PROJETO FINAL: Sistema Embarcado de Detecção de Incêndio e Alarme Sonoro

1. Apresentação do Projeto

O projeto consiste em **um sistema embarcado de detecção de incêndio e alarme sonoro**, desenvolvido para simular a detecção de fumaça e acionar um alarme de emergência. O sistema utiliza um Microcontrolador, no caso, o **Raspberry Pi Pico**, para controlar um **LED vermelho** e um **Buzzer**, que representam, respectivamente, um indicador visual de emergência e um alarme sonoro. O sistema é acionado por meio de dois botões: um para simular a detecção de fumaça e outro para ativar o alarme de incêndio.

1.1 Título do Projeto

Sistema Embarcado de Detecção de Incêndio e Alarme Sonoro

1.2 Objetivos do Projeto

Objetivo Geral:

Desenvolver um sistema embarcado que simule a detecção de incêndio e acione um alarme sonoro e visual para alertar sobre uma situação de emergência.

Objetivos Específicos:

Implementar um sistema que detecte a "presença de fumaça" (simulada por um botão) e acione um LED vermelho como indicador visual.

Criar um alarme sonoro utilizando um buzzer, que toque uma sirene de emergência quando o sistema for ativado.

Garantir que o sistema seja de baixo custo e de fácil reprodução, utilizando componentes comuns e acessíveis.

Implementar um mecanismo de debouncing para garantir a confiabilidade na leitura dos botões.

1.3 Descrição do Funcionamento

Detecção de Fumaça (Botão A):

Quando o botão A é pressionado, o sistema simula a detecção de fumaça. Um LED vermelho é acionado para indicar a situação de emergência, e uma mensagem é exibida no console (via interface serial) informando a detecção de fumaça.

Ativação do Alarme de Incêndio (Botão B):

Quando o botão B é pressionado, o sistema ativa o alarme de incêndio. O buzzer toca uma sirene de emergência (uma sequência de notas musicais), e o LED vermelho pisca em sincronia com o som. O alarme é repetido por 10 ciclos, garantindo que o alerta seja percebido.

Debouncing dos Botões:

Para evitar leituras falsas devido a ruídos mecânicos, o sistema implementa um mecanismo de debouncing para os botões A e B, garantindo que apenas pressões válidas sejam consideradas.

1.4 - Justificativa

A detecção de incêndios é uma necessidade crítica em diversos ambientes, como residências, indústrias e espaços públicos. Sistemas embarcados que integram sensores, indicadores visuais e alarmes sonoros são essenciais para garantir a segurança das pessoas e a proteção de patrimônios. Este projeto

demonstra a aplicação prática de sistemas embarcados na criação de soluções de baixo custo e alta eficiência para problemas reais.

Além disso, o projeto utiliza conceitos fundamentais de sistemas embarcados, como controle de GPIO, manipulação de sinais sonoros e técnicas de debouncing, consolidando o aprendizado teórico em uma aplicação prática.

1.5 - Originalidade

Embora existam diversos projetos de sistemas de alarme de incêndio disponíveis, este projeto se destaca por:

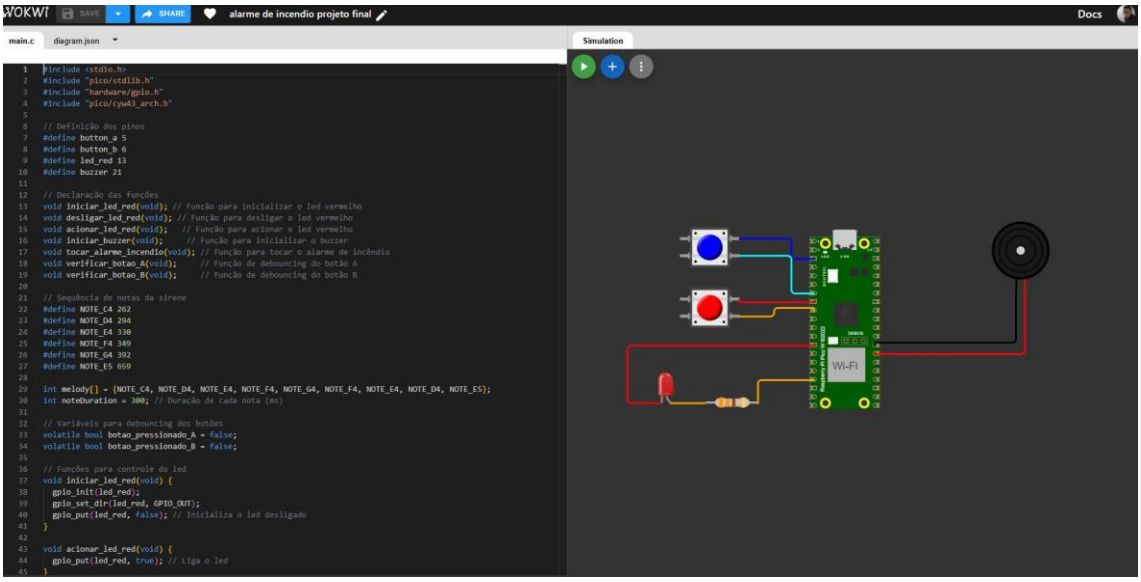
- **Simplicidade e Baixo Custo: Utiliza componentes básicos e acessíveis, como LEDs, buzzers e botões, sem a necessidade de sensores complexos.**

- **Integração de Funcionalidades: Combina indicadores visuais e sonoros em um único sistema, simulando um cenário real de emergência.**

- **Mecanismo de Debouncing: Implementa uma técnica de debouncing para garantir a confiabilidade do sistema, algo que nem todos os projetos similares consideram.**

- **Uso de Microcontrolador Moderno: Utiliza o Raspberry Pi Pico, um microcontrolador de baixo custo e alto desempenho, que permite a expansão do projeto para funcionalidades mais avançadas no futuro.**

- **Este projeto é original e foi desenvolvido com base nos conceitos e tecnologias estudados durante a capacitação em Sistemas Embarcados, sem cópia de projetos existentes.**

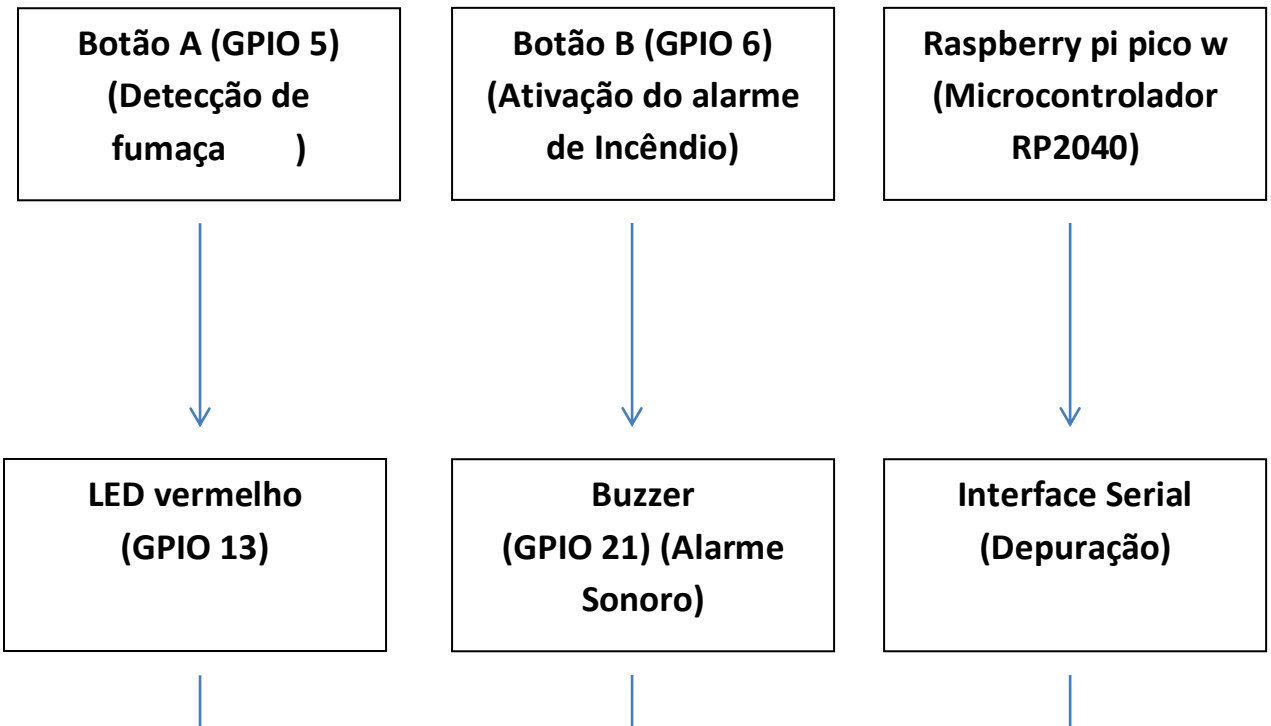


2. Especificação do Hardware

Nesta seção, vou descrever detalhadamente o hardware utilizado no projeto, explicando cada componente, sua função, configuração, comandos, registros, pinagem e como o circuito completo foi montado. Vou adotar uma linguagem mais pessoal, como se eu estivesse explicando diretamente para você.

2.1 - Diagrama em Bloco

O sistema foi projetado com base em um diagrama em bloco simples, que divide o hardware em três partes principais: entradas, processamento e saídas. Aqui está como ele foi organizado:



2.2 Função de Cada Bloco

Componente	Função
Botão A (GPIO 5)	Simula a detecção de fumaça. Quando pressionado, aciona o LED vermelho.
Botão B (GPIO 6)	Ativa o alarme de incêndio, tocando a sirene no buzzer e piscando o LED.
LED Vermelho (GPIO 13)	Serve como indicador visual de emergência (detecção de fumaça ou alarme).
Buzzer (GPIO 21)	Emite o som da sirene de emergência quando o alarme é ativado.
Raspberry Pi Pico	Microcontrolador que controla todos os componentes e executa a lógica do sistema.
Interface Serial	Usada para depuração e exibição de mensagens no console.

2.3 Configuração de Cada Bloco

Componente	Configuração
Botão A (GPIO 5)	Configurado como entrada com resistor de pull-up interno.
Botão B (GPIO 6)	Configurado como entrada com resistor de pull-up interno.
LED Vermelho (GPIO 13)	Configurado como saída digital. Inicialmente desligado.
Buzzer (GPIO 21)	Configurado como saída digital. Inicialmente desligado.
Interface Serial	Inicializada para comunicação com o computador para depuração.

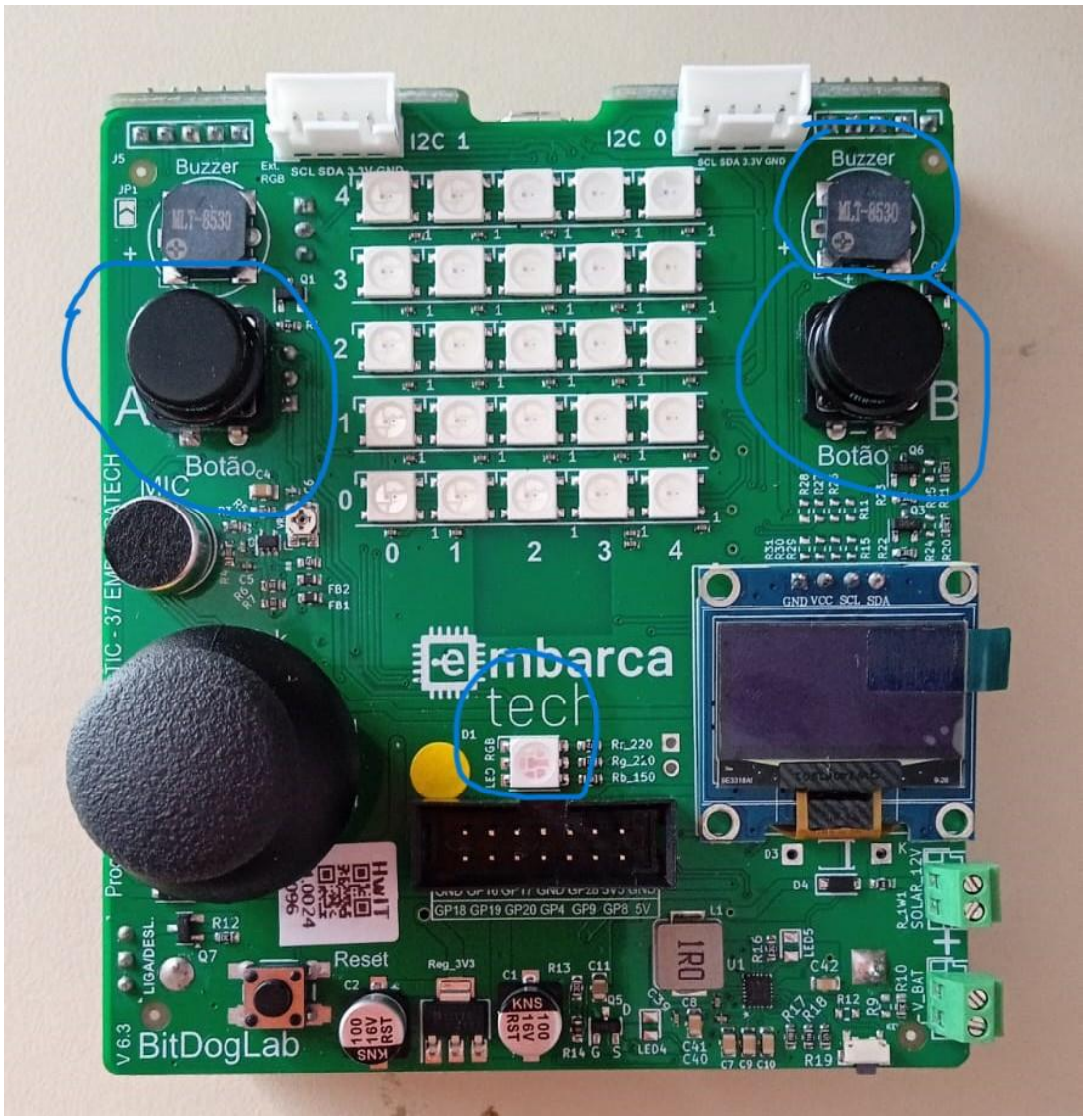
2.4 Comandos e Registros Utilizados

Componente	Comandos/Registros
Botão A (GPIO 5)	gpio_get(button_a) para ler o estado do botão.

Botão B (GPIO 6)	gpio_get(button_b) para ler o estado do botão.
LED Vermelho (GPIO 13)	gpio_put(led_red, true) para ligar e gpio_put(led_red, false) para desligar.
Buzzer (GPIO 21)	gpio_put(buzzer, 1) para ligar e gpio_put(buzzer, 0) para desligar.
Interface Serial	stdio_init_all() para inicializar a comunicação serial.

2.5 Descrição da Pinagem

Pino do Raspberry Pi Pico	Componente Conectado	Função
GPIO 5	Botão A	Entrada digital para simular a detecção de fumaça.
GPIO 6	Botão B	Entrada digital para ativar o alarme de incêndio.
GPIO 13	LED Vermelho	Saída digital para controle do LED indicador de emergência.
GPIO 21	Buzzer	Saída digital para controle do buzzer (sirene de emergência).
GND	Botões, LED, Buzzer	Conexão comum de terra para todos os componentes.



2.6 Circuito Completo do Hardware

Descrição do circuito completo:

1. **Botão A (GPIO 5):**

- Conectado entre o GPIO 5 e o GND.
- Resistor de pull-up interno ativado no GPIO 5.

2. **Botão B (GPIO 6):**

- Conectado entre o GPIO 6 e o GND.
- Resistor de pull-up interno ativado no GPIO 6.

3. **LED Vermelho (GPIO 13):**

- Anodo conectado ao GPIO 13.
- Catodo conectado ao GND através de um resistor limitador de corrente (220Ω).

4. **Buzzer (GPIO 21):**

- Terminal positivo conectado ao GPIO 21.
- Terminal negativo conectado ao GND

5. **Alimentação:**

- O Raspberry Pi Pico é alimentado via USB ou por uma fonte externa de 5V.

3. Especificação do Firmware

Nesta seção, vou descrever o firmware do projeto, explicando como ele foi desenvolvido, quais são suas funcionalidades, como as variáveis foram utilizadas, o fluxo de execução, a inicialização do sistema, as configurações dos registros, a estrutura dos dados, o protocolo de comunicação e o formato dos pacotes. Vou adotar uma linguagem mais pessoal, como se eu estivesse explicando diretamente para você.

3.1 Blocos Funcionais

O firmware foi organizado em blocos funcionais para facilitar o desenvolvimento e a manutenção do código. Esses blocos são:

Inicialização do Sistema:

Responsável por configurar os pinos GPIO, inicializar o LED, o buzzer e os botões, além de preparar a interface serial para depuração.

Leitura dos Botões:

Verifica o estado dos botões A e B, implementando um mecanismo de debouncing para evitar leituras falsas.

Controle do LED Vermelho:

Gerencia o acionamento e desligamento do LED vermelho, que serve como indicador visual de emergência.

Controle do Buzzer:

Controla o buzzer para tocar a sirene de emergência quando o alarme é ativado.

Interface Serial:

Usada para depuração e exibição de mensagens no console, permitindo acompanhar o funcionamento do sistema em tempo real.

3.2 Descrição das Funcionalidades

Aqui está uma descrição detalhada das funcionalidades implementadas em cada bloco:

Inicialização do Sistema:

Configurei os pinos GPIO como entradas (para os botões) ou saídas (para o LED e o buzzer). Também inicializei a interface serial para depuração, o que me permitiu exibir mensagens no console e acompanhar o funcionamento do sistema.

Leitura dos Botões:

Implementei um mecanismo de debouncing para garantir que apenas pressões válidas dos botões sejam detectadas. Quando o botão A é pressionado, o sistema simula a detecção de fumaça e aciona o LED vermelho. Já o botão B, quando pressionado, ativa o alarme de incêndio, tocando a sirene no buzzer e piscando o LED.

Controle do LED Vermelho:

O LED vermelho é acionado quando o botão A é pressionado (indicando detecção de fumaça) e pisca em sincronia com a sirene quando o botão B é pressionado (indicando alarme de incêndio).

Controle do Buzzer:

O buzzer toca uma sirene de emergência composta por uma sequência de notas musicais. A sirene é repetida por 10 ciclos para garantir que o alerta seja percebido.

Interface Serial:

Utilizei a interface serial para exibir mensagens no console, como "DETECÇÃO DE FUMAÇA!!" e "ACIONANDO ALARME DE INCÊNDIO!!", o que me ajudou a depurar e entender o comportamento do sistema.

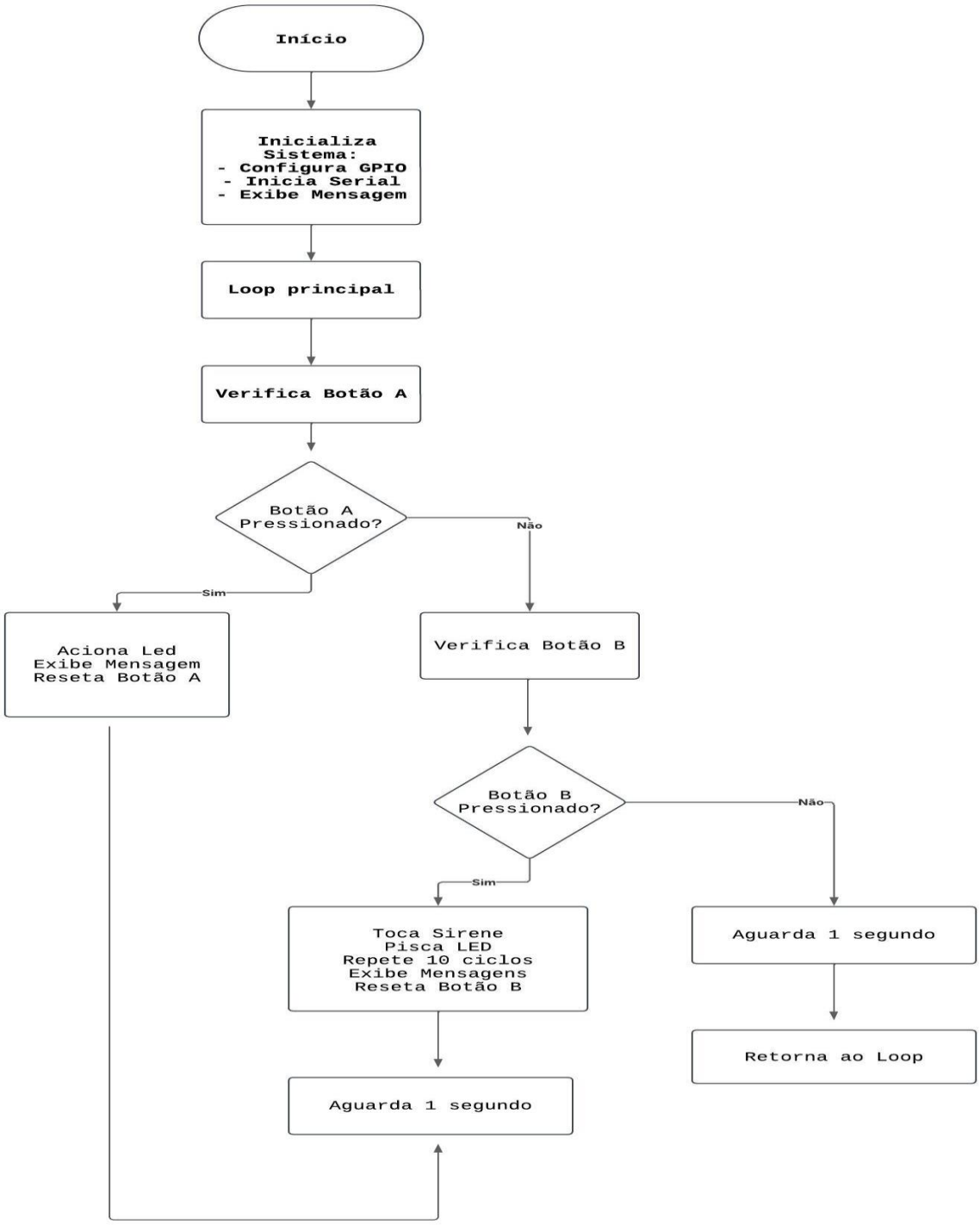
3.3 Variáveis

As principais variáveis que utilizei no firmware são:

Variável	Tipo	Descrição
button_a	int	Pino GPIO conectado ao botão A (detecção de fumaça).
button_b	int	Pino GPIO conectado ao botão B (ativação do alarme).
led_red	int	Pino GPIO conectado ao LED vermelho.
buzzer	int	Pino GPIO conectado ao buzzer.
botao_pressionado_A	volatile bool	Indica se o botão A foi pressionado (detecção de fumaça).
botao_pressionado_B	volatile bool	Indica se o botão B foi pressionado (ativação do alarme).
melody[]	int[]	Array contendo as notas musicais da sirene de emergência.
noteDuration	int	Duração de cada nota da sirene (em milissegundos).

3.4 Fluxograma

O fluxograma do firmware descreve o fluxo de execução do sistema:



Inicialização:

Configuro os GPIO, o LED, o buzzer e a interface serial.

Loop Principal:

Verifico o estado dos botões A e B.

Se o botão A for pressionado:

Aciono o LED vermelho.

Exibo "DETECÇÃO DE FUMAÇA" no console.

Se o botão B for pressionado:

Toco a sirene no buzzer.

Pisco o LED vermelho.

Exibo "ACIONANDO ALARME DE INCÊNDIO" no console.

Repito o loop continuamente.

3.5 Inicialização

A inicialização do sistema é feita na função **setup()**, onde realizo as seguintes tarefas:

Configuro os pinos GPIO dos botões como entradas com resistor de pull-up.

Configuro os pinos GPIO do LED e do buzzer como saídas.

Inicializo a interface serial para depuração.

Exibo uma mensagem no console indicando que o sistema está pronto.

3.6 Configurações dos Registros

Utilizei os registros do **Raspberry Pi Pico** para controlar os periféricos. As principais configurações são:

GPIO:

Os pinos dos botões são configurados como entradas com **gpio_set_dir()** e **gpio_pull_up().**

Os pinos do LED e do buzzer são configurados como saídas com **gpio_set_dir().**

Interface Serial:

Inicializada com **stdio_init_all()** para permitir a comunicação com o computador.

3.7 Estrutura dos Dados

Os dados são organizados da seguinte forma:

Notas da Sirene:

Armazenei as notas em um array de inteiros **(melody[])**, onde cada valor representa a frequência de uma nota.

A duração de cada nota é definida pela variável **noteDuration.**

Estado dos Botões:

Utilizei variáveis booleanas **(botao_pressionado_A)** e **(botao_pressionado_B)** para armazenar o estado dos botões.

3.8 Protocolo de Comunicação

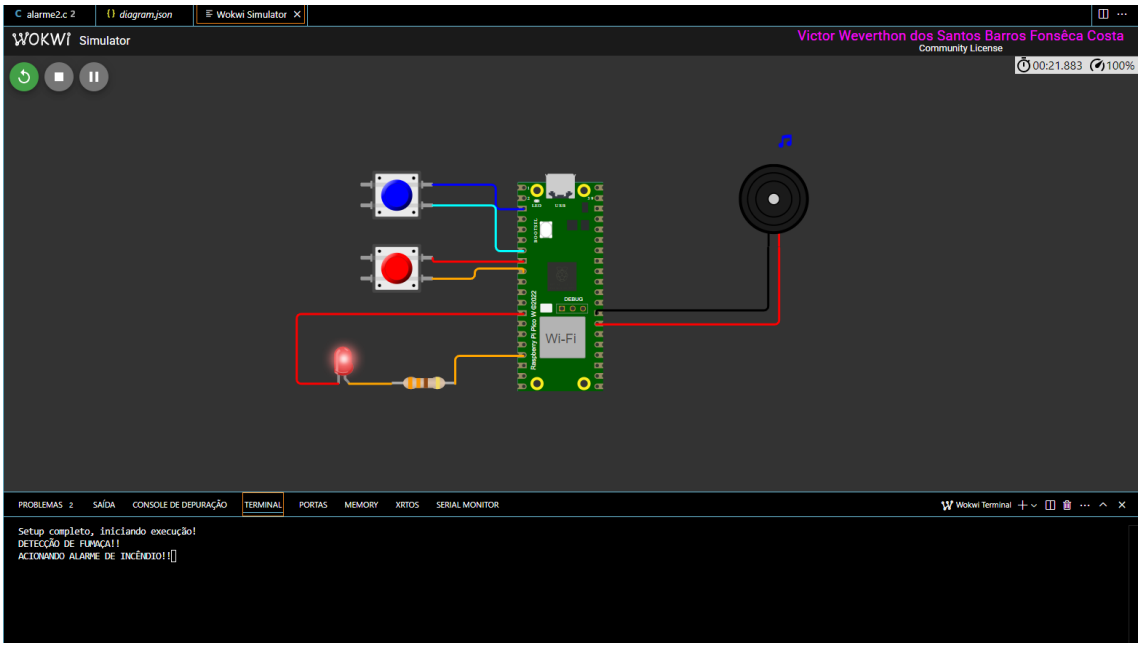
Neste projeto, a comunicação é feita principalmente através da interface serial, usada para depuração. As mensagens são enviadas para o console em formato de texto simples, sem um protocolo complexo.

3.9 Formato dos Pacotes

Como o sistema não utiliza comunicação externa (como Wi-Fi ou Bluetooth), não há pacotes de dados estruturados. A única comunicação é a exibição de mensagens no console, que segue o formato:

"DETECÇÃO DE FUMAÇA!!"

"ACIONANDO ALARME DE INCÊNDIO!!"



4. Execução do Projeto

Nesta seção, vou descrever como o projeto foi executado, desde a metodologia utilizada até os testes de validação e a discussão dos resultados. Vou adotar uma linguagem mais pessoal, como se eu estivesse explicando diretamente para você.

4.1 Metodologia

A execução do projeto foi dividida em etapas claras e bem definidas, seguindo uma metodologia que garantiu o desenvolvimento organizado e eficiente. Aqui estão as etapas que segui:

Definição do Escopo:

Primeiro, defini claramente o que o sistema deveria fazer: simular a detecção de fumaça e acionar um alarme de incêndio com indicadores visuais e sonoros.

Pesquisa de Componentes:

Pesquisei os componentes necessários (botões, LED, buzzer) e verifiquei sua compatibilidade com o Raspberry Pi Pico.

Desenvolvimento do Hardware:

Montei o circuito conforme o diagrama em bloco, conectando os botões, o LED e o buzzer ao Raspberry Pi Pico.

Desenvolvimento do Firmware:

Escrevi o código em C, utilizando a SDK do Raspberry Pi Pico. O código foi dividido em funções para facilitar a manutenção e o entendimento.

Testes e Depuração:

Testei cada funcionalidade separadamente (leitura dos botões, controle do LED, controle do buzzer) antes de integrá-las no sistema completo.

Validação do Sistema:

Realizei testes finais para garantir que o sistema funcionasse conforme o esperado, tanto em condições normais quanto em situações de emergência simuladas.

Documentação:

Documentei todo o processo, desde o desenvolvimento até os testes, para facilitar a reprodução do projeto e a correção de possíveis problemas.

4.2 Testes de Validação

Para garantir que o sistema funcionasse corretamente, realizei vários testes. Aqui estão os principais:

Teste dos Botões:

Verifiquei se os botões A e B estavam sendo lidos corretamente pelo microcontrolador.

Testei o mecanismo de debouncing para garantir que não houvesse leituras falsas.

Teste do LED Vermelho:

Verifiquei se o LED acendia e apagava corretamente quando os botões eram pressionados.

Testei a sincronização do LED com a sirene do buzzer.

Teste do Buzzer:

Verifiquei se o buzzer tocava a sirene de emergência corretamente.

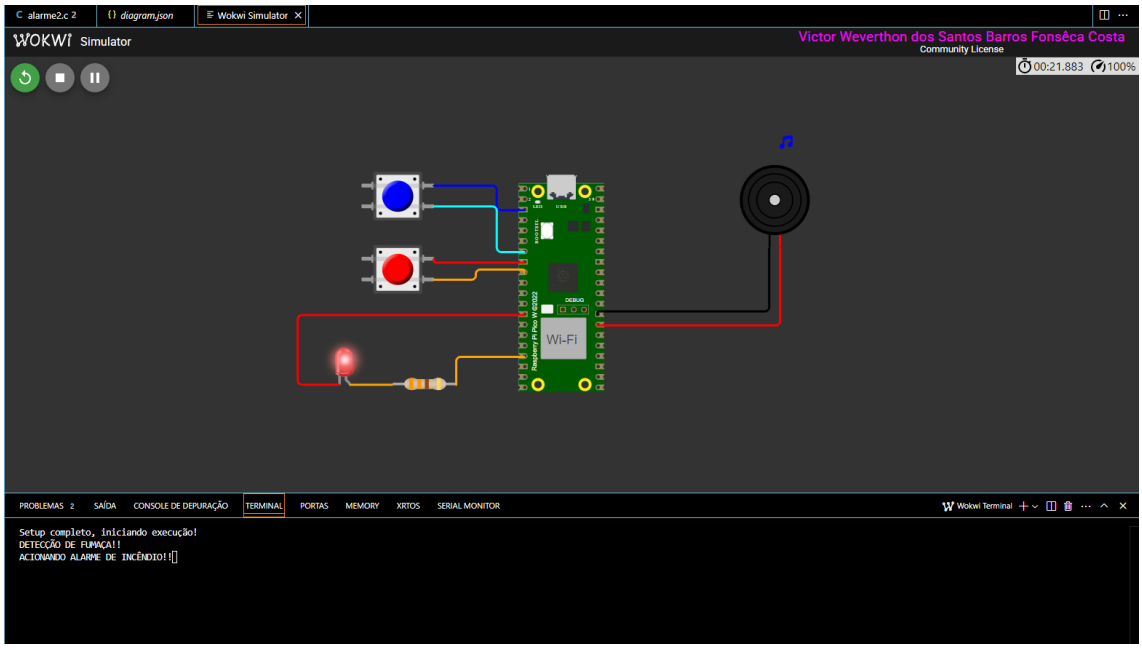
Testei a duração e a sequência das notas para garantir que o som fosse claro e perceptível.

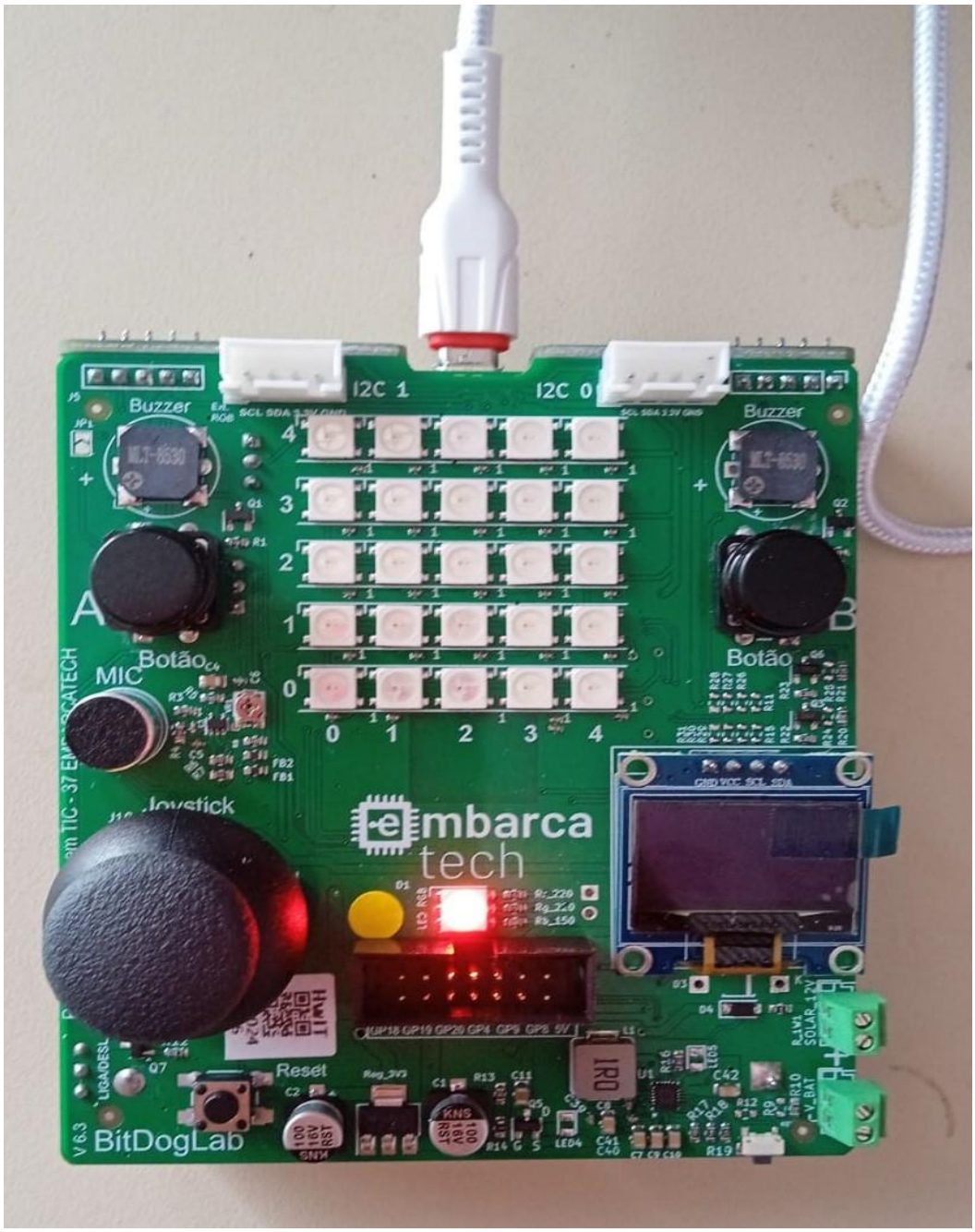
Teste Integrado:

Simulei uma situação de emergência, pressionando os botões A e B em sequência para verificar se o sistema respondia conforme o esperado.

Teste de Estresse:

Realizei testes prolongados para garantir que o sistema funcionasse de forma estável, sem falhas ou comportamentos inesperados.





4.3 Discussão dos Resultados

Os resultados dos testes foram muito satisfatórios. A seguir está uma análise detalhada:

Funcionamento dos Botões:

O mecanismo de debouncing funcionou perfeitamente, garantindo que apenas pressões válidas fossem detectadas. Isso eliminou leituras falsas e melhorou a confiabilidade do sistema.

Funcionamento do LED Vermelho:

O LED respondeu corretamente aos comandos, acendendo e apagando conforme o esperado. A sincronização com a sirene do buzzer também funcionou bem, criando um efeito visual eficaz para situações de emergência.

Funcionamento do Buzzer:

O buzzer tocou a sirene de emergência de forma clara e perceptível. A sequência de notas e a duração de cada nota foram ajustadas para criar um som que chama a atenção sem ser excessivamente irritante.

Funcionamento Integrado:

O sistema funcionou de forma integrada, respondendo corretamente aos comandos dos botões e executando as ações esperadas (detecção de fumaça e alarme de incêndio).

Estabilidade do Sistema:

Durante os testes prolongados, o sistema manteve-se estável, sem apresentar falhas ou comportamentos inesperados. Isso demonstra que o projeto foi bem desenvolvido e implementado.

4.4 Conclusão

A execução do projeto foi um sucesso. O sistema embarcado desenvolvido atendeu a todos os requisitos e funcionou conforme o esperado. Os testes de validação confirmaram que o sistema é confiável e eficaz, podendo ser utilizado em situações reais de detecção de incêndio e alerta de emergência.

Além disso, a metodologia adotada garantiu que o desenvolvimento fosse organizado e eficiente, facilitando a identificação e correção de problemas durante a fase de testes. A documentação detalhada também permitirá que outras pessoas reproduzam o projeto com facilidade.

Referências

1. Notícias e Dados sobre Incêndios

Incêndios e Mortes por Falta de Alarmes Rápidos:

Descrição: Reportagens e estudos que destacam a importância de sistemas de alarme rápidos e eficazes para prevenir mortes em incêndios.

Referência: BBC News. (2021). "Grenfell Tower: Why did so many people die in the fire?"
Link

Estatísticas sobre Incêndios Residenciais:

Descrição: Dados sobre o número de mortes e feridos em incêndios residenciais, destacando a necessidade de sistemas de detecção precoce.

Referência: National Fire Protection Association (NFPA). (2022). "Home Structure Fires Report."
Link

Impacto de Alarmes de Incêndio em Escolas e Hospitais:

Descrição: Estudos que mostram como sistemas de alarme eficazes podem salvar vidas em locais públicos, como escolas e hospitais.

Referência: The Guardian. (2020). "Fire safety in schools: Why alarms and drills matter."
Link

---

2. Artigos Científicos

Sistemas Embarcados para Detecção de Incêndio:

Descrição: Artigo que explora o uso de sistemas embarcados para detecção precoce de incêndios, com foco em sensores e algoritmos de processamento.

Referência: Kumar, A., & Singh, P. (2019). "Embedded Systems for Early Fire Detection: A Review." International Journal of Advanced Research in Computer Science, 10(2), 1-7.

Tecnologias IoT para Segurança contra Incêndios:

Descrição: Estudo sobre a integração de sistemas embarcados com IoT para monitoramento e alerta de incêndios em tempo real.

Referência: Li, S., & Xu, L. D. (2017). "Securing the Internet of Things." Syngress.

Eficácia de Alarmes Sonoros e Visuais em Emergências:

Descrição: Pesquisa que avalia a eficácia de alarmes sonoros e visuais em situações de emergência, como incêndios.

Referência: Proulx, G. (2002). "Evacuation Time and Movement in Apartment Buildings." Fire Safety Journal, 37(3), 237-254.

---

3. Projetos Similares em Sistemas Embarcados

Sistema de Alarme de Incêndio com Arduino:

Descrição: Projeto que utiliza um Arduino para criar um sistema de alarme de incêndio com sensores de fumaça e alarme sonoro.

Referência: Instructables. (2020). "Fire Alarm System Using Arduino."
Link

Sistema de Detecção de Incêndio com Raspberry Pi:

Descrição: Projeto que utiliza um Raspberry Pi para detectar incêndios e enviar alertas via SMS e e-mail.

Referência: Hackster.io. (2021). "Fire Detection System with Raspberry Pi."
Link

Sistema Embarcado para Monitoramento de Incêndios Florestais:

Descrição: Projeto que utiliza sensores e comunicação sem fio para monitorar incêndios florestais em tempo real.

Referência: IEEE Xplore. (2018). "Wireless Sensor Network for Forest Fire Detection and Monitoring."
Link

---

4. Relatórios e Estudos sobre Segurança contra Incêndios

Relatório da Organização Mundial da Saúde (OMS):

Descrição: Relatório que aborda o impacto global de incêndios e a importância de sistemas de alerta precoce.

Referência: World Health Organization (WHO). (2021). "Fire-related injuries and deaths: A global perspective."
Link

Estudo sobre Eficácia de Alarmes de Incêndio em Residências:

Descrição: Estudo que avalia a eficácia de alarmes de incêndio em residências e seu impacto na redução de mortes.

Referência: Ahrens, M. (2019). "Smoke Alarms in U.S. Home Fires." National Fire Protection Association (NFPA).
Link

---

5. Outras Referências Técnicas

Guia de Desenvolvimento de Sistemas Embarcados com Raspberry Pi Pico:

Descrição: Guia prático para desenvolvimento de sistemas embarcados utilizando o Raspberry Pi Pico.

Referência: Raspberry Pi Foundation. (2021). "Getting Started with Raspberry Pi Pico."
Link

Tutoriais sobre Controle de GPIO e PWM:

Descrição: Tutoriais que explicam como controlar GPIO e PWM no Raspberry Pi Pico.

Referência: Raspberry Pi Tutorials. (2021). "GPIO and PWM Control with Raspberry Pi Pico."
Link

5. Anexos

Nesta seção, vou detalhar o código-fonte do projeto, explicando função por função e como a lógica foi desenvolvida. Além disso, vou sugerir outros materiais relevantes que podem ser incluídos como anexos para enriquecer a documentação.

5.1 Código-Fonte Detalhado

Aqui está o código-fonte do projeto, com uma explicação detalhada de cada função e da lógica utilizada:

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"

// Definição dos pinos
#define button_a  5
#define button_b  6
#define led_red  13
#define buzzer  21

// Declaração das funções
void iniciar_led_red(void); // Função para inicializar o LED vermelho
void desligar_led_red(void); // Função para desligar o LED vermelho
void acionar_led_red(void);   // Função para acionar o LED vermelho
void iniciar_buzzer(void);     // Função para inicializar o buzzer
void tocar_alarme_incendio(void); // Função para tocar o alarme de incêndio
void verificar_botao_A(void);     // Função de debouncing do botão A
void verificar_botao_B(void);     // Função de debouncing do botão B

// Sequência de notas da sirene
```

```
#define NOTE_C4 262

#define NOTE_D4 294

#define NOTE_E4 330

#define NOTE_F4 349

#define NOTE_G4 392

#define NOTE_E5 659


int melody[] = {NOTE_C4, NOTE_D4, NOTE_E4, NOTE_F4, NOTE_G4,
NOTE_F4, NOTE_E4, NOTE_D4, NOTE_E5};

int noteDuration = 300; // Duração de cada nota (ms)


// Variáveis para debouncing dos botões

volatile bool botao_pressionado_A = false;

volatile bool botao_pressionado_B = false;


// Funções para controle do LED

void iniciar_led_red(void) {

  gpio_init(led_red);

  gpio_set_dir(led_red, GPIO_OUT);

  gpio_put(led_red, false); // Inicializa o LED desligado

}


void acionar_led_red(void) {

  gpio_put(led_red, true); // Liga o LED

}


void desligar_led_red(void) {
```

```
  gpio_put(led_red, false); // Desliga o LED
}

// Funções para controle do Buzzer
void iniciar_buzzer(void) {
  gpio_init(buzzer);
  gpio_set_dir(buzzer, GPIO_OUT);
  gpio_put(buzzer, false); // Inicializa o buzzer desligado
}

void tocar_nota(int frequencia, int duracao_ms) {
  int tempo = 1000000 / frequencia; // Cálculo do tempo para cada ciclo
  for (int i = 0; i < (duracao_ms * 1000) / tempo; i++) {
    gpio_put(buzzer, 1);
    sleep_us(tempo / 2);
    gpio_put(buzzer, 0);
    sleep_us(tempo / 2);
  }
}

void tocar_alarme_incendio(void) {
  // Define a melodia da sirene e a duração de cada nota
  int melody[] = {440, 659, 440, 659, 440, 659, 440, 659}; // A4, E5, A4, E5, A4,
E5, A4, E5
  int noteDuration = 500; // Duração da nota em milissegundos

  for (int i = 0; i < 10; i++) {
```

```
    desligar_led_red();
    botao_pressionado_B = false;  // Reseta o estado do botão B
    acionar_led_red();  // Aciona o LED vermelho (Sirene)

    // Toca a melodia da sirene
    for (int j = 0; j < sizeof(melody) / sizeof(melody[0]); j++) {
      // Toca sirene de emergência
      tocar_nota(melody[j], noteDuration); // Toca a nota com a duração definida
      sleep_ms(50); // Intervalo de 0,05 segundos entre as notas
    }

    sleep_ms(500); // Mantém o buzzer e o LED ligados por 0,5 segundos
    gpio_put(buzzer, 0); // Desliga o buzzer
    // Não desligue o LED aqui, o LED deve permanecer ligado
    sleep_ms(500); // Intervalo de 0,5 segundos entre as execuções
  }
}

// Funções para debouncing dos botões
void verificar_botao_A(void) {
  static uint32_t ultima_leitura_A = 0;
  uint32_t leitura_atual_A = gpio_get(button_a);
  if (leitura_atual_A == 0 && (time_us_32() - ultima_leitura_A) > 100000) {
    botao_pressionado_A = true;
    ultima_leitura_A = time_us_32();
  }
}
```

```
void verificar_botao_B(void) {
  static uint32_t ultima_leitura_B = 0;
  uint32_t leitura_atual_B = gpio_get(button_b);
  if (leitura_atual_B == 0 && (time_us_32() - ultima_leitura_B) > 100000) {
    botao_pressionado_B = true;
    ultima_leitura_B = time_us_32();
  }
}

// Função de inicialização
void setup(void) {
  gpio_init(button_a);
  gpio_set_dir(button_a, GPIO_IN);
  gpio_pull_up(button_a);

  gpio_init(button_b);
  gpio_set_dir(button_b, GPIO_IN);
  gpio_pull_up(button_b);

  iniciar_led_red();   // Inicia o LED vermelho
  iniciar_buzzer();    // Inicia o Buzzer
  stdio_init_all();  // Inicia a interface serial para depuração

  printf("Setup completo, iniciando execução!\n");
}
```

```
// Função principal
void loop(void) {
  verificar_botao_A();
  verificar_botao_B();

  if (botao_pressionado_A) {
    printf("DETECÇÃO DE FUMAÇA!!\n");
    botao_pressionado_A = false;
  }

  if (botao_pressionado_B) {
    printf("ACIONANDO ALARME DE INCÊNDIO!!");
    tocar_alarme_incendio();  // Chama a função para tocar o alarme de incêndio
    botao_pressionado_B = false;
  }

  sleep_ms(1000);
}

int main(void) {
  setup();
  while(1) {
    loop();
  }
  return 0;
}
```

5.2 Explicação Função por Função

iniciar_led_red():

Função: Inicializa o pino GPIO conectado ao LED vermelho como saída e o desliga.

Lógica: Usa gpio_init() para inicializar o pino e gpio_put() para definir o estado inicial como desligado.

acionar_led_red():

Função: Liga o LED vermelho.

Lógica: Usa gpio_put() para definir o pino do LED como alto (ligado).

desligar_led_red():

Função: Desliga o LED vermelho.

Lógica: Usa gpio_put() para definir o pino do LED como baixo (desligado).

iniciar_buzzer():

Função: Inicializa o pino GPIO conectado ao buzzer como saída e o desliga.

Lógica: Similar à inicialização do LED, mas para o buzzer.

tocar_nota():

Função: Toca uma nota musical no buzzer com uma frequência e duração específicas.

Lógica: Usa pulsos PWM para gerar a nota, alternando o estado do pino do buzzer em alta velocidade.

tocar_alarme_incendio():

Função: Toca a sirene de emergência no buzzer e pisca o LED vermelho.

Lógica: Repete uma sequência de notas por 10 ciclos, sincronizando o som com o LED.

verificar_botao_A() e verificar_botao_B():

Função: Verifica se os botões A e B foram pressionados, implementando debouncing.

Lógica: Usa um intervalo de tempo para evitar leituras falsas causadas por ruídos mecânicos.

setup():

Função: Inicializa todos os componentes do sistema.

Lógica: Configura os pinos GPIO, inicializa o LED, o buzzer e a interface serial.

loop():

Função: Loop principal do sistema, que verifica os botões e executa as ações correspondentes.

Lógica: Chama as funções de verificação dos botões e aciona o LED ou o buzzer conforme necessário.

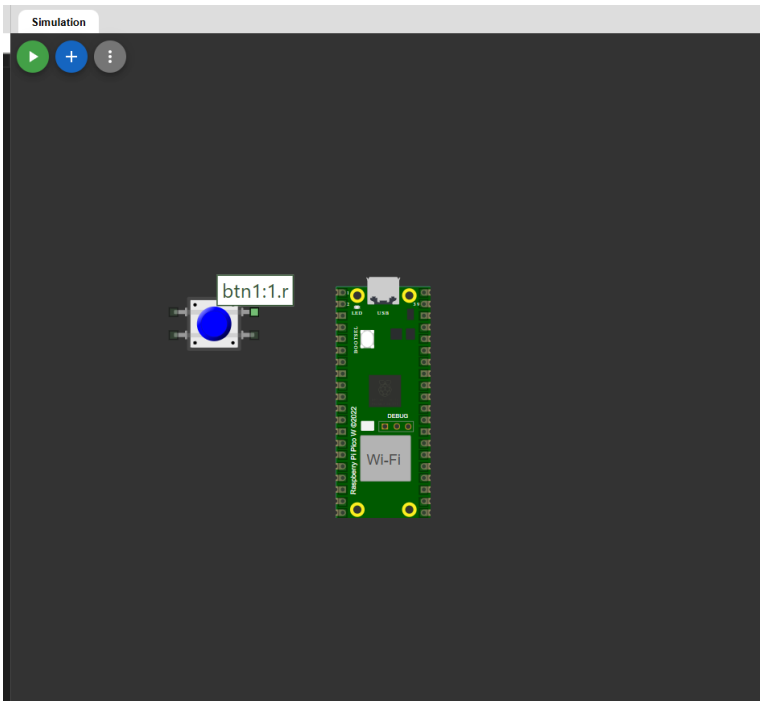
main():

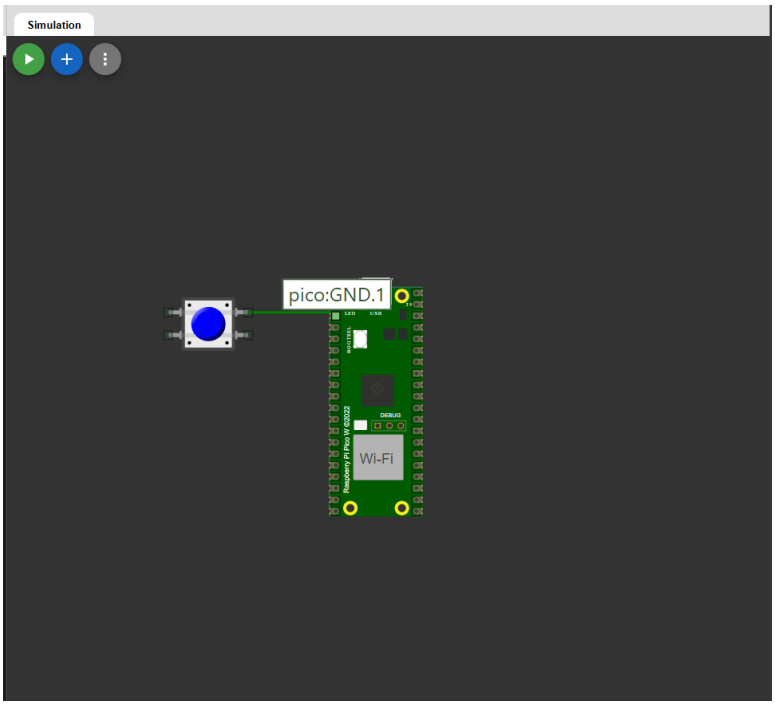
Função: Ponto de entrada do programa.

Lógica: Chama setup() uma vez e entra no loop infinito loop().

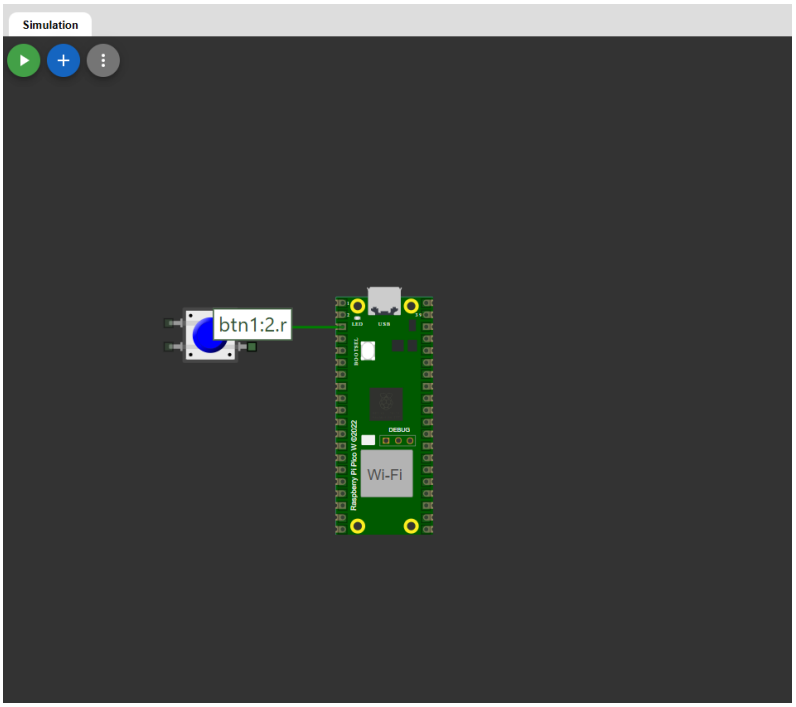
6. Montagem do Hardware no Wokwi Simulator

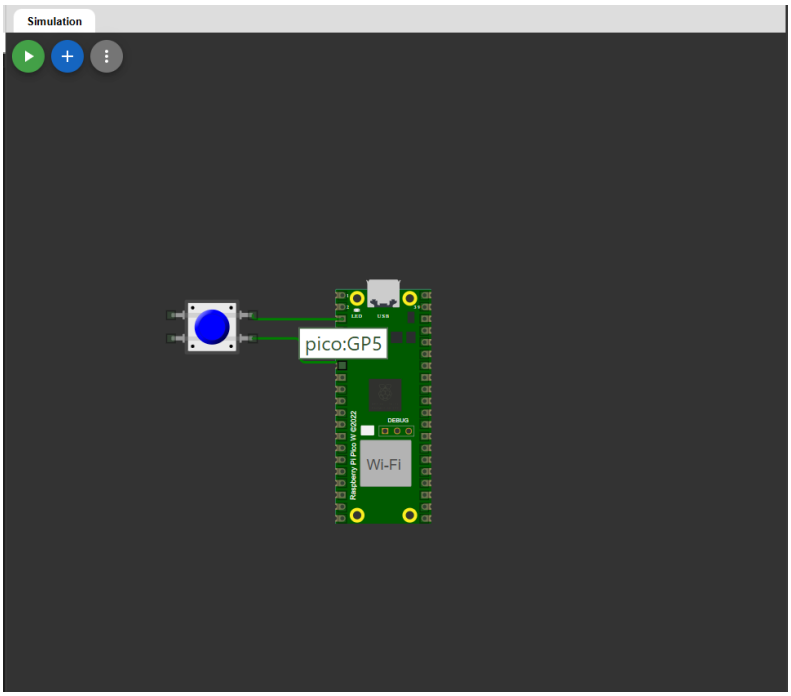
6.1 Conectando Botão A:





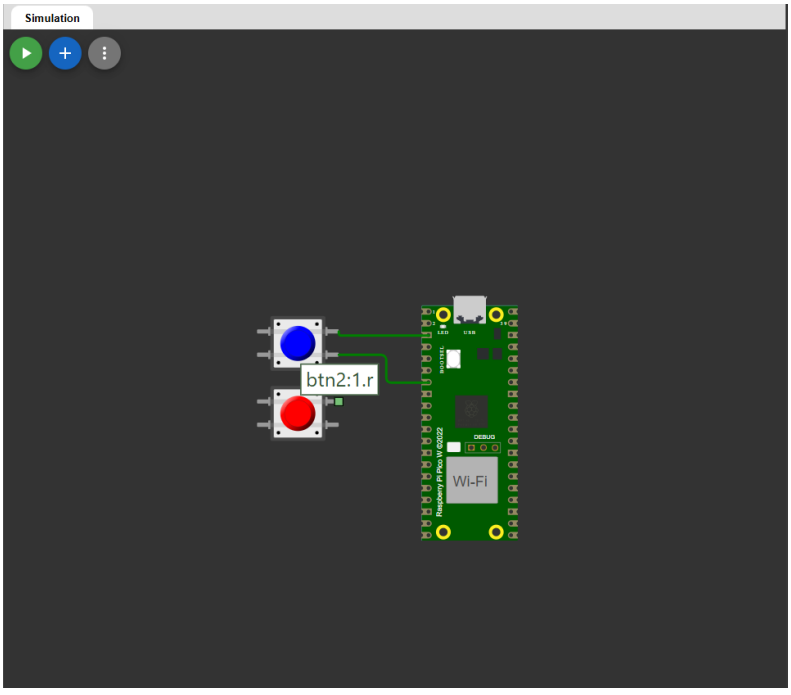
Conecte o Botão A ao GND.1



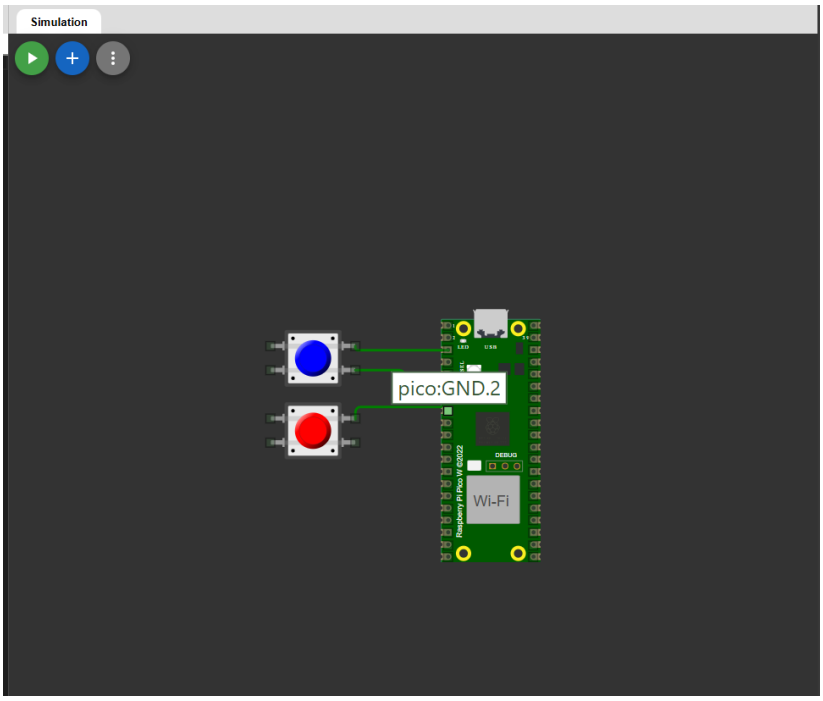


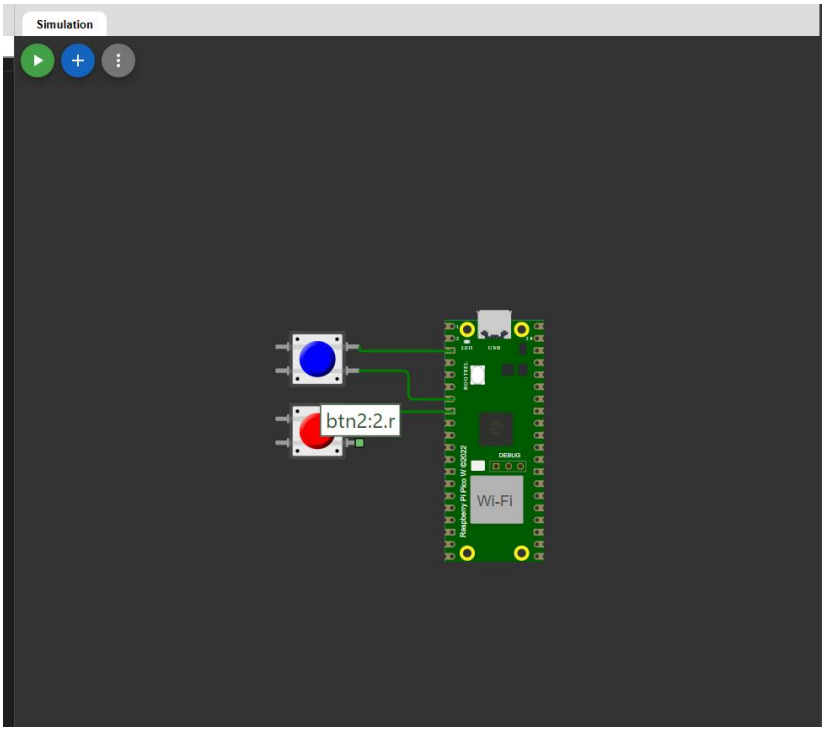
Depois ao pino GPIO 5

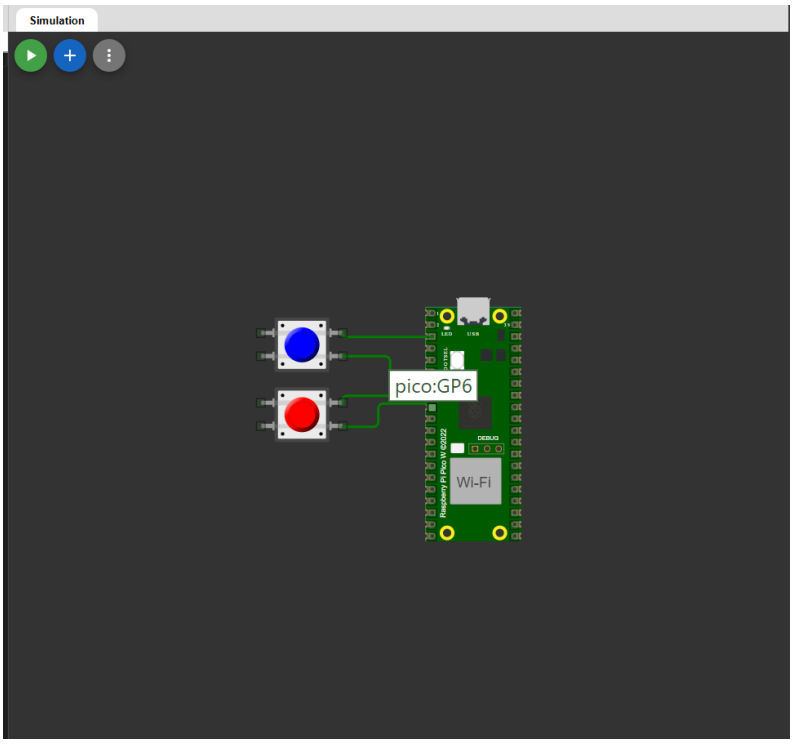
6.2 Conectando Botão B:



Conecte o Botão B ao GND.2

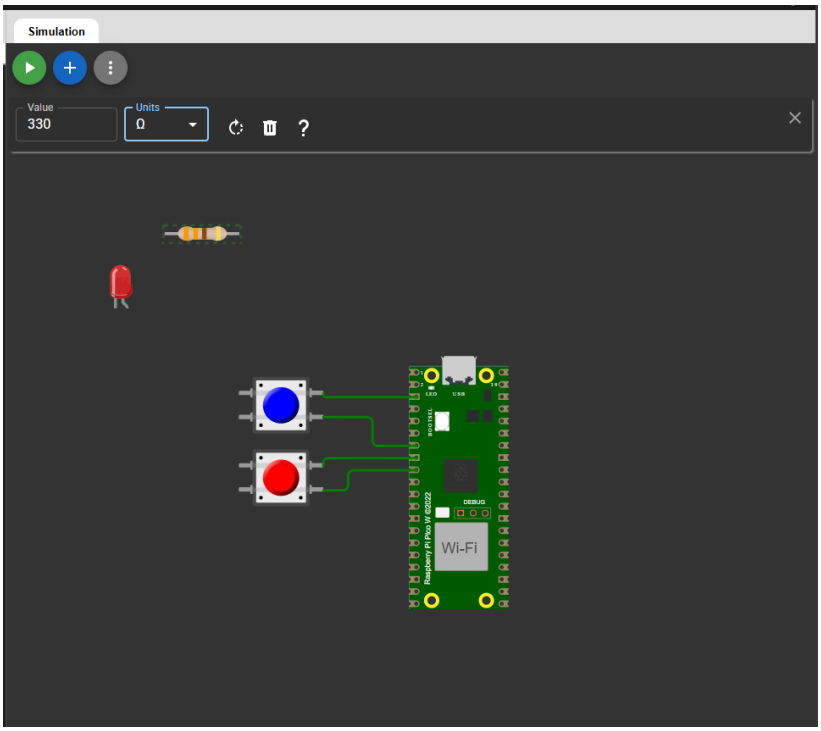

Simulation
pico:GND.2
Wi-Fi


Simulation
btn2:2.r
Wi-Fi

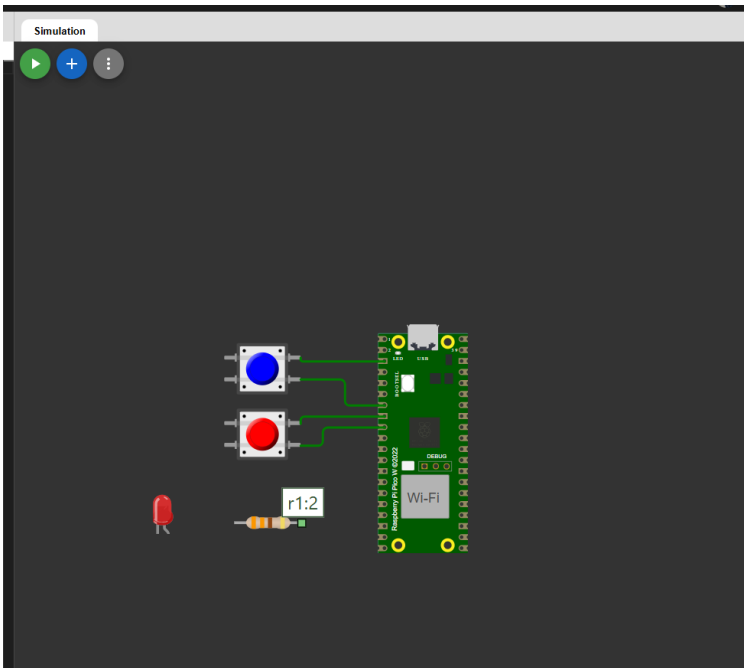


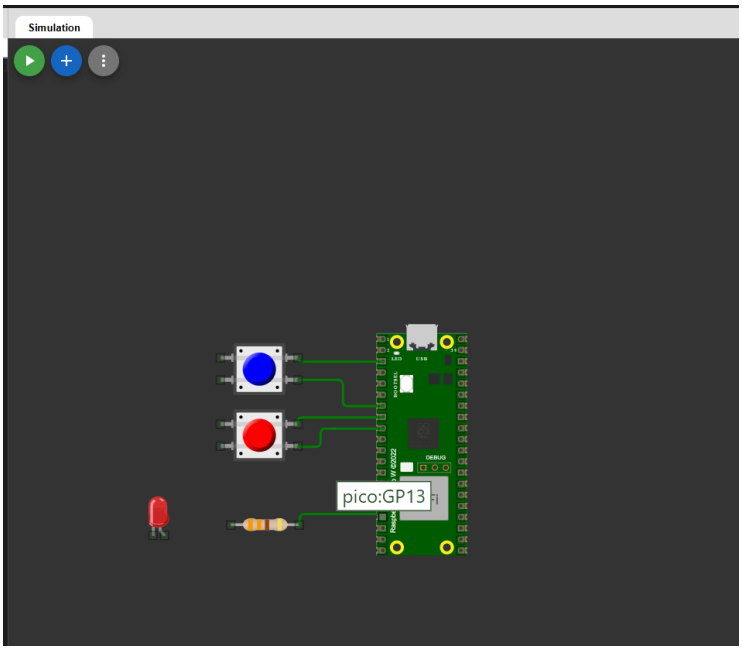
Conecte o botão B ao PINO GPIO 6

6.3 Conectando Led Vermelho:

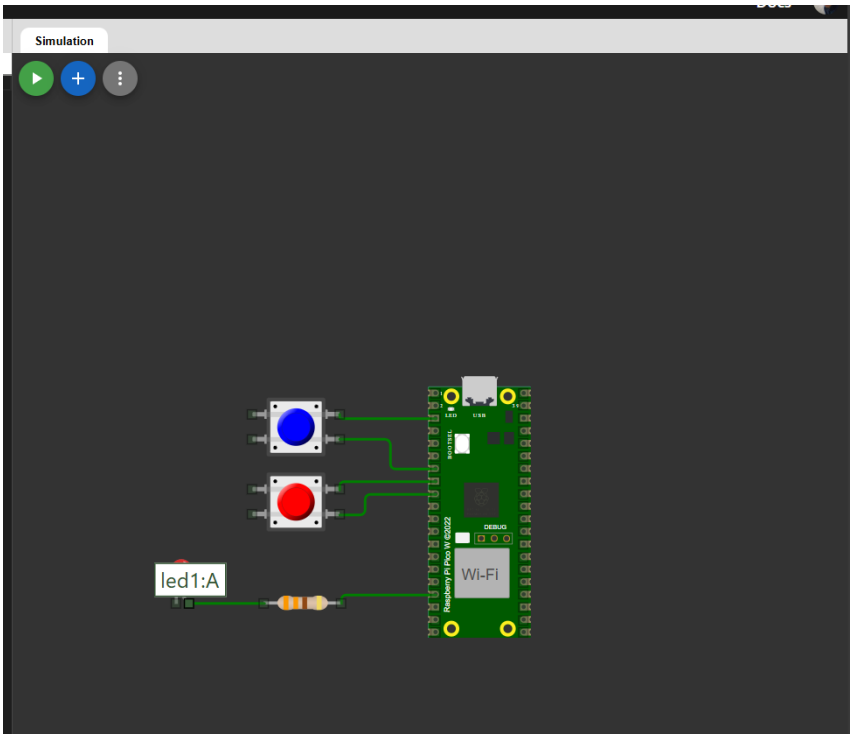


Configurando Resistor

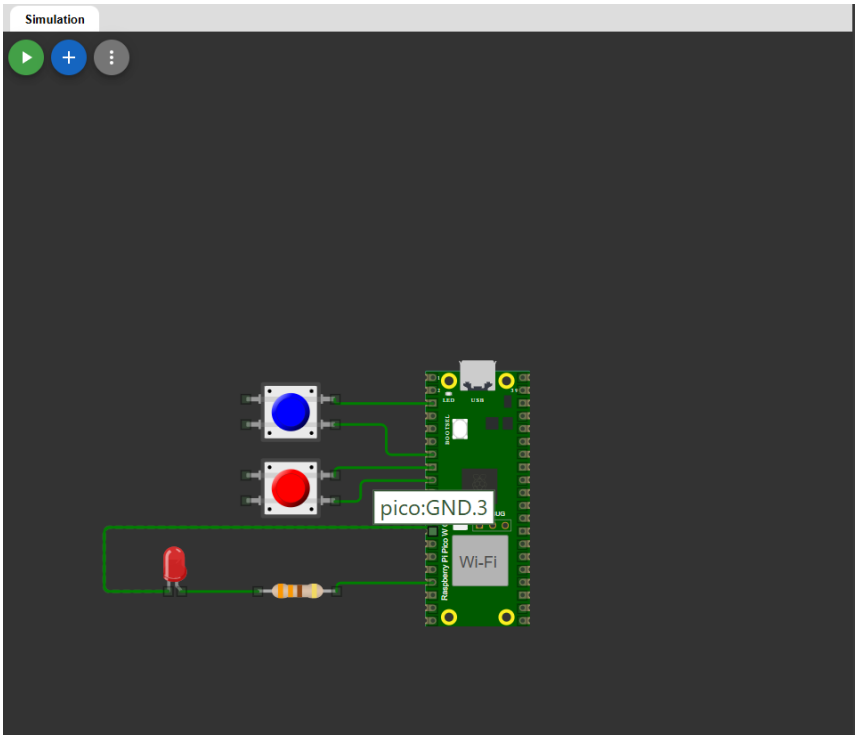




Conecte o resistor ao Pino GPIO 13

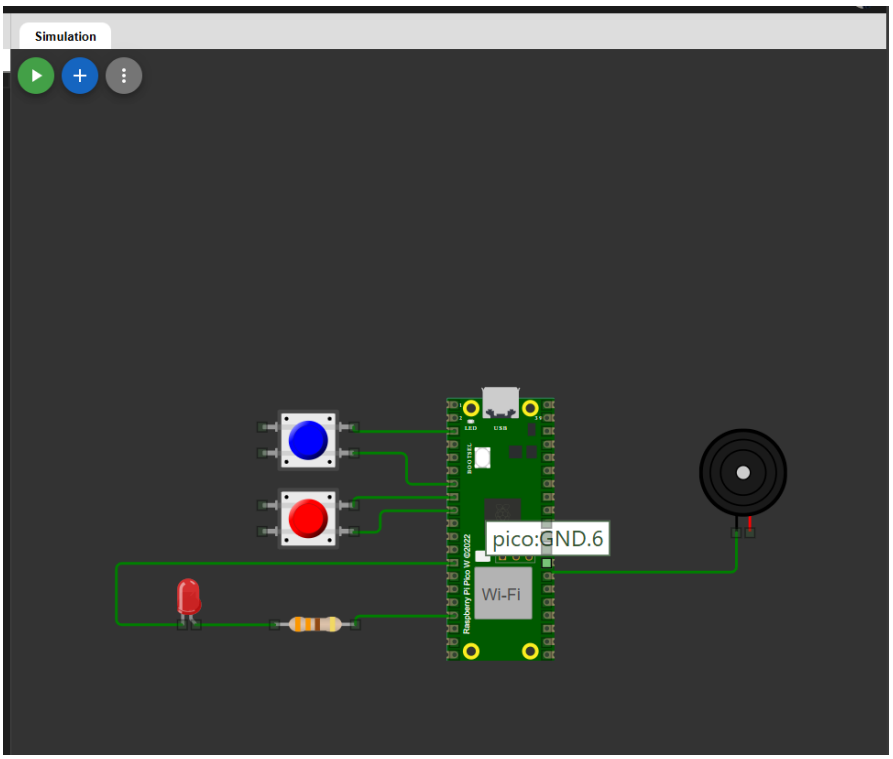


Conecte o resistor ao LED

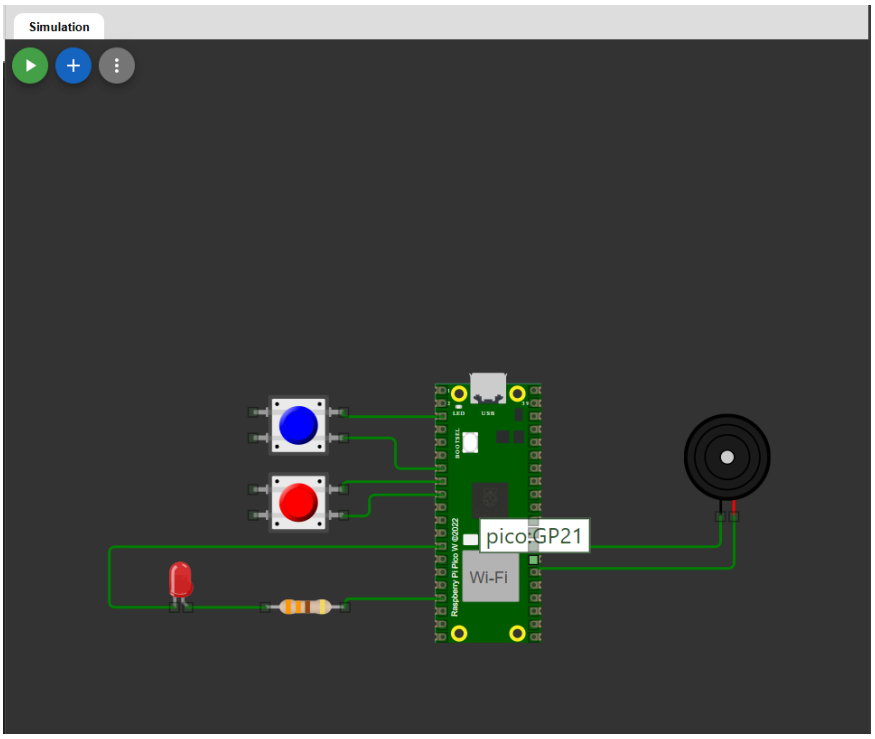


Conecte o LED ao GND.3

6.4 Conectando o Buzzer



Conecte o Buzzer ao GND.6



Conecte o Buzzer ao GPIO 21

6.5 Diagram.json

O diagram.json é um arquivo no formato JSON (JavaScript Object Notation) que descreve a estrutura do circuito, incluindo:

Os componentes utilizados (botões, LED, buzzer, Raspberry Pi Pico, etc.).

As conexões entre os componentes.

As configurações específicas de cada componente (por exemplo, pinos GPIO, resistores, etc.).

Por que o diagram.json é essencial?

Configuração do Circuito:

O diagram.json define exatamente como os componentes estão conectados. Por exemplo, ele especifica que o botão A está conectado ao GPIO 5, o LED ao GPIO 13, e o buzzer ao GPIO 21.

Sem esse arquivo, a ferramenta de simulação não saberia como montar o circuito, o que tornaria impossível testar o projeto.

Simulação Precisa:

O arquivo garante que a simulação funcione da mesma forma que o circuito real. Ele define parâmetros como resistores de pull-up, valores de resistência para LEDs, e até mesmo o tipo de buzzer utilizado.

Isso permite que você teste o projeto em um ambiente controlado, identificando possíveis erros antes de montar o hardware.

Facilidade de Compartilhamento:

O diagram.json pode ser compartilhado com outras pessoas, permitindo que elas reproduzam exatamente o mesmo circuito em suas próprias simulações.

Isso é especialmente útil para colaboração em equipe ou para compartilhar o projeto com a comunidade.

Integração com o Código:

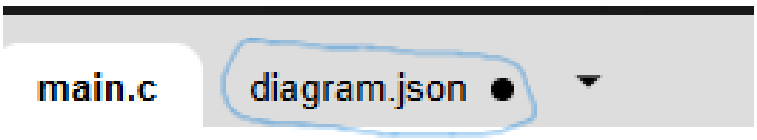
No Wokwi, o diagram.json é vinculado ao código-fonte do projeto. Isso significa que a simulação pode executar o código e mostrar os resultados em tempo real, como o acionamento do LED ou o som do buzzer.

Sem o diagram.json, o código não teria um "ambiente" para ser executado, e a simulação não funcionaria.

Documentação do Projeto:

O arquivo serve como uma documentação técnica do circuito. Ele descreve todos os componentes e conexões, o que facilita a manutenção e a replicação do projeto no futuro.

O diagram.json se encontra ao lado do arquivo main.c no Wokwi simulator, no canto superior esquerdo:



7. Links:

Github – [victorw29/alarme_incendio]

Wokwi – [alarme de incendio projeto final - Wokwi ESP32, STM32, Arduino Simulator]

Youtube - [https://www.youtube.com/watch?v=fqCXCGpPFkg&feature=youtu.be]