Aluno do Embarcatech_37 no IFMA

Nome: Manoel Felipe Costa Furtado

Matrícula: 20251RSE.MTC0086

Atividade 10 – Referente ao capítulo 10 da unidade 01 – Foreground/Background

Prazo dia 01/06/2025 as 23:59

Enunciado: Complementação da Atividade 5 Ao executar a atividade 5, percebemos que o BOTÃO do Joystick não foi utilizado. Também observamos que o contador de tarefa não é zerado quando chegamos no fim da fila.

Instruções de Implementação:

- Faça o projeto no VSCode e execute na placa BitdogLab.

- Qual a melhoria que deve ser realizada no novo projeto:

- Manter toda lógica já existente de controle da fila com
- os BOTÕES A e B e o uso da Matriz NeoPixel.
- Inserir uma nova lógica de Controle no BOTÃO do JOYSTICK:
    - AO PRESSIONAR O BOTÃO DO JOYSTICK, O SISTEMA ZERA O CONTADOR DE EVENTOS E APAGA TODA A MATRIZ DE NEOPIXEL.
- Simulando uma fila vazia que está pronta para iniciar. Os contadores que monitoram a fila e de eventos deverão ser ajustados de modo que tenhamos a ideia de uma nova fila que está sendo executada pela primeira vez.

- Nome do arquivo principal: “Atividade_10.c” → Na pasta src.
- Na pasta lib tem os arquivos sobre a Matriz de Leds etc.
- Vídeo mostrando o seu funcionamento
  Link: https://youtu.be/FRb3d9jr7Pg
- GitHub:
  https://github.com/ManoelFelipe/Embarcatech_37/tree/main/Unidade_01/Cap_10/Atividade_10

O que mudou:

- A nova rotina de reset no botão do joystick dentro de funcao_atividade_.c:
- Joystick (id 2): chama reiniciar_sistema(), que apaga toda a matriz NeoPixel (npClear + npWrite);
- zera index_neo, fila (inicio, fim, quantidade) e contador;
- atualiza os LEDs externos (Azul ligado indicando fila vazia);
- registra a ação no console.
- Toda a lógica anterior dos botões A e B foi preservada.

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 para usar de cada parte do código.
3. Arquivos .c (implementações)
   Explicam como cada coisa funciona.

Pastas principais:

- lib/neopixel/: cuida dos LEDs endereçáveis (WS2812) com PIO.
- 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_10.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 para facilitar.

- Código: src/funcao_atividade_.c

```c
/**
 * @file funcao_atividade_.c (versão aprimorada)
 * @brief Implementação das funções de controle de eventos, GPIO e lógica de fila.
 *        Agora inclui suporte ao botão do JOYSTICK para reinicializar a fila
 *        e a matriz NeoPixel, permitindo que o sistema comece uma nova fila
 *        do zero a qualquer momento.
 *
 * Esta versão mantém toda a lógica anterior para os botões A e B e adiciona:
 *   • Quando o botão do JOYSTICK (índice 2) é pressionado, todos os contadores
 *     são resetados, a fila é esvaziada, a matriz NeoPixel é apagada e os LEDs
 *     externos são atualizados (LED Azul ligado para indicar fila vazia).
 *
 * A nova rotina de reset no botão do joystick dentro de funcao_atividade_.c:
 * Joystick (id 2): chama reiniciar_sistema(), que
 * apaga toda a matriz NeoPixel (npClear + npWrite);
 * zera index_neo, fila (inicio, fim, quantidade) e contador;
 * atualiza os LEDs externos (Azul ligado indicando fila vazia);
 * registra a ação no console.
 * Toda a lógica anterior dos botões A e B foi preservada.
 *
 * @version 0.1
 * @author  Manoel Furtado
 * @date    01 junho 2025
 * @copyright Modificado por Manoel Furtado (MIT License) (veja LICENSE.md)
 */

#include "funcao_atividade_.h" //
#include "funcoes_neopixel.h"  //

// ================ VARIÁVEIS GLOBAIS DA FILA ================
int fila[TAM_FILA];         /**< Array que armazena os elementos da fila. */ //
int inicio      = 0;        /**< Índice do início da fila. */ //
int fim         = 0;        /**< Índice do próximo local vago no final da fila. */ //
int quantidade  = 0;        /**< Número de elementos atualmente na fila. */ //
int contador    = 0;        /**< Contador para gerar valores (tarefa/evento) inseridos na fila. */ //

// ================ OUTRAS VARIÁVEIS GLOBAIS ================
absolute_time_t ultimo_toque[NUM_BOTOES];          // Para eventual debouncing por tempo
const uint      BOTOES[NUM_BOTOES] = {BOTAO_A, BOTAO_B, BOTAO_JOYSTICK}; //
const uint      LEDS[NUM_BOTOES]   = {LED_VERMELHO, LED_AZUL, LED_VERDE}; //
volatile bool   eventos_pendentes[NUM_BOTOES] = {false, false, false}; //
volatile bool   estado_leds[NUM_BOTOES]       = {false, false, false}; //
volatile bool   core1_pronto = false; // Flag de inicialização do core1

// ================ PROTÓTIPOS INTERNOS ======================
static void reiniciar_sistema();
```

```c
// ============================================================
//                    FUNÇÃO DE CALLBACK GPIO
// ============================================================
void gpio_callback(uint gpio, uint32_t events) { //
    for (int i = 0; i < NUM_BOTOES; i++) { //
        // Verifica se a interrupção é do botão monitorado e se foi borda de descida
        if (gpio == BOTOES[i] && (events & GPIO_IRQ_EDGE_FALL)) { //
            multicore_fifo_push_blocking(i); // Envia índice do botão ao core1
        }
    }
}

// ============================================================
//                 INICIALIZAÇÃO DE PINO GPIO
// ============================================================
void inicializar_pino(uint pino, uint direcao, bool pull_up, bool pull_down) { //
    gpio_init(pino);                    // Prepara o pino
    gpio_set_dir(pino, direcao);        // Define direção

    if (direcao == GPIO_IN) { // Configura pulls apenas para entrada
        if (pull_up) {
            gpio_pull_up(pino);
        } else if (pull_down) {
            gpio_pull_down(pino);
        } else {
            gpio_disable_pulls(pino);
        }
    }
}

// ============================================================
//               LOOP PRINCIPAL NO CORE1 (tratamento)
// ============================================================
void tratar_eventos_leds() { //
    core1_pronto = true; // Sinaliza que o core1 está pronto

    while (true) {
        // Bloqueia até receber o índice do botão
        uint32_t id1 = multicore_fifo_pop_blocking();

        // Debounce simples
        sleep_ms(DEBOUNCE_MS);

        // Confirma se ainda está pressionado
        if (!gpio_get(BOTOES[id1])) {
            // Ignora se outro botão também está pressionado
            bool outro_pressionado = false;
            for (int i = 0; i < NUM_BOTOES; i++) {
```

```
            if (i != id1 && !gpio_get(BOTOES[i])) {
                outro_pressionado = true;
                break;
            }
        }
        if (outro_pressionado) {
            while (!gpio_get(BOTOES[id1])) tight_loop_contents();
            continue;
        }

        // ================ AÇÕES POR BOTÃO ================
        if (id1 == 0 && index_neo < LED_COUNT) {
            // BOTÃO A → Inserir elemento / acender próximo LED
            uint8_t r = numero_aleatorio(1, 255);
            uint8_t g = numero_aleatorio(1, 255);
            uint8_t b = numero_aleatorio(1, 255);
            npAcendeLED(index_neo, r, g, b);
            index_neo++;

            if (quantidade < TAM_FILA) {
                fila[fim] = contador++;
                fim = (fim + 1) % TAM_FILA;
                quantidade++;
                imprimir_fila();
            }

        } else if (id1 == 1 && index_neo > 0) {
            // BOTÃO B → Remover elemento / apagar último LED
            index_neo--;
            npAcendeLED(index_neo, 0, 0, 0);

            if (quantidade > 0) {
                inicio = (inicio + 1) % TAM_FILA;
                quantidade--;
                imprimir_fila();
            }

        } else if (id1 == 2) {
            // BOTÃO DO JOYSTICK → Reiniciar sistema
            reiniciar_sistema();
        }

        // Atualiza LEDs externos após cada ação
        gpio_put(LED_VERMELHO, (index_neo == LED_COUNT)); // Todos acesos → vermelho
        gpio_put(LED_AZUL,     (index_neo == 0));         // Fila vazia → azul
        gpio_put(LED_VERDE,    0);                        // Não utilizado

        // Aguarda soltar botão
```

```
            while (!gpio_get(BOTOES[id1])) {
                tight_loop_contents();
            }
        }
    }
}

// ============================================================
//                  FUNÇÃO AUXILIAR: RESET
// ============================================================
/**
 * @brief Reinicia completamente os contadores, a fila e os LEDs.
 *
 *  • Zera "index_neo" e apaga toda a matriz NeoPixel.
 *  • Reinicia variáveis da fila (inicio, fim, quantidade, contador).
 *  • Liga o LED Azul externo (indicativo de fila vazia) e apaga os demais.
 *  • Imprime no console que o sistema foi reiniciado.
 */
static void reiniciar_sistema() {
    // Zera ponteiro de NeoPixel e apaga matriz
    index_neo = 0;
    npClear();
    npWrite();

    // Reinicia fila e contadores
    inicio     = 0;
    fim        = 0;
    quantidade = 0;
    contador   = 0;

    // Atualiza LEDs externos
    gpio_put(LED_VERMELHO, 0);
    gpio_put(LED_AZUL,     1); // Azul ligado indica fila vazia
    gpio_put(LED_VERDE,    0);

    // Log para depuração
    printf("Joystick pressionado: Sistema reiniciado. Fila vazia.\n");
}

// ============================================================
//              IMPRESSÃO DO CONTEÚDO DA FILA
// ============================================================
void imprimir_fila() { //
    printf("Fila [tam=%d]: ", quantidade);
    int i = inicio;
    for (int c = 0; c < quantidade; c++) {
        printf("%d ", fila[i]);
        i = (i + 1) % TAM_FILA;
```

```
    }
    printf("\n");
}
```