Nome Aluno: Manoel Felipe Costa Furtado

Matrícula: 20251RSE.MTC0086

Turma: 20251.1.RSE.MTC.1317.1N

Atividade 03 – Referente ao capítulo 3 da unidade 01

Prazo dia 09/05/2025 as 23:59

Enunciado: Monitoramento de Som com Interrupção de Timer

Você deve criar um programa em C/C++ utilizando o SDK oficial do Raspberry Pi Pico (pico-sdk) no VSCode, que configure um timer periódico para realizar leituras do microfone analógico. Sempre que o som captado ultrapassar um limiar definido (nível de ruído considerado “alto”), a matriz de LED WS2812 deverá ser ativada com uma animação ou padrão de cores.

Instruções: Descrição do Sistema - O programa deverá:

- Realizar a leitura do sinal do microfone através do ADC do RP2040;
- Utilizar um timer baseado em interrupção periódica para fazer as leituras;
- Estabelecer um limiar de detecção de som (ex: valor médio do ADC > 800);
- Ativar um padrão de luz na matriz WS2812 quando o limiar for ultrapassado.

Requisitos técnicos:

- Leitura de microfone via ADC (ex: GPIO26/ADC0)
- Uso de timer periódico com call-back (função add_repeating_timer_ms())
- Controle de matriz WS2812

Nome do arquivo principal: Atividade_Cap_03.c

Vídeo mostrando o seu funcionamento
Link: https://youtube.com/shorts/JNt56qNlopU?feature=share

Código: Atividade_Cap_03.c

```c
/**
 * @file    Atividade_Cap_03.c
 * @brief   Detector de som com animação do nome "MANOEL" – Pico W
 * @details Este programa lê o sinal de microfone analógico
 *          periodicamente e, ao ultrapassar o limiar,
 *          exibe o nome "MANOEL" letra por letra em uma matriz
 *          5×5 de LEDs WS2812b dentro de 3000 ms. Cada letra
 *          recebe uma cor aleatória fixa durante sua exibição,
 *          e a cor é renovada aleatoriamente a cada troca de letra.
 *          Mantém estilo Doxygen e comentários completos.
 * @author  Manoel Furtado
 * @date    2025-05-09
 * @copyright 2025 Manoel Furtado (MIT License) (veja LICENSE.md)
 */

  /**
  *  Pode mudar em THRESHOLD que é o Limiar de detecção de som alto (0-4095)
  *  Este firmware bare-metal lê um microfone (GPIO 28 / ADC 2) a cada
  *  @ref SAMPLE_PERIOD_MS ms através de um timer periódico com
  *  interrupção. Se o sinal ultrapassar @ref THRESHOLD, uma animação
  *  colorida é exibida na matriz 5 × 5 de LEDs NeoPixel ligada ao
  *  GPIO 7, controlada por PIO.
  *  Essa animação é o nome "MANOEL" letra por letra, dentro da duração de 3000ms.
  *  A cada letra terá uma cor de forma aletória fixa por todo seu tempo de exebição.
  *  E toda vez que mudar a letra cor deve mudar de forma aleatória.
  *
  *  Pinagem:
  *    - GPIO 7 → DIN da fita/matriz WS2812b (saída PIO)
  *    - GPIO 28 → Saída do microfone eletreto (entrada ADC 2)
  *
  *  Principais recursos:
  *    - ADC + Timer IRQ para amostragem periódica
  *    - PIO para protocolo de 800 kHz dos WS2812
  *    - Código todo comentado em Doxygen para fins didáticos
*/

#include <stdio.h>           // biblioteca padrão de E/S
#include <stdlib.h>          // funções utilitárias padrão
#include "pico/stdlib.h"     // funções básicas do SDK Pico
#include "hardware/adc.h"    // controle do ADC (conversor analógico-digital)
#include "hardware/timer.h"  // timers e interrupções periódicas
#include "hardware/sync.h"   // primitivas de sincronização
#include "hardware/pio.h"    // PIO para comunicação customizada
#include "ws2812.pio.h"      // driver PIO para WS2812
```

```
// ╔═══════════════════════════╗
// ║ Parâmetros ajustáveis     ║
// ╚═══════════════════════════╝
/** @defgroup parameters Configurações de amostragem e limiar
 *  @{ */
#define SAMPLE_PERIOD_MS     5     /**< Intervalo entre leituras ADC (ms) */
#define THRESHOLD            800   /**< Limiar de som alto (0–4095) */
#define NAME_DURATION_MS     3000  /**< Duração total da animação (ms) */
/** @} */

// ╔═══════════════════════════╗
// ║ Parâmetros da animação    ║
// ╚═══════════════════════════╝
/** @defgroup animation Nome "MANOEL"
 *  Configura duração e subdivisão por letra
 *  @{ */
#define NAME_LETTERS         6
#define NAME_TOTAL_TICKS     (NAME_DURATION_MS / SAMPLE_PERIOD_MS)
#define TICKS_PER_LETTER     (NAME_TOTAL_TICKS / NAME_LETTERS)
/** @} */

// ╔═══════════════════════════╗
// ║ Definições de hardware    ║
// ╚═══════════════════════════╝
/** @defgroup hardware GPIO e periféricos
 *  @{ */
#define WS2812_PIN    7      /**< GPIO para DIN da fita WS2812b */
#define NUM_PIXELS    25     /**< LEDs na matriz 5×5 */
#define MIC_ADC_CH    2      /**< Canal ADC (GPIO 28) do microfone */
#define IS_RGBW       false  /**< Matriz RGB (false) ou RGBW (true) */
/** @} */

// ╔═══════════════════════════════╗
// ║ Bitmaps 5×5 do nome "MANOEL" ║
// ╚═══════════════════════════════╝
/** @brief Mapas de intensidade para cada letra (1=acende, 0=apaga)
 *  @details Cada posição da matriz define se o LED está aceso (1) ou apagado (0).
 *           As letras são armazenadas na ordem: M, A, N, O, E, L.
 *           A letra 'L' foi ajustada para orientação correta na matriz.
 */
static const uint8_t name_bitmaps[NAME_LETTERS][5][5] = {
    // M
    {{1,0,0,0,1},{1,1,0,1,1},{1,0,1,0,1},{1,0,0,0,1},{1,0,0,0,1}},
    // A
    {{0,1,1,1,0},{1,0,0,0,1},{1,1,1,1,1},{1,0,0,0,1},{1,0,0,0,1}},
    // N
    {{1,0,0,0,1},{1,1,0,0,1},{1,0,1,0,1},{1,0,0,1,1},{1,0,0,0,1}},
```

```
    // O
    {{0,1,1,1,0},{1,0,0,0,1},{1,0,0,0,1},{1,0,0,0,1},{0,1,1,1,0}},
    // E
    {{1,1,1,1,1},{1,0,0,0,0},{1,1,1,1,1},{1,0,0,0,0},{1,1,1,1,1}},
    // L (ajustado para orientação correta)
    {{0,0,0,0,1},{1,0,0,0,0},{0,0,0,0,1},{1,0,0,0,0},{1,1,1,1,1}}
};

// ╔═══════════════════════════╗
// ║    Variáveis globais      ║
// ╚═══════════════════════════╝
/** @defgroup globals Variáveis de controle
 *  @brief Variáveis utilizadas para controle interno do sistema
 *  @{ */
static PIO         pio            = pio0;     /**< Bloco PIO utilizado */
static uint        sm             = 0;         /**< Índice de state-machine */
static volatile int32_t pattern_ticks = 0;      /**< Ticks restantes da animação */
static int        prev_idx       = -1;        /**< Índice da letra anterior */
static uint32_t   letter_color   = 0;         /**< Cor atual da letra (GRB) */
/** @} */

// ╔═══════════════════════════╗
// ║ Utilitário: Cor (RGB→GRB) ║
// ╚═══════════════════════════╝
/**
 * @brief Converte componentes RGB para formato GRB do WS2812.
 * @param r Intensidade do canal vermelho (0–255).
 * @param g Intensidade do canal verde (0–255).
 * @param b Intensidade do canal azul (0–255).
 * @return Valor GRB empacotado em 24 bits apropriado para PIO.
 */
static inline uint32_t rgb_to_grb(uint8_t r, uint8_t g, uint8_t b) {
    return ((uint32_t)g << 16) | ((uint32_t)r << 8) | b;
}

// ╔═══════════════════════════╗
// ║ PIO: Envio de dados WS2812 ║
// ╚═══════════════════════════╝
/**
 * @brief Envia um pixel para a fita/matriz WS2812 usando PIO.
 * @param grb Cor no formato GRB empacotado (24 bits).
 * @note Utiliza a state-machine configurada para comunicação em 800 kHz.
 */
static inline void put_pixel(uint32_t grb) {
    pio_sm_put_blocking(pio, sm, grb << 8u); // Shift para alinhamento com protocolo WS2812
}
```

```c
/**
 * @brief Apaga todos os LEDs da matriz.
 * @details Envia valor zero para todos os 25 pixels.
 */
static void leds_off(void) {
    for (uint i = 0; i < NUM_PIXELS; ++i) {
        put_pixel(0);
    }
}

// ╔═══════════════════════════════╗
// ║  Animação: Exibição do nome   ║
// ╚═══════════════════════════════╝
/**
 * @brief Renderiza a animação do nome "MANOEL".
 * @details Divide a animação em intervalos iguais para cada letra.
 *          Gera uma nova cor aleatória ao mudar de letra.
 *          A matriz é renderizada com flip vertical para correção de orientação.
 * @param tick Número de ticks restantes na animação.
 */
static void leds_write_name(uint32_t tick) {
    // Calcula o índice da letra atual baseado no tick
    uint32_t idx = tick / TICKS_PER_LETTER;
    if (idx >= NAME_LETTERS) idx = NAME_LETTERS - 1; // Limite máximo

    // Gere nova cor para cada letra nova
    if (idx != prev_idx) {
        prev_idx = idx;
        uint8_t r = rand() & 0xFF;
        uint8_t g = rand() & 0xFF;
        uint8_t b = rand() & 0xFF;
        letter_color = rgb_to_grb(r, g, b); // Atualiza cor fixa para a letra
    }

    // Seleciona bitmap da letra atual
    const uint8_t (*bitmap)[5] = name_bitmaps[idx];

    // Desenha flip vertical: linha 4 → 0
    for (int row = 4; row >= 0; --row) {
        for (uint col = 0; col < 5; ++col) {
            if (bitmap[row][col]) {
                put_pixel(letter_color); /**< LED aceso */
            } else {
                put_pixel(0);            /**< LED apagado */
            }
        }
    }
}
```

```c
// ╔═══════════════════════════════╗
// ║ Callback: Amostragem ADC      ║
// ║ INTERRUPÇÃO DE AMOSTRAGEM ADC ║
// ╚═══════════════════════════════╝
/**
 * @brief IRQ de timer: amostra ADC e dispara animação.
 * @details Se a amostra exceder o limiar, reinicia contador de ticks.
 * @param rt Ponteiro para estrutura de timer (não utilizado).
 * @return true para manter o timer ativo.
 */
static bool adc_sample_callback(struct repeating_timer *rt) {
    (void) rt; // Evita aviso de parâmetro não utilizado
    uint16_t sample = adc_read();
    if (sample > THRESHOLD) {
        pattern_ticks = NAME_TOTAL_TICKS; // Reinicia a animação
    }
    return true;
}

// ╔═══════════════════╗
// ║  Função Principal ║
// ╚═══════════════════╝
/**
 * @brief Inicializa hardware e executa loop principal.
 * @details Etapas:
 *   1. Inicializa USB CDC para debug.
 *   2. Semeia rand() com time_us_32().
 *   3. Configura PIO/state-machine p/ WS2812.
 *   4. Inicializa ADC e GPIO do microfone.
 *   5. Configura timer periódico para amostragem.
 *   6. Loop infinito: atualiza animação ou apaga LEDs.
 * @return Nunca retorna (loop infinito).
 */
int main(void) {
    stdio_init_all();             /**< Habilita USB/Serial */
    srand(time_us_32());          /**< Seed baseada no tempo de inicialização */

    // Configura PIO para controle dos LEDs WS2812b
    uint offset = pio_add_program(pio, &ws2812_program);
    ws2812_program_init(pio, sm, offset, WS2812_PIN, 800000.0f, IS_RGBW);
    leds_off();                   /**< Assegura LEDs apagados */

    // Configuração do ADC (microfone)
    adc_init();
    adc_gpio_init(28);
    adc_select_input(MIC_ADC_CH);
```

```
    // Configura timer para amostragem periódica
    struct repeating_timer timer;
    add_repeating_timer_ms(
        -SAMPLE_PERIOD_MS,      // Intervalo negativo para precisão absoluta
        adc_sample_callback,
        NULL,
        &timer
    );

    // Loop principal: atualiza animação ou mantém LEDs apagados
    uint32_t frame = 0; /**< Contador de frames para índice */
    while (true) {
        if (pattern_ticks > 0) {
            leds_write_name(frame++); // Renderiza frame atual
            --pattern_ticks;
        } else {
            leds_off(); // Desliga LEDs se não houver animação
            frame = 0;
        }
        sleep_ms(SAMPLE_PERIOD_MS); // Sincroniza com a taxa de amostragem
    }
}
```