Nome Aluno: Manoel Felipe Costa Furtado

Matrícula: 20251RSE.MTC0086

Atividade 04 – Referente ao capítulo 4 da unidade 01

Prazo dia 18/05/2025 as 23:59

Enunciado: Semáforo de Trânsito Interativo. Criar um semáforo de trânsito, com acionamento de travessia para pedestres e indicação de tempo restante.

Nome do arquivo principal: Atividade_04.c

Na pasta src.

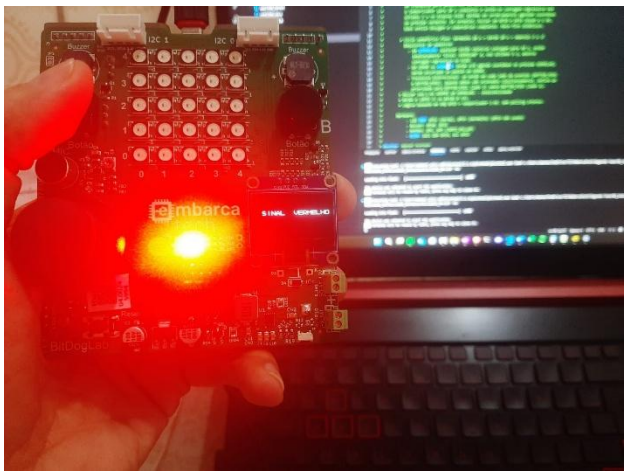
Na pasta lib tem os arquivos para display oled.

Rodando no Simulador Wokwi.

Link: https://wokwi.com/projects/431065994975581185

Vídeo mostrando o seu funcionamento
Link: https://youtu.be/IhhFuaj7UxE

Foto:



Código: Atividade_04.c

```c
/**
 * ************* Unidade 01 - cap.04 - Atividade 04 *********************************
 * @file    Atividade_04.c
 * @brief   Simulação de semáforo de trânsito com acessibilidade para pedestres
 * @details Implementa ciclo semafórico fixo (Vermelho → Verde → Amarelo) e lógica
 *          de travessia para pedestres. Ao acionar botão em vermelho, reinicia-se
 *          o temporizador para 10 s completos e exibe-se contagem regressiva nos
 *          últimos 5 s no display OLED. Botões em verde/amarelo agendam countdown
 *          no próximo vermelho. Em vermelho, buzzer em GP10 e GP21 bipam a 1 Hz.
 *          Usei estilo Doxygen e comentários completos para fins didáticos.
 *
 *        • Ciclo semafórico fixo: Vermelho 10 s → Verde 10 s → Amarelo 3 s ↺
 *          Requisitos de pedestre:
 *            - Em VERMELHO: qualquer botão reinicia contagem para 10 s, exibe
 *              imediatamente "Sinal: VERMELHO" e, nos últimos 5 s, mostra
 *              contagem regressiva no OLED.
 *            - Em VERDE ou AMARELO: botão A/B agenda countdown no próximo VERMELHO;
 *              Verde encurtado se pedido em VERDE.
 *            - Travessia bidirecional: um botão em cada lado da avenida (BUTTON_PIN1
 *              e BUTTON_PIN2). Quando ambos são acionados simultaneamente, define-se
 *              prioridade para BUTTON_PIN1 (lado 1) sobre BUTTON_PIN2 (lado
2).
 *        • Buzzer soa 1 Hz SEMPRE em VERMELHO. - Acessibilidade
 *        • Botão A (GPIO 5) ou Botão B (6).
 *        • Lógica baseada em IRQ GPIO + temporizador 1 Hz. Sem polling intenso.
 *
 *       Hardware:
 *            - LED RGB: GP11 (Verde), GP13 (Vermelho) (GP12 não usado)
 *            - Buzzer: GP10, GP21
 *            - Botões: GP5, GP6 (INPUT_PULLUP)
 *            - OLED: I2C1 SDA GP14, SCL GP15
 *
 * @author  Manoel Furtado
 * @date    15 mai 2025
 * @copyright 2025 Manoel Furtado (MIT License) (veja LICENSE.md)
 */

#include <stdio.h>
#include <string.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/timer.h"
#include "hardware/i2c.h"
#include "ssd1306_i2c.h"
#include "ssd1306.h"
#include "ssd1306_font.h"
```

```c
/*———————————————— Constantes de tempo (segundos) ————————————————*/
/** @brief Tempo total de sinal vermelho (s) */
#define RED_TIME_SEC       10
/** @brief Tempo total de sinal verde (s) */
#define GREEN_TIME_SEC     10
/** @brief Tempo total de sinal amarelo (s) */
#define YELLOW_TIME_SEC     3
/** @brief Início da contagem regressiva (últimos segundos) */
#define COUNTDOWN_START     5

/*———————————————— GPIO Mapping ————————————————*/
/** @brief Pino do LED vermelho */
#define LED_RED_PIN        13
/** @brief Pino do LED verde */
#define LED_GREEN_PIN      11
/** @brief Pino do LED azul (não usado) */
#define LED_BLUE_PIN       12

/** @brief Pino do buzzer principal */
#define BUZZER_PIN1        10
/** @brief Pino do buzzer secundário */
#define BUZZER_PIN2        21

/** @brief Botão lado 1 (prioridade em empate) */
#define BUTTON_PIN1         5
/** @brief Botão lado 2 */
#define BUTTON_PIN2         6

// I²C0 para OLED SSD1306
#define I2C_SDA_PIN        14   // GP14: SDA0 (I2C0)
#define I2C_SCL_PIN        15   // GP15: SCL0 (I2C0)

/*———————————————— Tipos e variáveis ————————————————*/
/** @brief Estados possíveis do semáforo */
typedef enum {
    ST_RED = 0,   /**< Vermelho */
    ST_GREEN,     /**< Verde */
    ST_YELLOW     /**< Amarelo */
} state_t;

/** @brief Estado corrente do semáforo */
static volatile state_t current_state = ST_RED;

/** @brief Contador de segundos restante no estado atual */
static volatile int     countdown_sec = RED_TIME_SEC;

static volatile int   ped_request_dir = 0;  /**< Travessia agendada: 1 ou 2 */
```

```c
static volatile int   red_from_ped_dir= 0;  /**< Vermelho iniciado por pedestre */

/** @brief Timer periódico de 1 Hz */
static struct repeating_timer timer_1s;

/** @brief Buffer interno para renderização no OLED */
static uint8_t           oled_buf[ssd1306_buffer_length];

/** @brief Área de renderização full-screen do OLED */
static struct render_area full_area;

/*———————————————— Protótipos de funções ————————————————*/
/** @brief Configura hardware: GPIOs, I2C e inicialização do OLED */
static void hw_init(void);

/** @brief Inicializa o display OLED (comandos, buffer e mensagem) */
static void oled_init(void);

/** @brief Limpa completamente o display OLED */
static void oled_clear(void);

/** @brief Exibe texto do estado atual no OLED
 *  @param st Estado do semáforo (Vermelho, Verde ou Amarelo)
 */
static void oled_show_state(state_t st);

/** @brief Exibe um número centralizado no OLED
 *  @param n Valor a ser exibido (contagem regressiva)
 */
static void oled_show_countdown(int n);

/** @brief Atualiza LEDs conforme estado
 *  @param st Estado do semáforo para definir cor
 */
static void set_led_color(state_t st);

/** @brief Callback chamado a cada segundo pelo timer
 *  @param t Ponteiro para struct de timer (não usado)
 *  @return true para manter o timer ativo
 */
static bool timer_cb(struct repeating_timer *t);

/** @brief Tratador de interrupção dos botões de pedestre
 *  @param gpio Pino que gerou a interrupção
 *  @param events Tipo de evento (não usado)
 */
static void button_irq(uint gpio, uint32_t events);
```

```c
/*———————————————— Implementação das funções ————————————————*/
/**
 * @brief Inicializa GPIOs de LEDs, buzzers, botões e I2C para o OLED
 * @details Configura direções, pull-ups, IRQs de botão e chama oled_init().
 */
static void hw_init(void) {
    stdio_init_all();

    /* Configura LEDs como saída */
    gpio_init(LED_RED_PIN);   gpio_set_dir(LED_RED_PIN,   GPIO_OUT);
    gpio_init(LED_GREEN_PIN); gpio_set_dir(LED_GREEN_PIN, GPIO_OUT);
    gpio_init(LED_BLUE_PIN);  gpio_set_dir(LED_BLUE_PIN,  GPIO_OUT);

    /* Configura buzzers como saída, estado inicial baixo */
    gpio_init(BUZZER_PIN1);   gpio_set_dir(BUZZER_PIN1,  GPIO_OUT);
    gpio_init(BUZZER_PIN2);   gpio_set_dir(BUZZER_PIN2,  GPIO_OUT);
    gpio_put(BUZZER_PIN1, 0);
    gpio_put(BUZZER_PIN2, 0);

    /* Configura botões como entrada com pull-up e IRQ em borda de descida */
    gpio_init(BUTTON_PIN1);   gpio_set_dir(BUTTON_PIN1, GPIO_IN);
    gpio_pull_up(BUTTON_PIN1);
    gpio_init(BUTTON_PIN2);   gpio_set_dir(BUTTON_PIN2, GPIO_IN);
    gpio_pull_up(BUTTON_PIN2);
    gpio_set_irq_enabled_with_callback(
        BUTTON_PIN1, GPIO_IRQ_EDGE_FALL, true, &button_irq);
    gpio_set_irq_enabled(
        BUTTON_PIN2, GPIO_IRQ_EDGE_FALL, true);

    /* Inicializa interface I2C1 para o OLED */
    i2c_init(i2c1, ssd1306_i2c_clock * 1000);
    gpio_set_function(I2C_SDA_PIN, GPIO_FUNC_I2C);
    gpio_set_function(I2C_SCL_PIN, GPIO_FUNC_I2C);
    gpio_pull_up(I2C_SDA_PIN);
    gpio_pull_up(I2C_SCL_PIN);

    /* Inicializa o display OLED */
    oled_init();
}

/**
 * @brief Envia comandos de inicialização ao SSD1306 e exibe mensagem inicial
 */
static void oled_init(void) {
    /* Limpa buffer e define área full-screen */
    memset(oled_buf, 0, sizeof(oled_buf));
    full_area.start_column = 0;
```

```
    full_area.end_column   = ssd1306_width - 1;
    full_area.start_page   = 0;
    full_area.end_page     = ssd1306_n_pages - 1;
    calculate_render_area_buffer_length(&full_area);

    /* Inicializa o controlador SSD1306 e exibe texto */
    ssd1306_init();
    oled_clear();
    ssd1306_draw_string(oled_buf, 8, 24, "Inicializando...");
    render_on_display(oled_buf, &full_area);
}

/**
 * @brief Limpa o buffer e atualiza o display OLED
 */
static void oled_clear(void) {
    memset(oled_buf, 0, sizeof(oled_buf));
    render_on_display(oled_buf, &full_area);
}

/**
 * @brief Exibe mensagem de estado no OLED (VERMELHO, VERDE ou AMARELO)
 * @param st Estado do semáforo
 */
static void oled_show_state(state_t st) {
    oled_clear();
    switch (st) {
        case ST_RED:
            ssd1306_draw_string(oled_buf, 8, 24, "Sinal: VERMELHO");
            break;
        case ST_GREEN:
            ssd1306_draw_string(oled_buf, 8, 24, "Sinal: VERDE");
            break;
        case ST_YELLOW:
            ssd1306_draw_string(oled_buf, 8, 24, "Sinal: AMARELO");
            break;
    }
    render_on_display(oled_buf, &full_area);
}

/**
 * @brief Exibe um valor centralizado no display para contagem regressiva
 * @param n Segundo restante a ser exibido
 */
static void oled_show_countdown(int n) {
    char txt[8];
    snprintf(txt, sizeof(txt), "%d", n);
    oled_clear();
```

```
    ssd1306_draw_string(oled_buf, 60, 24, txt);
    render_on_display(oled_buf, &full_area);
}

/**
 * @brief Atualiza estado dos LEDs RGB conforme estado do semáforo
 * @param st Estado atual (vermelho, verde ou amarelo)
 */
static void set_led_color(state_t st) {
    gpio_put(LED_RED_PIN,   (st == ST_RED)   || (st == ST_YELLOW));
    gpio_put(LED_GREEN_PIN, (st == ST_GREEN) || (st == ST_YELLOW));
    /* LED azul não utilizado */
}

/**
 * @brief Callback de 1 Hz responsável por decremento e transições
 * @param t Ponteiro para a struct do timer (unused)
 * @return true para manter o timer ativo
 */
static bool timer_cb(struct repeating_timer *t) {
    static bool buzz_state = false;

    /* Buzzer bip 1 Hz enquanto estiver em vermelho */
    if (current_state == ST_RED) {
        buzz_state = !buzz_state;
        gpio_put(BUZZER_PIN1, buzz_state);
        gpio_put(BUZZER_PIN2, buzz_state);
    } else {
        gpio_put(BUZZER_PIN1, 0);
        gpio_put(BUZZER_PIN2, 0);
    }

    /* Decrementa contador de segundos */
    if (countdown_sec > 0) {
        countdown_sec--;
    }

    /* Exibe últimos segundos se iniciado por pedestre */
    if (red_from_ped_dir && current_state == ST_RED &&
        countdown_sec <= COUNTDOWN_START && countdown_sec >= 1) {
        oled_show_countdown(countdown_sec);
    }
    /* Transição de estados no fim da contagem */
    if (countdown_sec == 0) {
        switch (current_state) {
            case ST_RED:
                current_state   = ST_GREEN;
                countdown_sec   = GREEN_TIME_SEC;
```

```
                red_from_ped_dir= 0;
                ped_request_dir = 0;
                break;
            case ST_GREEN:
                current_state = ST_YELLOW;
                countdown_sec = YELLOW_TIME_SEC;
                break;
            case ST_YELLOW:
                current_state = ST_RED;
                countdown_sec = RED_TIME_SEC;
                if (ped_request_dir) red_from_ped_dir = ped_request_dir;
                ped_request_dir = 0;
                break;
        }
        set_led_color(current_state);
        oled_show_state(current_state);
    }

    /* Se pedido em VERDE, encurta para próximo ciclo */
    if (ped_request_dir && current_state == ST_GREEN) {
        countdown_sec = 0;
    }

    return true; /* mantém callback ativo */
}

/**
 * @brief Handler de interrupção para botões de travessia
 * @param gpio Pino que gerou a IRQ
 * @param events Tipo de evento (unused)
 */
static void button_irq(uint gpio, uint32_t events) {
    int dir = (gpio == BUTTON_PIN1 ? 1 : 2);
    /* Em VERMELHO: reinicia contagem e registra direção */
    if (current_state == ST_RED) {
        countdown_sec     = RED_TIME_SEC;
        red_from_ped_dir  = dir;
        oled_show_state(ST_RED);
    } else {
        /* Em VERDE/AMARELO: agenda travessia no próximo VERMELHO */
        if (ped_request_dir == 0 || dir == 1) {
            ped_request_dir = dir;
        }
        oled_clear();
        ssd1306_draw_string(oled_buf, 0, 24, "Pedido travessia");
        render_on_display(oled_buf, &full_area);
    }
}
```

```c
/**
 * @brief Função principal: inicializa hardware e entra em loop de espera
 * @details
 *   1. Chama hw_init() para configurar GPIOs, I2C e display OLED.
 *   2. Define cor inicial do semáforo para vermelho e exibe no OLED.
 *   3. Adiciona timer periódico de 1 Hz para gerenciar transições de estado
 *      e contagem regressiva.
 *   4. Entra em loop infinito com tight_loop_contents(), permitindo que a
 *      CPU aguarde de forma eficiente por interrupções de botões e callbacks.
 * @return Esta função não retorna.
 */
int main(void) {
    /* 1) Inicialização de hardware (GPIOs, I2C, OLED) */
    hw_init();

    /* 2) Estado inicial: vermelho (LED e display) */
    set_led_color(ST_RED);
    oled_show_state(ST_RED);

    /* 3) Configura e inicia o timer periódico de 1 Hz para gerenciamento do semáforo:
     *    - Intervalo de 1000 ms (1 segundo) entre callbacks.
     *    - Offset de -1000 ms indica que o primeiro callback ocorre imediatamente
     *      após a chamada, sem aguardar o primeiro intervalo completo.
     *    - `timer_cb` é a função de callback que decrementa o contador,
     *      controla o buzzer e gerencia as transições de estado do semáforo.
     *    - `NULL` é o argumento de user_data passado ao callback (não utilizado).
     *    - `&timer_1s` recebe o handle do timer, permitindo futura remoção, se necessário.
     */
    add_repeating_timer_ms(-1000, timer_cb, NULL, &timer_1s);

    /* 4) Loop principal: CPU aguarda em idle, acordando apenas por IRQ */
    while (true) {
        tight_loop_contents();
    }
}
```