Aluno do Embarcatech_37 no IFMA

Nome: Manoel Felipe Costa Furtado

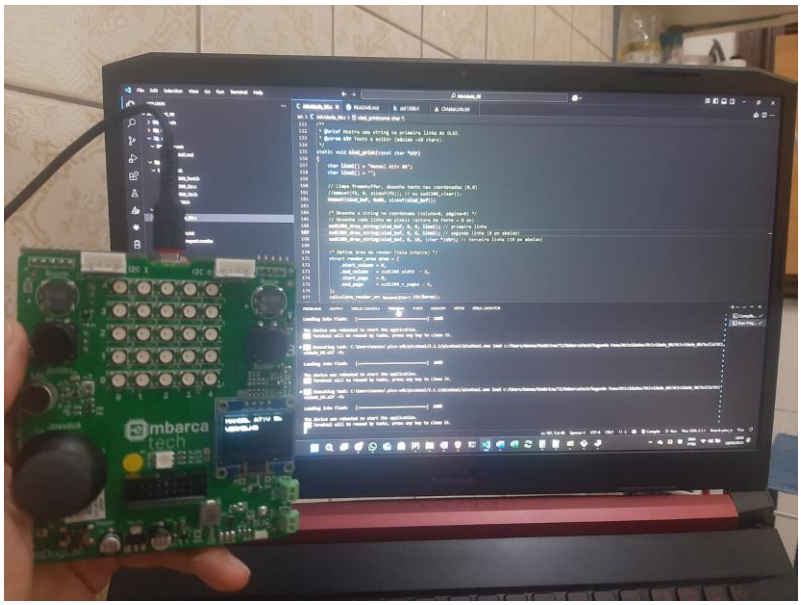
Matrícula: 20251RSE.MTC0086

Atividade 06 – Referente ao capítulo 6 da unidade 01 - USB

Prazo dia 25/05/2025 as 23:59

Enunciado: Dispositivo CDC (Communication Device Class) responsivo e com identificação visual: Criar um Dispositivo CDC (Communication Device Class), que responda aos comandos realizados no computador através de eco e indique visualmente o comando executado. O projeto deve ser implementado com a biblioteca TinyUSB.

- GitHub: https://github.com/ManoelFelipe/Embarcatech_37/tree/main/Unidade_01/Cap_06/Atividade_06
- Nome do arquivo principal: “Atividade_06.c” → Na pasta src.
- Na pasta lib tem os arquivos sobre o display OLED.
- Vídeo mostrando o seu funcionamento
  Link: https://youtu.be/HAi9CNqdWUI
- Foto do funcionamento, mais detalhes no vídeo.



- Código: Atividade_06.c

```c
/**
 * @file    Atividade_06.c
 * @brief   Dispositivo USB-CDC com eco e indicação visual – Raspberry Pi Pico W
 * @details Exemplo educacional (BitDogLab) que demonstra como:
 *           • Criar um dispositivo USB Communication Device Class (CDC) usando TinyUSB
 *           • Receber comandos de texto pelo Monitor Serial (VS Code) e ecoar
 *           • Acender LEDs e buzzer conforme o comando recebido
 *           • Exibir o comando no display OLED SSD1306 via I²C1 (GP14/GP15)
 *          Funcionalidades:
 *          1. Ecoa comandos recebidos via USB-CDC.
 *          2. Aciona LEDs (GPIO 11: verde, 12: azul, 13: vermelho) conforme comandos "verde", "azul", "vermelho".
 *          3. Aciona buzzer (GPIO 10) com o comando "som".
 *          4. Exibe comandos no display OLED SSD1306 via I2C (GPIO 14-SDA, 15-SCL).
 *
 *          Comandos reconhecidos (case-insensitive):
 *          └ "vermelho"  → acende LED vermelho  (GPIO-13)
 *          └ "verde"     → acende LED verde     (GPIO-11)
 *          └ "azul"      → acende LED azul      (GPIO-12)
 *          └ "som"       → aciona buzzer        (GPIO-10)
 *          Cada indicação visual/sonora permanece ativa por 1 s.
 *
 * @note    Hardware-alvo  : Raspberry Pi Pico W (RP2040) + placa BitDogLab
 *          Bibliotecas    : - TinyUSB (via pico_stdio_usb)
 *                           - SSD1306 driver em lib/
 *          Todas as funções estão extensivamente comentadas, estilo Doxygen para fins didáticos.
 *          GuitHub: https://github.com/ManoelFelipe/Embarcatech_37
 *
 * @author  Manoel Furtado
 *
 * @date    19 mai 2025
 * @copyright 2025 Manoel Furtado (MIT License) (veja LICENSE.md)
 */

/*-----------------------------------------------------------------------
 *  INCLUDES
 *-----------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#include "pico/stdlib.h"
#include "hardware/i2c.h"

/* Biblioteca TinyUSB – dispositivo USB */
#include "tusb.h"
```

```c
/* Bibliotecas SSD1306 fornecidas pelo usuário */
#include "ssd1306.h"
#include "ssd1306_i2c.h"

/*=========================================================================
 *                            DEFINES & CONSTANTS
 *=======================================================================*/
#define LED_GREEN_PIN   11    /**< GPIO do LED VERDE    */
#define LED_BLUE_PIN    12    /**< GPIO do LED AZUL     */
#define LED_RED_PIN     13    /**< GPIO do LED VERMELHO */
#define BUZZER_PIN      10    /**< GPIO do BUZZER       */

#define ONE_SECOND_MS   1000u /**< Duração fixa de 1 s  */
#define CMD_BUF_MAX     32    /**< Tamanho máximo do buffer de comando */

/*=========================================================================
 *                              PROTÓTIPOS
 *=======================================================================*/
/**
 * @brief Inicializa stdio USB, GPIOs (LEDs/buzzer), I²C e OLED.
 */
static void board_init(void);

/**
 * @brief Aguarda o host abrir a porta CDC e exibe status no OLED/serial.
 */
static void cdc_wait_connect(void);

/**
 * @brief Desenha texto no OLED (limpa buffer e atualiza display).
 * @param str Texto zero-terminado a mostrar (linha única).
 */
static void oled_print(const char *str);

/**
 * @brief Gera pulso de 1 s em um pino GPIO (LED ou buzzer).
 * @param gpio Número do pino GPIO já configurado como saída.
 */
static void pulse_pin(uint gpio);

/**
 * @brief Lê dados da CDC, ecoa, converte, exibe no OLED e dispara ação.
 */
static void process_cdc(void);

/*=========================================================================
 *                             VARIÁVEIS GLOBAIS
```

```c
 *============================================================================*/
/** Buffer local de RAM para o OLED (tamanho definido na lib SSD1306) */
static uint8_t oled_buf[ssd1306_buffer_length] = { 0 };

/* Buffer temporário para receber dados da CDC */
static char cmd_buf[CMD_BUF_MAX] = { 0 };

/* Comprimento atual do comando armazenado em cmd_buf */
static size_t cmd_len = 0;

/*============================================================================
 *                            IMPLEMENTAÇÃO
 *============================================================================*/

/**
 * @brief Configura periferais de placa: stdio USB, LEDs, buzzer, I²C e OLED.
 */
static void board_init(void)
{
    /* Inicializa stdin/stdout sobre USB-CDC */
    stdio_init_all();

    /* Configura LEDs e buzzer como saídas, inicialmente APAGADOS */
    const uint pins[] = { LED_GREEN_PIN, LED_BLUE_PIN, LED_RED_PIN, BUZZER_PIN };
    for (size_t i = 0; i < sizeof(pins)/sizeof(pins[0]); i++) {
        gpio_init(pins[i]);
        gpio_set_dir(pins[i], GPIO_OUT);
        gpio_put(pins[i], false);
    }

    /* Inicializa I²C1 a 400 kHz (configurado na lib SSD1306) */
    i2c_init(i2c1, ssd1306_i2c_clock * 1000);
    gpio_set_function(14, GPIO_FUNC_I2C);  /* SDA */
    gpio_set_function(15, GPIO_FUNC_I2C);  /* SCL */
    gpio_pull_up(14);
    gpio_pull_up(15);

    /* Inicializa display OLED */
    ssd1306_init();
}

/**
 * @brief Aguarda conexão USB-CDC; atualiza OLED e emite mensagem serial.
 */
static void cdc_wait_connect(void)
{
    oled_print("Aguardando CDC...");
    /* Loop até que o host abra a porta CDC */
```

```
    while (!tud_cdc_connected()) {
        tud_task();     /* Processa stack TinyUSB */
        sleep_ms(100);
    }
    /* Conectado: exibe status */
    oled_print("CDC conectado!");
    printf("CDC conectado!\r\nComandos: vermelho | verde | azul | som\r\n");
}

/**
 * @brief Mostra uma string na primeira linha do OLED.
 * @param str Texto a exibir (máximo ~20 chars).
 */
static void oled_print(const char *str)
{
    char line1[] = "Manoel Ativ 06";
    char line2[] = "";

    // Limpa framebuffer, desenha texto nas coordenadas (0,0)
    //memset(fb, 0, sizeof(fb)); // ou ssd1306_clear();
    memset(oled_buf, 0x00, sizeof(oled_buf));

    /* Desenha a string na coordenada (coluna=0, página=0) */
    // Desenha cada linha em pixels (altura da fonte = 8 px)
    ssd1306_draw_string(oled_buf, 0, 0, line1); // primeira linha
    ssd1306_draw_string(oled_buf, 0, 8, line2); // segunda linha (8 px abaixo)
    ssd1306_draw_string(oled_buf, 0, 16, (char *)str); // terceira linha (16 px abaixo)

    /* Define área de render (tela inteira) */
    struct render_area area = {
        .start_column = 0,
        .end_column   = ssd1306_width  - 1,
        .start_page   = 0,
        .end_page     = ssd1306_n_pages - 1,
    };
    calculate_render_area_buffer_length(&area);
    /* Envia buffer ao display */
    render_on_display(oled_buf, &area);
}

/**
 * @brief Acende um GPIO (LED ou buzzer) por EXACTAMENTE 1 s.
 * @param gpio Pino configurado como saída.
 */
static void pulse_pin(uint gpio)
{
    gpio_put(gpio, true);
    sleep_ms(ONE_SECOND_MS);
```

```c
    gpio_put(gpio, false);
}

/**
 * @brief Lê dados da CDC, ecoa, converte para minúsculas, exibe e aciona periféricos.
 */
static void process_cdc(void)
{
    /* Enquanto houver dados disponíveis na CDC */
    while (tud_cdc_available()) {
        uint8_t buf[64];
        /* Leitura não-bloqueante em bloco */
        uint32_t count = tud_cdc_read(buf, sizeof(buf));
        if (count == 0) return;

        /* Garante string C-zero-terminated */
        if (count < sizeof(buf)) buf[count] = '\0';
        else buf[sizeof(buf)-1] = '\0';

        /* Ecoa de volta ao host */
        tud_cdc_write(buf, count);
        tud_cdc_write_flush();

        /* Converte buffer para minúsculas para simplificar comparações */
        for (uint32_t i = 0; i < count; i++) {
            buf[i] = (char)tolower((unsigned char)buf[i]);
        }

        /* Exibe o comando no OLED */
        oled_print((const char *)buf);

        /* Despacha ação conforme comando */
        if      (strcmp((char*)buf, "vermelho") == 0) pulse_pin(LED_RED_PIN);
        else if (strcmp((char*)buf, "verde")    == 0) pulse_pin(LED_GREEN_PIN);
        else if (strcmp((char*)buf, "azul")     == 0) pulse_pin(LED_BLUE_PIN);
        else if (strcmp((char*)buf, "som")      == 0) pulse_pin(BUZZER_PIN);
        else {
            /* Comando não reconhecido */
            tud_cdc_write_str("\r\nComando desconhecido!\r\n");
            tud_cdc_write_flush();
        }
    }
}

/**
 * @brief Função principal: inicializa hardware, aguarda CDC e entra em loop.
 */
int main(void)
```

```
{
    board_init();          /* Configura LEDs, buzzer, I²C, OLED, USB-CDC */
    cdc_wait_connect();    /* Aguarda host e exibe status */

    /* Loop infinito de serviço USB e CDC */
    while (1) {
        process_cdc();     /* Trata leitura/escrita CDC */
        tud_task();        /* Mantém TinyUSB rodando */
    }

    return 0; /* Nunca alcançado */
}
```