9 da unidade 01 – Executor Cíclico

Prazo dia 25/05/2025 as 23:59

Enunciado: Complementação do Projeto TempCycleDMA. Não foi utilizado outra estratégia para gerenciar o tempo de execução das tarefas em função da Tarefa 1, considerara a principal.

Qual a melhoria que deve ser realizada no novo projeto:

Sincronizar as tarefas em função da primeira utilizado add_repeating_timer_ms nas demais tarefas e repeating_timer_callback para a tarefa 1.

- Nome do arquivo principal: “Atividade_09.c” → Na pasta src.
- Na pasta lib tem os arquivos sobre o display OLED, Matriz de Leds etc.
- Vídeo mostrando o seu funcionamento
  Link: https://youtu.be/pMi5-I88c6Q
- GitHub:
  https://github.com/ManoelFelipe/Embarcatech_37/tree/main/Unidade_01/Cap_09/Atividade_09
- Drive: Atividade_09.zip

O que mudou:

- Removido o laço while com chamadas diretas.
- Criados 5 temporizadores: 1× repeating_timer_callback (T1) e 4 × add_repeating_timer_ms (T5 → T4).
- Implementados callbacks que apenas sinalizam a execução via flags; o processamento pesado ocorre no loop principal, evitando travar a IRQ.
- Mantida a ordem tarefa_1 → tarefa_5 → tarefa_2 → tarefa_3 → tarefa_4 com pequenos offsets (10 ms, 20 ms, ... 40 ms) em relação ao disparo da Tarefa 1.
- Impressão de diagnóstico via USB e alimentação do watchdog a cada ciclo.
- Período do ciclo: 1 000 ms (definido em PERIODO_CICLO_MS).
- Basta ajustar esse valor se precisar de outra cadência.

Como funciona agora

- T1 dispara a cada segundo (timer_cb_t1).
- No fim do callback, são criados alarmes únicos que, após 10/20/30/40 ms, levantam as flags das demais tarefas — garantindo a sequência.
- O loop principal verifica as flags, executa a tarefa correspondente e volta ao modo ocioso (tight_loop_contents()).

O projeto está dividido em três partes principais:

1. CMakeLists.txt
   Define como compilar o projeto. Mostra onde estão os arquivos (src/ e lib/) e quais bibliotecas usar (como o SDK do Raspberry Pi Pico).
2. Arquivos .h (headers)
   São os arquivos que dizem o que cada módulo faz. Mostram as funções disponíveis, constantes e tipos. É por onde você entende o que dá pra usar de cada parte do código.
3. Arquivos .c (implementações)
   Explicam como cada coisa funciona. Aqui estão os detalhes: como controlar os LEDs, como desenhar no OLED, como lidar com timers, etc.

Pastas principais:

- lib/LabNeoPixel/: cuida dos LEDs endereçáveis (WS2812) com PIO.
- lib/ssd1306/: controla o display OLED via I2C.
- src/: é onde está a lógica principal da aplicação, incluindo as tarefas e o main().

Como entender o projeto:

1. Comece pelo main() em Atividade_09.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 pra facilitar.

- Código: src/Atividade_09.c

```c
/**
 * @file Atividade_09.c
 * @brief Comentários detalhados sobre o arquivo.
 * @details Este arquivo faz parte do projeto Atividade_09. Contém implementações e definições
 *          relacionadas à funcionalidade do módulo representado pelo caminho `src/Atividade_09.c`.
 *          Todos os comentários seguem o padrão Doxygen
 * @author  Manoel Furtado
 * @date    25 maio 2025
 * @copyright 2025 Manoel Furtado (MIT License) (veja LICENSE.md)
 */
/**
 * ------------------------------------------------------------
 *  Arquivo: Atividade_09.c
 *  Projeto: TempCycleDMA (versão sincronizada por timers)
 * ------------------------------------------------------------
 *  Descrição:
 *      Implementação revisada para utilizar temporizadores
 *      repetitivos (hardware alarm pool do RP2040) e sincronizar
 *      todas as tarefas em função da Tarefa 1 (principal).
 *
 *      ✓  Tarefa 1  - Leitura da temperatura (callback timer →
 *                     repeating_timer_callback)
 *      ✓  Demais    - Executadas por add_repeating_timer_ms,
 *                     sincronizadas ao mesmo período da 1ª.
 *
 *      Ordem garantida em cada ciclo:
 *          1️⃣ tarefa_1()
 *          2️⃣ tarefa_5()
 *          3️⃣ tarefa_2()
 *          4️⃣ tarefa_3()
 *          5️⃣ tarefa_4()
 *
 *      NOTA IMPORTANTE
 *      ---------------
 *      Callbacks de temporizador na SDK rodarão em contexto
 *      de interrupção. Como cada tarefa pode levar dezenas de
 *      milissegundos, optamos por SINALIZAR a execução no
 *      callback (setando flags) e processar de fato no laço
 *      principal. Assim evitamos bloqueios longos em IRQ.
 *
 *      Data da revisão: 25/05/2025
 * ------------------------------------------------------------
 */

#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/watchdog.h"
```

```c
#include "pico/time.h"

#include "setup.h"
#include "tarefa1_temp.h"
#include "tarefa2_display.h"
#include "tarefa3_tendencia.h"
#include "tarefa4_controla_neopixel.h"
#include "neopixel_driver.h"
#include "testes_cores.h"

// ---------- Constantes do escalonador ----------
#define PERIODO_CICLO_MS      1000    // 1 s entre execuções
#define OFFSET_T5_MS            10    // delay após T1
#define OFFSET_T2_MS            20
#define OFFSET_T3_MS            30
#define OFFSET_T4_MS            40

// ---------- Flags de execução (setadas nos timers) ----------
static volatile bool run_t1 = false;
static volatile bool run_t2 = false;
static volatile bool run_t3 = false;
static volatile bool run_t4 = false;
static volatile bool run_t5 = false;

// ---------- Protótipos ----------
static bool timer_cb_t1(struct repeating_timer *t);
static bool timer_cb_generic(struct repeating_timer *t);

// Estruturas dos temporizadores
static struct repeating_timer timer_t1;
static struct repeating_timer timer_t5;
static struct repeating_timer timer_t2;
static struct repeating_timer timer_t3;
static struct repeating_timer timer_t4;


// --- Alarm callbacks usados para defasar a execução das tarefas 5→4 ---
/**
 * @brief Descrição da função alarm_cb_t5.
 *
 * @details Explique aqui a lógica da função, parâmetros de entrada e o que ela retorna.
 * @param id Descrição do parâmetro id.
 * @param user_data Descrição do parâmetro user_data.
 * @return Valor de retorno descrevendo o significado.
 */
static int64_t alarm_cb_t5(alarm_id_t id, void *user_data) {
    run_t5 = true;
    return 0;
}
```

```c
/**
 * @brief Descrição da função alarm_cb_t2.
 *
 * @details Explique aqui a lógica da função, parâmetros de entrada e o que ela retorna.
 * @param id Descrição do parâmetro id.
 * @param user_data Descrição do parâmetro user_data.
 * @return Valor de retorno descrevendo o significado.
 */
static int64_t alarm_cb_t2(alarm_id_t id, void *user_data) {
    run_t2 = true;
    return 0;
}
/**
 * @brief Descrição da função alarm_cb_t3.
 *
 * @details Explique aqui a lógica da função, parâmetros de entrada e o que ela retorna.
 * @param id Descrição do parâmetro id.
 * @param user_data Descrição do parâmetro user_data.
 * @return Valor de retorno descrevendo o significado.
 */

static int64_t alarm_cb_t3(alarm_id_t id, void *user_data) {
    run_t3 = true;
    return 0;
}
/**
 * @brief Descrição da função alarm_cb_t4.
 *
 * @details Explique aqui a lógica da função, parâmetros de entrada e o que ela retorna.
 * @param id Descrição do parâmetro id.
 * @param user_data Descrição do parâmetro user_data.
 * @return Valor de retorno descrevendo o significado.
 */
static int64_t alarm_cb_t4(alarm_id_t id, void *user_data) {
    run_t4 = true;
    return 0;
}
// Variáveis globais usadas pelas tarefas antigas
float media = 0.0f;
tendencia_t t = TENDENCIA_ESTAVEL;

// Marcação de tempo p/ diagnóstico
absolute_time_t ini_tarefa1, fim_tarefa1;
absolute_time_t ini_tarefa2, fim_tarefa2;
absolute_time_t ini_tarefa3, fim_tarefa3;
absolute_time_t ini_tarefa4, fim_tarefa4;

// ----------------------------------------------------------------
```

```c
//                      Funções de Callback
// ----------------------------------------------------------------
/**
 * Callback principal (Tarefa 1).
 * Retorna true para manter o timer ativo.
 */
/**
 * @brief Descrição da função timer_cb_t1.
 *
 * @details Explique aqui a lógica da função, parâmetros de entrada e o que ela retorna.
 * @param t Descrição do parâmetro t.
 * @return Valor de retorno descrevendo o significado.
 */
static bool timer_cb_t1(struct repeating_timer *t) {
    run_t1 = true;

    // Agenda flags das demais tarefas na sequência desejada.
    // Cada flag é armada com pequenos offsets através de alarmas
    // únicos (add_alarm_in_ms) para manter ordem.
    // Opcional: poderíamos simplesmente setar as flags já aqui
    //           e confiar no laço principal, mas a abordagem a
    //           seguir mantém espaçamento mínimo entre tarefas.
    add_alarm_in_ms(OFFSET_T5_MS, alarm_cb_t5, NULL, true);

    add_alarm_in_ms(OFFSET_T2_MS, alarm_cb_t2, NULL, true);

    add_alarm_in_ms(OFFSET_T3_MS, alarm_cb_t3, NULL, true);

    add_alarm_in_ms(OFFSET_T4_MS, alarm_cb_t4, NULL, true);

    return true; // repetir
}

/**
 * Fallback genérico (NÃO USADO NESTA VERSÃO)
 * Foi mantido apenas como referência caso queira realizar
 * todo o processamento dentro dos callbacks. Optamos por usar
 * flags + laço principal para evitar longas execuções em IRQ.
 */
/**
 * @brief Descrição da função timer_cb_generic.
 *
 * @details Explique aqui a lógica da função, parâmetros de entrada e o que ela retorna.
 * @param t Descrição do parâmetro t.
 * @return Valor de retorno descrevendo o significado.
 */

static bool timer_cb_generic(struct repeating_timer *t) {
```

```
    // Nunca entra aqui
    return true;
}

// ----------------------------------------------------------------
//                            main()
// ----------------------------------------------------------------
/**
 * @brief Descrição da função main.
 *
 * @details Explique aqui a lógica da função, parâmetros de entrada e o que ela retorna.
 * @return Valor de retorno descrevendo o significado.
 */
int main(void) {
    stdio_init_all();
    setup();  // ADC, DMA, OLED, etc.

    // Watchdog opcional
    watchdog_enable(3000, false);

    // ---------------- Configuração dos Timers ----------------
    // Tarefa 1 - repeating_timer_callback
    add_repeating_timer_ms(PERIODO_CICLO_MS, timer_cb_t1, NULL, &timer_t1);

    // Demais tarefas — criamos timers apenas para manter
    // referência; o trabalho real é disparado via flags.
    add_repeating_timer_ms(PERIODO_CICLO_MS, timer_cb_generic, NULL, &timer_t5);
    add_repeating_timer_ms(PERIODO_CICLO_MS, timer_cb_generic, NULL, &timer_t2);
    add_repeating_timer_ms(PERIODO_CICLO_MS, timer_cb_generic, NULL, &timer_t3);
    add_repeating_timer_ms(PERIODO_CICLO_MS, timer_cb_generic, NULL, &timer_t4);

    // ---------------- Loop Principal ----------------
    while (true) {
        tight_loop_contents();   // baixa-prioridade RP2040

        if (run_t1) {
            run_t1 = false;
            ini_tarefa1 = get_absolute_time();
            media = tarefa1_obter_media_temp(&cfg_temp, DMA_TEMP_CHANNEL);          // ajustado para retornar
média
            fim_tarefa1 = get_absolute_time();
        }

        if (run_t5) {
            run_t5 = false;
            // Mantém sem bloqueio: usa piscar neopixel quando
            // temperatura < 1°C como no código original
            if (media < 1.0f) {
```

```
                npSetAll(COR_BRANCA);
                npWrite();
                sleep_ms(1000);
                npClear();
                npWrite();
            }
        }

        if (run_t2) {
            run_t2 = false;
            ini_tarefa2 = get_absolute_time();
            tarefa2_exibir_oled(media, t);
            fim_tarefa2 = get_absolute_time();
        }

        if (run_t3) {
            run_t3 = false;
            ini_tarefa3 = get_absolute_time();
            t = tarefa3_analisa_tendencia(media);
            fim_tarefa3 = get_absolute_time();
        }

        if (run_t4) {
            run_t4 = false;
            ini_tarefa4 = get_absolute_time();
            tarefa4_matriz_cor_por_tendencia(t);
            fim_tarefa4 = get_absolute_time();
        }

        // Alimente watchdog a cada iteração
        watchdog_update();

        // ---------- Diagnóstico via USB ----------
        static uint32_t last_print = 0;
        uint32_t agora = to_ms_since_boot(get_absolute_time());
        if (agora - last_print >= 1000) {
            last_print = agora;
            printf("📎  %.2f °C | Tend: %s \n",
                   media,
                   tendencia_para_texto(t));
        }
    }

    return 0;
}
```