Nome Aluno: Manoel Felipe Costa Furtado

Matrícula: 20251RSE.MTC0086

Turma: 20251.1.RSE.MTC.1317.1N

Atividade 01 – Prazo dia 29/04/2025 as 23:59

Enunciado: Estação de Monitoramento Interativo

Criar um sistema de dois núcleos que simula uma estação de monitoramento com sensores e atuadores. Os núcleos se comunicam entre si utilizando FIFO, compartilham estado com flags e usam alarmes para executar tarefas periódicas.

Código:

```
/**
 * @file   Atividade_01_commented.c
 * @brief  Estação de Monitoramento Interativo – exemplo multicore no RP2040
 *
 * ──────────────────────────────────────────────────────────────────────────
 * Este programa demonstra como usar os DOIS núcleos do RP2040 para criar uma
 * mini “estação de monitoramento” que:
 *
 * • Core 0
 *   – Lê **apenas o eixo X** (ADC-1) de um joystick analógico;
 *   – Converte o valor lido em três níveis (baixo / moderado / alto);
 *   – Atualiza uma flag global (`system_state`);
 *   – Envia o estado pela FIFO a cada 2s (callback do alarme).
 *
 * • Core 1
 *   – Bloqueia na FIFO aguardando um novo estado.
 *   – Ajusta a cor do LED RGB e o buzzer conforme a tabela:
 *      1 – VERDE   – Atividade Baixa
 *      2 – AZUL    – Atividade Moderada
 *      3 – VERMELHO + BUZZER – Atividade Alta/Critica
 *
 * • Requisitos atendidos
 *   – Uso de variável `volatile` para partilha de estado.
 *   – add_alarm_in_ms() com período de 2 000 ms (Core 0).
 *   – Comunicação FIFO (queue_t) entre os núcleos.
 *   – Limiares configuráveis para o joystick.
 *
 * Hardware sugerido
 * ─────────────────
```

```
 *  GPIO-13  – LED vermelho
 *  GPIO-11  – LED verde
 *  GPIO-12  – LED azul
 *  GPIO-21  – Buzzer (pino A, via transistor)
 *  GPIO-10  – Buzzer (pino B, opcional/complementar)
 *  GPIO-27  – Joystick eixo X (ADC 1)
 *  GPIO-26  – Joystick eixo Y (ADC 0) – lido só para debug
 *
 * Compilação
 * ──────────
 *  • Adicione este arquivo ao seu CMakeLists.txt.
 *  • Compile com o SDK do Raspberry Pi Pico W.
 *
 * PWM no Buzzer
 * ─────────────────────────────
 * O acionamento ON/OFF do buzzer pelo
 * periférico PWM do RP2040, permite gerar tons audíveis em qualquer
 * frequência (≈ 20 Hz – 20 kHz), limitada apenas pela resolução do PWM
 * e pela resposta do piezo/alto-falante empregado.
 *
 * Como escolher a frequência?
 * (1) Compilação: altere o valor de `DEFAULT_BUZZER_FREQ_HZ` abaixo.
 * (2) Execução: chame `set_buzzer_frequency(novafreq)` a qualquer momento
 *     (por exemplo, dentro do `core1_main()` antes de ligar o buzzer,
 *      ou em um comando vindo da UART, etc.).
 *
 * Histórico
 * ─────────
 *  29-abr-2025 – Manoel F. C. Furtado
 */

/* ──────────────────────────────────────
 *  INCLUDES
 * ──────────────────────────────────────*/
#include <stdio.h>
#include "pico/stdlib.h"
#include "pico/multicore.h"
#include "pico/util/queue.h"
#include "hardware/adc.h"
#include "hardware/gpio.h"
#include "hardware/pwm.h"     // PWM
#include "hardware/clocks.h"  // clock_get_hz()

/* ╔══════════════════════════════════╗
 * ║  DEFINIÇÕES DE HARDWARE          ║
 * ╚══════════════════════════════════╝*/
#define LED_RED_PIN     13      // LED vermelho
#define LED_GREEN_PIN   11      // LED verde
```

```c
#define LED_BLUE_PIN    12      // LED azul

#define BUZZER_PIN_A    21      // Buzzer – lado A  (PWM)
#define BUZZER_PIN_B    10      // Buzzer – lado B  (opcional / fase oposta)

#define JOYSTICK_X_PIN  27      // ADC-1 – eixo X
#define JOYSTICK_Y_PIN  26      // ADC-0 – eixo Y (debug)

/* ╔═══════════════════════════════════╗
 * ║  PARÂMETROS DO PWM DO BUZZER      ║
 * ╚═══════════════════════════════════╝*/
#define DEFAULT_BUZZER_FREQ_HZ  2000u   // 2 kHz inicial – altere à vontade
#define BUZZER_PWM_DUTY         0.50f   // 50 % duty-cycle (onda quadrada)

/* ╔═══════════════════════════════════╗
 * ║  SISTEMA DE ESTADOS               ║
 * ╚═══════════════════════════════════╝*/
typedef enum {
    STATE_LOW = 1,      // Atividade Baixa    → LED verde
    STATE_MODERATE,     // Atividade Moderada → LED azul
    STATE_HIGH          // Atividade Alta     → LED vermelho + buzzer
} system_state_t;

/* Limiar inferior / superior do ADC (0-4095). Ajuste se necessário. */
#define LOW_THRESHOLD       1365    // 1/3 do range (4095/3)
#define MODERATE_THRESHOLD  2730    // 2/3 do range (4095*2/3)

/* ╔═══════════════════════════════════╗
 * ║  VARIÁVEIS GLOBAIS PARTILHADAS    ║
 * ╚═══════════════════════════════════╝*/
volatile uint8_t system_state = STATE_LOW;   // Flag de estado (visível a
ambos os cores)
queue_t state_fifo;                          // FIFO com 1 byte (uint8_t)

/* ╔═══════════════════════════════════╗
 * ║  PROTÓTIPOS DE FUNÇÃO             ║
 * ╚═══════════════════════════════════╝*/
static void  config_leds(void);          // Configura os LEDs
static void  config_buzzer(void);        // Configura os LEDs
static void  config_joystick(void);      // Configura os Joystick
static void  set_leds(uint8_t state);    // Seta os LEDs
static void  read_joystick(void);        // Lê e processa o Joystick
static void  core1_main(void);           // Uso do Core 1
static void  activate_buzzer(bool on);   // Ativação do Buzzer
static void  set_buzzer_frequency(uint32_t freq_hz);
static int64_t alarm_cb(alarm_id_t id, void *user_data);

/* ╔═══════════════════════════════════╗
```

```
 * ╠  FUNÇÃO PRINCIPAL – Core 0       ╣
 * ╚═══════════════════════════════════╝*/
int main(void)
{
    stdio_init_all();         // Inicializa USB CDC. UART para debug
    sleep_ms(2000);           // Aguarda USB estabilizar

    config_leds();
    config_buzzer();
    config_joystick();

    queue_init(&state_fifo, sizeof(uint8_t), 1);   // FIFO capacidade 1
    multicore_launch_core1(core1_main);            // Inicia Core 1

    /* Alarme periódico (2 s) → garante atualização mesmo parado */
    add_alarm_in_ms(2000, alarm_cb, NULL, true);

    /* Loop principal – lê joystick e converte em estado */
    while (true)
    {
        read_joystick();
        sleep_ms(40);     // ~25 Hz
    }
}

/* ╔═══════════════════════════════════╗
 * ╠  ALARME – Callback periódico     ╣
 * ╚═══════════════════════════════════╝*/
static int64_t alarm_cb(__unused alarm_id_t id, __unused void* user_data)
{
    queue_try_add(&state_fifo, (const void*)&system_state);   // “refresca”
Core 1
    return 2000;  // re-agenda em 2 s
}

/* ╔═══════════════════════════════════╗
 * ╠  CORE 1 – Thread de atuadores    ╣
 * ╚═══════════════════════════════════╝*/
static void core1_main(void)
{
    uint8_t incoming;
    printf("Core 1 ativo - aguardando estados…\n");

    while (true)
    {
        /* Bloqueia até receber novo estado */
        queue_remove_blocking(&state_fifo, &incoming);
```

```
        /* Atuação sobre LEDs */
        set_leds(incoming);

        /* Exemplo: alterar tom aqui, se desejar */
        /* if (incoming == STATE_HIGH)  set_buzzer_frequency(1000); */

        if (incoming == STATE_HIGH) {
            set_buzzer_frequency(1000);  // 1 KHz
            activate_buzzer(true);
        } else {
            activate_buzzer(false);
        }
        //activate_buzzer(incoming == STATE_HIGH);
    }
}

/* ╔═══════════════════════════════════════╗
 * ║  JOYSTICK - Configuração e leitura    ║                          ║
 * ╚═══════════════════════════════════════╝*/
static void config_joystick(void)
{
    adc_init();
    adc_gpio_init(JOYSTICK_X_PIN);
    adc_gpio_init(JOYSTICK_Y_PIN);  // apenas debug
    printf("Joystick configurado (X→ADC1/%d, Y→ADC0/%d)\n",
           JOYSTICK_X_PIN, JOYSTICK_Y_PIN);
}

/**
 * @brief Lê o eixo X, determina estado e envia à FIFO se mudou.
 * O eixo Y é lido apenas para depuração no console.
 */
static void read_joystick(void)
{
    /* ---------- Leitura eixo X (ADC-1) ---------- */
    adc_select_input(1);
    uint16_t raw_x = adc_read();

    /* ---------- Leitura eixo Y (ADC-0) - debug --- */
    adc_select_input(0);
    uint16_t raw_y = adc_read();

    /* ---------- Converte valor em estado ---------- */
    uint8_t new_state;
    if (raw_x < LOW_THRESHOLD)
        new_state = STATE_LOW;
    else if (raw_x < MODERATE_THRESHOLD)
        new_state = STATE_MODERATE;
```

```
    else
        new_state = STATE_HIGH;

    /* ---------- Atualiza flag / envia à FIFO se mudou ---------- */
    if (new_state != system_state)
    {
        system_state = new_state;
        queue_try_add(&state_fifo, (const void*)&system_state);
    }

    /* ---------- Log de depuração ---------- */
    printf("ADC  X=%4u  Y=%4u  → estado=%u\n", raw_x, raw_y, system_state);
}

/* ╔══════════════════════════════════╗
 * ║  LEDs                            ║
 * ╚══════════════════════════════════╝*/
static void config_leds(void)
{
    const uint led_pins[] = { LED_RED_PIN, LED_GREEN_PIN, LED_BLUE_PIN };

    for (size_t i = 0; i < 3; ++i)
    {
        gpio_init(led_pins[i]);
        gpio_set_dir(led_pins[i], GPIO_OUT);
        gpio_put(led_pins[i], 0);           // começa apagado
    }
    printf("LEDs prontos (R=%d, G=%d, B=%d)\n",
           LED_RED_PIN, LED_GREEN_PIN, LED_BLUE_PIN);
}

static void set_leds(uint8_t state)
{
    /* Apaga todos antes */
    gpio_put(LED_RED_PIN,   0);
    gpio_put(LED_GREEN_PIN, 0);
    gpio_put(LED_BLUE_PIN,  0);

    switch (state)
    {
        case STATE_LOW:      gpio_put(LED_GREEN_PIN, 1); break;
        case STATE_MODERATE: gpio_put(LED_BLUE_PIN,  1); break;
        case STATE_HIGH:     gpio_put(LED_RED_PIN,   1); break;
        default: /* nada */ break;
    }
}

/* ╔══════════════════════════════════╗
```

```
 * ╟  BUZZER - PWM                                  ╢
 * ╚══════════════════════════════════════════════╝*/
/**
 * @brief Configura o pino do buzzer como saída PWM (desabilitado).
 */
static void config_buzzer(void)
{
    /* ❸ Configura pino A como função PWM */
    gpio_set_function(BUZZER_PIN_A, GPIO_FUNC_PWM);
    uint slice = pwm_gpio_to_slice_num(BUZZER_PIN_A);

    /* Pino B continua como GPIO ON/OFF (fase oposta ou reforço) */
    gpio_init(BUZZER_PIN_B);
    gpio_set_dir(BUZZER_PIN_B, GPIO_OUT);
    gpio_put(BUZZER_PIN_B, 0);

    /* Define frequência padrão e duty-cycle */
    set_buzzer_frequency(DEFAULT_BUZZER_FREQ_HZ);

    pwm_set_enabled(slice, false);  // começa em silêncio

    printf("Buzzer PWM configurado (pino %d • freq. inicial %u Hz)\n",
           BUZZER_PIN_A, DEFAULT_BUZZER_FREQ_HZ);
}

/**
 * @brief Ajusta a frequência do buzzer em tempo de execução.
 * @param freq_hz Frequência desejada (Hz). Máx. ≈ 20 kHz
 *
 * Fórmula: `freq = clk_sys / (div * (TOP + 1))`,
 * onde `div` é o divisor (1 - 255, fracionário) e `TOP` (wrap) é o valor
 * máximo do contador PWM.
 */
static void set_buzzer_frequency(uint32_t freq_hz)
{
    uint slice = pwm_gpio_to_slice_num(BUZZER_PIN_A);

    /* Escolha de TOP - valor intermediário para boa resolução */
    const uint32_t top = 2499;           // TOP = 2499 → 2500 passos
    pwm_set_wrap(slice, top);            // define período

    /* Calcula divisor fracionário (float) */
    uint32_t clk = clock_get_hz(clk_sys);   // 125 MHz padrão
    float div = (float)clk / (freq_hz * (top + 1));

    /* Limita faixa aceita (SDK: 1 ≤ div ≤ 255) */
    if (div < 1.0f)   div = 1.0f;
    if (div > 255.0f) div = 255.0f;
```

```
    pwm_set_clkdiv(slice, div);

    /* Duty-cycle (50 %) */
    uint32_t level = (uint32_t)((top + 1) * BUZZER_PWM_DUTY);
    pwm_set_gpio_level(BUZZER_PIN_A, level);
}

/**
 * @brief Liga ou desliga o buzzer.
 *        Ao ligar, apenas habilita o slice (freq. já configurada).
 */
static void activate_buzzer(bool on)
{
    uint slice = pwm_gpio_to_slice_num(BUZZER_PIN_A);
    pwm_set_enabled(slice, on);

    /* Opcional: espelha nível no pino B */
    gpio_put(BUZZER_PIN_B, on ? 1 : 0);
}
```