SISTEMA DE MONITORAMENTO DE PERTURBAÇÃO SONORA PARA CONDOMÍNIOS

Ilhéus-BA

2025

LEONAM SOUSA RABELO

SISTEMA DE MONITORAMENTO DE PERTURBAÇÃO SONORA PARA CONDOMÍNIOS

Relatório do projeto final apresentado como meio de avaliação para primeira fase do curso EmbarcaTech, realizado pelo Centro de Pesquisa, Desenvolvimento Tecnológico e Inovação, localizada em Ilhéus.

Ilhéus-BA

2025

Sumário

1. Apresentação do projeto

O projeto possui como meta principal realizar o monitoramento e identificação de ruídos sonoros elevados que causam perturbação em ambientes fechados e de convívio coletivo, como os condomínios. Para isso, será utilizada uma rede de sensores de nível sonoro integrados a microcontroladores, localizados de forma estratégica para realizar o monitoramento eficiente e inteligente.

O sistema será capaz de detectar ruídos excessivos em tempo real, acionando alertas automáticos para os responsáveis pelo ambiente e permitindo um controle mais preciso sobre infrações. Além disso, poderá ser integrado a um sistema centralizado de monitoramento, garantindo maior eficiência na fiscalização e proporcionando um ambiente mais harmonioso para os moradores.

2. Objetivos do projeto

O presente projeto tem como objetivo desenvolver um sistema inteligente de monitoramento e identificação de níveis elevados de ruído em diferentes áreas do condomínio. Para isso, busca-se a implementação de sensores de nível sonoro estrategicamente distribuídos, permitindo a detecção automática de ruídos excessivos e a determinação aproximada da origem do som. Essa funcionalidade possibilita uma abordagem mais assertiva na resolução de possíveis incômodos causados por perturbação sonora.

Além da identificação da origem do som, o sistema será responsável por emitir alertas automáticos para informar síndicos ou responsáveis sobre a ocorrência de ruídos excessivos. Dessa forma, a gestão condominial se torna mais eficiente, uma vez que os dados e registros coletados poderão embasar decisões administrativas e facilitar a implementação de medidas corretivas.

A acessibilidade e a facilidade de implementação também são aspectos fundamentais do projeto. A solução proposta busca ser viável para condomínios

de diferentes portes, garantindo uma instalação simplificada e baixa necessidade de manutenção.

Outro diferencial do sistema é sua integração com câmeras de segurança, permitindo que, ao detectar níveis elevados de ruído, a gravação seja automaticamente ativada. Esse mecanismo proporciona um registro visual do ocorrido, auxiliando na análise precisa dos incidentes e na resolução de possíveis conflitos entre moradores.

Com a implementação desse sistema, espera-se promover maior tranquilidade e bem-estar aos residentes, garantindo um ambiente mais organizado e harmônico. A tecnologia embarcada permitirá uma abordagem proativa no controle da poluição sonora, assegurando registros confiáveis e auxiliando na tomada de decisões para uma gestão condominial mais eficaz.

3. Descrição do funcionamento

O sistema será composto por sensores do tipo decibelímetro, por meio da utilização de microfones e amplificadores de sinal, estrategicamente distribuídos pelo condomínio. Esses sensores são conectados a microcontroladores, como a Raspberry Pi Pico W ou ESP32, ambos equipados com conectividade WiFi para transmissão de dados.

Os microcontroladores serão responsáveis por processar os sinais captados pelos sensores, verificando se os níveis de ruído ultrapassam um limite predefinido por um período contínuo. Quando uma infração for detectada, os dados serão enviados para um servidor MQTT, que repassará as informações à central de monitoramento do condomínio via Wi-Fi.

Na central do condomínio, o síndico ou responsável terá acesso a um painel de consulta do ambiente, recebendo as informações em tempo real. O sistema também permitirá a integração com câmeras de segurança, que, além da gravação contínua, poderão destacar e armazenar automaticamente os trechos correspondentes ao momento da infração. Isso facilitará a obtenção de evidências visuais para embasar ações administrativas e garantir um controle mais eficiente das ocorrências.

Para garantir uma cobertura eficiente da rede de sensores, algumas estratégias podem ser adotadas, como por exemplo, a utilização de um sensor por residência, garantindo maior precisão na identificação do infrator, porém pode ter alto custo de implementação, dependendo do número de residências. Uma alternativa estratégica é utilizar um sensor para cobrir determinada área, como por exemplo, uma rua ou quadra do condomínio, realizando identificação da origem mais aproximada do ruído para que seja verificada. Uma interessante estratégia a ser utilizada em conjunto com alguma das anteriores é o uso de sensores para cobertura em espaços coletivos, como salões de festas, piscinas, entre outras áreas comuns, realizando registros precisos do local.

4. Justificativa

A perturbação sonora é uma das principais causas de conflitos em condomínios, gerando reclamações constantes e dificuldades na fiscalização por parte dos síndicos. O sistema proposto oferece uma solução automatizada, reduzindo a necessidade de intervenção humana e garantindo registros objetivos sobre as ocorrências.

Além disso, a implementação de um monitoramento eficaz pode contribuir para o cumprimento das normas de silêncio estabelecidas pelo regimento interno do condomínio e pela legislação vigente, promovendo um ambiente mais tranquilo e harmonioso para todos os moradores. A integração com câmeras de segurança potencializa a eficácia do sistema, proporcionando maior segurança e transparência na gestão dos eventos registrados.

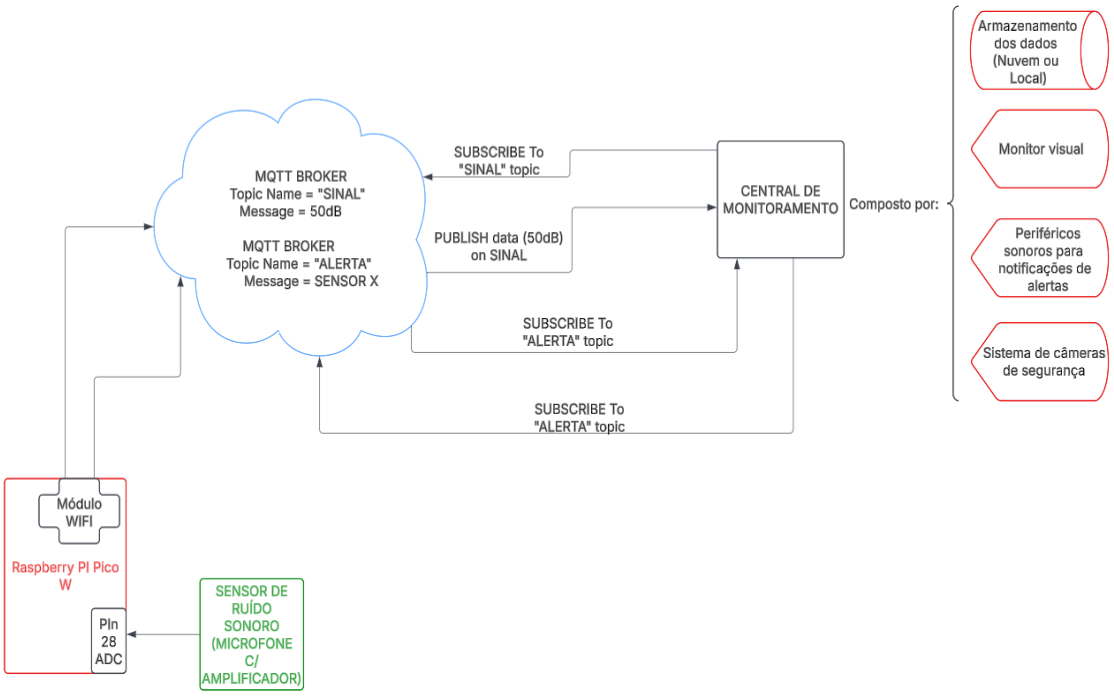
5. Originalidade

Embora existam sistemas de monitoramento de ruído disponíveis, a maioria se concentra em soluções industriais ou de grande escala. O diferencial deste projeto está na sua adaptação específica para condomínios, considerando questões como privacidade, custo-benefício e integração com sistemas de gestão condominial.

Uma pesquisa preliminar aponta que soluções comerciais, como medição de som, fiscalização policial para medir ruído de escapamentos de veículos, geralmente se baseiam em dispositivos individuais de medição de ruído, sem uma abordagem automatizada e inteligente, diferente do caso proposto para condomínios. O projeto busca preencher essa lacuna, oferecendo uma solução acessível, eficiente e personalizada para esse ambiente.

6. Especificação do hardware

6.1. Diagrama em bloco



6.2. Função de configuração de cada Bloco

Raspberry Pi Pico W (RP2040): Processa os dados recebidos do sensor de captação sonora (microfone) e decide se há violação dos níveis de ruído aceitáveis. Realiza controle de toda lógica do sistema, além do processamento de informações.

Microfone: Captura sinais sonoros e envia os dados para o microcontrolador, no nosso caso por meio do bloco ADC (Conversor Analógico-Digital). Em um

microfone de eletreto, precisamos de um circuito amplificador, como LM358, MAX9814 ou MCP602, para elevar o sinal a um nível adequado, logo precisamos utilizar um módulo amplificador do sinal. Temos outra opção também, que é utilizar um microfone com saída já amplificada, como por exemplo MAX9814.

Módulo Wi-Fi da placa: Uma das vantagens de se utilizar a Raspberry Pi Pico W é a utilização do seu módulo WiFi. Sua função aqui é enviar dados para o servidor MQTT, permitindo monitoramento remoto.

Servidor MQTT: Responsável pela comunicação dos dados com a central de monitoramento.

Central de monitoramento: Engloba todo monitoramento do sistema de sensores pelo condomínio, recebendo informações de forma visual através das câmeras de segurança por meio de monitores, além de periféricos sonoros como caixas de som para alertar sobre o acontecimento de incidentes de perturbação em algum dos sensores. Além do mais, existem um bloco de armazenamento que pode ser de forma local ou em nuvem, registrando os incidentes e níveis de ruído captados, além da gravação do trecho do incidente.

6.3. Configuração de cada bloco

Microfone: Configurado através do bloco de conversor analógico-digital disponível na placa, para capturar áudio em uma taxa de amostragem adequada para análise de ruído.

RP2040: Configurado para receber dados, processá-los e enviar as informações e alertas via módulo Wi-Fi da placa.

WiFi - Configurado para conexão ao servidor MQTT para envio de alertas.

Central de monitoramento – Infraestrutura equipada e configurada com periféricos e comunicação para monitoramento do sistema.

6.4. Comandos e registros utilizados

Microfone: Uso de biblioteca da Raspberry PI Pico para configuração e captura dos sinais por meio do ADC, além das pinagem correta na placa:

```
#include "hardware/adc.h"

#define MIC_PIN 28

adc_init();

adc_gpio_init(MIC_PIN);
```

Wi-Fi: Uso de biblioteca para utilização do módulo WiFi da placa Raspberry PI Pico W, e realizar configurações da rede:

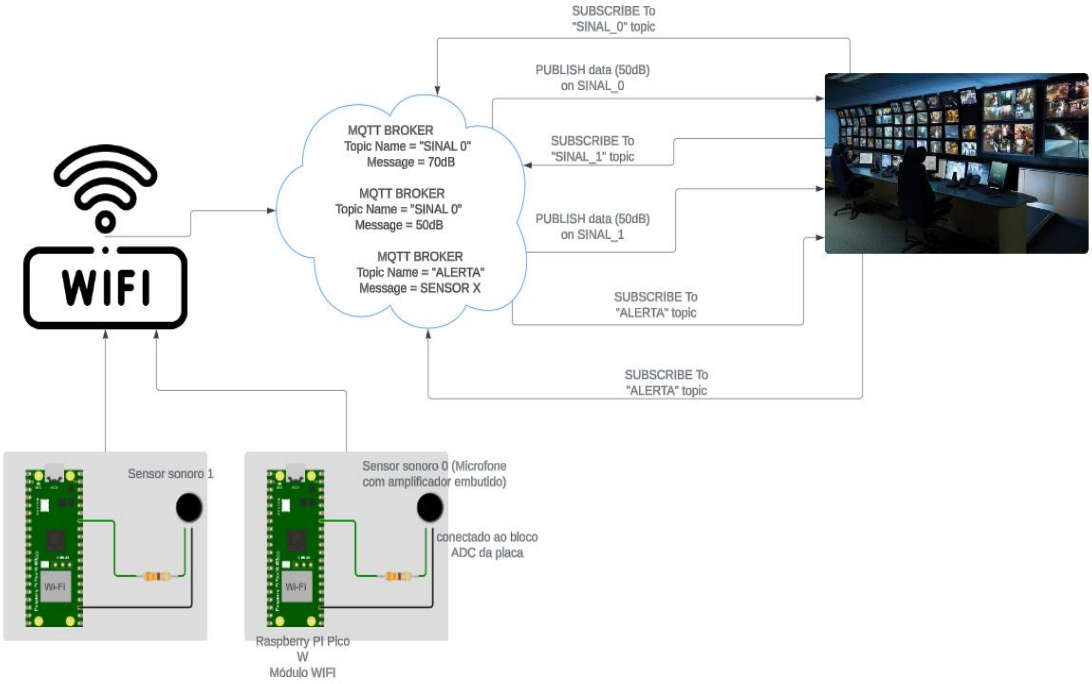
```
#include "pico/cyw43_arch.h"

#define WIFI_SSID "NomeDaRedeWiFi"

#define WIFI_PASS "SenhaDaRedeWiFi"
```

Comunicação via MQTT: Configuração de comandos de publicação e subscrição.

6.5. Descrição da pinagem

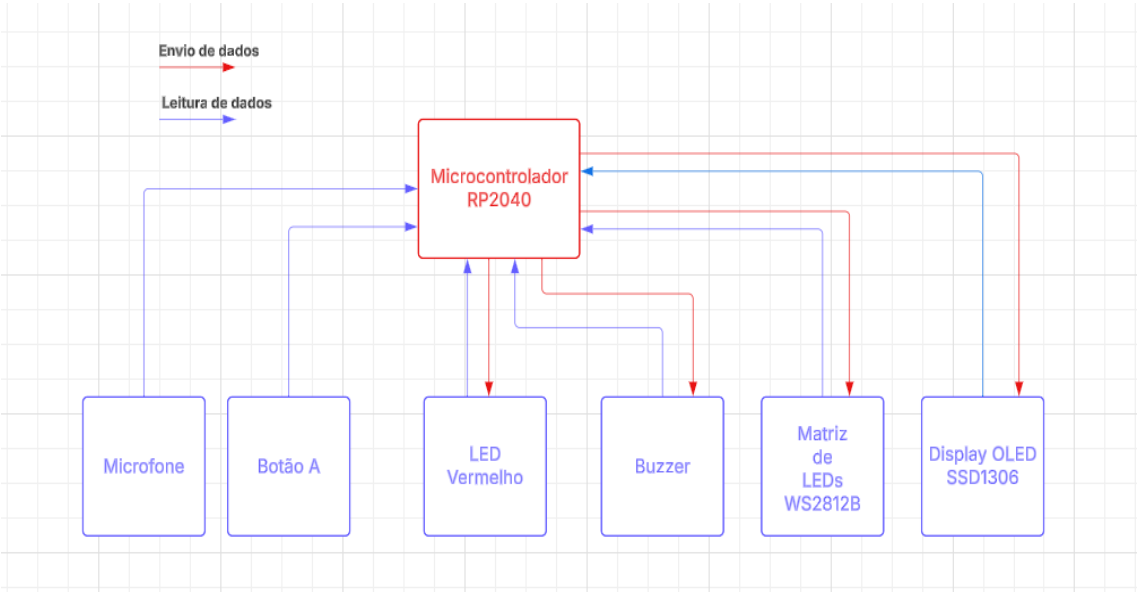
Componente	**Pino do Microcontrolador**	**Função**
Microfone	GPIO 28	Captação de sinais sonoros.

6.6. Circuito completo do hardware



7. Especificação do hardware simulando na BitDogLab

7.1. Diagrama em bloco



7.2. Função e configuração de cada bloco

Microcontrolador RP2040 – Realiza controle de toda lógica do sistema, além do processamento de informações, interligando também a comunicação de

todos os periféricos da placa. Programado em C, configurado para leituras ADC do microfone, uso de PWM para controle de LED, e envio de dados ao display.

Microfone - Captura sinais sonoros e envia os dados para o microcontrolador, no nosso caso por meio do bloco ADC (Conversor Analógico-Digital), configurado para leitura via ADC, com limiar predefinido. Simula um sensor do tipo decibelímetro no projeto, realizando leituras de sinais sonoros.

Matriz de Led (WS2812B) – Demonstra informações visuais por meio do controle dos leds dispostos em uma matriz 5x5, sendo controlado por meio de uma máquina PIO que realiza as demandas de configuração e envio de dados de forma independente do processador. Simula qual sensor está sendo monitorado, no caso, mostra números de 0 a 9, representando endereços de ruas do condomínio.

Buzzer – Emite notificação por meio de alertas sonoros quando a leitura do sinal sonoro do microfone ultrapassa o limite pré-definido no código. Controlado via PWM (Pulse Width Modulation), ajustando a intensidade e frequência do som, emitindo bipes intercalados quando ativado. Simula um alerta sonoro na central de monitoramento, notificando uma ocorrência de perturbação registrada pelo sensor.

Botão A – Permite interação do usuário com o sistema, mapeados para GPIOs. Botão A simula qual sensor está sendo monitorado, incrementando os endereços representados de forma numérica.

LED vermelho – Permite integração com o sistema para determinadas situações, mapeado e controlado para GPIO. Simula a gravação do trecho do incidente quando ativado, registrando a ocorrência de forma visual.

Display OLED 128x64 - Mostra informações sobre o funcionamento e status do sistema, simulando o monitoramento em tempo real do condomínio, como visualização de câmeras de segurança. Utiliza comunicação I2C para exibição das informações de forma gráfica.

7.3. Comandos e registros utilizados

Matriz de Leds (WS2812B):

Definimos a pinagem da matriz de leds na nossa placa: #define WS2812_PIN 7

Além de incluir nossa máquina PIO: #include "ws2812.pio.h"

Configuramos a máquina PIO para controle:

```
ws2812.pio
.pio_version 0 // only requires PIO version 0

.program ws2812
.side_set 1

.define public T1 3
.define public T2 3
.define public T3 4

.wrap_target
bitloop:
    out x, 1       side 0 [T3 - 1]
    jmp !x do_zero side 1 [T1 - 1]
do_one:
    jmp  bitloop   side 1 [T2 - 1]
do_zero:
    nop            side 0 [T2 - 1]
.wrap


% c-sdk {
#include "hardware/clocks.h"

static inline void ws2812_program_init(PIO pio, uint sm, uint offset, uint pin, float freq, bool rgbw) {

    pio_gpio_init(pio, pin);
    pio_sm_set_consecutive_pindirs(pio, sm, pin, 1, true);

    pio_sm_config c = ws2812_program_get_default_config(offset);
    sm_config_set_sideset_pins(&c, pin);
    sm_config_set_out_shift(&c, false, true, rgbw ? 32 : 24);
    sm_config_set_fifo_join(&c, PIO_FIFO_JOIN_TX);
```

```
    int cycles_per_bit = ws2812_T1 + ws2812_T2 + ws2812_T3;
    float div = clock_get_hz(clk_sys) / (freq * cycles_per_bit);
    sm_config_set_clkdiv(&c, div);

    pio_sm_init(pio, sm, offset, &c);
    pio_sm_set_enabled(pio, sm, true);
}
%}
```

No nosso código, inicializamos a máquina PIO:

```
//Inicializa o pio
PIO pio = pio0;
int sm = 0;
uint offset = pio_add_program(pio, &ws2812_program);
ws2812_program_init(pio, sm, offset, WS2812_PIN, 800000, IS_RGBW);
```

E dessa forma, conseguimos enviar dados e controlar os leds da matriz, com auxílio de funções específicas no código.

Microfone:

Incluímos nossa biblioteca do SDK da Raspberry PI Pico:

#include "hardware/pio.h"

Definimos o pino utilizado pelo microfone:

```
#define MIC_PIN 28
```

Inicializamos o ADC no pino GPIO do microfone:

```
//Inicializa o ADC
adc_init();
adc_gpio_init(MIC_PIN);  //Configura GPIO28 como entrada ADC para o microfone
```

Lemos o canal 2, que representa o microfone, e com isso podemos realizar a captura dos sinais:

```
// Lê o valor do microfone (canal 2)
adc_select_input(2);
uint16_t valor_microfone = adc_read();
```

Buzzer:

Definimos o pino para o Buzzer:

```
#define BUZZER_PIN 21
```

Realizamos a configuração via PWM:

```
gpio_set_function(BUZZER_PIN, GPIO_FUNC_PWM);   //Define o pino do buzzer como PWM
uint slice_num = pwm_gpio_to_slice_num(BUZZER_PIN); //Pega o slice do pino do buzzer
//Calcula o divisor do clock para atingir a frequência desejada
float clock_div = 125.0;  // lock divisor ajustável
uint16_t wrap_value = (125000000 / (clock_div * BUZZER_FREQ)) - 1; //125 MHz é o clock base
pwm_set_clkdiv(slice_num, clock_div);   //Define o clock divisor
pwm_set_wrap(slice_num, wrap_value);    //Define o valor de wrap
pwm_set_gpio_level(BUZZER_PIN, wrap_value / 2); //50% do ciclo de trabalho (onda quadrada)
```

Botão A:

Define a pinagem para o botão:

```
#define BOTAO_A 5       //Pino do botão A
```

Inicializa-se a GPIO do botão como entrada, e define como botão de pull-up:

```
//Inicializa pino do botao
gpio_init(BOTAO_A);
gpio_set_dir(BOTAO_A, GPIO_IN);
gpio_pull_up(BOTAO_A);
```

LED Vermelho:

Define o pino do LED vermelho na placa:

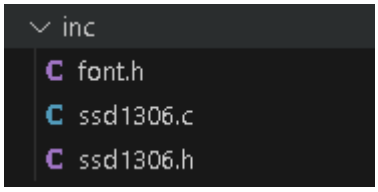
```
#define LED_PIN_RED 13
```

Inicializamos o LED como GPIO de saída:

```
    //Inicializa pino do led
    gpio_init(LED_PIN_RED);
    gpio_set_dir(LED_PIN_RED, GPIO_OUT);
    gpio_put(LED_PIN_RED, 0);
```

Display OLED 128x64:

Deve ser avisado que utilizamos bibliotecas com funções implementadas para o envio de dados para o display:



Definimos as portas e conexões do display na placa:

```
#define I2C_PORT i2c1
#define I2C_SDA 14
#define I2C_SCL 15
#define endereco 0x3C
```

Para a inicialização:

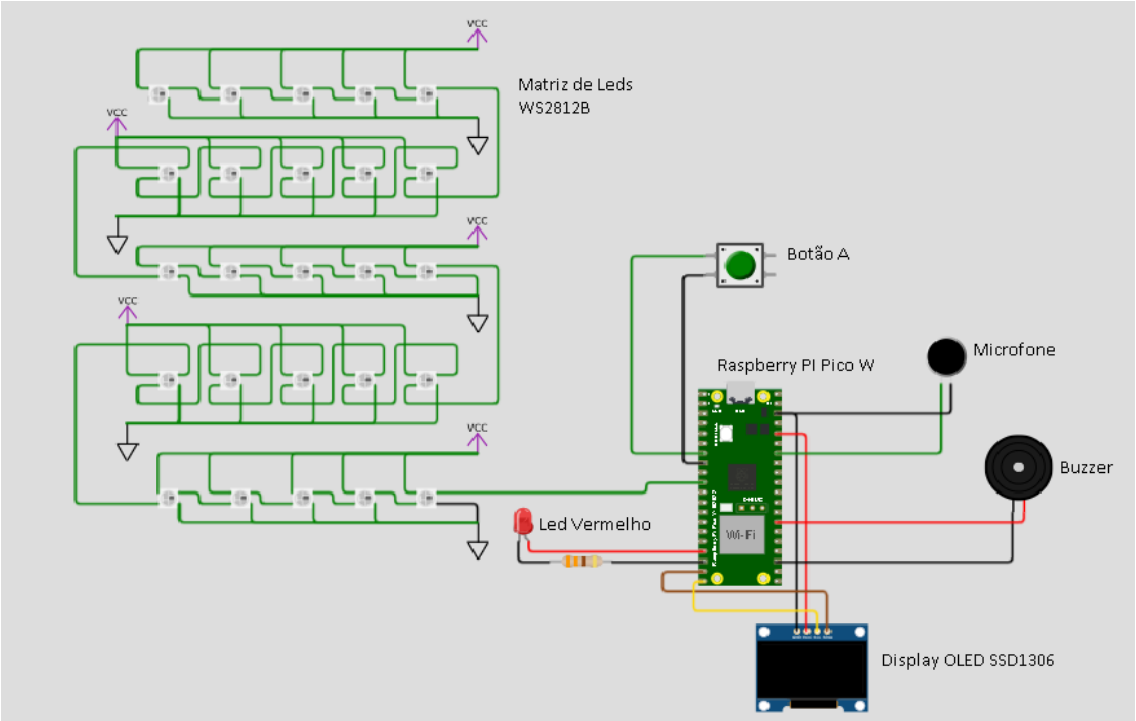
```
  //I2C IInicialização. Usando 400Khz.
  i2c_init(I2C_PORT, 400 * 1000);
  gpio_set_function(I2C_SDA, GPIO_FUNC_I2C); //Envio do pino para I2C SDA (dados)
  gpio_set_function(I2C_SCL, GPIO_FUNC_I2C); //Envio do pino para I2C SCL (clock)
  gpio_pull_up(I2C_SDA); // Pull up para data line
  gpio_pull_up(I2C_SCL); // Pull up para clock line
  ssd1306_init(&ssd, WIDTH, HEIGHT, false, endereco, I2C_PORT);
  ssd1306_config(&ssd);
  ssd1306_send_data(&ssd);
  ssd1306_fill(&ssd, false);  //Limpa o display
  ssd1306_send_data(&ssd);
```

7.4. Descrição da pinagem

Componente	Pino do Microcontrolador	Função
Matriz WS2812B	GPIO 7	Controle da matriz de LEDs

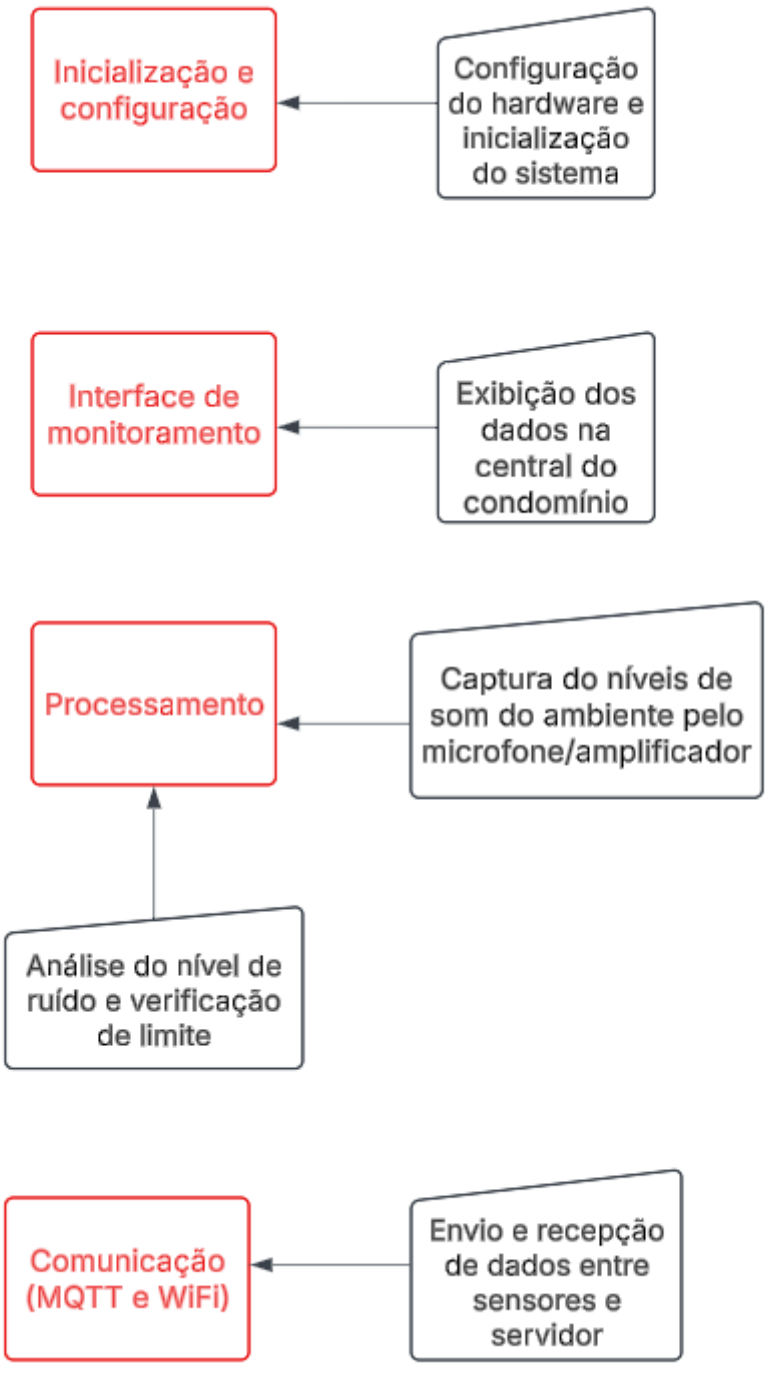
Componente	Pino do Microcontrolador	Função
Display SSD1306	GPIO 14 (SDA), GPIO 15 (SCL)	Interface I2C
Microfone	GPIO 28	Entrada de som
Buzzer	GPIO 21	Saída de som
LED vermelho	GPIO 13	Ativa led vermelho, indica gravação
Botão A	GPIO 5	Entrada para alterar o sensor

7.5. Circuito completo do hardware



8. Especificação do firmware

8.1. Blocos Funcionais



8.2. Descrição das Funcionalidades

Inicialização e Configuração: Configura os microcontroladores, sensores e conexões.

Interface de Monitoramento: Painel de interface web ou aplicativo que exibe os dados do nível de ruído em tempo real.

Comunicação: Utiliza WiFi para transmitir dados via MQTT para um servidor central.

Processamento de Dados: O firmware analisa os valores recebidos do sensor e verifica se o ruído ultrapassa um limite definido. Leitura dos sensores: O microfone e o amplificador captam o som ambiente, convertendo-o em sinal digital.

8.3. Definição das Variáveis

limite_ruido: Define o nível máximo de decibéis permitido antes de gerar uma notificação.

nivel_ruido: Armazena o nível atual de ruído captado pelo sensor.

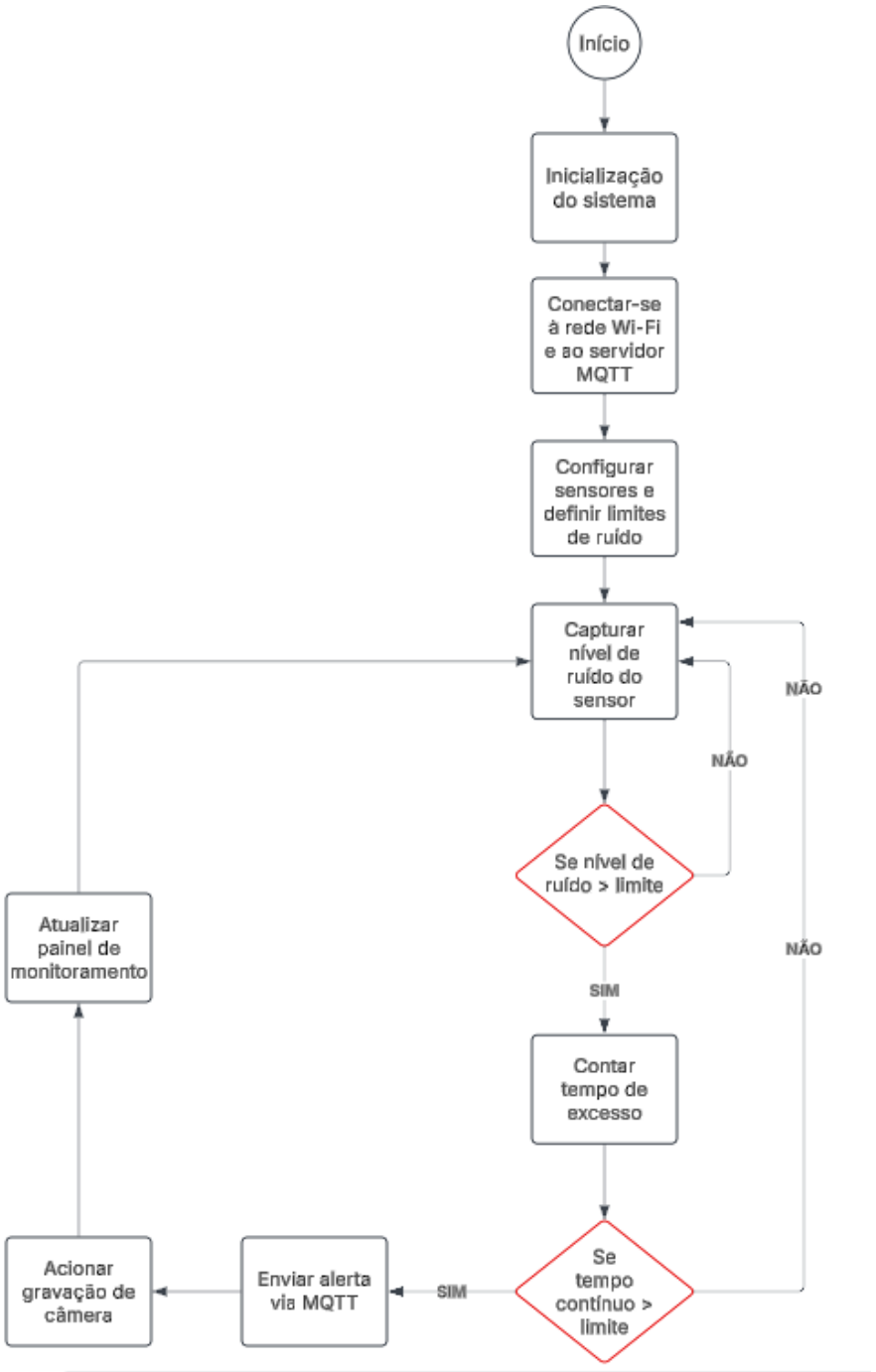
tempo_excedente: Conta o tempo durante o qual o ruído está acima do limite.

wifi_ssid, wifi_password: Configurações de conexão Wi-Fi.

mqtt_server, mqtt_topic: Definem o servidor MQTT e o tópico para envio de dados.

status_alerta: Indica se uma notificação foi enviada.

8.4. Fluxograma do Software



8.5. Inicialização do Software

1. Configuração da conexão Wi-Fi e servidor MQTT.

2. Definição dos parâmetros do sensor e limites de ruído.

3. Configuração dos pinos do microcontrolador para leitura do microfone.

4. Início da coleta de dados e processamento em tempo real.

8.6. Configuração dos Registros

- Wi-Fi: Configuração da conexão via WiFi passando o usuário e senha da rede (ssid, password).

- MQTT: Conexão e assinatura do tópico de envio.

- GPIOs: Definição dos pinos de entrada para leitura do sensor.

8.7. Estrutura e Formato dos Dados

Os dados são enviados no seguinte formato JSON via MQTT:

```
{
  "sensor_id": "001",
  "nivel_ruido": 78.5,
  "limite_ruido": 75,
  "tempo_excedente": 10,
  "status_alerta": "ativo"
}
```

8.8. Protocolo de Comunicação

- Protocolo: MQTT sobre TCP/IP.

- Servidor: MQTT Broker (pode ser local ou em nuvem).

- Tópico: /condominio/ruido/sensores.

8.9. Formato do Pacote de Dados

Cada mensagem enviada via MQTT segue o formato JSON descrito acima, garantindo compatibilidade e integração com outros sistemas.

9. Especificação do firmware na BitDogLab

9.1. Simulação na placa BitDogLab:

Com o uso do microfone conectado ao microcontrolador da BitDogLab, será realizado a captação dos sons do ambiente. O sinal analógico será convertido via ADC (Conversor Analógico-Digital) e analisado para determinar se o nível sonoro excede um limiar pré-definido, simulando um limite de decibéis.

Ação do Sistema:

- Caso o ruído ultrapasse o nível aceitável por um tempo contínuo, um alerta será ativado. Um Buzzer emitirá um sinal sonoro para alertar, utilizando PWM para controle do som, simulando um alerta sonoro notificando os responsáveis na central.

- Na nossa simulação, definimos que exista um sensor por rua do condomínio. A matriz de led será acionada via PIO mostrando números de 0 a 9, supondo que cada rua tenha endereço de forma enumerado, simulando qual sensor está sendo monitorado.

- Ultrapassando o limite predefinido, será simulado a ativação da câmera de maneira inteligente, focando onde o ruido está vindo por meio de envio de mensagens via comunicação serial USB, e a ativação do led vermelho, simulando que está sendo feita a gravação do trecho do incidente para registro, garantindo análise visual e auditiva, além de preservar a segurança do responsável do condomínio para tomar as devidas providências, com consciência prévia do que está acontecendo.

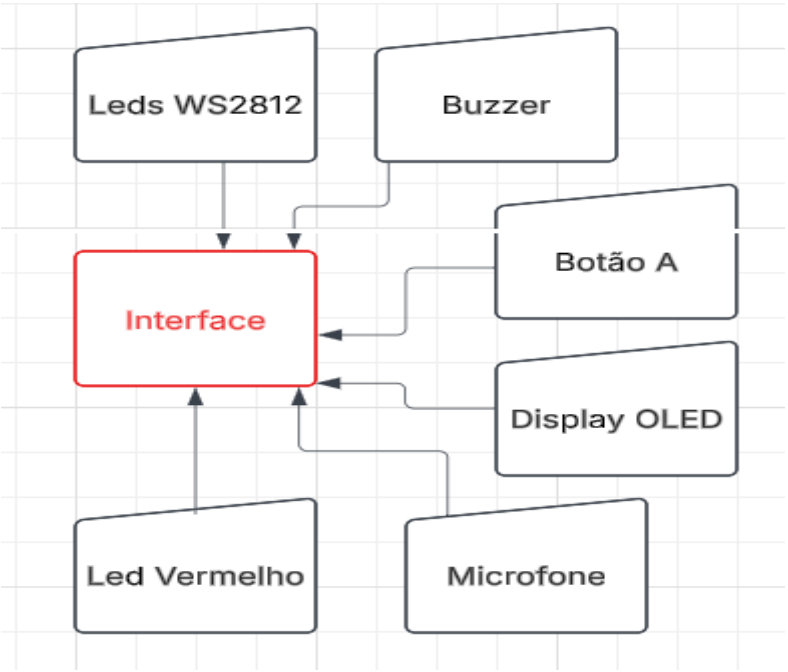
- Já que estamos simulando, o botão A incrementa o valor enumerado que identifica os sensores (no caso de 0 a 9), simulando a leitura do ruido pelo microfone naquele sensor.

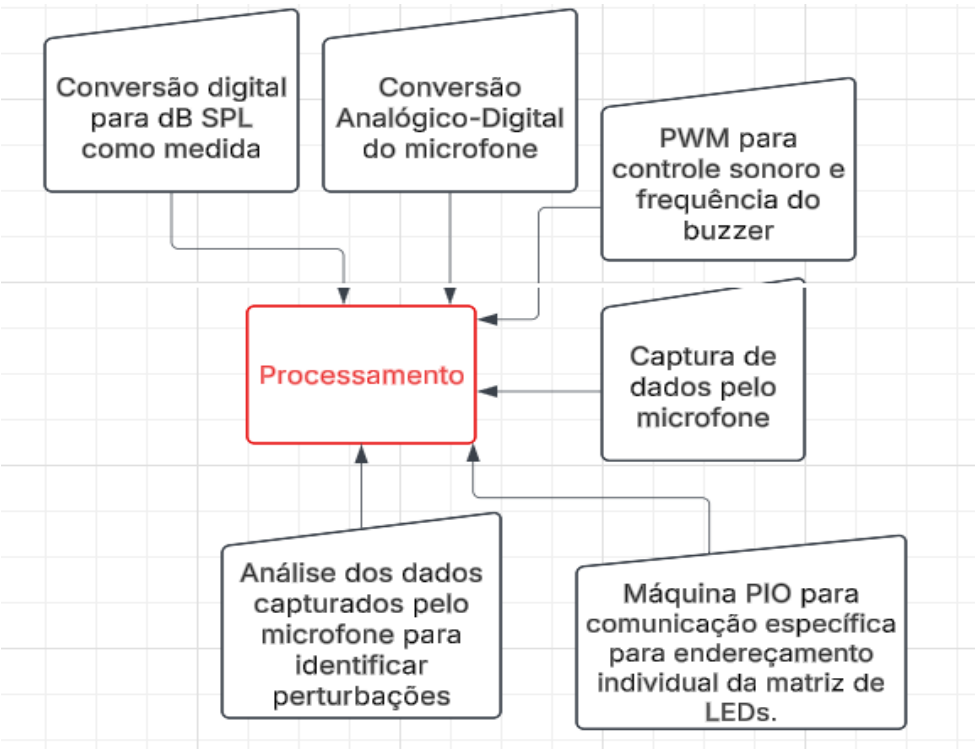
- O display OLED SSD1306 da placa BitDogLab irá simular o monitor da central do condomínio, informando a localização do incidente e vigilância das câmeras.

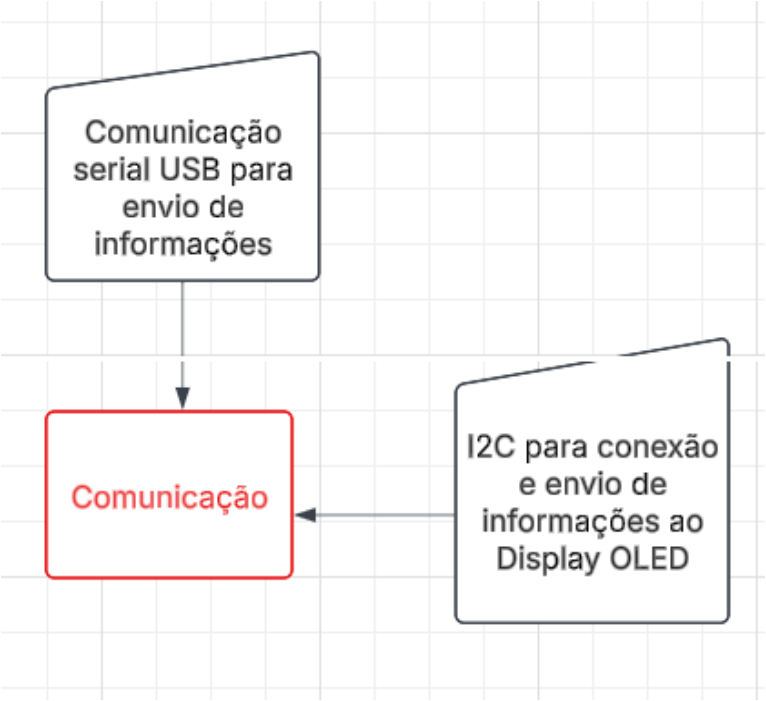
- Após encerramento do incidente, onde a leitura do sinal pelo microfone esteja abaixo do limiar predefinido, em um tempo continuo garantindo encerramento do registro, a comunicação serial USB enviará uma mensagem informando que o

registro foi salvo, simulando o armazenamento dos dados do incidente na central do condomínio.

9.2. Blocos funcionais







9.3. Descrição das funcionalidades dos blocos

Interface: Fornece feedbacks sonoros e visuais através dos Leds, display e buzzer, além de interação com o usuário por meio do botão A da placa.

Processamento: Toda parte de captura, cálculos e análises dos sinais. Os sinais do microfone são capturados, convertidos de analógico para digital, e ainda realizamos cálculo para deixar em formato dB SPL (decibéis). Além disso, realizamos a análise dos dados obtidos para verificar se há excesso comparado ao valor do limiar predefinido. Também controlamos o buzzer por meio do PWM, efetuando os cálculos de Duty Cycle, e garantido 50% de intensidade. Em relação ao controle da matriz de Leds, configuramos uma máquina PIO para auxiliar no gerenciamento da matriz, trabalhando de forma independente.

Comunicação: Utilizamos comunicação I2C para conectar o nosso display OLED SSD1306 e enviar as informações de forma visual sobre as medições de sinal pelo microfone. Adicionamos também uma comunicação serial via USB para simular o estado do sistema.

9.4. Definição das variáveis

Uma das variáveis mais importante é a limiar, qual define que faixa de valor é considerado um valor de ruído elevado, perturbação, ou algo normal. Definimos como 3000, o que equivale a 80dB SPL para realizar nossos testes:

```
const uint limiar = 3000;     //predefinição do Limiar -> aproximadamente 80dB SPL
```

Temos também nossas variáveis de controle de tempo ou estado dos periféricos, como o buzzer, e o endereço do sensor, simulado pela matriz de Leds mostrando graficamente endereços enumerados (variável ‘numero’).

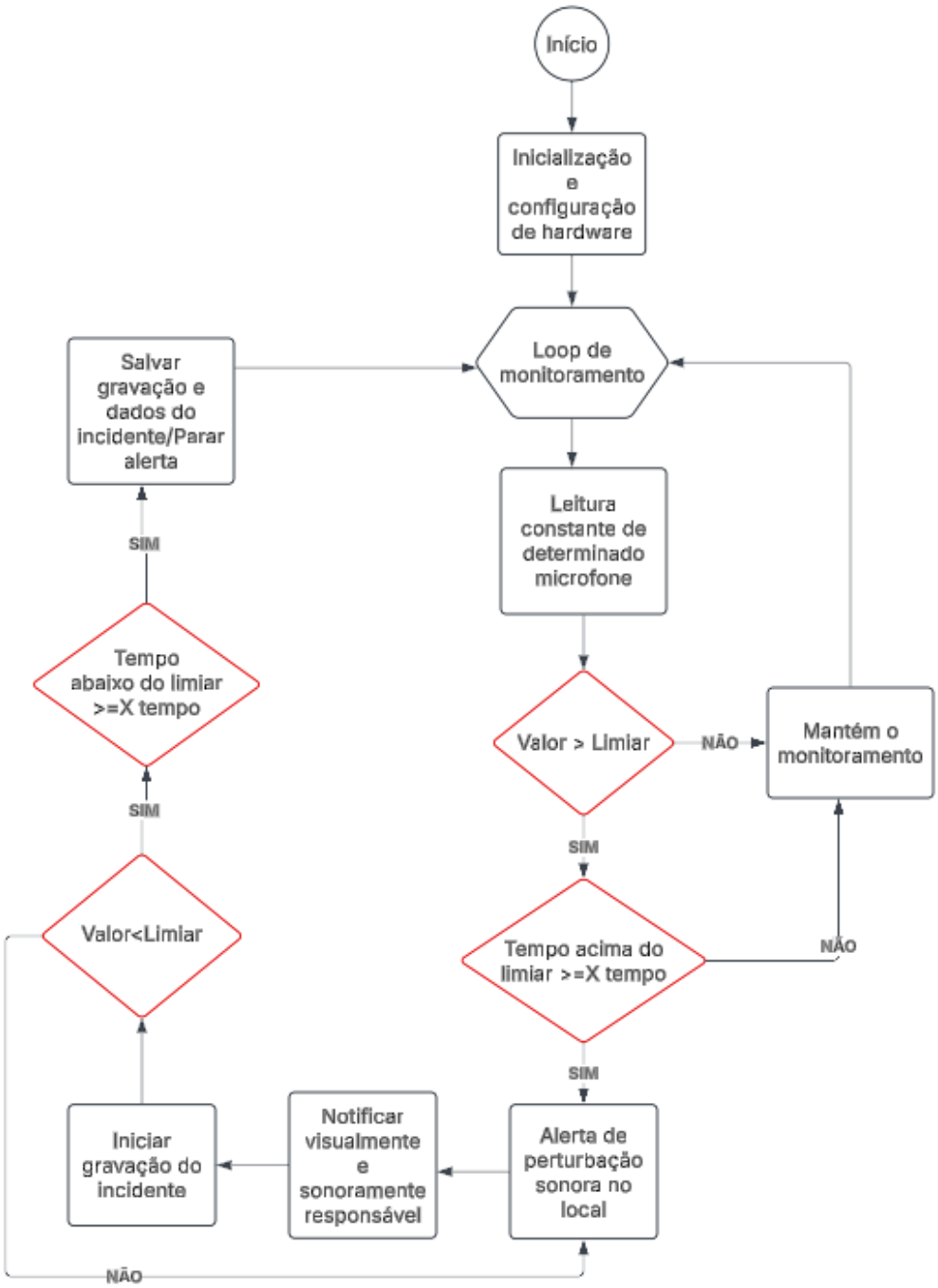
```
uint32_t volatile tempo_real = 0;       //Guarda o tempo real do ruído
uint32_t volatile silencio_tempo = 0;   //Guarda o tempo do silencio
bool volatile alerta_sonoro = false;    //Controle do estado do buzzer
uint volatile numero = 0;               //Variável para inicializar o numero com 0, indicando a rua 0 (WS2812B)
uint32_t volatile tempo_espera = 0;     //Guarda o tempo que começou a espera de 30s apos clicar no botão B
```

```
bool buzzer_ligado = false; //Controle para alternar os bipes
```

E claro, não podemos esquecer da nossa variável que captura os sinais do microfone:

```
    uint16_t valor_microfone = adc_read();
```

9.5. Fluxograma



9.6. Inicialização

O processo de inicialização do firmware envolve:

Definição das bibliotecas:

```c
#include <stdio.h>              //Biblioteca padrão C
#include <math.h>               //Biblioteca funções matemáticas
#include "pico/stdlib.h"        //Biblioteca padrão Pico
#include "hardware/i2c.h"       //Biblioteca I2C
#include "hardware/adc.h"       //Biblioteca ADC
#include "hardware/pio.h"       //Biblioteca PIO
#include "hardware/timer.h"     //Biblioteca Timer
#include "hardware/clocks.h"    //Biblioteca Clock
#include "hardware/pwm.h"       //Biblioteca PWM
#include "ws2812.pio.h"         //Incluir nossa maquina PIO para a matriz de leds
#include "inc/ssd1306.h"        //Incluir nossa biblioteca para controle e envio de dados para o display
#include "inc/font.h"           //Incluir nossa biblioteca que armazena os numeros, e letras para o display
```

Definição das constantes com pinagem dos periféricos, valor do limiar e configurações padrões a serem utilizadas:

```c
#define VREF 3.3          //Tensão de referência do ADC
#define SPL_MAX 120       //SPL máximo do microfone a 3,3V
#define ADC_MAX 4095      //Resolução do ADC
#define I2C_PORT i2c1     //I2C1
#define I2C_SDA 14        //Pino SDA (dados)
#define I2C_SCL 15        //Pino SCL (clock)
#define endereco 0x3C     //Endereço do display
#define IS_RGBW false     //Maquina PIO para RGBW
#define NUM_PIXELS 25     //Quantidade de LEDs na matriz
#define NUM_NUMBERS 11    //Quantidade de numeros na matriz
#define BOTAO_A 5         //Pino do botão A
#define WS2812_PIN 7      //Pino do WS2812
#define LED_PIN_RED 13    //Pino do led vermelho
#define MIC_PIN 28        //Pino do microfone
#define BUZZER_PIN 21     //Pino do buzzer
#define BUZZER_FREQ 2000 //Frequência do som em Hz
const uint limiar = 3000;     //predefinição do Limiar -> aproximadamente 80dB SPL
```

Definição de variáveis globais para controle do programa:

```c
uint32_t volatile tempo_real = 0;       //Guarda o tempo real do ruído
uint32_t volatile silencio_tempo = 0;   //Guarda o tempo do silencio
bool volatile alerta_sonoro = false;    //Controle do estado do buzzer
uint volatile numero = 0;               //Variável para inicializar o numero com 0, indicando a rua 0 (WS2812B)
uint32_t volatile tempo_espera = 0;     //Guarda o tempo que começou a espera de 30s apos clicar no botão B
```

Algumas funções para configuração da matriz de Leds, além da matriz que contém os desenhos dos números que serão utilizados na matriz de leds:

```
bool led_numeros[NUM_NUMBERS][NUM_PIXELS] = {
    //Número 0
    {
    0, 1, 1, 1, 0,
    0, 1, 0, 1, 0,
    0, 1, 0, 1, 0,
    0, 1, 0, 1, 0,
    0, 1, 1, 1, 0
    },

    //Número 1
    {0, 1, 1, 1, 0,
    0, 0, 1, 0, 0,
    0, 0, 1, 0, 0,
    0, 1, 1, 0, 0,
    0, 0, 1, 0, 0
    },

    //Número 2
    {0, 1, 1, 1, 0,
    0, 1, 0, 0, 0,
    0, 1, 1, 1, 0,
    0, 0, 0, 1, 0,
    0, 1, 1, 1, 0
    },
```

```
    //Número 3
    {0, 1, 1, 1, 0,
    0, 0, 0, 1, 0,
    0, 1, 1, 1, 0,
    0, 0, 0, 1, 0,
    0, 1, 1, 1, 0
    },

    //Número 4
    {0, 1, 0, 0, 0,
    0, 0, 0, 1, 0,
    0, 1, 1, 1, 0,
    0, 1, 0, 1, 0,
    0, 1, 0, 1, 0
    },

    //Número 5
    {0, 1, 1, 1, 0,
    0, 0, 0, 1, 0,
    0, 1, 1, 1, 0,
    0, 1, 0, 0, 0,
    0, 1, 1, 1, 0
    },
```

```
  //Número 6
  {0, 1, 1, 1, 0,
  0, 1, 0, 1, 0,
  0, 1, 1, 1, 0,
  0, 1, 0, 0, 0,
  0, 1, 1, 1, 0
  },

  //Número 7
  {0, 1, 0, 0, 0,
  0, 0, 0, 1, 0,
  0, 1, 0, 0, 0,
  0, 0, 0, 1, 0,
  0, 1, 1, 1, 0
  },

  //Número 8
  {0, 1, 1, 1, 0,
  0, 1, 0, 1, 0,
  0, 1, 1, 1, 0,
  0, 1, 0, 1, 0,
  0, 1, 1, 1, 0
  },
```

```
    //Número 9
    {0, 1, 1, 1, 0,
    0, 0, 0, 1, 0,
    0, 1, 1, 1, 0,
    0, 1, 0, 1, 0,
    0, 1, 1, 1, 0
    },

    //APAGAR OS LEDS, representado pelo número (posição) 10
    {0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0
    }
};
```

```c
//Variável global para armazenar a cor (Entre 0 e 255 para intensidade)
uint8_t led_r = 20; //Intensidade do vermelho
uint8_t led_g = 0; //Intensidade do verde
uint8_t led_b = 0; //Intensidade do azul

//Função para ligar um LED
Codeium: Refactor | Explain | X
static inline void put_pixel(uint32_t pixel_grb){
    pio_sm_put_blocking(pio0, 0, pixel_grb << 8u);
}

//Função para converter cores RGB para um valor de 32 bits
Codeium: Refactor | Explain | X
static inline 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);
}
```

```c
//Função para envio dos dados para a matriz de leds
Codeium: Refactor | Explain | X
void set_one_led(uint8_t r, uint8_t g, uint8_t b, int numero){
    //Define a cor com base nos parâmetros fornecidos
    uint32_t color = urgb_u32(r, g, b);

    //Define todos os LEDs com a cor especificada
    for(int i = 0; i < NUM_PIXELS; i++){
        if(led_numeros[numero][i]){     //Chama a matriz de leds com base no numero passado
            put_pixel(color);           //Liga o LED com um no buffer
        }else{
            put_pixel(0);               //Desliga os LEDs com zero no buffer
        }
    }
}
```

Inicialização e configuração dos pinos e máquina de estado:

- Configurando o Led vermelho como saída;

- Botão A como entrada e ativando o resistor de pull-up;

- Inicializamos a máquina PIO para controle da matriz de Leds WS2812B

- Inicializamos o display OLED SSD1306 por meio do protocolo I2C, utilizando a estrutura ‘ssd’, definindo o Data Line, Clock Line e endereço do display;

- Configuramos o microfone utilizando a inicialização do bloco ADC no pino.

- Configuração do buzzer por meio da utilização do PWM, definindo o slice, wrap, e frequência, assim pode-se configurá-lo com um ciclo de trabalho de 50% de intensidade.

```c
ssd1306_t ssd; //Inicializa a estrutura do display

//Função para inicializar as pinagens, e toda configuração de comunicação com os periféricos
Codeium: Refactor | Explain | X
void inicializar_GPIOs(){
    stdio_init_all();

    //Inicializa pino do led
    gpio_init(LED_PIN_RED);
    gpio_set_dir(LED_PIN_RED, GPIO_OUT);
    gpio_put(LED_PIN_RED, 0);

    //Inicializa pino do botao
    gpio_init(BOTAO_A);
    gpio_set_dir(BOTAO_A, GPIO_IN);
    gpio_pull_up(BOTAO_A);

    //Inicializa o pio
    PIO pio = pio0;
    int sm = 0;
    uint offset = pio_add_program(pio, &ws2812_program);
    ws2812_program_init(pio, sm, offset, WS2812_PIN, 800000, IS_RGBW);
```

```c
    //I2C IInicialização. Usando 400Khz.
    i2c_init(I2C_PORT, 400 * 1000);
    gpio_set_function(I2C_SDA, GPIO_FUNC_I2C); //Envio do pino para I2C SDA (dados)
    gpio_set_function(I2C_SCL, GPIO_FUNC_I2C); //Envio do pino para I2C SCL (clock)
    gpio_pull_up(I2C_SDA); // Pull up para data line
    gpio_pull_up(I2C_SCL); // Pull up para clock line
    ssd1306_init(&ssd, WIDTH, HEIGHT, false, endereco, I2C_PORT);
    ssd1306_config(&ssd);
    ssd1306_send_data(&ssd);
    ssd1306_fill(&ssd, false);  //Limpa o display
    ssd1306_send_data(&ssd);

    //Inicializa o ADC
    adc_init();
    adc_gpio_init(MIC_PIN);  //Configura GPIO28 como entrada ADC para o microfone

    gpio_set_function(BUZZER_PIN, GPIO_FUNC_PWM);   //Define o pino do buzzer como PWM
    uint slice_num = pwm_gpio_to_slice_num(BUZZER_PIN); //Pega o slice do pino do buzzer
    //Calcula o divisor do clock para atingir a frequência desejada
    float clock_div = 125.0;  //Clock divisor ajustável
    uint16_t wrap_value = (125000000 / (clock_div * BUZZER_FREQ)) - 1; //125 MHz é o clock base
    pwm_set_clkdiv(slice_num, clock_div);   //Define o clock divisor
    pwm_set_wrap(slice_num, wrap_value);    //Define o valor de wrap
    pwm_set_gpio_level(BUZZER_PIN, wrap_value / 2); //50% do ciclo de trabalho (onda quadrada)
}
```

Funções para ligar/desligar o buzzer por meio de PWM:

- Habilita ou desabilita o buzzer por meio da função ‘pwm_set_enabled(slice_num, true/false)’

```c
//Função para iniciar o buzzer por meio de PWM
void iniciar_buzzer(){
    uint slice_num = pwm_gpio_to_slice_num(BUZZER_PIN);
    pwm_set_enabled(slice_num, true);
    //Aguarda 1 segundo para manter o som ativo
    sleep_ms(1000);

    //Desativa o PWM
    pwm_set_enabled(slice_num, false);
    //Aguarda 2 segundos antes de poder tocar novamente
    sleep_ms(2000);
}

//Desliga o buzzer
void parar_buzzer(){
    uint slice_num = pwm_gpio_to_slice_num(BUZZER_PIN);
    pwm_set_enabled(slice_num, false);
}
```

Função para leitura do microfone e configuração de ações do sistema com base na leitura:

```c
bool buzzer_ligado = false; //Controle para alternar os bipes

//Função para ler o valor do microfone e controlar ações do sistema
Codeium: Refactor | Explain | X
uint16_t ler_microfone(){
    //Lê o valor do microfone (canal 2)
    adc_select_input(2);
    uint16_t valor_microfone = adc_read();
    uint32_t periodo = to_ms_since_boot(get_absolute_time());

    //Controle de perturbação no andar, considerando 3 segundos de tolerância
    if(valor_microfone > limiar){
        if(!alerta_sonoro){
            // Se não há alerta e o ruído começou agora, registra o tempo inicial
            tempo_real = periodo;
            alerta_sonoro = true;
        }

        //Se o ruído continua acima do limiar e já passaram 3 segundos desde o início do ruído
        if((periodo - tempo_real) >= 3000){
            char bufferRua[20];
            ssd1306_fill(&ssd, false);                          //Limpa o display antes de escrever
            ssd1306_draw_string(&ssd, "VIGILANCIA", 30, 10);    //Escreve no display

            sprintf(bufferRua, "Rua %d", numero);               //Converte o numero da rua para string
            ssd1306_draw_string(&ssd, bufferRua, 10, 30);       //Escreve no display
            ssd1306_draw_string(&ssd, "ALERTA RUIDO", 10, 50);  //Escreve no display
            ssd1306_send_data(&ssd);                            //Atualiza o display

            printf("Perturbação no andar %d\nGRAVANDO...\n\n", numero);
```

```
            printf("Perturbação no andar %d\nGRAVANDO...\n\n", numero);
            sleep_ms(500);
            gpio_put(LED_PIN_RED, 1);   //Ativa o LED vermelho indicando início da gravação

            //Alterna o estado do buzzer para criar bipes intercalados
            if(periodo % 1000 < 500){ //Alterna a cada 500ms
                if (!buzzer_ligado) {
                    iniciar_buzzer();
                    buzzer_ligado = true;
                }
            }else{
                if(buzzer_ligado){
                    parar_buzzer();
                    buzzer_ligado = false;
                }
            }
        }

        silencio_tempo = 0; //Reinicia o tempo de silêncio
    }else{
```

```
        silencio_tempo = 0; //Reinicia o tempo de silêncio
    }else{
        //Se o ruído parou, inicia a contagem do silêncio
        if(alerta_sonoro){
            if(silencio_tempo == 0){
                silencio_tempo = periodo;
            }else if(periodo - silencio_tempo >= 3000){
                gpio_put(LED_PIN_RED, 0);   //Desativa o LED vermelho indicando fim do trecho da gravação
                printf("Registrando dados de gravação e horário.\n\n");
                parar_buzzer(); //Garante que o buzzer seja desligado
                alerta_sonoro = false;
                buzzer_ligado = false; //Reseta o estado do buzzer
            }
        }
    }
    return valor_microfone;
}
```

Função de debouncing para o botão:

- Com a chamada da função habilita o tratamento de boucing no botão, evitando leituras erróneas.

```
//Debounce do botão (evita leituras falsas)
Codeium: Refactor | Explain | X
bool debounce_botao(uint gpio){
    static uint32_t ultimo_tempo = 0;
    uint32_t tempo_atual = to_ms_since_boot(get_absolute_time());
    //Verifica se o botão foi pressionado e se passaram 200ms
    if (gpio_get(gpio) == 0 && (tempo_atual - ultimo_tempo) > 200){ //200ms de debounce
        ultimo_tempo = tempo_atual;
        return true;
    }
    return false;
}
```

Configuração das chamadas de interrupção do botão:

- Configura as ações na chamada de interrupção do botão A.

```c
//Função de interrupção com Debouncing
Codeium: Refactor | Explain | X
void gpio_irq_handler(uint gpio, uint32_t events){
    uint32_t current_time = to_ms_since_boot(get_absolute_time());

        // Caso o botão A seja pressionado
        if(gpio == BOTAO_A && !alerta_sonoro && debounce_botao(BOTAO_A)){
            gpio_put(LED_PIN_RED, 0);    //Desliga o LED vermelho
            numero++;   //Incrementa o valor da rua (matriz de leds)
            if (numero == 10){
                numero = 0; //Retorna ao andar 0
            }
        set_one_led(led_r, led_g, led_b, numero);
        }
}
```

Função para realizar conversão do valor lido pelo microfone para a medida decibéis:

- Converte o valor de leitura pelo microfone de digital para a medida de decibéis, por meio do cálculo abaixo:

```c
//Função para converter o valor do ADC para dB SPL
Codeium: Refactor | Explain | X
float converter_adc_para_db(int adc_value) {
    float v_out = (adc_value / (float)ADC_MAX) * VREF; //Converte ADC para tensão
    float spl = 20 * log10((v_out / VREF) * SPL_MAX);  //Converte para dB SPL
    return spl;
}
```

Por fim, temos nossa chamada main, com o nosso laço principal while:

- Chama a função de inicialização das GPIOs, máquina de estado PIO, e demais configurações;

- Configura interrupção no botão A, utilizando o evento GPIO_IRQ_EDGE_FALL, pegando a borda de descida do nível do botão, configurado como pull-up;

- Envia dados inicializando nossa matriz de leds;

- Dentro do loop infinito, realizamos a leitura constante do valor do sinal por meio da leitura do microfone, através da função ‘ler_microfone()’, que faz toda verificação de presença de ruídos acima do limiar.

- Enviamos mensagem e informações do nível de ruído em decibéis para o display SSD1306 que faz conexão via protocolo I2C, por meio de funções presentes na biblioteca ‘ssd1306.h’.

```c
int main(){
    //Chamamos as função de inicialização
    inicializar_GPIOs();

    //Configuramos as interrupções nos botão A
    gpio_set_irq_enabled_with_callback(BOTAO_A, GPIO_IRQ_EDGE_FALL, true, &gpio_irq_handler);

    set_one_led(led_r, led_g, led_b, 0); //Inicia a simulação monitorando o rua de endereço 0

    while(true){
        //Variáveis usadas sendo usadas para armazenar os dados
        uint16_t valor_microfone= 0;
        float decibeis = 0.0;
        char bufferRua[20];
        char bufferDecibeis[20];

        valor_microfone = ler_microfone();                    //Le o valor do microfone
        decibeis = converter_adc_para_db((float)valor_microfone);  //Calcula o decibeis com base no valor lido

        ssd1306_fill(&ssd, false);                          //Limpa o display antes de escrever
        ssd1306_draw_string(&ssd, "VIGILANCIA", 30, 10);    //Escreve no display

        sprintf(bufferRua, "Rua %d", numero);               //Converte o numero da rua para string
        ssd1306_draw_string(&ssd, bufferRua, 10, 30);       //Escreve no display

        sprintf(bufferDecibeis, "Decibeis %.2f", decibeis); //Converte o decibeis para string
        ssd1306_draw_string(&ssd, bufferDecibeis, 10, 50);  //Escreve no display
        ssd1306_send_data(&ssd);                            //Atualiza o display

    }
}
```

9.7. Configuração dos registros

As configurações específicas dos registros incluem:

GPIOs: Definir pinos como entrada ou saída conforme a necessidade dos componentes conectados, além de configuração como o uso de conversão analógico-digital.

PWM: Configurar módulos PWM para controlar a intensidade e frequência do buzzer.

Comunicação com LEDs WS2812B: Uso da máquina de estado independente, a PIO, para controle e manipulação da matriz de leds 5x5, como intensidade e cores dos leds.

9.8. Estrutura e formato dos dados

Os dados manipulados pelo firmware são principalmente valores inteiros representando nível de ruído, por meio de valor digital correspondente à amplitude do sinal capturado pelo microfone e tempos, onde marcas temporais são armazenadas para cálculo de duração de eventos de ruído.

Além claro, de números de ponto flutuante utilizados na conversão dos valores digitais para medida decibéis SPL.

9.9. Protocolo de comunicação

No contexto da BitDogLab, a comunicação ocorre internamente entre os componentes através de protocolos como I2C para conexão com o display SSD1306, possibilitando a leitura e envio de dados para serem mostrados visualmente e comunicação serial USB para envio de informações como a simulação do estado das câmeras ou envio dos registros.

9.10. Formato do pacote de dados

Embora o sistema não envolva comunicação externa complexa na simulação, os dados internos seguem formatos específicos:

Leitura do Microfone: Valores dos sinais digitais representando a intensidade do som, que são transformados em formato decibéis SPL;

Comandos para LEDs: Estruturas de dados contendo informações de cor e intensidade para cada LED, realizada através da máquina PIO configurada para essa tarefa específica.

10. Execução do projeto

10.1. Metodologia

A execução do projeto seguiu uma abordagem iterativa, com pesquisas, desenvolvimento, testes e ajustes contínuos para garantir um funcionamento confiável do sistema.

Antes da implementação, foram realizadas pesquisas para definir os melhores componentes de hardware e software para o projeto. Foram considerados os seguintes aspectos:

- Sensores de ruído: Análise de microfones MEMS e módulos de amplificação para captar níveis de som com precisão.

- Microcontroladores: Comparação entre ESP32 e Raspberry Pi Pico W devido à necessidade de conectividade Wi-Fi para envio dos dados.

- Protocolos de Comunicação: Estudo sobre MQTT para transmissão eficiente das informações de ruído.

10.2. Escolha do Hardware

Com base nas pesquisas, os seguintes componentes foram escolhidos para o projeto:

- Microcontrolador: Optou-se por utilizar a Raspberry Pi Pico W devido ao maior conhecimento prévio da placa e ao fato de ter sido objeto de estudo no curso, além de ter um módulo WiFi, necessário para a implementação do nosso projeto. Embora a ESP32 apresente vantagens em termos de processamento e suporte nativo ao protocolo I2S (útil para leitura de sinais de áudio), a familiaridade com a Pico proporcionou maior controle e eficiência na implementação do sistema.

- Sensor de Som: Foi escolhido o microfone com amplificador MAX9814, pois oferece ganho automático (AGC), facilitando a captura precisa de variações de ruído, independentemente da intensidade do som ambiente.

- Comunicação e Monitoramento: Para transmissão de dados, utilizou-se do módulo Wi-Fi da Raspberry Pi Pico W, garantindo conectividade com o servidor MQTT para envio de alertas em tempo real.

- Interface de Monitoramento: Painel acessível ao síndico e responsáveis pelo condomínio.

10.3. Definição das Funcionalidades do Software

O software foi desenvolvido com base nos componentes presentes na placa BitDogLab que utiliza como base a Raspberry PI Pico W e microcontrolador RP2040, composto por periféricos ao seu redor.

A ideia do projeto real é desenvolver um software para capturar níveis de ruído sonoro em tempo real e realizar o processamento dos dados e verificação de incidentes no ambiente condominial. Indo mais além, incluímos ao projeto um servidor MQTT para envio das informações até o centro de monitoramento do condomínio, que conta com integração de um sistema de câmeras de segurança posicionadas pelo ambiente, garantindo automatização e controle do monitoramento.

Como foi definido o uso exclusivo da nossa placa BitDogLab juntamente com seus periféricos, e todo conhecimento demonstrados no curso, com intuito de simulação do projeto, foi feito o desenvolvimento de um software capaz de simular especialmente a leitura do sensor de ruido, através do microfone da placa, realizando a análise e ações semelhantes ao projeto real, embora com uma teoria bastante sólida a respeito.

Utilizando a configuração de vários periféricos da placa, foi possível uma simulação aproximada em escala menor, no quesito de dados, distâncias e até mesmo ambiente. O maior desafio dessa confecção da simulação, foi a parte do funcionamento do envio ao servidor MQTT e utilização de uma rede WiFi, já que não era possível realizá-los por conta das limitações aos periféricos da placa, e conhecimentos em relação a configuração de redes.

Sendo assim, foi definido a utilização da comunicação serial via USB na placa para simulação do envio de dados, como o registro da gravação e monitoramento das câmeras no momento da detecção da perturbação sonora. Toda parte de comunicação em rede, envio dos dados de um sensor, foram feitas através da comunicação serial, e utilização dos periféricos como a matriz de leds 5x5, display OLED e led vermelho.

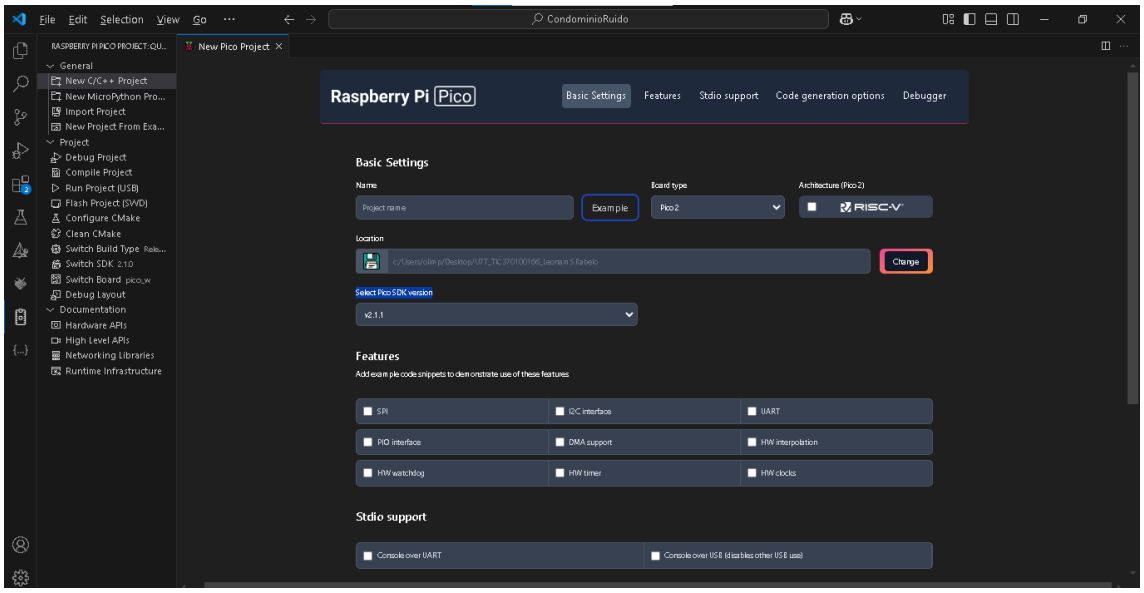
Em relação a captação dos sinais vindos do nosso microfone, foi bastante útil, pois simulou bem o que seria no nosso projeto real embora em menor escala, inclusive demonstrando alguns desafios a serem trabalhados.

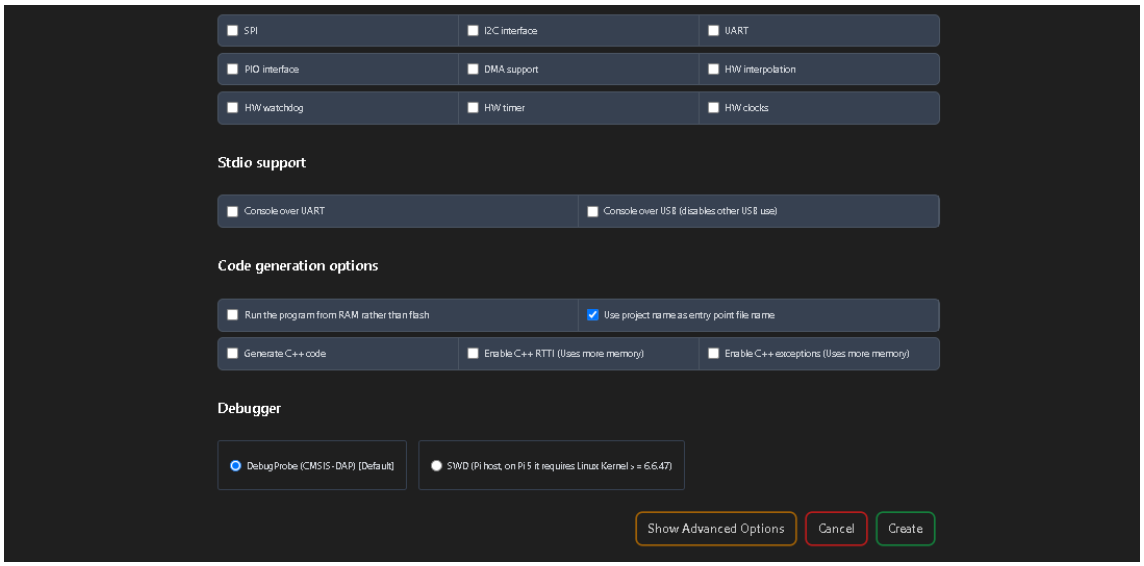
10.4. Inicialização da IDE

O desenvolvimento foi realizado no Visual Studio Code, na linguagem C. Foi feita toda configuração de instalação do kit de desenvolvimento da Raspberry PI Pico SDK, assim como feita a criação de variáveis de ambiente do sistema. Foi instalado alguns plugins no nosso Visual Studio Code para configuração do projeto, como o ‘C/C++’, ‘Raspberry PI Pico Project’ que faz toda inicialização do projeto, além de compilar e até mesmo instalar o firmware criado na nossa placa BitDogLab, além de inúmeras ferramentas para se utilizar no desenvolvimento, e claro ‘CMake’ e ‘CMake Tools’, que realizará todo script de configuração do seu código, inclusive a compilação.

10.5. Programação na IDE

O primeiro passo é criar nosso projeto por meio da extensão ‘Raspberry PI Pico Project’:





A programação na IDE envolveu alguns aspectos, como a configuração do CMakeLists.txt, ajustando a compilação, importação de bibliotecas que serão utilizadas no nosso código, além da configuração da comunicação serial, e máquina PIO utilizada na nossa simulação.

```
# Generated Cmake Pico project file

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(CondominioRuido 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(CondominioRuido
               CondominioRuido.c
               inc/ssd1306.c)

pico_set_program_name(CondominioRuido "CondominioRuido")
pico_set_program_version(CondominioRuido "0.1")

# Generate PIO header
pico_generate_pio_header(CondominioRuido ${CMAKE_CURRENT_LIST_DIR}/ws2812.pio)

# Modify the below lines to enable/disable output over UART/USB
pico_enable_stdio_uart(CondominioRuido 0)
pico_enable_stdio_usb(CondominioRuido 1)

# Add the standard library to the build
target_link_libraries(CondominioRuido
        pico_stdlib)

# Add the standard include files to the build
target_include_directories(CondominioRuido PRIVATE
        ${CMAKE_CURRENT_LIST_DIR}
        ${CMAKE_CURRENT_LIST_DIR}/inc
)
```

```
# Add any user requested libraries
target_link_libraries(CondominioRuido
        hardware_i2c
        hardware_adc
        hardware_pio
        hardware_pwm
        hardware_timer
        hardware_clocks
        )

pico_add_extra_outputs(CondominioRuido)
```

Com isso, a programação se inicia, importando bibliotecas, definindo variáveis, pinos de entrada e saída, e implementações de funções para o objetivo final do nosso projeto.

10.6. Depuração

A depuração do código foi realizada por meio de logs na comunicação serial, validando captura correta dos dados do microfone, envio das informações

ao display. Além dos logs, a IDE juntamente com a extensão da Raspberry PI Pico possibilita captura das variáveis por meio do Debugger, deixando o ambiente de desenvolvimento ainda mais rico.

11. Testes de validação

Após a implementação, foram realizados testes para validar o funcionamento do sistema. Foi verificado que o funcionamento de leitura dos sinais digitais pelo microfone estava correto, registrando em média 35.50dB alimentando a placa via conexão USB, e 34.50dB apenas na bateria da placa, sendo que o teste foi feito em ambiente silencioso para manter um padrão.

Colocando em prática o objetivo da simulação do projeto, foi utilizado testes básicos de ruídos com músicas no smartphone próximo a placa, e até mesmo assobios, e o resultado foi exatamente como esperado, ao passar os 80dB por um período constante de 3 segundos, o buzzer disparou enquanto estava com o ruído acima do limiar, a comunicação serial disparou simulando a gravação por meio das câmeras, o led vermelho acendeu, indicando início de gravação do trecho daquele incidente, e um alerta foi enviado ao display informado o endereço do sensor (enumerado de 0 a 9).

Retirando o som próximo a placa, a captação sonora já não indicava mais valores acima do limiar, porém o registro continuou por um tempo de 3 segundos garantindo que realmente aquele incidente havia parado, como esperado. Após a verificação da parada após o tempo de três segundos, um registro foi enviado via comunicação serial simulando o armazenamento do registro do incidente, além claro, do led vermelho apagar, indicando finalização do trecho da gravação.

Os botões alteram incrementando o valor do endereço do sensor que está sendo vigiado e enviando os dados captados pelo microfone, como esperado.

12. Discussão dos resultados

Os resultados da simulação do projeto foram surpreendentes, o microfone foi exposto a variados níveis de intensidade e distância sonora, e mesmo sendo uma simulação apresentou ser fiel na sua captação dos sinais.

O mais interessante foi o nível de ruídos externos na placa, e a interferência direta no microfone na captação do sinal, como por exemplo a alimentação USB e apenas na bateria, onde houve uma diferença nos valores registrados. O que leva a pensar em um tratamento mais cuidadoso na implementação a ambientes externos como ruas de um condomínio, seja via hardware como cases de proteção a placa, e até mesmo via software, com utilização de inteligência artificial para a detecção de barulhos de chuva por exemplo, evitando falsos positivos. No hardware do microfone escolhido, o MAX9814, vem equipado com AGC (Automatic Gain Control), função que controla automaticamente o volume do som captado, o que é útil para gravar ou detectar sons em ambientes com níveis de ruído que variam constantemente.

Contudo, a simulação se mostrou eficiente e funcional para o projeto de monitoramento de ruído em condomínios, permitindo uma gestão mais eficiente de infrações sonoras. Com pequenas otimizações, pode ser escalado para diferentes tipos de ambientes, como postos policiais e hospitais.

Link para o repositório

Segue o link para o repositório no GitHub:
https://github.com/LeonamRabelo/CondominioRuido

Link do vídeo

Segue o link para o vídeo com breves explicações e demonstração na BitDogLab:

https://www.youtube.com/watch?v=xOp6AUnd_g4

Referências

[1] Projeto de decibelímetro utilizando microfone de eletro. Disponível em: https://www.youtube.com/watch?v=wRxv3A3aHjg. Acesso em: 26 fev. 2025.

2] RASPBERRY PI FOUNDATION. RP2040 Datasheet. Disponível em: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf. Acesso em: 26 fev. 2025.

[3] RASPBERRY PI FOUNDATION. Raspberry Pi Pico Documentation. Disponível em: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf. Acesso em: 26 fev. 2025.

[4] ANALOG DEVICES. MAX9814: Microphone Amplifier with AGC and Low-Noise Microphone Bias. Disponível em: https://www.analog.com/media/en/technical-documentation/data-sheets/MAX9814.pdf. Acesso em: 26 fev. 2025.

[5] MQTT. MQTT Protocol Specifications. Disponível em: https://mqtt.org/mqtt-specification/. Acesso em: 26 fev. 2025.

[6] JUSBRASIL. A perturbação do sossego nos condomínios: o que fazer?. Disponível em: https://www.jusbrasil.com.br/artigos/a-perturbacao-do-sossego-nos-condominios-o-que-fazer/614630937. Acesso em: 26 fev. 2025.

[7] TENDA. Como cumprir a Lei do Silêncio em condomínio e evitar problemas?. Disponível em: https://www.tenda.com/blog/viver-em-condominio/como-cumprir-a-lei-do-silencio-em-condominio-e-evitar-problemas. Acesso em: 26 fev. 2025.