



PROJETO FINAL EMBARCATECH

**TÍTULO: SISTEMA EMBARCADO PARA TESTES
AUTOMATIZADOS DE LUMINÁRIAS E DISPLAY EM LINHAS DE
ENVELHECIMENTO (*AGING LINE*).**

**ALUNO: RAVEL SOUZA
CPF: 094.366.673-29**

APRESENTAÇÃO DO PROJETO

A crescente demanda por equipamentos eletrônicos confiáveis e duráveis torna essencial a realização de testes rigorosos antes que esses produtos cheguem ao mercado. Para garantir sua qualidade e resistência ao uso contínuo, é necessário submetê-los a condições reais de operação por longos períodos. Nesse contexto, a Linha de Envelhecimento – ou Aging Line – surge como uma solução fundamental para avaliar o desempenho e a estabilidade de dispositivos como luminárias, displays e outros equipamentos elétricos.

A Aging Line é um sistema projetado para simular o funcionamento prolongado dos produtos, expondo-os a variações elétricas controladas e analisando seu comportamento ao longo do tempo. Durante o teste, os dispositivos permanecem conectados à máquina, onde são submetidos a diferentes tensões de entrada – variando de 110VAC a 380VAC – e operam continuamente até que um período mínimo de teste seja atingido. Esse processo permite identificar falhas prematuras, validar a eficiência dos componentes e garantir que os produtos atendam aos mais altos padrões de qualidade e segurança.

Segue a imagem de uma Aging Line Machine real:



Fonte: http://www.gdduoli.com/e_products_show/?id=52

Durante um ano atuando no Setor da Qualidade de uma fábrica de luminárias LED, identifiquei diversas oportunidades de automação. Entre todas essas possibilidades, a mais relevante e com maior potencial de impacto foi, sem dúvida, a automação de uma Linha de Envelhecimento presente na fábrica. Esse teste desempenha um papel fundamental na garantia da qualidade dos produtos, permitindo:

- Verificação de erros de ligação, assegurando que o produto foi montado corretamente.
- Identificação de irregularidades em componentes fabricados externamente, garantindo a conformidade dos materiais utilizados.
- Avaliação da operação em diferentes faixas de tensão, garantindo que o produto funcione adequadamente em qualquer região do país.
- Validação da durabilidade do produto, certificando que ele pode operar de forma contínua sem falhas prematuras.

Entretanto, a ausência de um sistema embarcado na Linha de Envelhecimento que possibilite a coleta de dados durante a operação representa um grande desperdício de tempo, energia e mão de obra. Atualmente, a máquina apenas submete os equipamentos a variações de tensão, sem registrar informações detalhadas sobre seu desempenho. Como consequência, as mesmas luminárias que permanecem conectadas por horas na Aging Line precisam ser testadas individualmente pelo Setor de Qualidade, prolongando o tempo necessário para obtenção de variáveis críticas que poderiam ser coletadas na primeira etapa do processo. Durante meu período na indústria, observei que, para cada 6 horas de teste na Linha de Envelhecimento, eram necessárias mais 3 horas de testes complementares no Setor de Qualidade, evidenciando uma ineficiência significativa no fluxo de validação.

Além dessas limitações operacionais, a falta de automação também traz riscos à saúde dos trabalhadores. O monitoramento contínuo das luminárias durante o teste exige que um operador esteja presente no ambiente por longos períodos, identificando LEDs queimados ou luminárias apagadas. Mesmo com o uso de óculos de proteção, era comum que os responsáveis pela atividade relatassem desconforto ocular e enxaquecas devido à exposição prolongada à luminosidade intensa. Com a automação e o monitoramento remoto do sistema, a necessidade de presença constante dos operadores no ambiente de testes seria eliminada, reduzindo significativamente os riscos à visão e melhorando as condições de trabalho.

Diante desse cenário, este Projeto Final propõe uma solução automatizada para otimizar o processo de qualidade das luminárias e minimizar os impactos à saúde dos trabalhadores em indústrias de tecnologia. A implementação da placa BitDogLab permitirá a coleta e análise automática de dados durante os testes, eliminando a necessidade de verificações manuais posteriores e garantindo um processo mais eficiente, seguro e confiável. Além disso, a solução apresentada não se limita apenas à indústria de luminárias, podendo ser aplicada a fábricas de displays, lâmpadas e outros processos industriais que utilizam a Aging Line como parte de seus testes de qualidade.

Toda a documentação do projeto está disponível em:
github.com/ravelsouza/ProjetoFinalEmbarcatech_RavelSouza_Linha_de_Envelhecimento

Vídeos com a execução do projeto estão disponíveis em
<https://youtu.be/UFJkonJxTpl> e <https://youtu.be/PXuLB3xBjKk>.

OBJETIVOS DO PROJETO

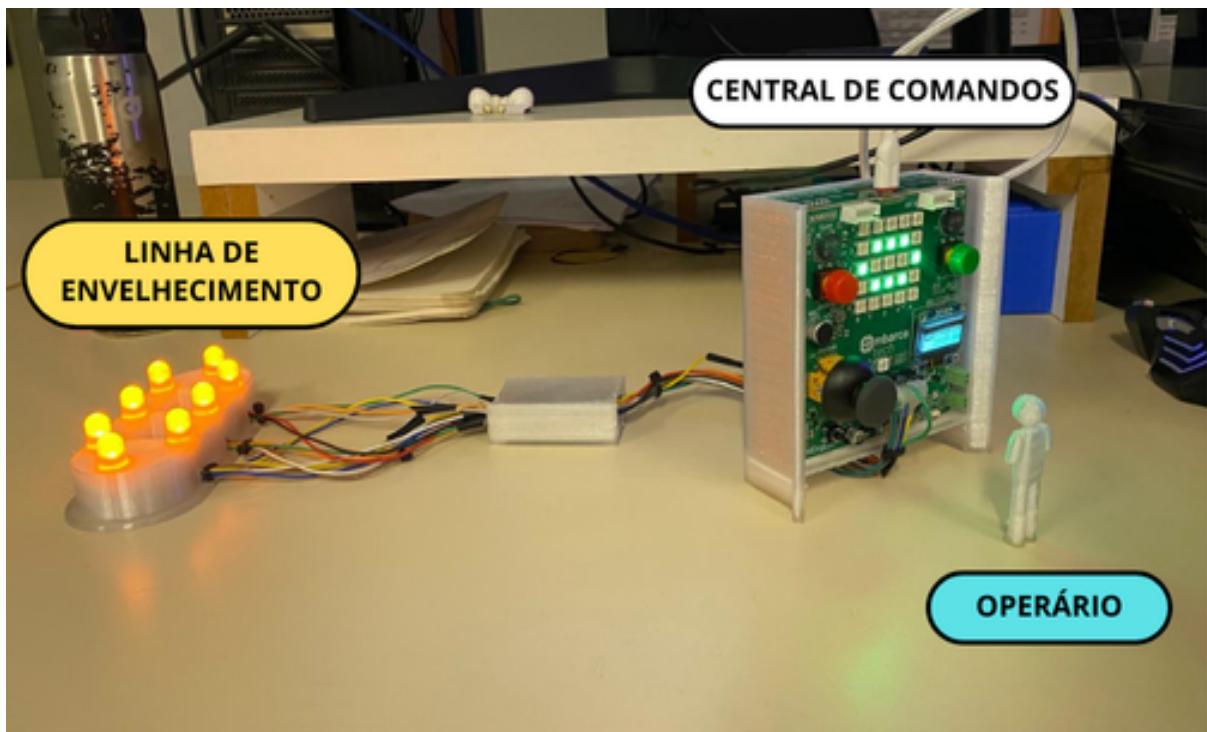
Os principais objetivos desse projeto são:

- Testar meus conhecimentos práticos em Sistemas Embarcados.
- Contribuir com um projeto Open-Source para a comunidade de desenvolvedores.
- Propor melhorias para as indústrias de luminárias e displays.
- Economizar tempo, mão-de-obra e energia.
- Potencializar a produção e o diagnóstico de falhas.
- Prevenir doenças do trabalho advindas do processo de Aging Line.
- Executar um projeto pessoal proposto por mim à alguns anos e provar que tal proposta é válida.

DESCRIÇÃO DO FUNCIONAMENTO

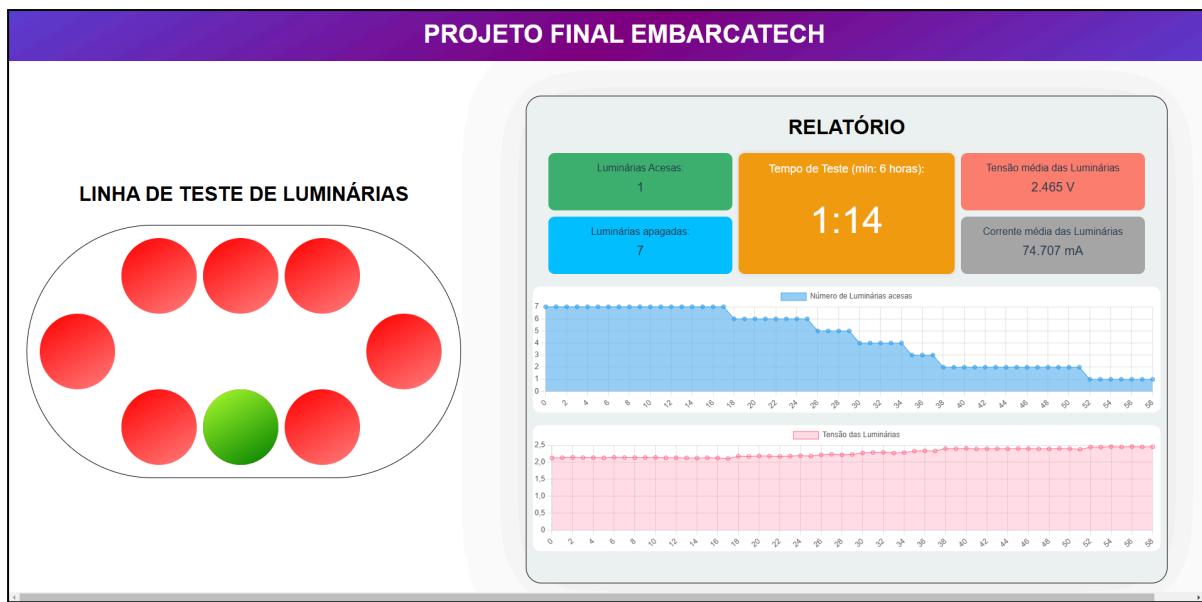
O projeto possui duas principais camadas para o seu funcionamento: Hardware e Software (Firmware e Servidor Flask). É necessário o pleno entendimento de ambas para a execução da proposta.

De início, é importante destacar que o **hardware** é composto, basicamente, pela placa BitDogLab com uma Raspberry Pi Pico W embarcada e por um conjunto de LEDs e resistores. Dessa maneira, para fins de simulação, a placa BitDogLab representará uma central de comandos à parte da Linha de Envelhecimento e os LEDs representarão as luminárias em teste. Em resumo, a placa embarcada - por meio dos General Purpose Input/Output - lê dados digitais de tensão dos LEDs para classificá-los entre acesos e apagados, a fim de representar cada um na Matriz de LEDs RGB presente na placa. Sucessivamente, utiliza-se um dos canais ADC para obter o valor analógico de tensão nos resistores e calcular a corrente do sistema em tempo real. Por fim, os Buzzers são configurados para gerar um alarme depois que o tempo de teste for atingido. No caso do projeto, o tempo base será de 6 minutos, mas vale lembrar que em uma Linha de Envelhecimento real esse tempo é de no mínimo 6 horas. Para prototipar a proposta, foram utilizadas modelagem e impressão 3D. Abaixo encontra-se uma representação do sistema físico desse projeto (Esse esquema será apresentado mais detalhadamente ao longo do trabalho):



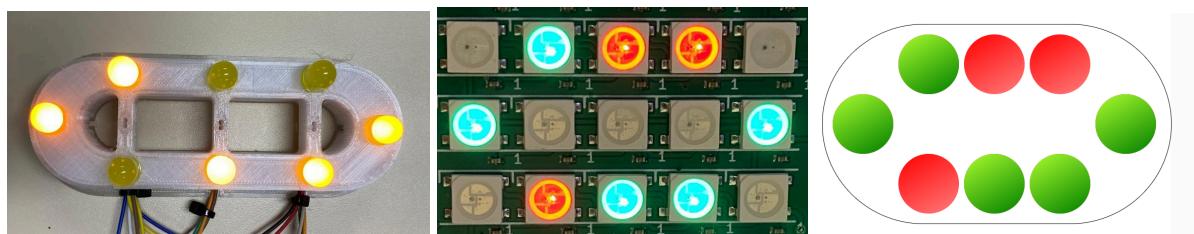
Fonte: Autor, Câmera.

Sucessivamente, o software embarcado (**Firmware**) no microcontrolador deve seguir um ciclo de instruções específicas para essa aplicação. Primeiramente, deve ser medida o nível lógico em cima de cada LED, a fim de identificar os componentes acesos (nível lógico 1) e os apagados (nível lógico 0). Feito essas medições, parte-se para a medição do valor analógico da tensão média entre os LEDs e o cálculo da corrente levando-se em consideração os valores dos resistores. Por fim, calcula-se também o tempo de teste para o melhor aproveitamento de energia e produção. Todos os dados apresentados são enviados via Serial pela Raspberry Pi Pico W e capturados por um servidor Flask, que exibe essas informações e outras que serão apresentadas ao longo do texto. Destaca-se que o monitoramento e a intervenção serão de responsabilidade do Setor de Qualidade, visto que não precisarão mais testar o produto manualmente, é de suma importância a atenção na análise do dashboard gerado pelo servidor. Nessa perspectiva, segue uma imagem do Dashboard do sistema (Esse esquema será apresentado mais detalhadamente ao longo do projeto):



Fonte: Autor, Screenshot.

Perceba a similaridade da “*LINHA DE TESTE DE LUMINÁRIAS*” no Dashboard com o protótipo apresentado anteriormente. Assim como na matriz de LEDs da placa, no servidor os componentes acesos são representados pela cor verde e os componentes apagados pela cor vermelha.



JUSTIFICATIVA

É correto afirmar, conforme experiência pessoal na indústria de luminárias, que o maior empecilho para o aumento da produção na fábrica era o tempo mínimo de teste. Com 6 horas de *Envelhecimento* e 3 horas de ensaios manuais, o Setor de Montagem era limitado a produzir apenas o que o Setor de Qualidade poderia inspecionar dentro do tempo mínimo, pois, havendo uma fabricação excedente, ou as luminárias seriam submetidas a um período de testes menor do que o exigido ou seria necessário a convocação de técnicos para horas-extras. Por esse motivo, os gestores não tinham liberdade para expandir a produção.

Somado a isso, o risco à saúde visual dos trabalhadores também deve ser levado em consideração para a implementação da iniciativa. O projeto, ao embarcar um sistema na planta industrial, consegue realocar os funcionários para ambientes com menor ou nenhuma incidência da forte luminosidade dos equipamentos em teste, prevenindo-os de lesões oculares consequentes desse processo.

Diante o exposto, é notória a relevância do escopo para as indústrias que seguem o padrão abordado, visto que, apesar de exigir a instalação de novos componentes na Linha de Envelhecimento, a proposta é totalmente viável e sustentável. Com a automatização, os técnicos da qualidade economizariam 3 horas de testes manuais e os auxiliares de montagem ganhariam 3 horas de produção.. Caso a empresa não necessite de um crescimento expressivo na produção, pode utilizar-se da iniciativa para adicionar novas etapas na fabricação, com o intuito de elevar a qualidade e o valor de mercado do produto.

ORIGINALIDADE

Embora existam diversas publicações relacionadas à automatização de esteiras e máquinas de testes de equipamentos industriais, a maioria dessas contribuições apresenta uma abordagem mais geral, uma vez que soluções mais específicas frequentemente permanecem em sigilo devido a questões de competitividade e propriedade intelectual. Entre os trabalhos acadêmicos disponíveis, podemos destacar o artigo "**AUTOMAÇÃO NOS PROCESSOS INDUSTRIAS: PROCESSO DE IMPLEMENTAÇÃO E O PAPEL DO GESTOR DE TECNOLOGIA DA INFORMAÇÃO**", de Eric Sampaio De Oliveira, e a dissertação "**AUTOMATIZAÇÃO E INTEGRAÇÃO DE UMA LINHA DE PRODUÇÃO INDUSTRIAL**", de Marco Miguel Marques Teixeira. Ambos os trabalhos abordam a automação e a integração de processos industriais, oferecendo uma visão abrangente sobre os desafios e as estratégias envolvidas.

No entanto, apesar de existirem essas referências, o projeto que estou desenvolvendo é distinto, uma vez que nasce da minha experiência prática e vivência no chão de fábrica. Embora outros tipos de implementação já tenham sido realizados, a proposta do meu projeto é criar o primeiro simulador Open-Source de uma máquina de testes automatizada. Este simulador será uma ferramenta valiosa para guiar as indústrias no processo de automatização, permitindo que elas possam testar e simular diferentes cenários de produção antes de implementar fisicamente o Sistema Embarcado em suas linhas de produção.

O objetivo principal deste projeto não é apenas oferecer uma solução prática, mas também democratizar o acesso a uma tecnologia de teste avançada, proporcionando às indústrias uma forma mais acessível e eficiente de integrar e testar suas máquinas automatizadas. Dessa forma, o simulador contribuirá significativamente para o aprimoramento das práticas de automação, permitindo que as empresas planejem, ajustem e otimizem suas operações de forma mais inteligente e econômica.

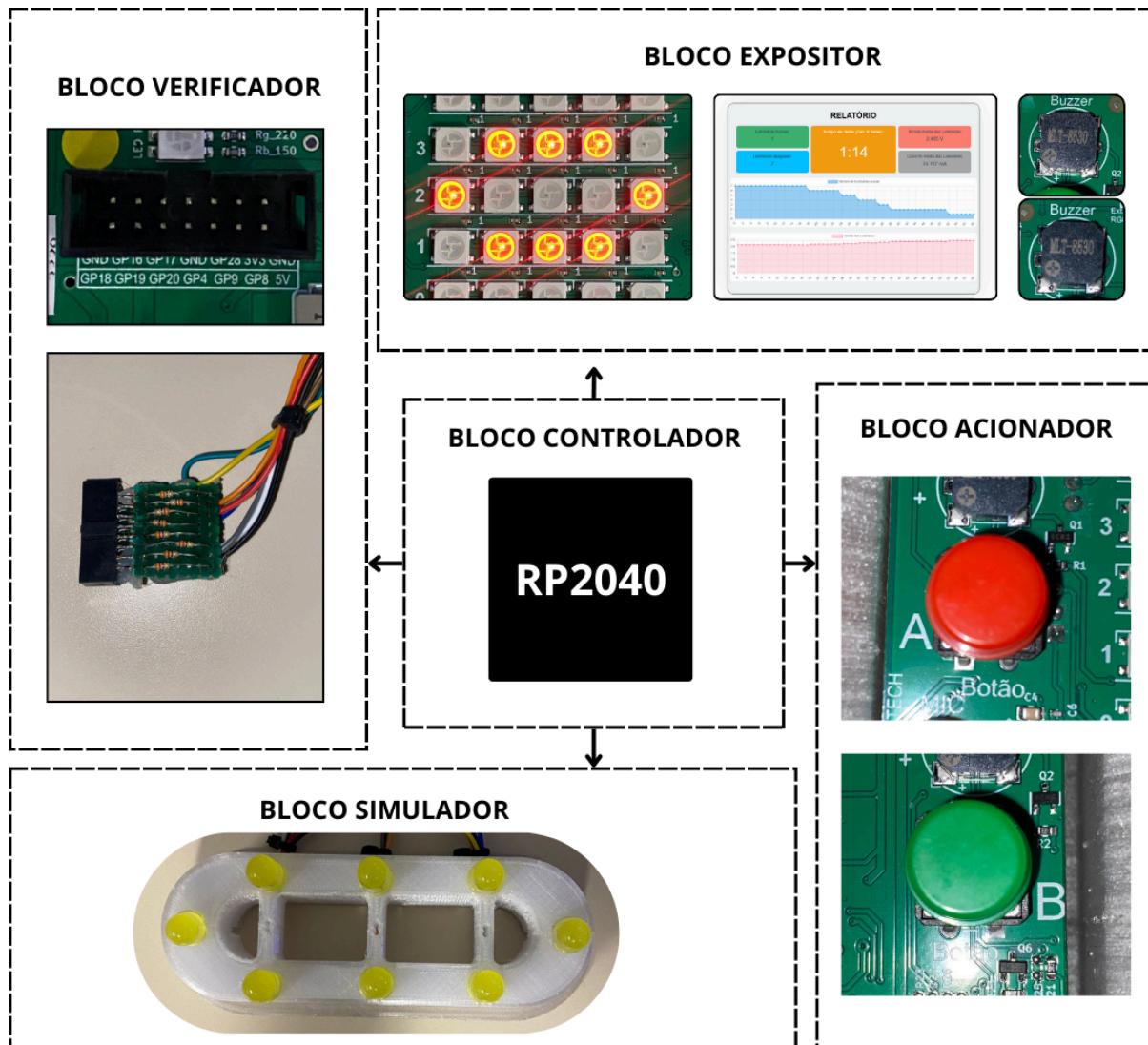
ESPECIFICAÇÕES DO HARDWARE

DIAGRAMA EM BLOCO

Da perspectiva do Hardware, o projeto pode ser dividido em 5 principais blocos: Controlador, Acionador, Simulador, Verificador e Expositor. Esses blocos possuem uma lógica interna de funcionamento que será abordada nesse tópico. Ao todo, essa aplicação embarcada necessita de 22 componentes:

- Raspberry Pi Pico W;
- Matriz de LEDs 5x5;
- 8 resistores de 33 Ohms;
- 2 Pushbuttons;
- 2 Buzzers ML-8530;
- 8 LEDs 5mm.

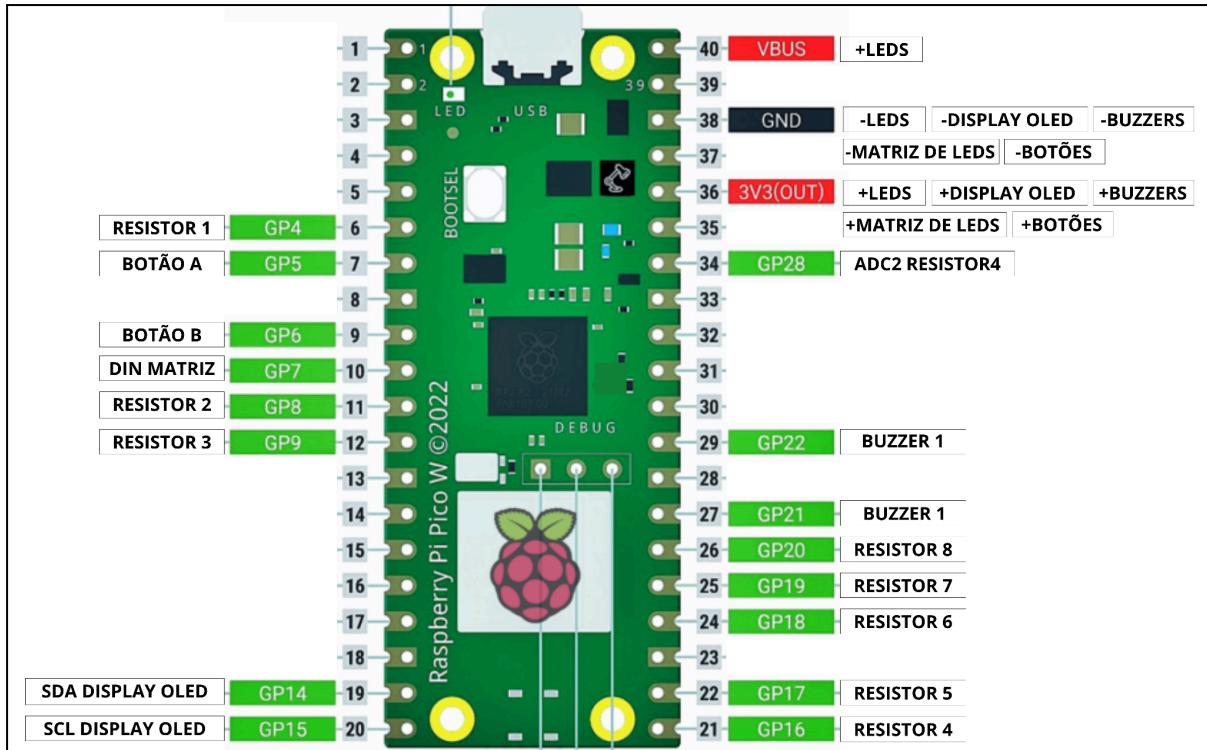
Segue um diagrama de blocos do Hardware:



Fonte: Autor, Canva.

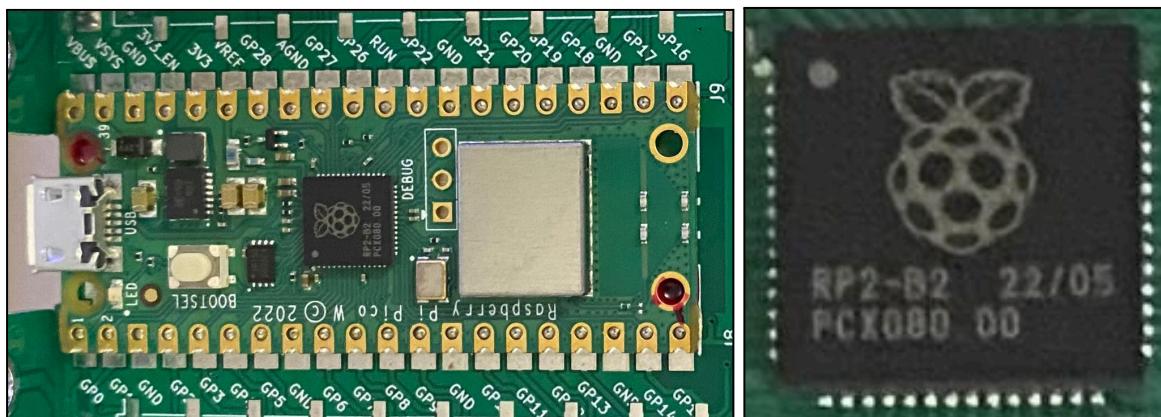
PINAGEM UTILIZADA

Para a interligação do Hardware foram utilizados ao todo 19 pinos da Raspberry Pi Pico W, dentre os quais 16 são pinos GPIO's e, 13 deles, são configurados como entrada digital.



Fonte: Autor, Canva.

BLOCO CONTROLADOR:



Função: Com o RP2040 embarcado, esse bloco será responsável pelo gerenciamento dos demais, recebendo o comando inicial do bloco acionador, acendendo os leds da simulação, obtendo os dados do bloco verificador e, por fim, definindo as configurações necessárias para o bloco expositor.

Configuração: A configuração desse bloco se dá basicamente pelas funções e operações que regem o firmware que está disponível no ANEXO A.

Comandos e Registros utilizados:

- Inclusão de bibliotecas:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pico/stdlib.h"
#include "ws2812.pio.h"
#include <ctype.h>
#include "pico/binary_info.h"
#include "pico/time.h"
#include "hardware/i2c.h"
#include "hardware/gpio.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"
#include "hardware/pio.h"
```

- Definição de Principais Constantes:

```
#define BUTTON_A 5
#define BUTTON_B 6
#define PINO_ADC 28
#define BUZZER_A 21
#define BUZZER_B 22
#define FREQUENCIA_BUZZER 5500
#define SSD1306_WIDTH 128
#define SSD1306_HEIGHT 64
#define WS2812_PIN 7
```

- Inicialização de Comunicação Serial:

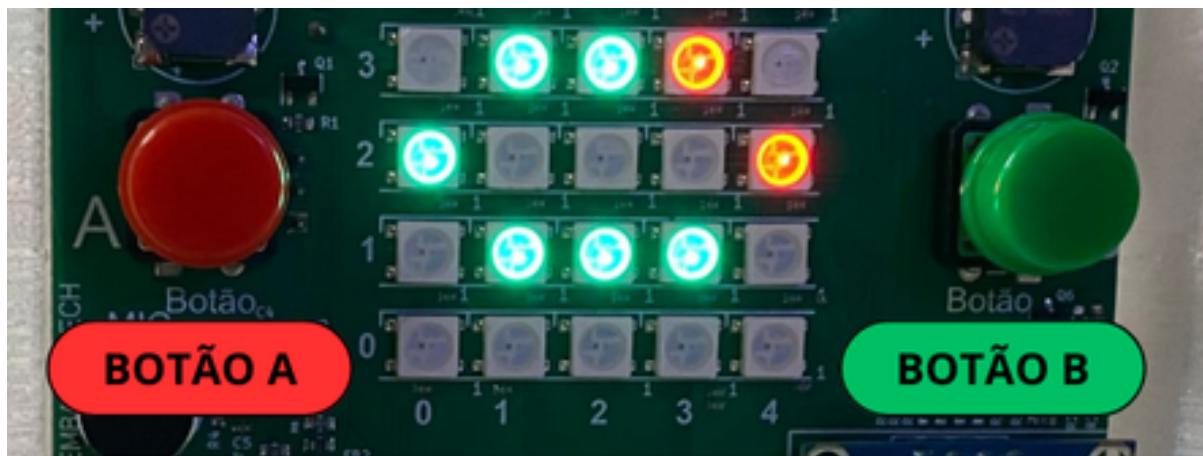
```
int main( ): stdio_init_all( );
CMakeLists.txt: pico_enable_stdio_uart(projetoFinal 1)
CMakeLists.txt: pico_enable_stdio_usb(projetoFinal 1)
```

Pinagem utilizada:

- **Comunicação Controlador-Acionador:** GPIO5 (BOTÃO A) e GPIO6 (BOTÃO B).
- **Alimentação Controlador-Simulador:** 5V e GND.
- **Comunicação Controlador-Verificador:** GPIO4, GPIO8, GPIO9, GPIO16, GPIO17, GPIO18, GPIO19, GPIO20 (Leitura Digital) e GPIO28 (Leitura Analógica).

- **Comunicação Controlador-Expositor:** GPIO7 (Matriz de Leds), GPIO14 e GPIO15 (I2C1 - Display OLED).

BLOCO ACIONADOR:



Função: Constituído por dois botões, será responsável pela ativação e desativação do sistema, uma vez que ele se inicia em stand-by até que o botão B seja pressionado e encerra o período de medição após desativação do alarme com o botão A.

Configuração: Com um botão conectado ao GPIO 5, outro ao GPIO6 e ambos definidos como PULL UP, esse bloco envia um comando de acionamento para o RP2040, sinalizando que o operador deseja inicializar a máquina de testes. Ao atingir o tempo de testes - nesse caso 6 minutos - o bloco acionador é responsável por enviar um comando de reinicialização para o Controlador.

Comandos e Registros utilizados:

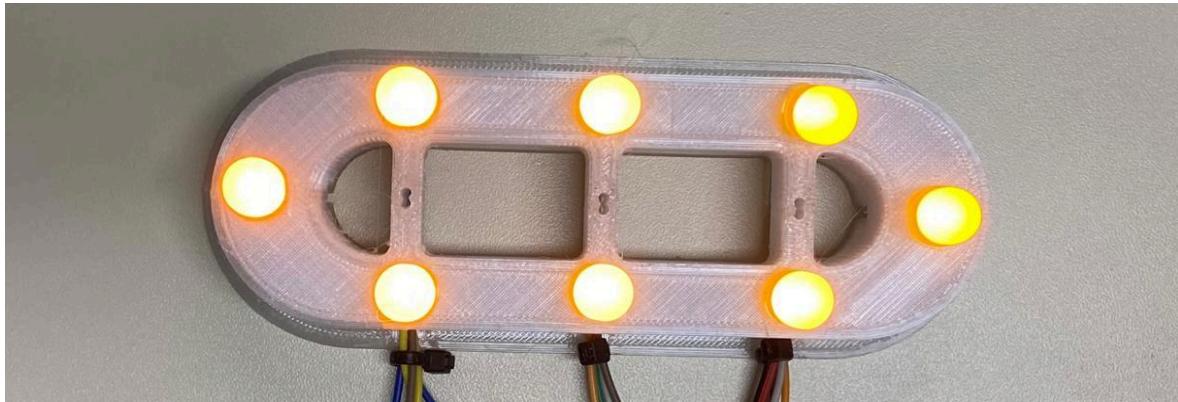
- Inicialização dos botões A e B como entrada e pull-up:

```
for(int i = 5; i < 7; i++){
    gpio_init(i);
    gpio_set_dir(i, GPIO_IN);
    gpio_pull_up(i);
}
```

Pinagem utilizada:

- **Botão A:** GPIO5, 5V e GND;
- **Botão B:** GPIO6, 5V e GND;

BLOCO SIMULADOR - LINHA DE ENVELHECIMENTO:



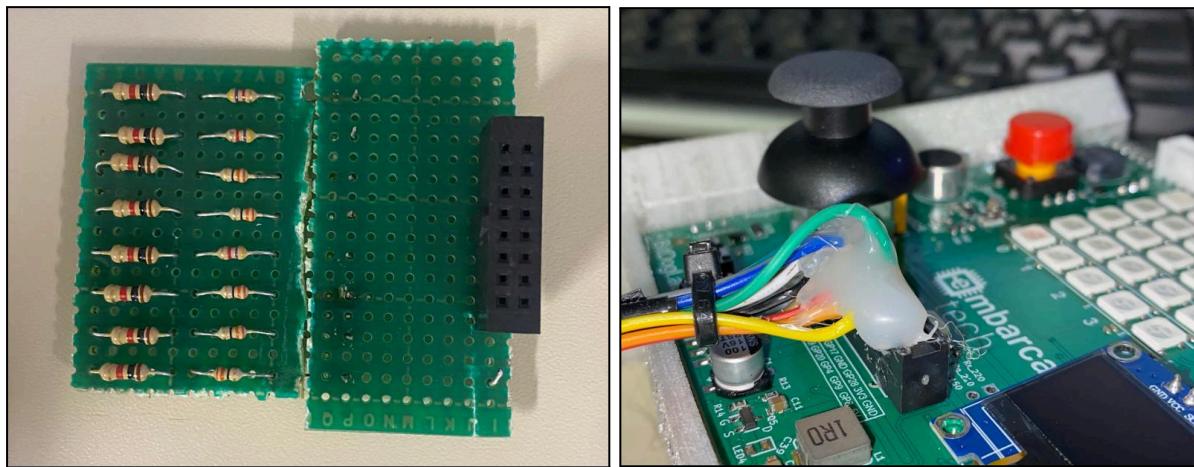
Função: Composto por 8 Leds conectados em paralelo, serve para representar uma Linha de Envelhecimento real, com possíveis falhas nos leds, ligações invertidas ou mal contato entre os terminais.

Configuração: Esse bloco é configurado apenas para acender os LEDS da simulação, que por sua vez podem estar queimados ou polarizados inversamente. Logo, apenas dois pinos são necessários para a sua alimentação , o GND e o 5V. Ao todos são 8 LEDS conectados em paralelo nesses pinos. Referente ao código, não há configurações necessárias para serem implementadas.

Pinagem utilizada:

- LEDS +: 5V.
- LEDS -: GND.

BLOCO VERIFICADOR:



Função: Com 8 pinos digitais e um pino Digital-Analógico definidos como entrada, esse bloco obterá medições sobre o estado dos LEDs na esteira de teste, enviando esses dados para o Controlador com o fito de que ele identifique e sinalize LEDs apagados ou com valores inadequados de tensão e corrente.

Configuração: Diferentemente do Bloco Simulador, essa unidade possui vários pinos configurados como entrada em seu código. São eles: GPIO4, GPIO8, GPIO9, GPIO16, GPIO17, GPIO18, GPIO19 e GPIO20 para as leituras digitais e GPIO28 para a leitura analógica do sistema. Os pinos digitais serão responsáveis por detectar níveis lógicos altos do Bloco Simulador - por meio de um divisor de tensão e da função `gpio_get()` - e enviar essas informações para o Controlador. Os pinos digitais configurados como entrada na Raspberry Pi Pico W definem como nível lógico alto valores entre 1.5V e 3,3V. Com esse conceito, pode-se utilizar um divisor de tensão para capturar valores nesse intervalo sob os resistores de 33 Ohms e obter matematicamente o valor da tensão e da corrente em cada LED.

O pino analógico-digital, por sua vez, será responsável por capturar o valor digital proporcional à tensão média nos LEDs e enviar para o Controlador armazenar e transformá-lo em valores reais de tensão para, posteriormente, encontrar a corrente média do sistema. São realizadas 360 leituras digitais e analógicas em 6 minutos de teste, ou seja, é feito uma leitura de todos os pinos digitais e do pino analógico a cada segundo.

Comandos e Registros utilizados:

- Declaração dos pinos de entrada:

```
const uint32_t pinos[10] = {4, 8, 9, 16, 17, 18, 19, 20, 21, 22};
const uint32_t pinos_config[10] = {0,0,0,0,0,0,0,0,1,1};
for(int cont = 0; cont < 10; cont++){
    gpio_init( pinos[cont] );
    gpio_set_dir( pinos[cont], pinos_config[cont] );
    sleep_ms(1);
}
adc_init();
adc_gpio_init(PINO_ADC);
adc_select_input(2);
```

- Declaração das variáveis de registro:

```
float tensao;
float corrente;
uint32_t luminarias_acesas = 0;
```

- Leitura digital dos pinos: Lê os pinos armazenados de pinos[0] a pinos[7] e se houver tensão, configura a cor como verde, caso contrário, como vermelho.

```
for( int cont = 0; cont < 8; cont ++ ){
```

```

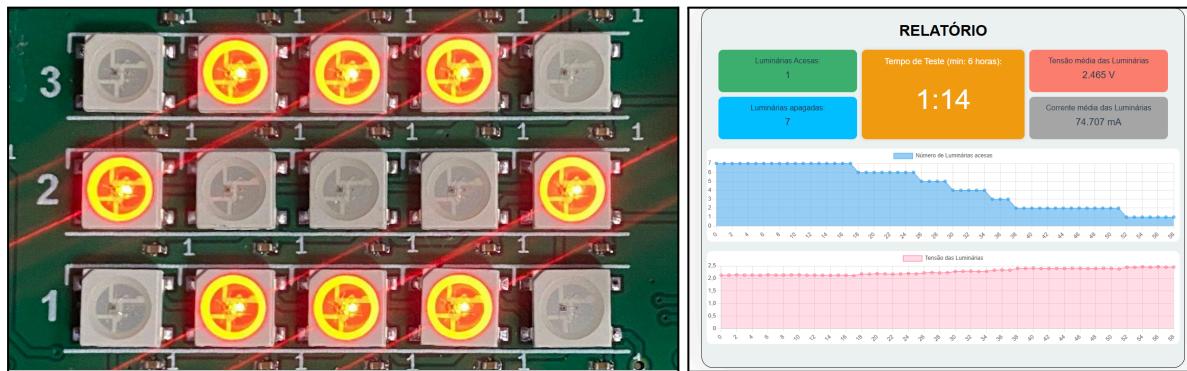
if(gpio_get( pinos[cont] ))
{
    quantidade[cont] = 1;
    luminarias_acesas += 1;
    simulacao[cont] = urgb_u32(0,5,0);
}
else
{
    quantidade[cont] = 0;
    simulacao[cont] = urgb_u32(5,0,0);
}
}
}

```

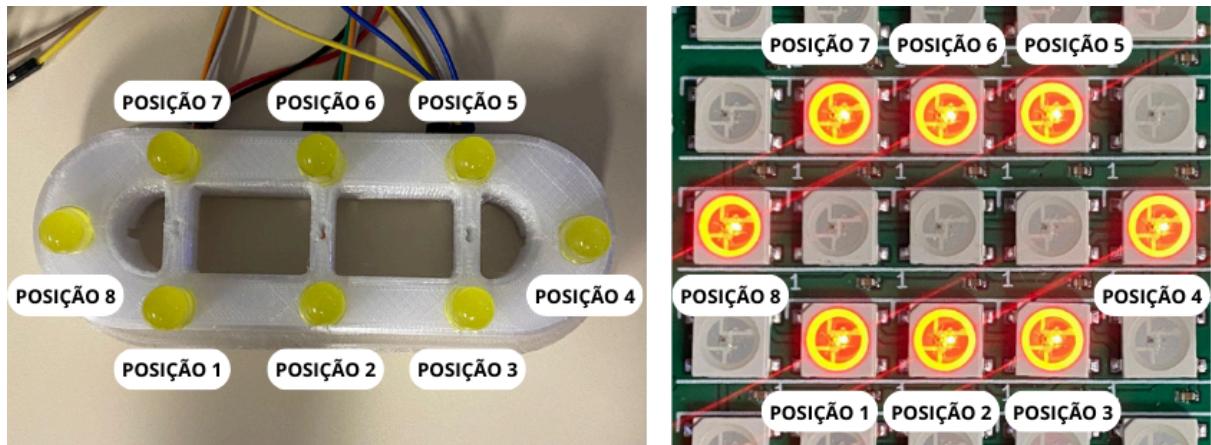
Pinagem utilizada:

- **Divisor de Tensão do LED1:** GPIO4.
- **Divisor de Tensão do LED2:** GPIO8.
- **Divisor de Tensão do LED3:** GPIO9.
- **Divisor de Tensão do LED4:** GPIO16.
- **Divisor de Tensão do LED5:** GPIO17.
- **Divisor de Tensão do LED6:** GPIO18.
- **Divisor de Tensão do LED7:** GPIO19.
- **Divisor de Tensão do LED8:** GPIO20.
- **Leitura Analógica de um dos Divisores de Tensão:** GPIO28.

BLOCO EXPOSITOR



Função: Por fim, após o tratamento dos dados, o Bloco Expositor comunicará ao operador o estado dos LEDs na máquina por meio de uma Matriz de Leds RGB e um servidor Flask (a construção do servidor Flask não será abordado nesse trabalho mas seu código estará disponível em https://github.com/ravelsouza/ProjetoFinalEmbarcatech_RavelSouza_Linha_de_Esvaescimento). Na Matriz, cada LED SMD aceso simboliza um LED/luminária conectado à máquina com suas respectivas posições. Veja o exemplo abaixo:



Na figura à esquerda está o protótipo de uma Linha de Envelhecimento e as respectivas posições das luminárias. Na figura à direita, está a Matriz de Leds da placa BitDogLab diferindo as luminárias acesas das apagadas e, também, sinalizando as posições delas ao longo da máquina. Essa rastreabilidade é muito importante para a praticidade na desconexão de luminárias defeituosas da máquina.

Configuração: A unidade Expositora recebe um comando do RP2040 após o processamento dos dados recebidos do Bloco Verificador. São acesos 8 dos 25 LEDS da matriz, simbolizando os 8 LEDS do Bloco Simulador, onde a cor do LED na matriz determina o estado da “luminária” na máquina (Verde = luminária acesa, Vermelho = luminária apagada). A Matriz de LEDs RGB está conectada ao GPIO7 da Raspberry Pi Pico W e é atualizada a cada segundo até que o tempo de teste seja excedido.

Comandos e Registros utilizados:

- Definição do pino de controle da matriz:

```
#define WS2812_PIN 7
```

- Definição da função de controle da matriz: Essa é a principal função do Bloco Expositor. Ela recebe, entre outros parâmetros, uma lista com os pinos dos leds que devem ser acesos na Matriz e outra lista com a cor com que esses LEDs devem acender.

```
static inline void teste_esteira(PIO pio, uint sm, uint len, int leds[], int row,
uint32_t* cor) {
```

```
    int cor_led = 0;
    for (uint i = 0; i < len; ++i) {
        bool led_acesso = false;
        for (int j = 0; j <= row; ++j) {
            if (leds[j] == i) {
                put_pixel(pio, sm, cor[cor_led]);
                cor_led += 1;
```

```

        led_aceso = true;
        break;
    }
}
if (!led_aceso) {
    put_pixel(pio, sm, urgb_u32(0, 0, 0));
}
}

```

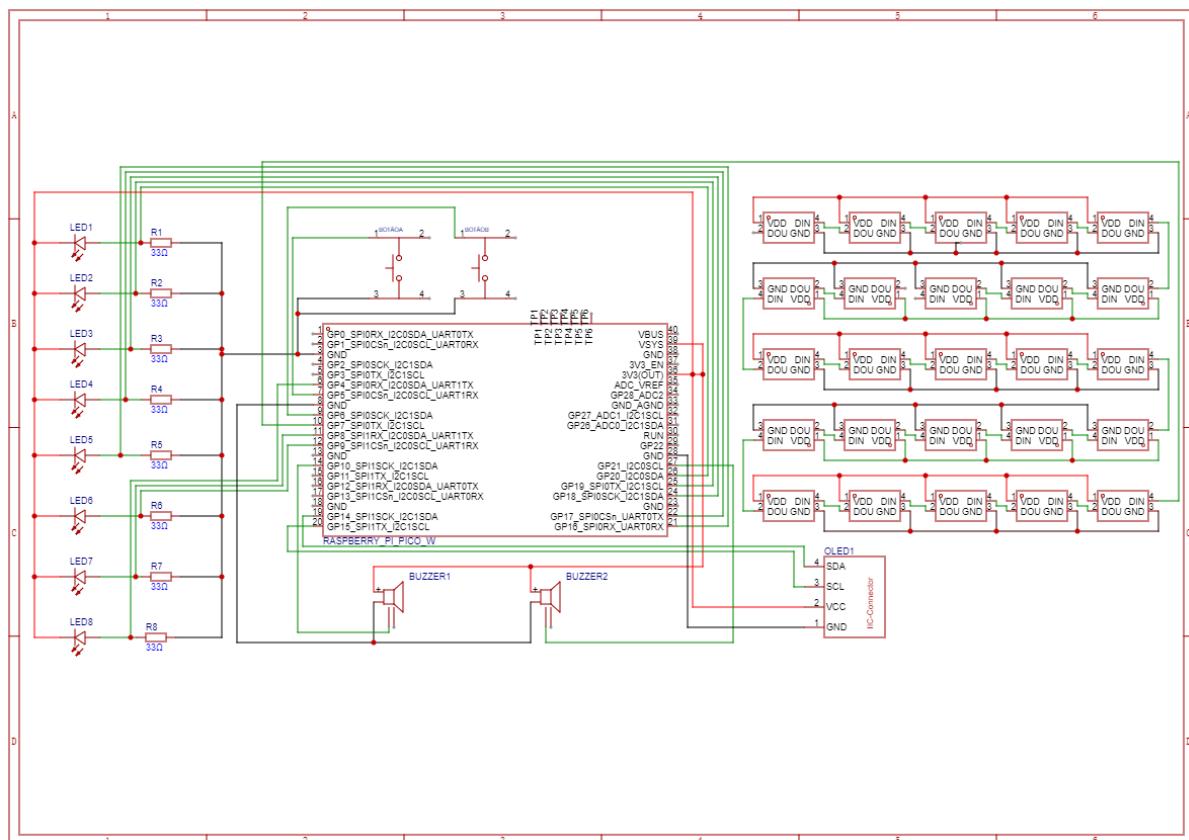
Pinagem utilizada:

- **Matriz de LEDs:** GPIO7, 5V e GND.

CIRCUITO COMPLETO DO HARDWARE (Esquemático)

O esquemático do circuito pode ser representado de diversas formas, como com simbologia eletrônica ou simulação de componentes reais. As duas formas estão representadas abaixo. Vale ressaltar que as ligações dos botões, buzzers, e demais componentes foram construídas levando em consideração a documentação da placa BitDogLab, disponível em: <https://github.com/BitDogLab/BitDogLab>.

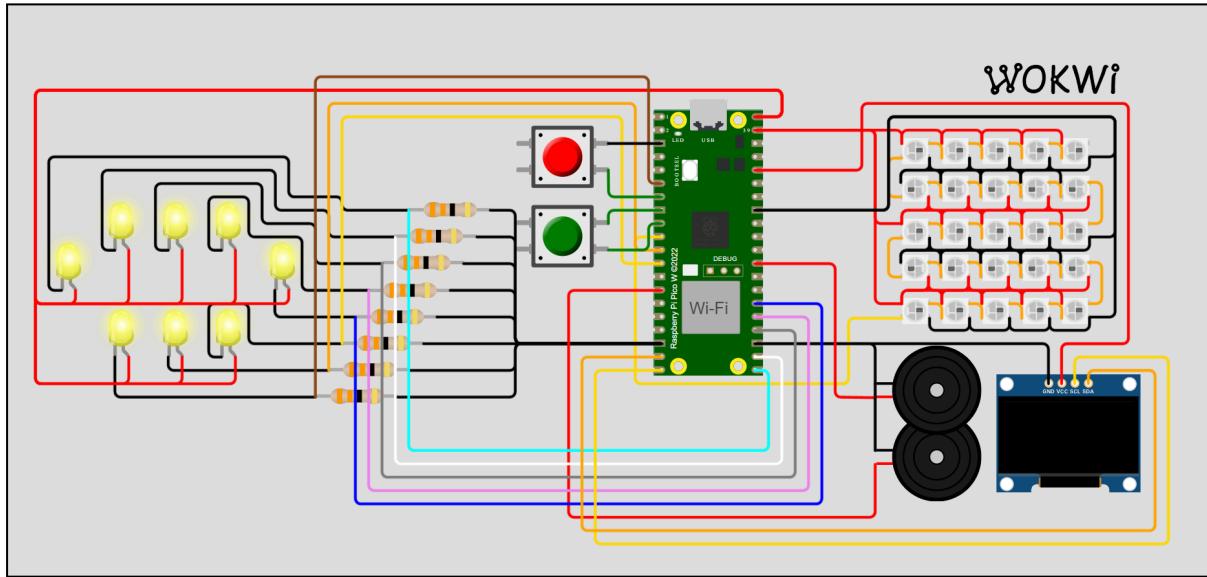
SIMBOLOGIA ELETRÔNICA:



Fonte: Autor, EasyEda.

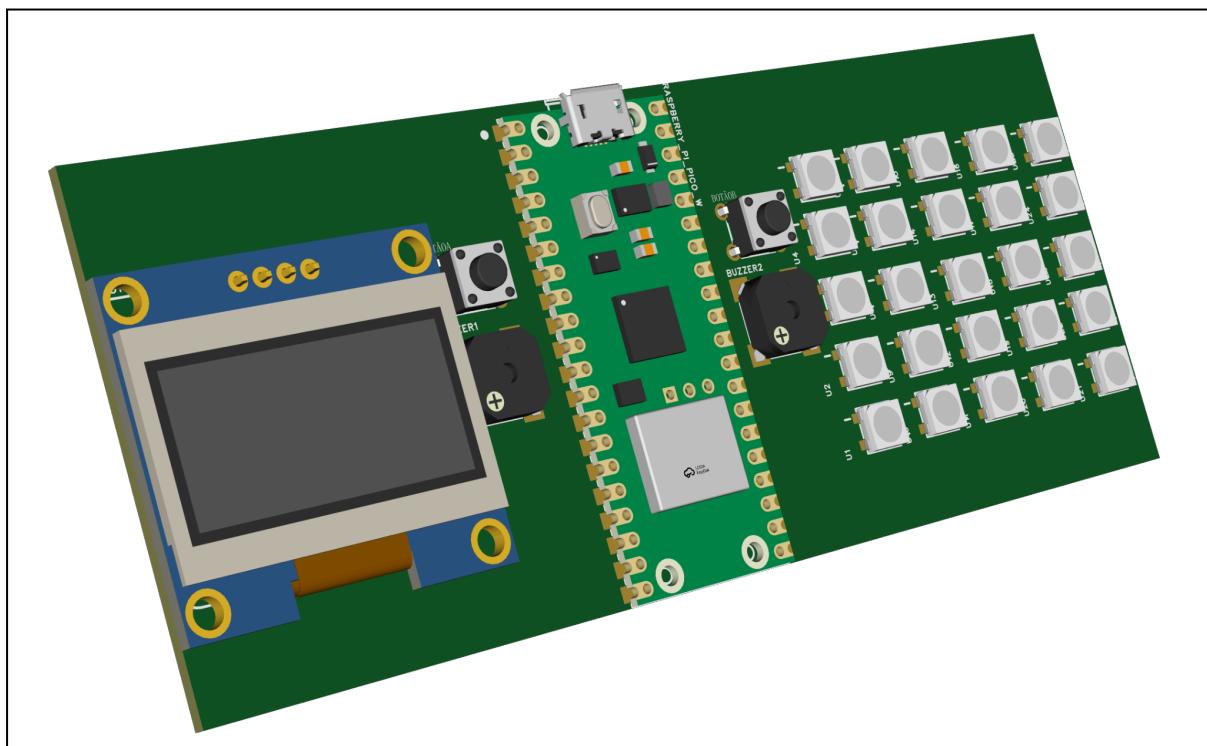
SIMULAÇÃO DE COMPONENTES REAIS:

<https://wokwi.com/projects/421537041654796289>



Fonte: Autor, Wokwi.

É possível, além de utilizar placas prontas como a BitDogLab, projetar placa própria para o projeto. O Software EasyEda possui uma alta performance de modelagem e possui versão gratuita. Abaixo é mostrado um exemplo de construção de PCB apenas com os componentes necessários para a aplicação:



Fonte: Autor, EasyEda.

ESPECIFICAÇÃO DO FIRMWARE

O firmware embarcado para a placa Raspberry Pi Pico W foi desenvolvido utilizando o SDK (Software Development Kit) oficial fornecido pela própria fabricante. O SDK oferece uma série de bibliotecas e ferramentas que facilitam a integração com o hardware do microcontrolador RP2040, permitindo o desenvolvimento eficiente de aplicações embarcadas. A linguagem escolhida para o desenvolvimento do firmware foi C, pois é amplamente utilizada em sistemas embarcados pela sua performance, controle de baixo nível e ampla compatibilidade com o hardware.

O ambiente de desenvolvimento utilizado para a codificação foi o VS Code (Visual Studio Code), um editor de código leve, mas poderoso, que oferece uma série de extensões para suportar a programação em C, além de facilitar a depuração e o controle de versão do projeto. Como ferramenta de simulação, foi utilizado o Wokwi, um simulador online que permite testar o código em ambientes virtuais, o que é útil para testar comportamentos sem a necessidade de hardware físico. O Wokwi, em particular, oferece suporte para placas como o Raspberry Pi Pico, facilitando a visualização do desempenho do código e ajudando na identificação precoce de erros.

Para organizar o código-fonte do projeto de forma modular e clara, ele foi estruturado em três arquivos principais, cada um com um papel específico na funcionalidade geral do firmware. A divisão foi feita para melhorar a legibilidade, reutilização do código e manutenção do sistema. Os principais arquivos são: ***“includes.h”***, ***“functions.h”***, ***“constantes.h”*** e ***“main.c”***.

Essa estrutura de organização facilita a manutenção do projeto, pois cada parte do código está claramente separada conforme sua função. Segue abaixo uma representação da organização dos arquivos no diretório raiz:

> generated	•
> templates	•
❖ .gitignore	U
Ⓜ CMakeLists.txt	U
Ⓒ constantes.h	U
Ⓒ functions.h	U
Ⓒ includes.h	U
Ⓒ main.c	U
≡ pico_sdk_import.cmake	U
⊕ serial_p.py	U
Ⓒ ws2812_parallel.c	U
≡ ws2812.pio	U

Fonte: Autor, Screenshot (VS Code).

BLOCOS FUNCIONAIS

- **BIBLIOTECAS:** As bibliotecas são componentes essenciais em qualquer projeto de firmware. No caso do **Raspberry Pi Pico W**, as bibliotecas utilizadas são fornecidas pelo **SDK** da própria Raspberry Pi, além de outras bibliotecas externas que podem ser utilizadas conforme a necessidade do projeto.

Para disponibilizá-las para uso, é necessário acrescentá-las no arquivo CMakeLists.txt do diretório da seguinte maneira (O arquivo para a configuração do CMakeLists.txt encontra-se no ANEXO E):

- *target_link_libraries(projetoFinal hardware_pwm hardware_i2c pico_stdlib hardware_pio hardware_adc)*

Descrição das Funcionalidades

#include <stdio.h>: Fornece funções para manipulação de entrada e saída, como printf, scanf e fopen.

#include <stdlib.h>: Fornece funções para alocação e desalocação de memória, controle de processos, e conversão de strings.

#include <string.h>: Contém funções para manipulação de strings, como strcmp.

#include "pico/stdlib.h": Oferece funções para inicializar o microcontrolador, manipular GPIOs, temporizadores, interagir com periféricos e outros aspectos do hardware.

#include "ws2812.pio.h": Fornece funções para controle de tiras ou matrizes de LEDs WS2812 usando o PIO (Programmable Input/Output) do Raspberry Pi Pico.

#include <cctype.h>: Fornece funções para manipulação de caracteres.

#include "pico/binary_info.h": Fornece macros e funções para incluir informações binárias adicionais no código compilado.

#include "pico/time.h": Fornece funções para gerenciar temporizadores e medir o tempo, permitindo operações como atrasos, contagem de tempo e gerenciamento de relógios.

#include "hardware/i2c.h": Fornece funções para configurar e gerenciar a comunicação I2C entre o Raspberry Pi Pico e outros dispositivos, como sensores e displays.

#include "hardware/gpio.h": Fornece funções para configurar, ler e escrever nos pinos GPIO do microcontrolador.

#include "hardware/adc.h": Permite ler valores analógicos dos pinos configurados como entradas ADC.

#include "hardware/pwm.h": Fornece funções para gerar sinais PWM, que podem ser usados para controle de velocidade de motores, brilho de LEDs, entre outras aplicações.

#include "hardware/pio.h": Permite usar os blocos PIO para implementar protocolos de comunicação personalizados.

- **FUNÇÃO PRINCIPAL:** A função principal (geralmente chamada de main()) é o ponto de entrada do Firmware. É onde o programa começa a ser executado e onde o fluxo de controle principal é gerenciado. Dentro da função principal, as inicializações do sistema são realizadas. Veja o ANEXO A.

Definição das Variáveis

```
// Constante usada para converter o valor digital-analógico em tensão.

const float conversion_factor = 3.3f / (1 << 12);

// Define os pinos dos Leds e dos Buzzers.

const uint32_t pinos[10] = {4, 8, 9, 16, 17, 18, 19, 20, 21, 10};

// Define as configurações dos pinos anteriormente declarados.

const uint32_t pinos_config[10] = {0,0,0,0,0,0,0,0,1,1};

// Variáveis de armazenamento.

float tensao;

float corrente;

uint32_t start_time;

uint8_t minutos = 0;

uint32_t simulacao[8] = {};

uint32_t quantidade[8] = {};

uint32_t luminarias_acesas = 0;

uint32_t current_time;

uint32_t tempo_de_teste;

uint16_t result;

uint sm;

uint offset;

PIO pio;

bool success;
```

Fluxo do Código: O Firmware foi desenvolvido, resumidamente, para Inicializar o Sistema, Ler dados Digitais e Analógicos dos LEDs, Expor esses dados na Matriz de LEDs, Enviar as informações por Comunicação Serial e Regular o Tempo de Teste dos Componentes. Abaixo está a abordagem simplificada do fluxo (Fluxo completo em ANEXO A).

- **Iniciar o Sistema:** Ao clicar do Botão B

```
if(!gpio_get(BUTTON_B))
```

- **Ler dados Digitais e Analógicos:**

```
gpio_get(pinos[cont])
```

- **Expor Dados na Matriz de LEDs:**

```
exibir_na_matriz(pio, sm, 25, todos_os_leds, 7, cor_na_matriz);
```

- **Enviar informações via Serial:**

```
for( int controle = 0; controle < 8; controle++) {
```

```
    printf( "%d ", estado_dos_leds[ controle ] );
}
```

```
printf("%d %d ", luminarias_acesas, 8 - luminarias_acesas);
```

```
printf("%d:%u ", minutos, tempo_de_teste);
```

```
printf("%0.3f %0.3f\n", tensao, corrente);
```

- **Regular Tempo de Teste:**

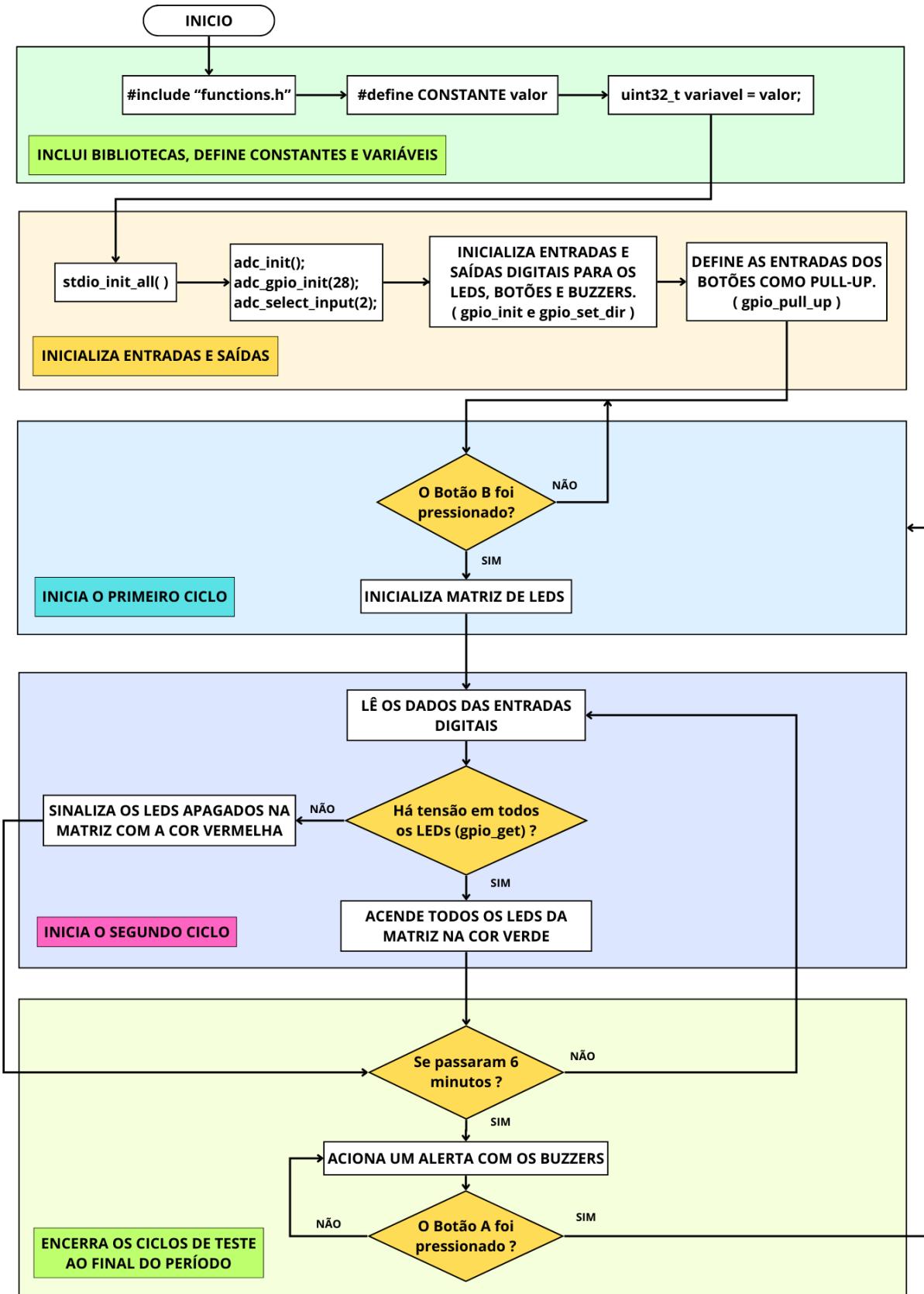
```
if( tempo_de_teste >= 59 ) // Contagem de minutos
```

```
{
    minutos += 1;
    tempo_inicial = time_us_32();
}
```

```
while( minutos == 6 ) // Checagem do tempo Limite (6 minutos)
```

```
{
    buzzer_alerta();
    if(!gpio_get( BUTTON_A ))
    {
        minutos = 0;
        break;
    }
}
```

FLUXOGRAMA DA FUNÇÃO PRINCIPAL



Fonte: Autor, Canva.

FUNÇÕES SECUNDÁRIAS: As funções secundárias são aquelas que realizam operações auxiliares dentro do firmware, mas não são responsáveis pela execução direta da lógica principal do sistema. Elas são chamadas pela função principal ou por outras funções secundárias para dividir a execução do código em módulos menores e mais fáceis de gerenciar.

Descrição das Funcionalidades - Principais Funções Secundárias

Todas as funções secundárias estão disponíveis no repositório do GitHub fornecido anteriormente ou no ANEXO B.

uint32_t urgb_u32(uint8_t r, uint8_t g, uint8_t b): A única função secundária que retorna uma variável, ela converte valores RGB (vermelho, verde e azul) em um valor único de 32 bits para representar uma cor em um sistema que utiliza um valor de cor de 32 bits (como em displays ou LEDs endereçáveis) e Um valor de 32 bits representando a cor no formato 0xRRGGBB, onde RR, GG e BB são os valores hexadecimais para as cores vermelha, verde e azul, respectivamente.

```
uint32_t urgb_u32(uint8_t r, uint8_t g, uint8_t b) {
    return
        ((uint32_t) (r) << 8) |
        ((uint32_t) (g) << 16) |
        (uint32_t) (b);
}
```

void SSD1306_init(): Inicializa um display OLED com o controlador SSD1306, que é frequentemente usado em displays gráficos de pequeno porte, como um display de 128x64 pixels. Essa função configura o controlador para que ele possa começar a exibir dados no display.

```
void SSD1306_init() {
    uint8_t cmd[] = {
        SSD1306_SET_DISP,           // desliga o display
        SSD1306_SET_MEM_MODE,      // define o modo de endereço da memória 0 =
horizontal, 1 = vertical, 2 = página
        0x00,                      // modo de endereçamento horizontal
        SSD1306_SET_DISP_START_LINE, // define a linha de início do display
para 0
        SSD1306_SET_SEG_REMAP | 0x01, // define o remapeamento de
segmentos, o endereço da coluna 127 é mapeado para SEG0
        SSD1306_SET_MUX_RATIO,     // define a razão de multiplexação
        SSD1306_HEIGHT - 1,         // Altura do display - 1
        SSD1306_SET_COM_OUT_DIR | 0x08, // define a direção de escaneamento
COM (comum). Escanear de baixo para cima, COM[N-1] para COM0
        SSD1306_SET_DISP_OFFSET,   // define o offset do display
```

```

    0x00,           // sem offset
    SSD1306_SET_COM_PIN_CFG,
#if ((SSD1306_WIDTH == 128) && (SSD1306_HEIGHT == 32))
    0x02,
#elif ((SSD1306_WIDTH == 128) && (SSD1306_HEIGHT == 64))
    0x12,
#else
    0x02,
#endif
    SSD1306_SET_DISP_CLK_DIV,    // define a razão de divisão do relógio do
display
    0x80,           // razão de divisão 1, frequência padrão
    SSD1306_SET_PRECHARGE,
    0xF1,
    SSD1306_SET_VCOM_DESEL,
    0x30,
    SSD1306_SET_CONTRAST,      // define o controle de contraste
    0xFF,
    SSD1306_SET_ENTIRE_ON,     // ativa o display inteiro para seguir o
conteúdo da RAM
    SSD1306_SET_NORM_DISP,     // define o display como normal (não
invertido)
    SSD1306_SET_CHARGE_PUMP,   // define a bomba de carga
    0x14,
    SSD1306_SET_SCROLL | 0x00,  // desativa o rolo horizontal se estiver
ativado.
    SSD1306_SET_DISP | 0x01, // liga o display
};
SSD1306_send_cmd_list(cmds, count_of(cmds));
}

```

void inicializa_oled(): Inicializa o Display OLED 128x64 com uma imagem da Logo do Projeto Embarcatech.

```

void inicializa_oled(){
    // Inicializa o barramento I2C com a frequência especificada pela constante
SSD1306_I2C_CLK
    i2c_init(i2c_default, SSD1306_I2C_CLK * 1000);
    // Configura os pinos SDA e SCL do I2C para os pinos padrão do PICO
    gpio_set_function(PICO_DEFAULT_I2C_SDA_PIN, GPIO_FUNC_I2C);
    gpio_set_function(PICO_DEFAULT_I2C_SCL_PIN, GPIO_FUNC_I2C);
    gpio_pull_up(PICO_DEFAULT_I2C_SDA_PIN);
    gpio_pull_up(PICO_DEFAULT_I2C_SCL_PIN);
    SSD1306_init();
}

```

```

struct render_area frame_area = {
    start_col: 0,           // Começo da coluna
    end_col: SSD1306_WIDTH - 1, // Fim da coluna (largura do display - 1)
    start_page: 0,          // Começo da página
    end_page: SSD1306_NUM_PAGES - 1 // Fim da página (número de páginas
do display - 1)
};

calc_render_area buflen(&frame_area);
static uint8_t buf[SSD1306_BUF_LEN];
memset(buf, 0, SSD1306_BUF_LEN); // Zera o buffer com zeros

// Renderiza a tela com o conteúdo vazio (tela apagada)
render(buf, &frame_area);

// Exibe uma animação carregando a imagem 'embarcatech' no display
for (int i=0; i<sizeof(embarcatech); i+=24) {
    memcpy(buf+i, &embarcatech[i], 24);
    render(buf, &frame_area);
    sleep_ms(1);
}
}

```

void inicializar_pio(PIO pio, uint sm, uint offset): configura uma máquina de estado (*State Machine*) de um periférico PIO (*Programmable Input/Output*) no microcontrolador. PIOS são usados para tarefas de controle de hardware de baixa e alta complexidade, como controle de LEDs.

```

void inicializar_pio(PIO pio, uint sm, uint offset){
    bool success =
pio_claim_free_sm_and_add_program_for_gpio_range(&ws2812_program, &pio,
&sm, &offset, WS2812_PIN, 1, true);
    hard_assert(success);
    ws2812_program_init(pio, sm, offset, WS2812_PIN, 800000, IS_RGBW);
}

```

void inicializa_bloco_expositor(PIO pio, uint sm): Responsável por gerar uma animação com os LEDs de uma matriz de 8 LEDs usados para sinalização antes do início das medições. Serve para indicar que o sistema está em modo de inicialização ou reinicialização.

```

void inicializa_bloco_expositor(PIO pio, uint sm)
{
    int cor_index = 0;
    uint32_t cores[4] = {
        urgb_u32(20, 0, 0), // Vermelho (R, G, B)

```

```

    urgb_u32(0, 0, 20), // Azul (R, G, B)
    urgb_u32(0, 20, 0), // Verde (R, G, B)
    urgb_u32(20, 20, 20) // Branco (R, G, B)
};

for (int ciclo = 0; ciclo < 4; ++ciclo) {
    for (int row = 0; row < 8; ++row) {
        acender_leds_na_matriz(pio, sm, 25, inicial[row], row, cores[cor_index]);
        sleep_ms(200);
    }

    for (int row = 6; row >= 0; --row) {
        acender_leds_na_matriz(pio, sm, 25, final[row], row, cores[cor_index]);
        sleep_ms(200);
    }
}

// Muda a cor para o próximo ciclo
cor_index = (cor_index + 1) % 4;
}
acender_leds_na_matriz(pio, sm, 25, inicial[7], 7, cores[3]);
sleep_ms(300);
for(int i = 0; i < 2; i++)
{
    acender_leds_na_matriz(pio, sm, 25, inicial[7], 7, urgb_u32(0,0,0));
    sleep_ms(300);
    acender_leds_na_matriz(pio, sm, 25, inicial[7], 7, cores[3]);
    sleep_ms(300);
}
}

```

void acender_leds_na_matriz(PIO pio, uint sm, uint len, int leds[], int row, uint32_t cor): Recebe uma lista contendo as posições dos LEDs que devem ser ligados na Matriz, além da cor em que eles dever ser ligados. acende LEDs em uma matriz, com base nas posições fornecidas na lista leds[], na linha especificada pela variável row. Ela também define a cor dos LEDs por meio da variável cor.

```

void acender_leds_na_matriz(PIO pio, uint sm, uint len, int leds[ ], int row, uint32_t
cor) {
    // Apaga todos os LEDs
    for (uint i = 0; i < len; ++i) {
        bool led_aceso = false;
        for (int j = 0; j <= row; ++j) {
            if (leds[j] == i) {
                put_pixel(pio, sm, cor); // Acende o LED na cor especificada
                led_aceso = true;
            }
        }
    }
}

```

```

        break;
    }
}
if (!led_aceso) {
    put_pixel(pio, sm, urgb_u32(0, 0, 0)); // Apaga os LEDs não escolhidos
}
}
}
}

void exibir_na_matriz(PIO pio, uint sm, uint len, int leds[ ], int row, uint32_t* cor): Similar à função anterior, essa função também acende LEDs em uma matriz, mas aqui ela usa uma lista de cores fornecidas pelo usuário, permitindo que diferentes LEDs tenham cores diferentes. Ela também utiliza um array de pinos para controlar quais LEDs devem ser acesos.
```

```

void exibir_na_matriz(PIO pio, uint sm, uint len, int leds[], int row, uint32_t* cor) {

    // Apaga todos os LEDs
    int cor_led = 0;
    for (uint i = 0; i < len; ++i) {
        bool led_aceso = false;
        // Verifica se o LED atual deve ser aceso (comparando com a linha do vetor de
        LEDs)
        for (int j = 0; j <= row; ++j) {
            if (leds[j] == i) {
                put_pixel(pio, sm, cor[cor_led]); // Acende o LED na cor especificada
                cor_led += 1;
                led_aceso = true;
                break;
            }
        }
        if (!led_aceso) {
            put_pixel(pio, sm, urgb_u32(0, 0, 0)); // Apaga os LEDs não escolhidos
        }
    }
}

}

}

}

void inicializa_buzzer(int BUZZER): Recebe o valor do pino de um dos Buzzers
para defini-lo como saída.
```

```

void inicializa_buzzer(int BUZZER) {
    gpio_set_function(BUZZER, GPIO_FUNC_PWM);
    uint slice_num = pwm_gpio_to_slice_num(BUZZER);
    pwm_config config = pwm_get_default_config();
```

```

pwm_config_set_clkdiv(&config, clock_get_hz(clk_sys) / (FREQUENCIA_BUZZER
* 4096));
pwm_init(slice_num, &config, 1);
pwm_set_gpio_level(BUZZER, 0);
}

```

void buzzer_alerta(): Gera um alerta sonoro utilizando o buzzer. Ele alterna o nível lógico do pino do buzzer entre alto e baixo a cada segundo, criando um som de alerta (frequência de 5.5 kHz).

```

void buzzer_alerta()
{
    pwm_set_gpio_level(BUZZER_A, FREQUENCIA_BUZZER);
    pwm_set_gpio_level(BUZZER_B, FREQUENCIA_BUZZER);
    sleep_ms(1000);
    pwm_set_gpio_level(BUZZER_A, 0);
    pwm_set_gpio_level(BUZZER_B, 0);
    sleep_ms(1000);
}

```

PROTOCOLOS DE COMUNICAÇÃO

A Raspberry Pi Pico W usa várias interfaces e protocolos para comunicação. Esses protocolos permitem que ela interaja de forma eficiente com uma variedade de sensores e realize tarefas como coleta e envio de dados. Para a implementação do projeto, é necessário utilizar três principais protocolos de comunicação da Raspberry Pi Pico W: *Uart*, *I2C* e *GPIO*. Veja abaixo as suas definições e formatos de aplicação no Firmware da placa.

UART

Definição: Uart significa *Universal Asynchronous Receiver/Transmitter* e é um protocolo de comunicação assíncrona, isto é, não requer um sinal de relógio. A sincronização é feita com base em taxas de baud (velocidade de transmissão) acordadas entre os dispositivos.

Formato de Aplicação: No trabalho, essa interface é útil para exibir para o usuário e para o técnico do Setor da Qualidade as variáveis mais importantes do sistema: Quantidade de Luminárias acesas e apagadas, Tensão Média das Luminárias, Corrente Média e Corrente Total e Tempo de Teste. Todas essas informações são enviadas por segundo via comunicação serial. Vale destacar que o servidor Python utiliza-se da porta COM4 para imprimir os dados, ou seja, estabelece uma comunicação serial com a Raspberry Pi Pico W para obter as variáveis.

I2C

Definição: O I2C (*Inter-Integrated Circuit*) é um protocolo de comunicação serial síncrona que permite que vários dispositivos (como sensores, displays, etc.) se comuniquem usando dois fios, um para dados (SDA) e outro para o relógio (SCL). Cada dispositivo tem um endereço único no barramento.

Formato de Aplicação: No projeto, esse protocolo é o responsável por imprimir o Logo do Embarcatech no Display OLED utilizando uma array contendo as informações necessárias da imagem a ser gerada.

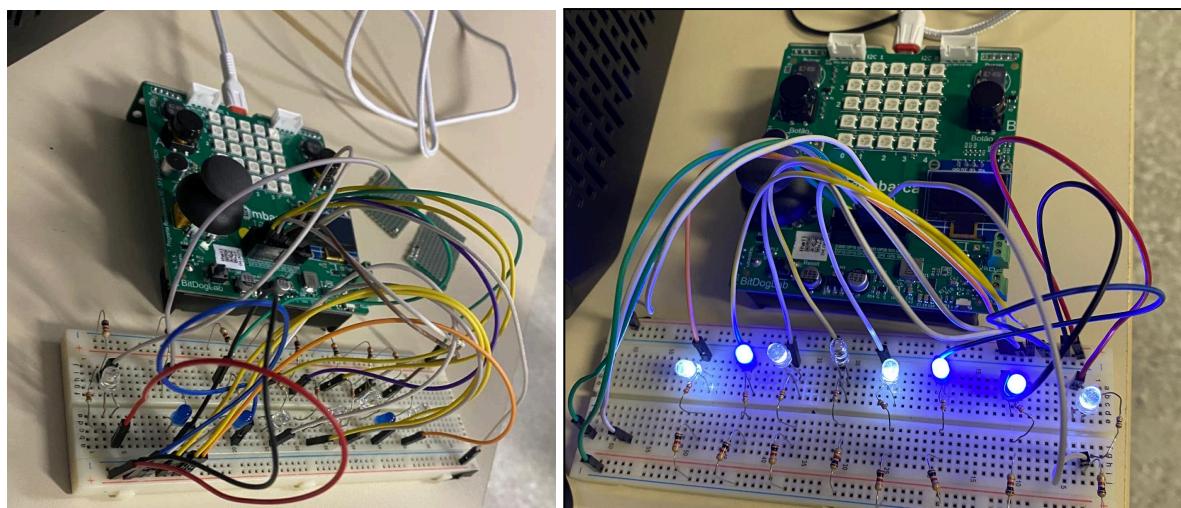
GPIO

Definição: Os pinos GPIO (*General Purpose Input/Output*) são pinos digitais na Raspberry Pi Pico W (e em outros microcontroladores) que podem ser configurados tanto para entrada quanto para saída, permitindo que interajam com sinais externos de forma analógica (em alguns casos) ou digital.

Formato de Aplicação: Por fim, os pinos GPIO's executam uma função fundamental para o desenvolvimento da proposta. São eles que capturam os dados de Tensão nos LEDs e os envia para o RP2040 para processamento, além de acionar o alarme nos buzzers e receber os comandos dos botões. Vale lembrar que a Matriz de LEDs também é controlada por um GPIO, o pino 7.

TESTES DE VALIDAÇÃO

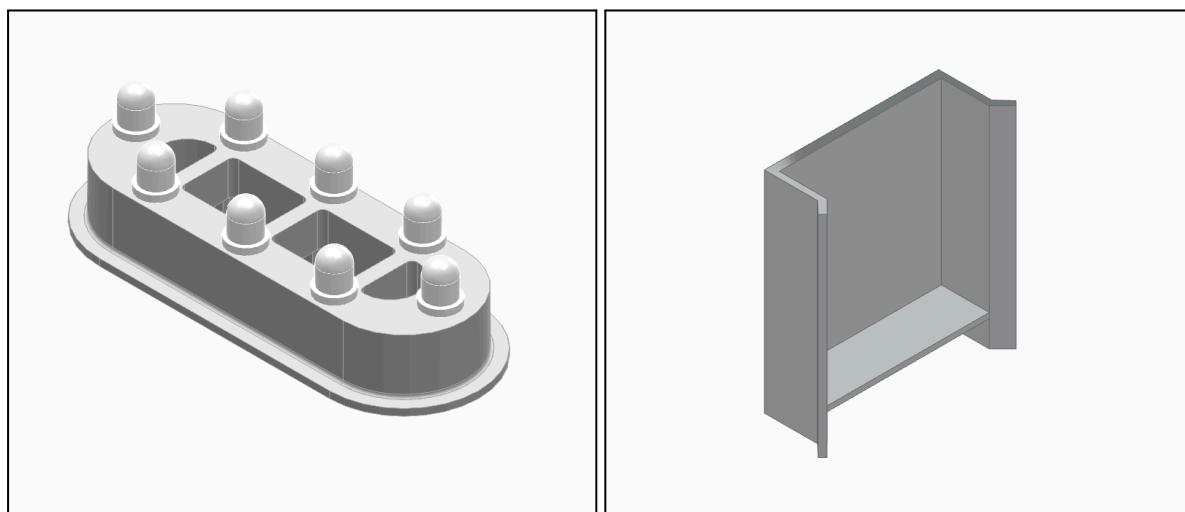
A princípio, foram realizados testes nos softwares Wokwi e TinkerCad em relação às ligações, aos divisores de tensão e à programação do sistema. Os testes foram satisfatórios e demonstraram segurança no dimensionamento dos componentes. Sucessivamente, o primeiro teste físico foi implementado utilizando protoboard e jumpers para conectar a placa de desenvolvimento aos resistores e LEDS. Segue uma imagem desse primeiro teste:



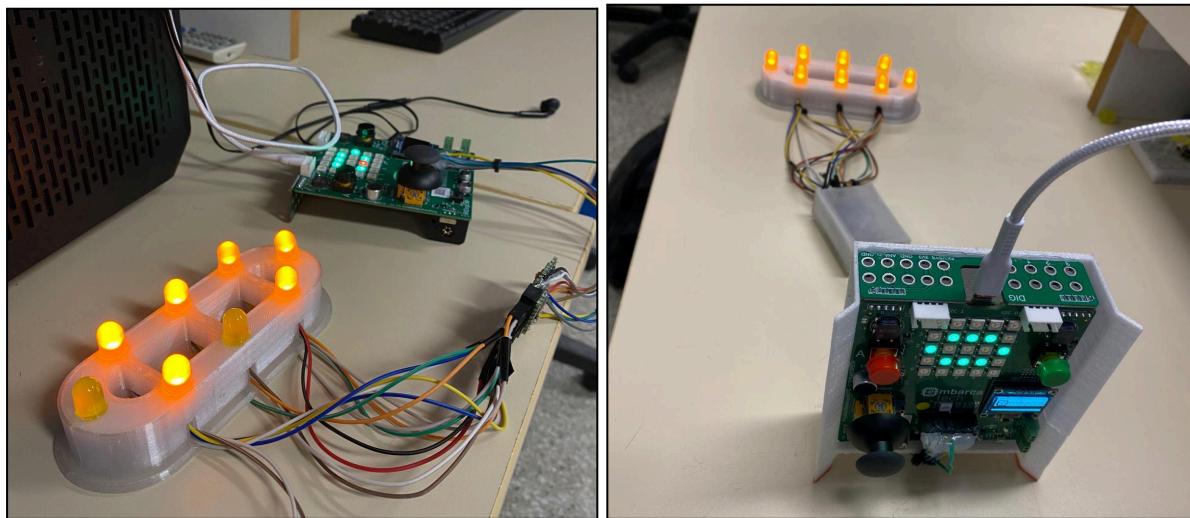
Fonte: Autor, Câmera.

Testadas as ligações e desenvolvido o Firmware básico, o próximo passo foi modelar o protótipo da Linha de Envelhecimento (ANEXO F) e do suporte da Central de Comandos (ANEXO G), trocar os LEDs 3mm por LEDs 5mm, soldar os terminais dos LEDs em jumpers e construir a placa do divisor de tensão. Todos os arquivos 3D (.part) do projeto estarão disponíveis em: https://github.com/ravelsouza/ProjetoFinalEmbarcatech_RavelSouza_Linha_de_Envelhecimento. Segue as imagens de como foram essas etapas:

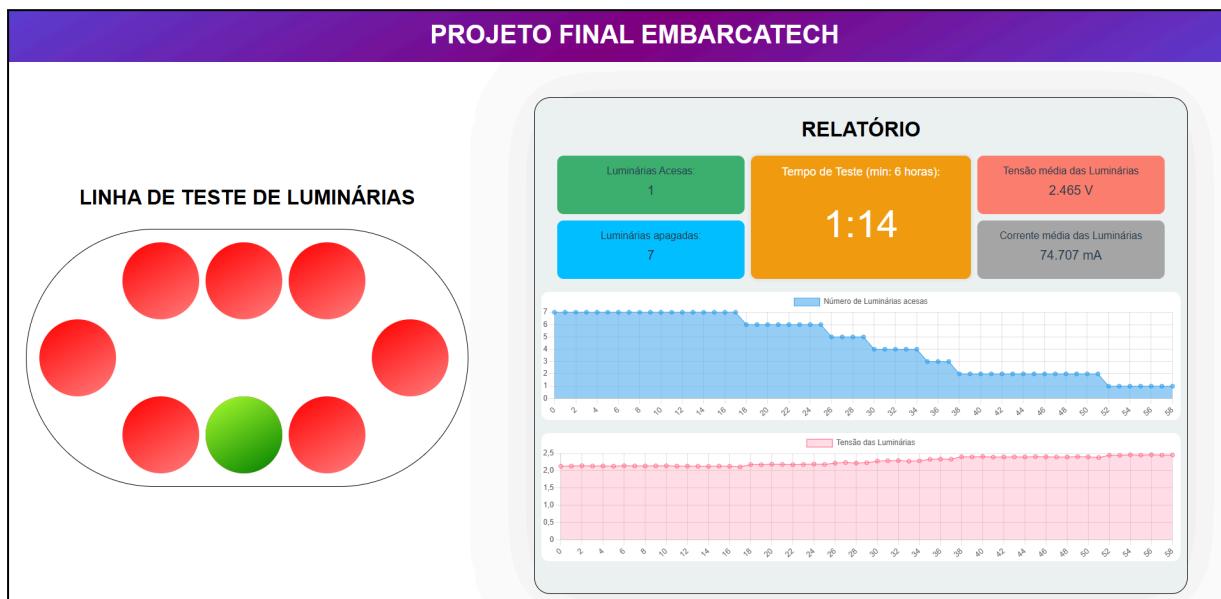
CONSTRUÇÃO DOS MODELOS:



INTERLIGAÇÃO DOS COMPONENTES:



Em sequência, foi feito o upload do Firmware final com todas as funcionalidades (ANEXO A) na placa Raspberry Pi Pico W e desenvolvido um servidor local utilizando Python para receber os dados via comunicação serial e imprimí-los de forma dinâmica. Propondo, assim, a possibilidade de um monitoramento ainda mais remoto do sistema. Segue um Screenshot do funcionamento do servidor Python (para melhor visualização, foi publicado um vídeo do funcionamento do servidor em <https://youtu.be/PXuLB3xBjKk>):



Fonte: Autor, Screenshot.

DISCUSSÃO DOS RESULTADOS

Portanto, diante dos resultados apresentados, fica claro o potencial de executabilidade do projeto nas indústrias de luminárias e displays. Como já discutido na seção de Originalidade, a implementação de sistemas embarcados que buscam melhorar o ambiente de trabalho para operadores tem ganhado cada vez mais

destaque nas fábricas ao redor do mundo. Esses sistemas oferecem um controle mais eficiente e monitoramento em tempo real, o que se traduz em maior segurança e otimização dos processos. Contudo, ainda existe uma resistência considerável à adoção de novas tecnologias no espaço fabril, frequentemente relacionada ao medo de mudanças e à necessidade de adaptação dos funcionários a novos métodos de trabalho.

Além disso, uma oportunidade significativa para aprimorar ainda mais a aplicação do Sistema Embocado no ambiente industrial é a implementação de uma análise de dados mais precisa para os dados de tensão e corrente obtidos pelo microcontrolador. Através de técnicas avançadas de processamento de sinal, como a filtragem e a média ponderada, é possível melhorar a precisão das medições e detectar variações mínimas que poderiam passar despercebidas com métodos tradicionais. Essas informações podem ser essenciais para a manutenção preditiva dos sistemas de iluminação e displays, permitindo identificar padrões de consumo e até antecipar falhas nos componentes elétricos antes que se tornem um problema crítico.

A coleta e análise em tempo real desses dados também podem ser complementadas com algoritmos de aprendizado de máquina, permitindo não apenas o monitoramento contínuo da performance dos sistemas, mas também a adaptação inteligente dos parâmetros de operação com base nas condições específicas de cada ambiente de trabalho. Isso poderia resultar em um sistema ainda mais eficiente e sustentável, reduzindo desperdícios de energia e prolongando a vida útil dos equipamentos. Assim, além de melhorar a segurança e a produtividade, esse aprimoramento possibilita uma gestão mais eficaz dos recursos elétricos, o que pode gerar uma vantagem competitiva significativa para as indústrias que adotarem essa tecnologia.

ANEXOS

ANEXO A - Firmware completo - arquivo main.c

```
#include "functions.h"

// Definições de Variáveis
const float fator_de_conversao = 3.3f / (1 << 12);
const uint32_t pinos[10] = {4, 8, 9, 16, 17, 18, 19, 20, 21, 22};
const uint32_t pinos_config[10] = {0,0,0,0,0,0,0,0,1,1};
const uint32_t todos_os_leds[8] = {6, 7, 8, 10, 18, 17, 16, 14};
float tensao;
float corrente;
uint8_t minutos = 0;
uint32_t cor_na_matriz[8] = {};
uint32_t estado_dos_leds[8] = {};
uint32_t luminarias_acesas = 0;
uint32_t tempo_inicial;
uint32_t tempo_atual;
uint32_t tempo_de_teste;
uint16_t leitura;
uint sm;
uint offset;
PIO pio;
bool success;

int main() {

    stdio_init_all();
    tempo_inicial = time_us_32();
    inicializa_oled();

    for(int cont = 0; cont < 10; cont++) // Inicializa os pinos
    {
        gpio_init(pinos[cont]);
        gpio_set_dir(pinos[cont], pinos_config[cont]);
        sleep_ms(1);
    }

    for(int i = 5; i < 7; i++) // Inicializa os botões
    {
        gpio_init(i);
        gpio_set_dir(i, GPIO_IN);
        gpio_pull_up(i);
    }
}
```

```

}

adc_init();
adc_gpio_init(PINO_ADC);
adc_select_input(2);

inicializa_buzzer(BUZZER_A);
inicializa_buzzer(BUZZER_B);

// Inicializa o PIO
success =
pio_claim_free_sm_and_add_program_for_gpio_range(&ws2812_program, &pio,
&sm, &offset, WS2812_PIN, 1, true);
hard_assert(success);
ws2812_program_init(pio, sm, offset, WS2812_PIN, 800000, IS_RGBW);

uint32_t cores[4] = {
    urgb_u32(20, 0, 0), // Vermelho
    urgb_u32(0, 0, 20), // Azul
    urgb_u32(0, 20, 0), // Verde
    urgb_u32(20, 20, 20) // Branco
};

while (1) {
    if(!gpio_get(BUTTON_B))
    {
        inicializa_bloco_expositor(pio, sm);
        while(1)
        {
            luminarias_acesas = 0;
            for(int cont = 0; cont < 8; cont++)
            {
                if(gpio_get(pin[cont])) // Se houver tensão no pino, o led correspondente na
                Matriz deve acender na cor verde.
                {
                    estado_dos_leds[cont] = 1;
                    luminarias_acesas += 1; // Adiciona-se 1 para cada luminária acesa na
esteira.
                    cor_na_matriz[cont] = urgb_u32(0,5,0);
                }
                else // Se não houver tensão no pino, o led correspondente deve acender na
cor vermelha.
                {
                    estado_dos_leds[cont] = 0;

```

```

    cor_na_matriz[cont] = urgb_u32(5,0,0);
}
}

// Recebe uma lista a posição dos leds na Matriz e outra com as suas
respectivas cores para setá-las na Matriz.
exibir_na_matriz(pio, sm, 25, todos_os_leds, 7, cor_na_matriz);

leitura = adc_read();
tensao = leitura * fator_de_conversao;
corrente = (tensao / 33)* 1000; // Usa o valor da tensão para calcular a
corrente. OBS: Resistor --> 33 Ohms
tempo_atual = time_us_32();
tempo_de_teste = (tempo_atual - tempo_inicial)/1000000;

// Imprime os dados
for(int controle = 0; controle < 8; controle++)
{
printf("%d ", estado_dos_leds[controle]);
}

printf("%d %d ", luminarias_acesas, 8 - luminarias_acesas);
printf("%d:%u ", minutos, tempo_de_teste);
printf("%0.3f %0.3f\n", tensao, corrente);

if(tempo_de_teste >= 59) // Contagem de minutos
{
    minutos += 1;
    tempo_inicial = time_us_32();
}

while(minutos == 6) // Checagem do tempo Limite (6 minutos)
{
    buzzer_alerta();
    if(!gpio_get(BUTTON_A))
    {
        minutos = 0;
        break;
    }
}

sleep_ms(1000);
}

```

```

        }
    }
}
```

ANEXO B - Funções do Firmware - arquivo functions.h

```

#include "includes.h"
#include "constantes.h"

struct render_area {
    uint8_t start_col;
    uint8_t end_col;
    uint8_t start_page;
    uint8_t end_page;

    int buflen;
};

int inicial[8][8] = {
    {6},
    {6, 7},
    {6, 7, 8},
    {6, 7, 8, 10},
    {6, 7, 8, 10, 18},
    {6, 7, 8, 10, 18, 17},
    {6, 7, 8, 10, 18, 17, 16},
    {6, 7, 8, 10, 18, 17, 16, 14}
};

int final[7][7] = {
    {14},
    {16, 14},
    {17, 16, 14},
    {18, 17, 16, 14},
    {10, 18, 17, 16, 14},
    {8, 10, 18, 17, 16, 14},
    {7, 8, 10, 18, 17, 16, 14}
};

void calc_render_area_buflen(struct render_area *area) {
    // Calcula o tamanho do buffer para uma área de renderização
    area->buflen = (area->end_col - area->start_col + 1) * (area->end_page -
    area->start_page + 1);
}
```

```

void SSD1306_send_cmd(uint8_t cmd) {
    // O processo de escrita I2C espera um byte de controle seguido pelos dados
    // Esse "dado" pode ser um comando ou dados que seguem um comando
    // Co = 1, D/C = 0 => o driver espera um comando
    uint8_t buf[2] = {0x80, cmd};
    i2c_write_blocking(i2c_default, SSD1306_I2C_ADDR, buf, 2, false);
}

void SSD1306_send_cmd_list(uint8_t *buf, int num) {
    for (int i=0; i<num; i++)
        SSD1306_send_cmd(buf[i]);
}

void SSD1306_send_buf(uint8_t buf[], int buflen) {
    // Em modo de endereçamento horizontal, o ponteiro de endereço da coluna
    // auto-incrementa
    // e então retorna para a próxima página, então podemos enviar o buffer inteiro
    // de uma vez!

    // Copia nosso buffer de frame para um novo buffer porque precisamos adicionar
    // o byte de controle
    // no início

    uint8_t *temp_buf = malloc(buflen + 1);

    temp_buf[0] = 0x40;
    memcpy(temp_buf+1, buf, buflen);

    i2c_write_blocking(i2c_default, SSD1306_I2C_ADDR, temp_buf, buflen + 1,
    false);

    free(temp_buf);
}

void SSD1306_init() {
    // Alguns desses comandos não são estritamente necessários, pois o processo de
    // reset
    // já define alguns desses valores, mas eles são mostrados aqui para demonstrar
    // como a sequência de inicialização parece
    // Alguns valores de configuração são recomendados pelo fabricante da placa

    uint8_t cmd[] = {
        SSD1306_SET_DISP,           // desliga o display

```

```

/* mapeamento de memória */
SSD1306_SET_MEM_MODE,      // define o modo de endereço da memória
0 = horizontal, 1 = vertical, 2 = página
0x00,                      // modo de endereçamento horizontal
/* resolução e layout */
SSD1306_SET_DISP_START_LINE, // define a linha de início do display
para 0
SSD1306_SET_SEG_REMAP | 0x01, // define o remapeamento de
segmentos, o endereço da coluna 127 é mapeado para SEG0
SSD1306_SET_MUX_RATIO,     // define a razão de multiplexação
SSD1306_HEIGHT - 1,         // Altura do display - 1
SSD1306_SET_COM_OUT_DIR | 0x08, // define a direção de escaneamento
COM (comum). Escanear de baixo para cima, COM[N-1] para COM0
SSD1306_SET_DISP_OFFSET,    // define o offset do display
0x00,                      // sem offset
SSD1306_SET_COM_PIN_CFG,    // define a configuração dos pinos COM
(comum). Número mágico específico da placa.
                                // 0x02 Funciona para 128x32, 0x12 Funciona
possivelmente para 128x64. Outras opções 0x22, 0x32
#if ((SSD1306_WIDTH == 128) && (SSD1306_HEIGHT == 32))
  0x02,
#elif ((SSD1306_WIDTH == 128) && (SSD1306_HEIGHT == 64))
  0x12,
#else
  0x02,
#endif
/* esquema de temporização e controle */
SSD1306_SET_DISP_CLK_DIV,   // define a razão de divisão do relógio do
display
0x80,                      // razão de divisão 1, frequência padrão
SSD1306_SET_PRECHARGE,     // define o período de pré-carga
0xF1,                      // Vcc internamente gerado na nossa placa
SSD1306_SET_VCOM_DESEL,    // define o nível de deseleção VCOMH
0x30,                      // 0.83xVcc
/* display */
SSD1306_SET_CONTRAST,      // define o controle de contraste
0xFF,
SSD1306_SET_ENTIRE_ON,     // ativa o display inteiro para seguir o
conteúdo da RAM
SSD1306_SET_NORM_DISP,     // define o display como normal (não
invertido)
SSD1306_SET_CHARGE_PUMP,   // define a bomba de carga
0x14,                      // Vcc internamente gerado na nossa placa

```

```

    SSD1306_SET_SCROLL | 0x00, // desativa o rolo horizontal se estiver
ativado. Isso é necessário, pois as gravações de memória irão corromper o
conteúdo se o rolo estiver ativo
    SSD1306_SET_DISP | 0x01, // liga o display
};

SSD1306_send_cmd_list(cmds, count_of(cmds));
}

void render(uint8_t *buf, struct render_area *area) {
// Atualiza uma parte do display com uma área de renderização
uint8_t cmd[ ] = {
    SSD1306_SET_COL_ADDR,
    area->start_col,
    area->end_col,
    SSD1306_SET_PAGE_ADDR,
    area->start_page,
    area->end_page
};
SSD1306_send_cmd_list(cmds, count_of(cmds));
SSD1306_send_buf(buf, area->buflen);
}

void inicializar_pio(PIO pio, uint sm, uint offset){
    bool success =
pio_claim_free_sm_and_add_program_for_gpio_range(&ws2812_program, &pio,
&sm, &offset, WS2812_PIN, 1, true);
    hard_assert(success);
    ws2812_program_init(pio, sm, offset, WS2812_PIN, 800000, IS_RGBW);
}

void inicializa_buzzer(int BUZZER) {
    gpio_set_function(BUZZER, GPIO_FUNC_PWM);
    uint slice_num = pwm_gpio_to_slice_num(BUZZER);

    pwm_config config = pwm_get_default_config();
    pwm_config_set_clkdiv(&config, clock_get_hz(clk_sys) / (FREQUENCIA_BUZZER
* 4096));
    pwm_init(slice_num, &config, 1);

    pwm_set_gpio_level(BUZZER, 0);
}

```

```

void buzzer_alerta()
{
    pwm_set_gpio_level(BUZZER_A, FREQUENCIA_BUZZER);
    pwm_set_gpio_level(BUZZER_B, FREQUENCIA_BUZZER);
    sleep_ms(1000);
    pwm_set_gpio_level(BUZZER_A, 0);
    pwm_set_gpio_level(BUZZER_B, 0);
    sleep_ms(1000);
}

void put_pixel(PIO pio, uint sm, uint32_t pixel_grb) {
    pio_sm_put_blocking(pio, sm, pixel_grb << 8u);
}

uint32_t urgb_u32(uint8_t r, uint8_t g, uint8_t b) {
    return
        ((uint32_t) (r) << 8) |
        ((uint32_t) (g) << 16) |
        (uint32_t) (b);
}

uint32_t urgbw_u32(uint8_t r, uint8_t g, uint8_t b, uint8_t w) {
    return
        ((uint32_t) (r) << 8) |
        ((uint32_t) (g) << 16) |
        ((uint32_t) (w) << 24) |
        (uint32_t) (b);
}

void acender_leds_na_matriz(PIO pio, uint sm, uint len, int leds[], int row, uint32_t cor) {
    // Apaga todos os LEDs
    for (uint i = 0; i < len; ++i) {
        bool led_aceso = false;
        // Verifica se o LED atual deve ser aceso (comparando com a linha do vetor de LEDs)
        for (int j = 0; j <= row; ++j) {
            if (leds[j] == i) {
                put_pixel(pio, sm, cor); // Acende o LED na cor especificada
                led_aceso = true;
                break;
            }
        }
        if (!led_aceso) {

```

```

        put_pixel(pio, sm, urgb_u32(0, 0, 0)); // Apaga os LEDs não escolhidos
    }
}
}

void exibir_na_matriz(PIO pio, uint sm, uint len, int leds[], int row, uint32_t* cor) {

// Apaga todos os LEDs
int cor_led = 0;
for (uint i = 0; i < len; ++i) {
    bool led_aceso = false;
    // Verifica se o LED atual deve ser aceso (comparando com a linha do vetor de
    LEDs)
    for (int j = 0; j <= row; ++j) {
        if (leds[j] == i) {
            put_pixel(pio, sm, cor[cor_led]); // Acende o LED na cor especificada
            cor_led += 1;
            led_aceso = true;
            break;
        }
    }
    if (!led_aceso) {
        put_pixel(pio, sm, urgb_u32(0, 0, 0)); // Apaga os LEDs não escolhidos
    }
}
}

void inicializa_oled(){
    i2c_init(i2c_default, SSD1306_I2C_CLK * 1000);
    gpio_set_function(PICO_DEFAULT_I2C_SDA_PIN, GPIO_FUNC_I2C);
    gpio_set_function(PICO_DEFAULT_I2C_SCL_PIN, GPIO_FUNC_I2C);
    gpio_pull_up(PICO_DEFAULT_I2C_SDA_PIN);
    gpio_pull_up(PICO_DEFAULT_I2C_SCL_PIN);

    SSD1306_init();

    struct render_area frame_area = {
        start_col: 0,
        end_col : SSD1306_WIDTH - 1,
        start_page : 0,
        end_page : SSD1306_NUM_PAGES - 1
    };
}
```

```

calc_render_area_buflen(&frame_area);

// Zera o Display
static uint8_t buf[SSD1306_BUF_LEN];
memset(buf, 0, SSD1306_BUF_LEN);
render(buf, &frame_area);

for (int i=0; i<sizeof(embarcatech); i+=24) {
    memcpy(buf+i, &embarcatech[i], 24);
    render(buf, &frame_area);
    sleep_ms(1);
}
}

void inicializa_bloco_expositor(PIO pio, uint sm)
{
    int cor_index = 0;
    uint32_t cores[4] = {
        urgb_u32(20, 0, 0), // Vermelho (R, G, B)
        urgb_u32(0, 0, 20), // Azul (R, G, B)
        urgb_u32(0, 20, 0), // Verde (R, G, B)
        urgb_u32(20, 20, 20) // Branco (R, G, B)
    };
    for (int ciclo = 0; ciclo < 4; ++ciclo) {
        // Loop para a matriz inicial
        for (int row = 0; row < 8; ++row) {
            acender_leds_na_matriz(pio, sm, 25, inicial[row], row, cores[cor_index]);
            sleep_ms(200); // Espera 200ms entre as linhas
        }

        // Loop para a matriz final
        for (int row = 6; row >= 0; --row) {
            acender_leds_na_matriz(pio, sm, 25, final[row], row, cores[cor_index]);
            sleep_ms(200); // Espera 200ms entre as linhas
        }

        // Muda a cor para o próximo ciclo
        cor_index = (cor_index + 1) % 4;
    }
}

```

```

acender_leds_na_matriz(pio, sm, 25, inicial[7], 7, cores[3]);
sleep_ms(300);
for(int i = 0; i < 2; i++)
{
    acender_leds_na_matriz(pio, sm, 25, inicial[7], 7, urgb_u32(0,0,0));
    sleep_ms(300);
    acender_leds_na_matriz(pio, sm, 25, inicial[7], 7, cores[3]);
    sleep_ms(300);
}
}

```

ANEXO C - Firmware completo - arquivo includes.h

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pico/stlolib.h"
#include "ws2812.pio.h"
#include <ctype.h>
#include "pico/binary_info.h"
#include "pico/time.h"
#include "hardware/i2c.h"
#include "hardware/gpio.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"
#include "hardware/pio.h"

```

ANEXO D - Firmware completo - arquivo constantes.h

```

#define SSD1306_WIDTH          128
#define SSD1306_HEIGHT         64
#define SSD1306_I2C_CLK        400
#define SSD1306_I2C_ADDR       _u(0x3C)
#define SSD1306_SET_MEM_MODE   _u(0x20)
#define SSD1306_SET_COL_ADDR   _u(0x21)
#define SSD1306_SET_PAGE_ADDR  _u(0x22)
#define SSD1306_SET_HORIZ_SCROLL _u(0x26)
#define SSD1306_SET_SCROLL     _u(0x2E)

#define SSD1306_SET_DISP_START_LINE _u(0x40)

#define SSD1306_SET_CONTRAST   _u(0x81)
#define SSD1306_SET_CHARGE_PUMP _u(0x8D)

#define SSD1306_SET_SEG_REMAP  _u(0xA0)

```

```

#define SSD1306_SET_ENTIRE_ON      _u(0xA4)
#define SSD1306_SET_ALL_ON        _u(0xA5)
#define SSD1306_SET_NORM_DISP    _u(0xA6)
#define SSD1306_SET_INV_DISP     _u(0xA7)
#define SSD1306_SET_MUX_RATIO   _u(0xA8)
#define SSD1306_SET_DISP         _u(0xAE)
#define SSD1306_SET_COM_OUT_DIR _u(0xC0)
#define SSD1306_SET_COM_OUT_DIR_FLIP _u(0xC0)

#define SSD1306_SET_DISP_OFFSET   _u(0xD3)
#define SSD1306_SET_DISP_CLK_DIV _u(0xD5)
#define SSD1306_SET_PRECHARGE    _u(0xD9)
#define SSD1306_SET_COM_PIN_CFG  _u(0xDA)
#define SSD1306_SET_VCOM_DESEL   _u(0xDB)

#define SSD1306_PAGE_HEIGHT      _u(8)
#define SSD1306_NUM_PAGES (SSD1306_HEIGHT / SSD1306_PAGE_HEIGHT)
#define SSD1306_BUF_LEN          (SSD1306_NUM_PAGES * SSD1306_WIDTH)

#define SSD1306_WRITE_MODE       _u(0xFE)
#define SSD1306_READ_MODE        _u(0xFF)

#define FREQUENCIA_BUZZER 5500

#define BUTTON_A 5
#define BUTTON_B 6

#define PINO_ADC 28

#define SENSOR_PIN 22
#define SHAKE_THRESHOLD 4
#define TIME_WINDOW_MS 5000

#define IS_RGBW false
#define NUM_PIXELS 25

#ifndef PICO_DEFAULT_WS2812_PIN
#define WS2812_PIN PICO_DEFAULT_WS2812_PIN
#else
#define WS2812_PIN 7
#endif

#if WS2812_PIN >= NUM_BANK0_GPIOS
#error Attempting to use a pin>=32 on a platform that does not support it

```

```
#endif

#define BUZZER_A 21      // Buzzer
#define BUZZER_B 10      // Buzzer

static uint8_t embarkatech[] = {0xff, 0xff, 0xff, 0xef, 0xff, 0xff, 0xf7, 0xff, 0x3f, 0x3f,
0xbff, 0xf7, 0xbff, 0x3f, 0x3f, 0x3f, 0xf7, 0xbff, 0x3f, 0x37, 0xff,
0xff, 0x1f, 0x3b, 0xff, 0xef, 0x3f, 0x3f, 0x37, 0xff, 0xfb, 0xff, 0xdf, 0xff, 0x7b, 0xff, 0xef,
0xff, 0xbb, 0xff, 0xdf, 0xfb, 0xff, 0xdf, 0xfb, 0x7f,
0xdf, 0xfb, 0xef, 0xff, 0xbb, 0xff, 0xef, 0x7b, 0xff, 0xdb, 0xff, 0xef, 0x7b, 0xff, 0xdb,
0xff, 0x6f, 0xfd, 0xf7, 0xbff, 0xfb, 0x7f, 0xed, 0xbf,
0xf7, 0xff, 0xdd, 0x77, 0xff, 0xdf, 0xf5, 0x7f, 0xdf, 0xf7, 0x7d, 0xdf, 0xfb, 0xef, 0xbd,
0xff, 0xd7, 0x7f, 0xfd, 0xd7, 0x7f, 0xfd, 0xd7, 0x7f,
0xfd, 0xd7, 0x7f, 0xfd, 0xd7, 0x7f, 0xdf, 0xd7, 0x7f, 0xdf, 0xd7, 0x7f,
0xdf, 0xd7, 0x7f, 0xfd, 0xd7, 0x7f, 0xdf, 0xd7, 0x7f, 0xdf, 0xd7, 0x7f,
0xdf, 0xd7, 0x7f, 0xfd, 0xd7, 0x7f, 0xdf, 0xd7, 0x7f, 0xdf, 0xd7, 0x7f,
0xc2, 0xc0, 0xc1, 0xc3, 0xc2, 0xc0, 0xc0, 0xc3, 0xc3, 0xc0, 0xc0, 0xc3, 0xc3, 0xc0,
0xc0, 0x03, 0x03, 0x02, 0x03, 0x82, 0xc0, 0xc0, 0xc3,
0xc2, 0xc0, 0xc1, 0xc3, 0xc2, 0xc0, 0xc0, 0xc3, 0xc3, 0xc0, 0xc0, 0xc3, 0xc3, 0xc0,
0xc0, 0x03, 0x03, 0x02, 0x03, 0x3f, 0x3f, 0x2f, 0x3d, 0xff,
0x6f, 0xfd, 0xde, 0xf7, 0xff, 0xbd, 0xff, 0xef, 0xfb, 0xff, 0xfe, 0xef, 0x7b, 0xff,
0xef, 0xbd, 0xff, 0xde, 0xf7, 0x7f, 0xef, 0xfd, 0xf7,
0x0f, 0x07, 0x0d, 0x4f, 0xfb, 0x7f, 0xed, 0xff, 0xb7, 0xfe, 0xdf, 0xfb, 0x6f, 0xfd, 0xbf,
0xf7, 0xfd, 0xaf, 0xff, 0x75, 0xff, 0xde, 0xfb, 0xbf,
0xed, 0xff, 0xb7, 0xfd, 0xdf, 0x77, 0xfd, 0xdf, 0x77, 0xfd, 0x77, 0xfd, 0x5f, 0xf7,
0xfd, 0x5f, 0xf7, 0x7d, 0xdf, 0xf7, 0x7d, 0xdf, 0x77, 0xfd, 0xdf, 0x77, 0xfd, 0x77, 0xfd,
0xf7, 0xbd, 0xef, 0xff, 0x5a, 0xff, 0x77, 0x33, 0x33, 0x33,
0x00, 0x00, 0x00, 0x00, 0xaa, 0x3f, 0x1f, 0x1f, 0x3f, 0x3b, 0x0f, 0x03, 0x03, 0x01, 0x31,
0x39, 0x31, 0x39, 0x31, 0x01, 0x03, 0x0f, 0xfe, 0xff, 0x00,
0x00, 0x00, 0x33, 0x33, 0x53, 0x31, 0xff, 0xff, 0xad, 0x01, 0x00, 0x01, 0xf3,
0x1f, 0x78, 0xf1, 0xe0, 0x01, 0x01, 0x07, 0xe3, 0xf1, 0xf9,
0xb0, 0xf1, 0x01, 0x01, 0x03, 0xff, 0xde, 0xff, 0x41, 0x00, 0x00, 0x00, 0xf3, 0xf1,
0xb8, 0xf1, 0xe0, 0x01, 0x03, 0x05, 0x5f, 0xef, 0xfb, 0x6e,
0xe3, 0x61, 0x61, 0x31, 0x38, 0x31, 0x30, 0x51, 0x01, 0x02, 0x07, 0xff, 0xfd, 0xdf,
0x01, 0x01, 0x01, 0xc1, 0xf1, 0x71, 0xf0, 0xd1, 0xff, 0x3d,
0x07, 0x03, 0x01, 0xe1, 0xf1, 0xb8, 0xf1, 0x71, 0xe1, 0xc1, 0xc3, 0xcf, 0x7d, 0xff,
0xe3, 0x63, 0x61, 0x31, 0x30, 0x59, 0x30, 0x39, 0x40, 0x03,
0x01, 0x0f, 0xff, 0xfb, 0x77, 0x63, 0x32, 0x67, 0x00, 0x00, 0x00, 0x00, 0x55, 0xff,
0xfe, 0xfe, 0x7e, 0xef, 0xfc, 0xf0, 0xe0, 0xe3, 0xc7, 0xc6,
0xc7, 0xe7, 0x62, 0xe3, 0xf7, 0x7d, 0xff, 0x00, 0x00, 0x00, 0x00, 0x63, 0x36,
0x63, 0x36, 0xf7, 0xff, 0xda, 0x80, 0x80, 0x80, 0xff, 0xff, 0xef,
0xfd, 0xbf, 0x80, 0x80, 0x80, 0xff, 0xdd, 0xf7, 0xff, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x8d,
```

0x87, 0xc0, 0xc0, 0x60, 0xfa, 0xff, 0xb7, 0xe0, 0xc0, 0x80, 0x8c, 0x8e, 0x96,
 0x8f, 0xca, 0xc6, 0x80, 0x80, 0x80, 0xff, 0xdd, 0xff, 0x80, 0x80,
 0x80, 0xff, 0xd5, 0x7f, 0xfb, 0xae, 0xff, 0xdd, 0xf0, 0xc0, 0xc0, 0x87, 0x8f, 0x8d,
 0x8f, 0x8b, 0x87, 0xc2, 0xe3, 0xb1, 0xff, 0xdf, 0xe0,
 0x80, 0x80, 0x86, 0x9e, 0x86, 0x8e, 0xcf, 0xc0, 0x80, 0x80, 0x88, 0xff, 0xef,
 0xff, 0xfe, 0xfe, 0x7e, 0xf0, 0xe0, 0xe0, 0xf0, 0x40, 0x01,
 0xa1, 0xe0, 0x61, 0x01, 0x80, 0xe1, 0x60, 0x01, 0x00, 0xe1, 0xe0, 0x01, 0x01,
 0xe1, 0xe1, 0x01, 0x00, 0xe0, 0xe0, 0xf0, 0xfe, 0xde,
 0xfe, 0xee, 0x7f, 0xff, 0x5d, 0x07, 0xff, 0x7f, 0x6d, 0xff, 0xbe, 0xf7, 0xff, 0x7b,
 0xdf, 0xfd, 0xff, 0x5e, 0x77, 0x7f, 0x7e, 0xd7, 0xff,
 0x7d, 0xdf, 0xf7, 0xbf, 0xfd, 0xd7, 0xff, 0xbe, 0xfb, 0xef, 0xbd, 0x7f, 0x77, 0x5e,
 0xfb, 0x7f, 0xef, 0xbb, 0xfe, 0xef, 0xbd, 0xff, 0x41,
 0x0b, 0xfd, 0xff, 0x76, 0x5f, 0xfd, 0x37, 0xff, 0xda, 0xff, 0x6d, 0xff, 0xb6, 0xff,
 0xdd, 0x37, 0xff, 0xed, 0x7f, 0xdb, 0xbe, 0x57, 0xbe,
 0x5f, 0xf5, 0xbf, 0x7d, 0xef, 0x5b, 0x7f, 0xae, 0xfb, 0x1f, 0xf6, 0x3f, 0xed, 0xbf,
 0x75, 0xdf, 0x7f, 0xd5, 0x7f, 0xaf, 0xfa, 0x7f, 0xf7,
 0xdd, 0xff, 0xbb, 0xff, 0xff, 0x77, 0xff, 0xff, 0x7b, 0xff, 0xbe, 0xfe, 0xfe, 0xdf,
 0xff, 0xfe, 0xde, 0xff, 0xef, 0xfe, 0xff, 0x77,
 0xff, 0xfe, 0xee, 0x7f, 0xff, 0xee, 0x7e, 0xff, 0xb7, 0xff, 0xfe, 0xb7, 0xff, 0xfd,
 0xef, 0xff, 0x76, 0xfe, 0x00, 0xfe, 0xfe, 0xb7, 0xfe,
 0xff, 0x6d, 0xff, 0xb7, 0x03, 0xb9, 0xbc, 0x3e, 0xb7, 0x3f, 0x6e, 0xbe, 0x3c,
 0x59, 0x83, 0xff, 0xbd, 0xef, 0xfd, 0xaf, 0x03, 0xf8, 0xfe,
 0x76, 0xde, 0xff, 0xf6, 0xbe, 0xee, 0x79, 0xf3, 0xaf, 0xfe, 0xdb, 0xff, 0x48, 0x01,
 0xfc, 0xfe, 0xb6, 0xff, 0xf6, 0x5e, 0xfe, 0xf8, 0x03,
 0xef, 0xbd, 0xff, 0x6b, 0xff, 0x24, 0xff, 0xbf, 0xeb, 0x7f, 0xfa, 0xcb, 0x7f, 0xe2,
 0xdf, 0xe5, 0x57, 0xfe, 0x57, 0xee, 0x6b, 0xd6, 0xff,
 0xc2, 0x5f, 0xf3, 0xae, 0xf7, 0x4e, 0xf3, 0xde, 0xd7, 0x7a, 0xe7, 0xcb, 0xff,
 0xbe, 0xeb, 0xff, 0xff, 0xff, 0x77, 0xff, 0xff, 0xdf,
 0xff, 0xf7, 0xff, 0xff, 0xdf, 0xfd, 0xef, 0xff, 0xfd, 0x77, 0xff, 0xfd, 0x7f, 0x7f,
 0xbff, 0xff, 0xf7, 0xff, 0xdf, 0xff, 0xfb, 0xdf,
 0xff, 0xed, 0xff, 0x7f, 0xf6, 0xbf, 0xff, 0xf7, 0xe0, 0xcf, 0x9f, 0xbd, 0xcf, 0x7f,
 0xfb, 0xff, 0xdd, 0xf0, 0x67, 0xdf, 0x9b, 0xbf, 0xaf,
 0xbf, 0x9b, 0xdf, 0xc7, 0xf7, 0xbe, 0xf7, 0xff, 0xda, 0x7f, 0xf0, 0xc7, 0xdb, 0x9f,
 0xbff, 0xaa, 0x9f, 0xdf, 0xd5, 0xaf, 0xe3, 0x7f, 0xfb,
 0xde, 0xff, 0xa0, 0xc5, 0xff, 0xbb, 0xfe, 0xd7, 0xff, 0xbd, 0xf7, 0xff, 0x80, 0xee,
 0xbb, 0xff, 0xb7, 0xfd, 0x4b, 0xfd, 0xbf, 0xeb, 0xff,
 0x8c, 0x79, 0xef, 0x9a, 0x7c, 0xee, 0x9b, 0x2f, 0xdb, 0xb8, 0x5f, 0xbd, 0x48,
 0xff, 0x54, 0xdf, 0xb8, 0x5b, 0xfc, 0x0f, 0xfa, 0xac, 0x5f,
 0xb8, 0xcf, 0x3c, 0xda, 0xff, 0xf7, 0xff, 0xff, 0x77, 0xff, 0xfb, 0xff, 0x7d, 0xff,
 0xfe, 0xdf, 0xff, 0xff, 0xdd, 0xff, 0xfe, 0xef,
 0xff, 0x7b, 0xff, 0xbf, 0xfb, 0xff, 0xfd, 0xfe, 0xff, 0x77, 0xff, 0xde, 0xff,
 0x7b, 0xff, 0xef, 0x7f, 0xfb, 0xdf, 0xff, 0xf6, 0xbf,
 0xff, 0xed, 0x7f, 0xff, 0xdb, 0xff, 0xfe, 0x7f, 0xfd, 0x7f,
 0xbff, 0xfb, 0xde, 0xff, 0x7b, 0xef, 0xbe, 0xff, 0xeb,

```

    0x7f, 0xf7, 0xbe, 0xfb, 0x7f, 0xed, 0xbff, 0xf7, 0xfe, 0xdb, 0xff, 0x6f, 0xfb, 0xbe,
    0xef, 0xfb, 0xfe, 0xaf, 0xfb, 0x7e, 0xef, 0xbb, 0xfe,
    0xef, 0xbb, 0xfe, 0x6f, 0xfd, 0xdf, 0x75, 0xff, 0xdf, 0xf5, 0xbf, 0xff, 0xea, 0xbf,
    0xff, 0xd5, 0x7f, 0xff, 0xd5, 0x7f, 0xff, 0xd5, 0x7f,
    0xfb, 0xdf, 0x7b, 0xef, 0xfd, 0x5f, 0xfd, 0xb7, 0xff, 0xed, 0xbf, 0xfb, 0xef, 0xbd,
    0xff, 0xeb, 0xbf, 0xf6, 0xff};

```

ANEXO E - Arquivo CMakeLists.txt

cmake_minimum_required(VERSION 3.13)

```

set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)

# Initialise pico_sdk from installed location
# (note this can come from environment, CMake cache etc)

# == DO NOT EDIT THE FOLLOWING LINES for the Raspberry Pi Pico VS Code
Extension to work ==
if(WIN32)
    set(USERHOME ${ENV{USERPROFILE}})
else()
    set(USERHOME ${ENV{HOME}})
endif()
set(sdkVersion 2.1.0)
set(toolchainVersion 13_3_Rel1)
set(picotoolVersion 2.1.0)
set(picoVscode ${USERHOME}/.pico-sdk/cmake/pico-vscode.cmake)
if (EXISTS ${picoVscode})
    include(${picoVscode})
endif()
#
=====
=====
set(PICO_BOARD pico_w CACHE STRING "Board type")

# Pull in Raspberry Pi Pico SDK (must be before project)
include(pico_sdk_import.cmake)

project(pio_ws2812 C CXX ASM)

# Initialise the Raspberry Pi Pico SDK
pico_sdk_init()

```

```

# Add executable. Default name is the project name, version 0.1

add_executable(pio_ws2812)

file(MAKE_DIRECTORY ${CMAKE_CURRENT_LIST_DIR}/generated)

# generate the header file into the source tree as it is included in the RP2040
datasheet
pico_generate_pio_header(pio_ws2812
${CMAKE_CURRENT_LIST_DIR}/ws2812.pio                         OUTPUT_DIR
${CMAKE_CURRENT_LIST_DIR}/generated)

target_sources(pio_ws2812 PRIVATE main.c)

target_link_libraries(pio_ws2812      hardware_pwm      hardware_i2c      pico_stdlib
hardware_pio hardware_adc)
pico_add_extra_outputs(pio_ws2812)
pico_enable_stdio_uart(pio_ws2812 1)
pico_enable_stdio_usb(pio_ws2812 1)
# add url via pico_set_program_url

add_executable(pio_ws2812_parallel)

pico_generate_pio_header(pio_ws2812_parallel
${CMAKE_CURRENT_LIST_DIR}/ws2812.pio                         OUTPUT_DIR
${CMAKE_CURRENT_LIST_DIR}/generated)

target_sources(pio_ws2812_parallel PRIVATE ws2812_parallel.c)

target_compile_definitions(pio_ws2812_parallel PRIVATE
PIN_DBG1=3)

target_link_libraries(pio_ws2812_parallel pico_stdlib hardware_pio hardware_dma )
pico_add_extra_outputs(pio_ws2812_parallel)

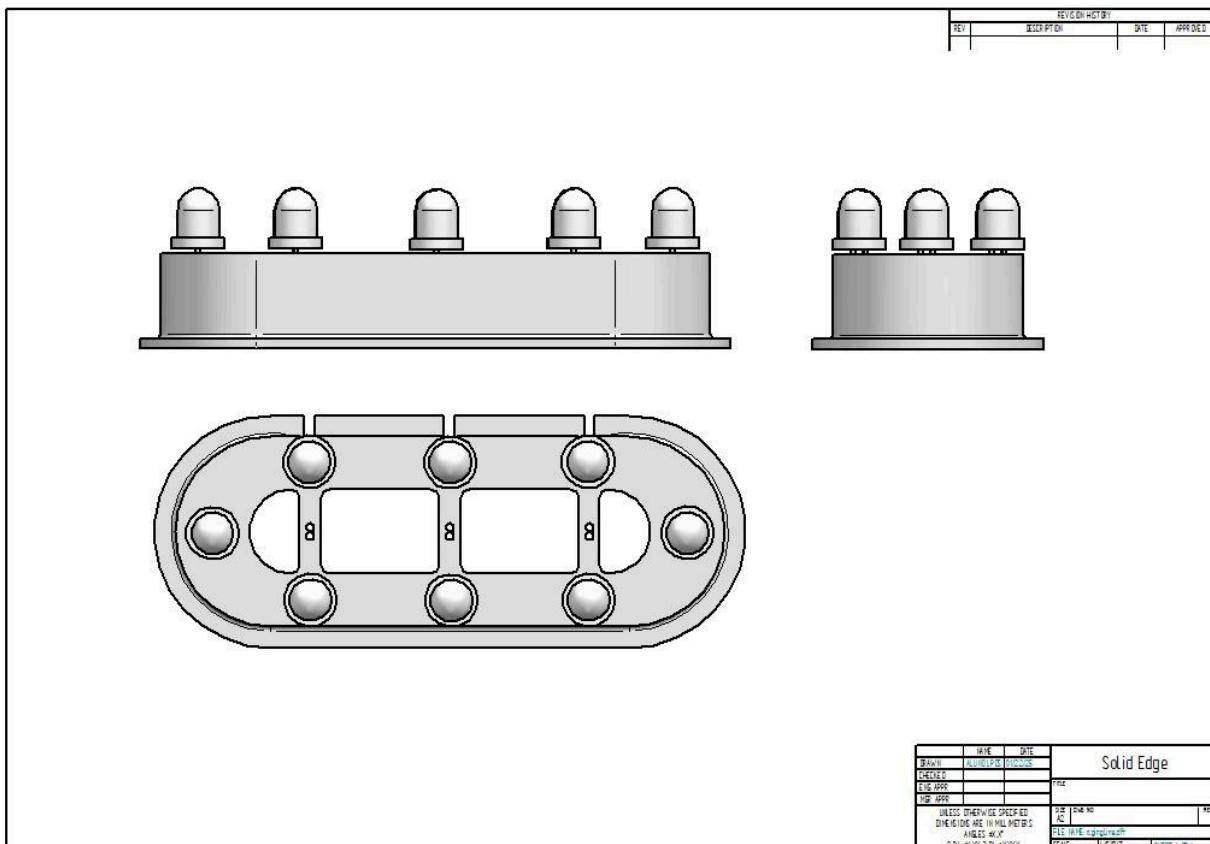
# add url via pico_set_program_url

# Additionally generate python and hex pioasm outputs for inclusion in the RP2040
datasheet
add_custom_target(pio_ws2812_datasheet                         DEPENDS
${CMAKE_CURRENT_LIST_DIR}/generated/ws2812.py)
add_custom_command(OUTPUT
${CMAKE_CURRENT_LIST_DIR}/generated/ws2812.py
DEPENDS ${CMAKE_CURRENT_LIST_DIR}/ws2812.pio

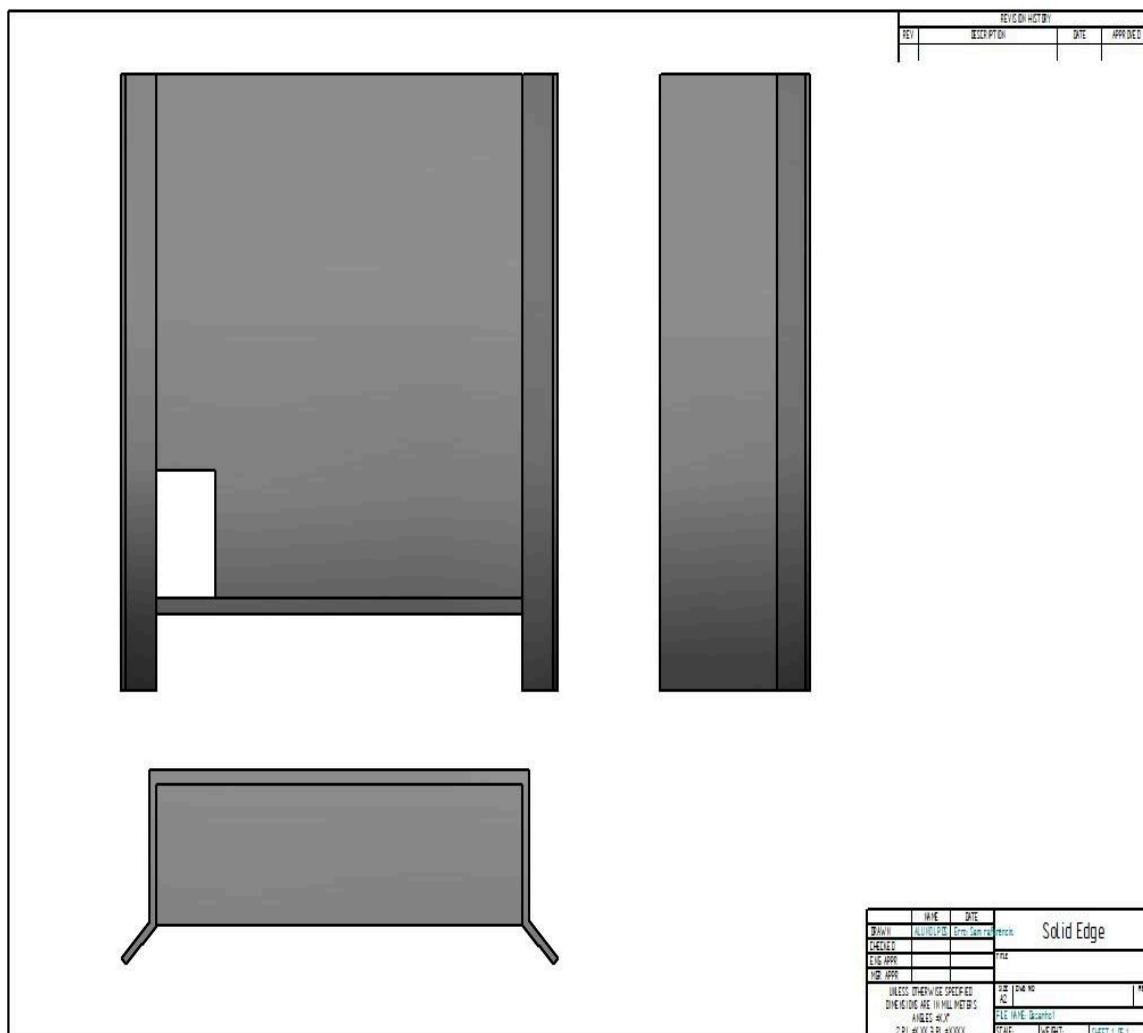
```

```
COMMAND pioasm -o python ${CMAKE_CURRENT_LIST_DIR}/ws2812.pio
${CMAKE_CURRENT_LIST_DIR}/generated/ws2812.py
VERBATIM)
add_dependencies(pio_ws2812 pio_ws2812_datasheet)
```

ANEXO F - Protótipo Esteira



Fonte: Autor, SolidEdge 2024.

ANEXO G - Protótipo do Suporte um Quadro de Comandos

REFERÊNCIAS

Software Wokwi: [Https://wokwi.com](https://wokwi.com)

Software Tinkercad: <https://www.tinkercad.com>

Software EasyEda: <https://easyeda.com>

Documentação da placa BitDogLab: <https://github.com/BitDogLab/BitDogLab>

Imagen: [Http://www.gdduoli.com/e_products_show/?id=52](http://www.gdduoli.com/e_products_show/?id=52)

AUTOMAÇÃO NOS PROCESSOS INDUSTRIAS: PROCESSO DE IMPLEMENTAÇÃO E O PAPEL DO GESTOR DE TECNOLOGIA DA INFORMAÇÃO, Eric Sampaio De Oliveira.

Link: <https://prospectus.fatecitapira.edu.br/index.php/pst/article/view/220>

AUTOMATIZAÇÃO E INTEGRAÇÃO DE UMA LINHA DE PRODUÇÃO INDUSTRIAL, Marco Miguel Marques Teixeira.

Link: <https://repositorio-aberto.up.pt/bitstream/10216/66949/2/27428.pdf>

CUGNASCA, C. E. Projetos de Sistema Embutidos. Departamento de Engenharia de Computação e Sistemas Digitais. Escola Politécnica da USP, 02/2018.

Link:

<https://integra.univesp.br/courses/2710/pages/texto-base-projetos-de-sistemas-embutidos-%7C-carlos-eduardo-cugnasca>

<https://github.com/makerportal/rpi-pico-ws2812>