



**INSTITUTO FEDERAL DO PIAUÍ
TERESINA ZONA SUL
LICENCIATURA EM COMPUTAÇÃO**

**CAPACITAÇÃO EM SISTEMAS EMBARCADOS
ALUNO: ANTONIO JOSÉ ALVES DE SOUSA**

**PROJETO EMBARCADO: LÂMPADA AJUSTÁVEL(BASE TEÓRICA, CIRCUITO
E CÓDIGO COMENTADO)**

**TERESINA - PI
16/02/2025**

ESCOPO DO PROJETO

Simulação de um Sistema de Iluminação Inteligente

O Sistema de Iluminação Inteligente utiliza os botões da Bit Dog Lab para controlar a intensidade e a cor dos LEDs, permitindo a criação de um ambiente iluminado de acordo com as preferências do usuário.

Pode ser particularmente útil em diversas situações, como:

Momentos de lazer com a família: Permite ajustar a iluminação para criar uma atmosfera ideal como durante a exibição de filmes, proporcionando uma melhor experiência.

Ambientes de Trabalho: Pode ser utilizado em escritórios, onde diferentes níveis de luminosidade podem ajudar na concentração e produtividade, especialmente em tarefas que exigem foco.

Acessibilidade: Para pessoas com problemas de visão que preferem ambientes não totalmente escuros, o sistema permite ajustes finos na luminosidade, garantindo conforto visual sem ofuscação.

Decoração de Eventos: Em festas e celebrações, a capacidade de mudar a cor e a intensidade das luzes.

Residências Inteligentes: Integrando-se a sistemas de automação residencial, permitindo controle remoto via aplicativos ou assistentes de voz.

Salas de Aula: Ajustes de luz podem ajudar a manter a atenção dos alunos e criar um ambiente propício ao aprendizado.

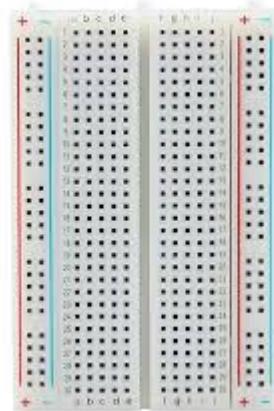
ESPECIFICAÇÃO DO HARDWARE

Componentes eletrônicos utilizados: Placa Bit Dog Lab, Protoboard, Infravermelho, Capacitor, Transistor, Resistor e fios para alimentação do circuito.

Placa Bit Dog Lab



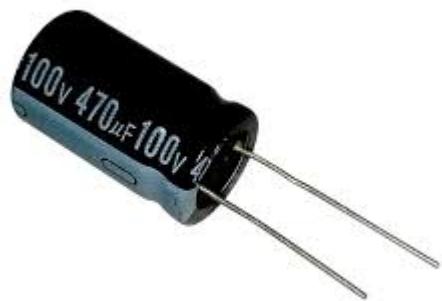
Protoboard



Infravermelho



Capacitor



Resistor



Transistor



Fios

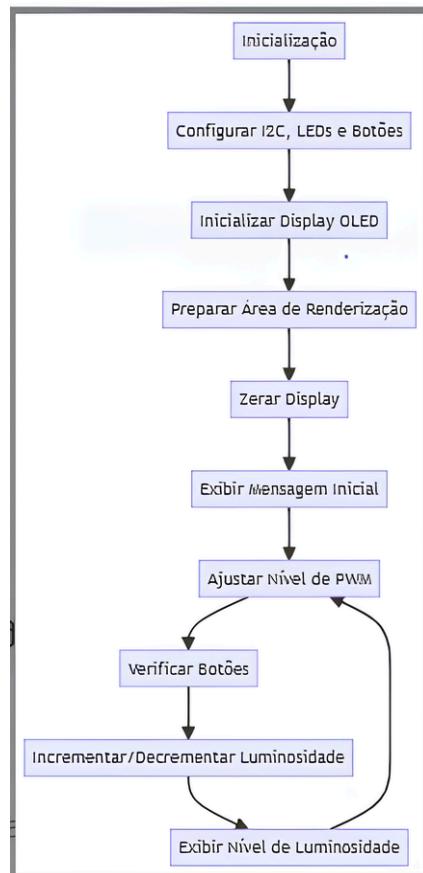


Pinagem Utilizada

```
/*
#           DEFINICOES E PINOS UTILIZADOS
# =====
# */

#define VERDE 11 //Unico utilizado no programa para o led verde
// #define BTA 5 // Botão A antes utilizado sem o circuito para incrementar a luminosidade
#define BTA 3 // Botão A
#define BTB 6 // Botão B
const uint I2C_SDA = 14; //PINO SDA utilizado para comunicação com o display
const uint I2C_SCL = 15; //PINO SCL utilizado para comunicação com o display
/* ----- */
```

Especificação do firmware:Fluxograma do software



Execução do projeto e Metodologia

Pesquisa das etapas do projeto e metodologias aplicadas: Pesquisa no Capítulo 4 da Capacitação Básica em Sistemas Embarcados: As etapas do

projeto e as metodologias aplicadas, incluindo o uso de botões e PWM, foram pesquisadas no capítulo 4 da capacitação básica em sistemas embarcados.

Exibição de Mensagens: Para exibir as mensagens, foi utilizado como exemplo o código de gravação de imagens e mensagens disponível no GitHub, que também foi usado no tutorial do Professor Jivago.

Uso dos Botões: Inicialmente, foram utilizados os botões A (pino 5) e B (pino 6) da placa para incrementar e decrementar a luminosidade do LED, respectivamente.

Conexões na Protoboard e Placa de Desenvolvimento: Na protoboard, foram feitas as seguintes conexões:

- Ligação do 3.3V da placa para a protoboard
- Ligação do GND (terra) da placa para a protoboard
- Ligação do pino 3 da placa para a protoboard

Círculo externo com infravermelho:

Posteriormente, troquei o botão A para o pino 3 da placa e montou um circuito externo em uma protoboard, utilizando componentes como resistores, capacitor, transistor e um receptor de infravermelho.

Com o circuito externo, o sinal de um transmissor de infravermelho (como de um controle remoto de TV) é recebido pelo receptor e enviado para a placa, permitindo o incremento da luminosidade do LED.

Decremento e Desligamento: O botão B é utilizado para decrementar a luminosidade do LED até que ele seja desligado.

Atualização do Display OLED: À medida que a luminosidade do LED é ajustada, a mensagem no display OLED é atualizada para refletir o novo nível de luminosidade.

MONTAGEM DO CIRCUITO

Imagens utilizadas como referência para criação do circuito

Infravermelho	
Circuito	

BASE TEÓRICA DO CIRCUITO

O circuito é composto pelos seguintes elementos:

TSOP1738: Este é um receptor de infravermelho que converte o sinal de infravermelho em um sinal elétrico.

BC557: Este é um transistor NPN que atua como um amplificador do sinal vindo do receptor TSOP1738.

10KΩ: Este é um resistor que serve como carga de polarização para o transistor BC557.

470Ω: Este é um resistor limitador de corrente para o LED.

LED: Este é um diodo emissor de luz que indica quando o receptor de infravermelho está recebendo um sinal.

1uF: Este é um capacitor de desacoplamento que ajuda a filtrar o sinal de alimentação.

O circuito recebe um sinal de +3.3V na entrada "SIGNAL" e o sinal de infravermelho é convertido pelo TSOP1738 em um sinal elétrico. Esse sinal é então amplificado pelo transistor BC557 e aciona o LED, indicando a recepção do sinal de infravermelho.

Conexões Principais:

TSOP1738 (Receptor IR):

- VCC (Alimentação): Conectado ao +3.3V (azul na imagem)
- GND: Conectado ao GND da fonte (cabo marrom)
- OUT (Sinal): Conectado ao pino 3 do Arduino via resistor de 10KΩ (cabo branco)

Transistor BC557 (PNP):

- Coletor (C): Ligado ao GND
- Base (B): Recebe sinal do TSOP1738 - Receptor de infravermelho via resistor de 10KΩ
- Emissor (E): Conectado ao LED via resistor de 470Ω

LED:

- Ânodo: Conectado ao resistor de 470Ω
- Cátodo: Ligado ao GND

Funcionamento do Circuito:

Recepção do Sinal IR:

- O TSOP1738 demodula sinais infravermelhos (38KHz) de controles remotos
- Saída digital: Nível baixo (0V) quando detecta sinal válido

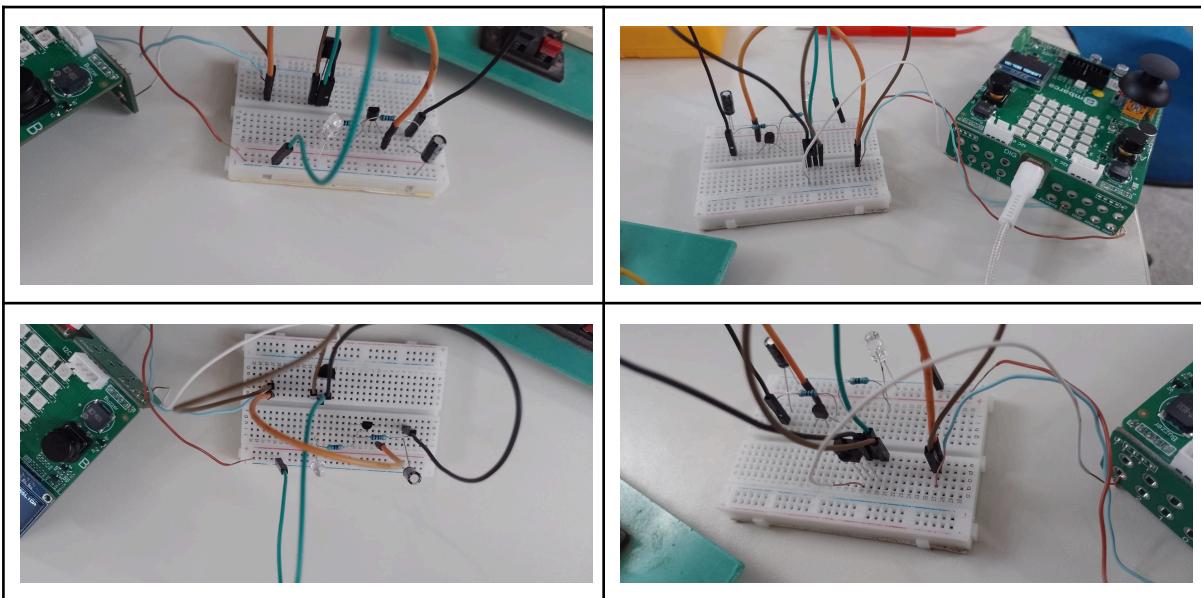
Amplificação com Transistor:

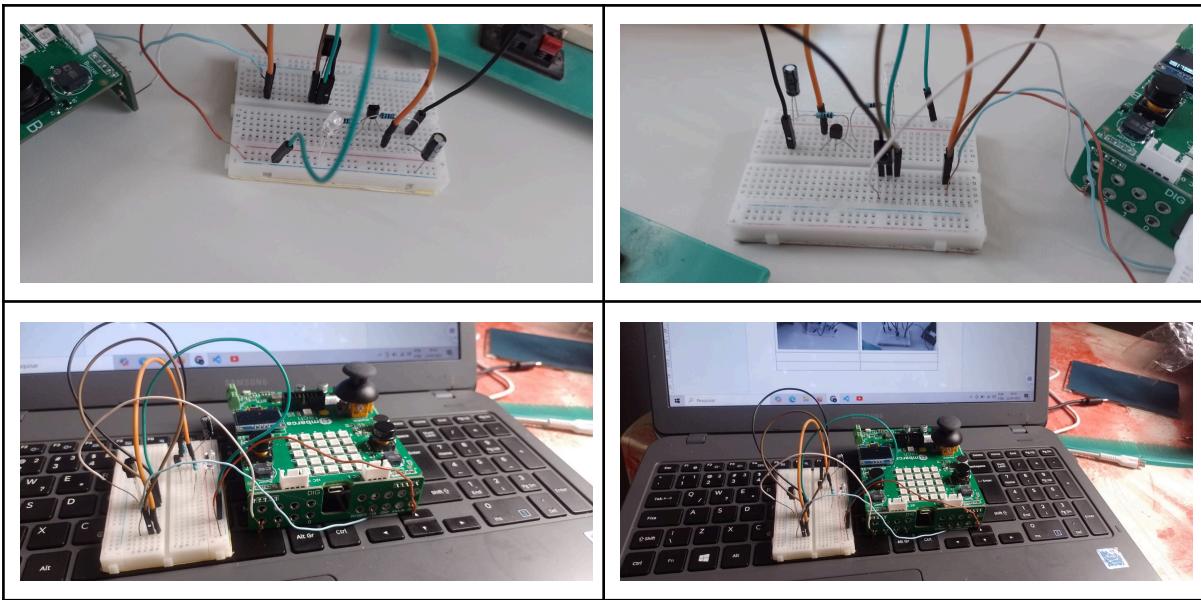
- Quando o TSOP1738 detecta sinal, ativa o transistor BC557
- O transistor funciona como chave para acionar o LED

Proteção e Filtragem:

- Capacitor de 1 μ F: Filtra ruídos na alimentação
- Resistor 10K Ω : Limita corrente na base do transistor
- Resistor 470 Ω : Limita corrente do LED (~10mA)

ANEXO (CIRCUITO/CÓDIGO)





```

    ProJeto_Lampada Ajustavel - PROJ_FINAL.1.5
    =====

    O objetivo deste projeto é mostrar o uso da aplicação de um sistema simples
    que pode ser aperfeiçoado para uso em situações reais
    Autor: Antonio José #File: PROJ_01 INT #Date : março, 2025

    ===== Aplicação utilizando Display OLED, PWM, LEDS e BOTÕES =====

    ===== Projeto_Residencia: EmbarcaTech =====

    ===== BIBLIOTECAS UTILIZADAS =====

#include <stdio.h> //Entrada e saida padrao
#include <string.h> //Manipulação de estring(ex: memset, sprintf)
#include <stdlib.h> //Utilidade geral e alocação na memoria
#include "pico/stlplib.h" //Guarda-Chuva
#include "pico/binary_info.h" //Para informaçoes binarias e depuração
#include "inc/ssi306.h" //Controle do DISPLAY_OLED SSD1306
#include "hardware/i2c.h" //Comunicação I2C
#include "hardware/pwm.h" //Controle do PWM
/* ===== DEFINICOES E PINOS UTILIZADOS =====

#define VERDE 11 //Unico utilizado no programa
#define AZUL 12 //definido
#define VERMELHO 13 //definido
// #define BTA 5 // Botão A antes utilizado sem o circuito para incrementar a luminosidade
#define BTA 3 // Botão A
#define BTB 6 // Botão B
const uint I2C_SDA = 14; //PINO SDA utilizado para comunicação com o display
const uint I2C_SCL = 15; //PINO SCL utilizado para comunicação com o display
/* ===== VARIAVEIS DE CONFIGURACAO DO PWM =====

const uint16_t periodo = 200; //Periodo do PWM (valor máximo de tensão)
const float divisor_pwm = 16.0; //Divisor fracional do clock do PWM
const uint16_t stemp_pwm = 40; //Passo de incremento e decremento do duty cycle do LED
/* ===== VARIAVEIS PARA AJUSTE DE LUMINOSIDADE =====

uint8_t niveluminosidade = 0; // Nivel inicial da luminosidade (1 a 5)
uint8_t contadorA = 0; // Contador para os botões
/* 
```

```

/*
-----#
#           VARIAVEIS PARA O TRATAMENTO DO DEBOUCE NOS BOTÕES          #
#           =====#
# -----#
//Definições para tratamento do debounce pra garantir que n ajam leituras incorretas      #
uint32_t last_button_a_time = 0; //Armazena o ultimo tempo em que o botao A foi precionado   #
uint32_t last_button_b_time = 0; //Armazena o ultimo tempo que o botao B foi precionado       #
const uint32_t DEBOUNCE_DELAY = 200; //Tempo de debounce em milisegundos                      #
/* -----#


/*
-----#
#           FUNÇÃO PARA HABILITAR O PWM DO PINO                         #
#           =====#
# -----#
void config_pwm(int led) { //Recebe um parametro do pino do led           #
    uint slice; //Variável para obter o slice do PWM, divisor da frequência   #
    gpio_set_function(led, GPIO_FUNC_PWM); //Configura para saída PWM           #
    slice = pwm_gpio_to_slice_num(led); //Retorna um slice permitindo o controle do sinal  #

    //Configuração do periodo wrap duty cycle / ajuste do sinal de saída        #
    pwm_set_clkdiv(slice, divisor_pwm); //Define o divisor do clock do PWM      #
    pwm_set_wrap(slice, periodo); //Configura o valor máximo do contador(periodo pwm)  #
    pwm_set_gpio_level(led, stemp_pwm); //Define o nível de pwm no pino correspondente  #
    pwm_set_enabled(slice, true); //Habilita o pwm no slice correspondente     #
}

/*
-----#
#           FUNÇÕES PARA O DISPLAY                                     #
#           =====#
#           FUNÇÃO DE REDENRIZAÇÃO DO DISPLAY                         #
#           =====#
# -----#
# A função Render_on_display é responsavel por redenrizar o conteudo do buffer SSD na area  #
# especificado pelo parametro do Display                                #
# -----#
#           =====PARAMETROS=====#
#     uint8_t *ssd: ponteiro para o buffer que contem os dados a serem exibidos      #
#     struct render_area *area: ponteiro para uma estrutura que define a area do      #
#     display onde o conteudo sera redenrizado                               #
#           ===== render_on_display atualiza a tela =====#
# -----#
void render_on_display(uint8_t *ssd, struct render_area *area);

/*
-----#
#           FUNÇÃO PARA DESENHAR A ESTRING NO DISPLAY                   #
#           =====#
# -----#
# A função extern void ssd1306_draw_string é responsavel por desenhar umma estring no      #
# buffer SSD a partir de coordenadas, manipula o buffer para incluir a representação grafica  #
# nas cordenadas corretas no limite do display                                #
# -----#
#           =====PARAMETROS=====#
#     uint8_t *ssd: buffer onde a string sera desenhada                  #
#     int16_t x: coordenada x onde a string comeca a ser desenhada       #
#     int16_t y: coordenada y onde a string comeca a ser desenhada       #
#     char *string: A string a ser desenhada                            #
#           ===== ssd1306_draw_string lida com a renderização de textos =====#
# -----#
extern void ssd1306_draw_string(uint8_t *ssd, int16_t x, int16_t y, char *string);

/*
-----#
#           FUNÇÃO PARA CALCULAR O CUMPRIMENTO DO BUFFER                 #
#           =====#
# -----#
# A função calculate_render_area_buffer_length calcula o cumprimento do buffer necessário  #
# para redenizar uma area especifica do Dysplay                                #
# A implementação calcula o tamanho do buffer com base nas dimensões da area(largura altura)#
# -----#
#           =====PARAMETROS=====#
#     struct render_area *area: ponteiro para estrutura que define a area a ser redenrizada.  #
# == calculate_render_area_buffer_length ajuda a garantir que haja espaço suficiente no ==  #
# == para a área que se deseja renderizar.buffer ==                           #
# -----#
void calculate_render_area_buffer_length(struct render_area *area);

/*
-----#
#           FUNCAO PARA EXIBIR A MENSAGEM NO DISPLAY                     #
#           =====#
# -----#
#           =====PARAMETROS=====#
#     uint8_t *ssd: ponteiro para o buffer onde os dados da mensagem serao armazenados  #
#     struct render_area *frame_area: ponteiro para a area do display para exibição da mensagem  #
#     const char *mensagem: string que sera exibida                            #
# -----#
void exibir_mensagem(uint8_t *ssd, struct render_area *frame_area, const char *mensagem) {
    memset(ssd, 0, ssd1306_buffer_length); //Usa o tamanho correto do buffer para: Limpar o buffer
    ssd1306_draw_string(ssd, 5, 0, mensagem); //chama a função ssd1306_draw_string para desenhar a mensagem a partir das cordenadas do display(5,0)
    render_on_display(ssd, frame_area); // chama a função render_on.display para redenizar o conteudo do buffer do Dysplay na area especifica
}
*/

```

```

/*
 * FUNCOES PARA EXIBIR AS MENSAGENS NO DISPLAY
 */
=====
# LAMPADA DESLIGADA
#
# A FUNÇÃO da função exibir_lampada é exibir a mensagem LAMDA DESLIGADA no display
# -----
# ===== PARAMETROS =====
# uint8_t *ssd: ponteiro para o buffer onde os dados da mensagem serao armazenados
# struct render_area *frame_area: ponteiro para area do display
# declara um array de char de 30 caracteris para armazenar a mensagem
# utiliza o sprintf() para formatar e armazenar em mensagem
# chamada da função EXIBIR_MENSAGEM passando buffer, area de redenização e a mensagem...
# formatada.
#
# -----
void exibir_lampada_desligada(uint8_t *ssd, struct render_area *frame_area) {
    char mensagem[30]; // array de 30 caracteris -----
    sprintf(mensagem, sizeof(mensagem), "Lampada desligada"); // mensagem e a função sizeof para percorrer os caracteris --
    exibir_mensagem(ssd, frame_area, mensagem); //buffer, area-redenização, mensagem formatada/realiza a tarefa de exibição
} //-----

/*
 * FUNÇÃO PARA ATUALIZAR A MENSAGEM DE LUMINOSIDADE
 */
=====
# Objetivo: exibir o nivel de luminosidade
# -----
# ===== PARAMETROS =====
# uint8_t *ssd: ponteiro para o buffer onde os dados da mensagem serao armazenados
# struct render_area *frame_area: ponteiro para area do display para exibição de imagem
# declara um array de char de 20 caracteris para armazenar a mensagem
# utiliza o sprintf(para formatar e gravar a string, o %d é substituido pelo valor da...
# variavel nivel de luminosidade.
# chama a função Exibir_mensagem: passando o buffer e area de redenização da mensagem...
# limpando o buffer e desenhando a mensagem
#
# -----
void exibir_nivel_luminosidade(uint8_t *ssd, struct render_area *frame_area) {
    char mensagem[20];//array 20caracteris
    sprintf(mensagem, sizeof(mensagem), "Luminosidade: %d", nivelLuminosidade); // mensagem e a função sizeof para percorrer os caracteris, exibir o nivel de
luminosidade#
    exibir_mensagem(ssd, frame_area, mensagem); //limpa e desenha a mensagem
} //-----

/*
 * FUNÇÃO PARA INCREMENTAR E DECREMENTAR A MENSAGEM DE LUMINOSIDADE
 */
=====
# Objetivo: ajustar os clicks no botao e luminosidade e exibir as mensagens
# -----
# ===== PARAMETROS =====
# pino do led
# pino do Botao
# uint8_t *ssd: ponteiro para o buffer onde os dados da mensagem serao armazenados
# struct render_area *frame_area: ponteiro para area do display para exibição de imagem
# -----
void Incrementar(int led, int botao, uint8_t *ssd, struct render_area *frame_area) {
    uint32_t current_time = to_ms_since_boot(get_absolute_time()); //a função que trata de Temporização, ela obtém o tempo atual em milis segundos sendo armazenada por
esta variável o tempo atual -----
    uint32_t *last_time = (botao == BTA) ? &last_button_a_time : &last_button_b_time; //a variável last_time determina qual botão foi precionado e armazena o tempo do
último precionamento em um ponteiro(last_time)#
    //depois verifica se o botão foi precionado e se o tempo desde o último precionamento é maior que um a trazo de debouce, para que n tenha leitura incorreta
    // Verifica se o botão foi pressionado e se passou tempo suficiente desde o último pressionamento
    if (gpio_get(botao) == 0 && (current_time - *last_time) >= DEBOUNCE_DELAY)) {
        *last_time = current_time; // Atualiza o tempo do último pressionamento
        if (botao == BTA && nivelLuminosidade < 5) {// if o botão for clicado e o nível de luminosidade estiver menor que 5
            contadorA++; //Incrementa 1
            nivelLuminosidade++; // incrementa a luminosidade
            printf("Botão A pressionado: %d - Nível de luminosidade da lampada: %d\n", contadorA, nivelLuminosidade); //exibe no display serial
            exibir_nivel_luminosidade(ssd, frame_area); // chama a função para exibir no display o nível de luminosidade atual
        } else if (botao == BTA && nivelLuminosidade > 1) {
            nivelLuminosidade--; //decrementa a luminosidade
            printf("Botão B pressionado - Nível de luminosidade da lampada: %d\n", nivelLuminosidade); // exibi no monitor serial
            exibir_nivel_luminosidade(ssd, frame_area); // chama a função para exibir no display o nível de luminosidade atual
        } else if (botao == BTA && nivelLuminosidade == 1) {// se o nível da luminosidade ao clicar no botão chegar a 1
            nivelLuminosidade--; // ao clicar no botão e chegar em 0 decrementa
            if (nivelLuminosidade == 0) {
                gpio_put(led, 0); //chegando em 0 apaga
                printf("Botão B pressionado - Lampada desligada - Nível de luminosidade: %d\n", nivelLuminosidade); // exibi no monitor serial
                exibir_lampada_desligada(ssd, frame_area); //chama a função que exibe a mensagem de apagada
            }
        }
    }
} //----- FUNÇÃO PRINCIPAL -----
int main() {
    stdio_init_all(); // Inicializa os tipos stdio padrão presentes ligados ao binário
    // Inicialização do i2c
    i2c_init(I2C1, ssd1306_i2c_clock * 1000); //interface I2C1, com frequência do clock definida
    gpio_set_function(I2C_SDA, GPIO_FUNC_I2C); //inicializa os pinos para saída I2C
    gpio_set_function(I2C_SCL, GPIO_FUNC_I2C); //inicializa os pinos para saída I2C
    gpio_pull_up(I2C_SDA); //utiliza o resistor que coloca em nível lógico alto
    gpio_pull_up(I2C_SCL); //utiliza o resistor que coloca em nível lógico alto
    // Inicializa os LEDs
}

```

```

gpio_init(VERDE);
gpio_set_dir(VERDE, GPIO_OUT);

// Inicializa os botões
gpio_init(BTA);
gpio_set_dir(BTA, GPIO_IN);
gpio_pull_up(BTA); // Resistor pull-up para o botão A

gpio_init(BTB);
gpio_set_dir(BTB, GPIO_IN);
gpio_pull_up(BTB); // Resistor pull-up para o botão B

config_pwm(VERDE); // configura o PWM para o LED verde chamando a função de configuração

// Processo de inicialização completa do OLED SSD1306
ssd1306_init();

// Preparar área de renderização para o display (ssd1306_width pixels por ssd1306_n_pages páginas)
struct render_area frame_area = { //especifica a area do display onde irá redenrizar
    .start_column = 0, //define a coluna de inicio 0
    .end_column = ssd1306_width - 1, // coluna de fim 1
    .start_page = 0, //pagina de inicio da redenrização e fim
    .end_page = ssd1306_n_pages - 1 //termina na coluna maxima do display abraangendo todas as paginas disponiveis
};

// FUNÇÃO de calculo do cumprimento do buffer necessário por armazenar os dados da area redenrizada
calculate_render_area_buffer_length(&frame_area);

// Zera o display inteiro
uint8_t ssd(ssd1306_buffer_length); //declara um array chamado ssd usado como buffer para armazenar os dados que serão enviados para o display
memset(ssd, 0, ssd1306_buffer_length); // inicializa todos os bytes do buffer SSD zerando o display
render_on_display(ssd, &frame_area);

// Parte do código para exibir a mensagem no display
render_on_display(ssd, &frame_area); //envia os dados para o Display

if (nivelLuminosidade == 0) {
    exibir_lampada_desligada(ssd, &frame_area);
} else {
    exibir_nivel_luminosidade(ssd, &frame_area);
}

while (true) {
    // Ajusta o nível de PWM baseado no nível de luminosidade
    if (nivelLuminosidade > 0) { //se o nível da luminosidade maior que 0
        pwm_set_gpio_level(VERDE, periodo / 5 * nivelLuminosidade); // ajusta o pwm. calcula a fração do periodo (valor maximo de tensao) / 5 ultipliado pelo nível
        de luminosidade
    } else {
        pwm_set_gpio_level(VERDE, 0); // Certifica-se que o LED está desligado quando nivelLuminosidade é 0
    }

    // Chama a função Incrementar com os parâmetros apropriados
    Incrementar(VERDE, BTA, ssd, &frame_area); // Exemplo de chamada com LED Verde e Botão A. ssd bufer que contem os dados a serem exibidos e &frame_area: area de
    redenrização usada na função
    Incrementar(VERDE, BTB, ssd, &frame_area); // Exemplo de chamada com LED Verde e Botão B. ssd bufer que contem os dados a serem exibidos e &frame_area: area
    de redenrização usada na função

    sleep_ms(10); // Delay de 1 segundo
}
return 0;
}

```

Links do Repositório GitHub e Youtube

Youtube: <https://youtu.be/wffGBtnpPms?si=Mup9qqN808QVuxWd>

GitHub: https://github.com/Anoxi-IFPI/Embarca_Tech.git