SISTEMA DE ANÁLISE E REPRODUÇÃO DE NOTAS MUSICAIS UTILIZANDO A BITDOGLAB

RICHARD LIMA RIBEIRO

Juazeiro – BA

2025

Introdução

Este projeto implementa um Sistema de Detecção e Reprodução de Notas Musicais utilizando os periféricos da BitdogLab. Ele oferece um menu interativo onde o usuário pode escolher entre detectar uma nota musical ou reproduzi-la.

A navegação no sistema é feita de forma intuitiva, utilizando um joystick e botões para selecionar as opções no menu. As notas são exibidas tanto na matriz de LED quanto no display SSD, proporcionando uma interface clara e dinâmica para o usuário.

Além disso, o sistema utiliza buzzers para a reprodução das notas musicais, permitindo que os usuários escutem os sons gerados. Com essa abordagem, o projeto pode ser aplicado tanto no aprendizado musical quanto em atividades interativas que envolvem reconhecimento sonoro.

Este projeto busca demonstrar a versatilidade da placa BitdogLab na integração de diferentes periféricos, proporcionando uma experiência prática e educativa no reconhecimento e na reprodução de sons musicais.

Objetivos

O principal objetivo deste projeto é desenvolver um sistema interativo de detecção e reprodução de notas musicais, utilizando os periféricos disponibilizados na BitdogLab. O sistema deve permitir que os usuários identifiquem notas musicais em tempo real e as reproduzam, promovendo uma experiência educativa e prática no reconhecimento de notas.

Objetivos específicos:

- Facilitar a detecção de notas musicais – O sistema deve ser capaz de identificar notas musicais e exibi-las de forma clara na matriz de LED e no display.
- Oferecer uma interface intuitiva – O menu interativo permite a navegação fácil por meio de um joystick e botões, tornando o sistema acessível para diferentes tipos de usuários.
- Reproduzir notas musicais – Utilizar buzzers para emitir sons das notas selecionadas, permitindo a audição e validação das notas musicais.
- Integrar os diferentes periféricos da BitdogLab – Demonstrar a aplicação prática dos componentes, como matriz de leds, display, joystick e buzzers, para criar um sistema funcional e interativo.
- Possibilitar aplicações educacionais – O projeto pode ser utilizado para auxiliar no ensino de música e no treinamento da percepção auditiva de notas musicais.

Descrição do funcionamento

O projeto é estruturado em diferentes **módulos funcionais**, que operam de forma independente, mas que, juntos, permitem a execução eficiente de todas as funcionalidades. Cada módulo desempenha um papel essencial, garantindo uma interação intuitiva com o usuário e a precisão na detecção e reprodução das notas musicais.

1. Menu Interativo

Para viabilizar a interação humano-máquina, o sistema conta com um menu interativo exibido no display SSD. Esse menu orienta o usuário sobre as ações disponíveis e a etapa do processo em que ele se encontra.

A navegação ocorre por meio de um joystick e botões físicos presentes na placa, permitindo a seleção de opções de forma intuitiva e ágil.

2. Feedback Visual pela Matriz de LEDs

Para tornar a visualização das notas musicais mais clara e intuitiva, o sistema utiliza uma matriz de leds. Essa matriz exibe a nota selecionada durante a reprodução ou a nota detectada pelo sistema, facilitando a compreensão do usuário.

As notas musicais são representadas seguindo a codificação usual:

- **Dó, Ré, Mi, Fá, Sol, Lá, Si → A, B, C, D, E, F, G**
- **Sustenidos (#)**: Indicados por um **LED azul na extremidade direita** da matriz.

Essa abordagem facilita a identificação das notas e seus subtons, tornando o sistema mais didático e acessível.

3. Emissão de Notas Musicais pelos Buzzers

Além da detecção de notas, o projeto permite a reprodução sonora das mesmas utilizando buzzers controlados por PWM. Isso garante que as notas sejam emitidas na frequência correta, permitindo que o usuário escute e compare os sons gerados.

4. Detecção de Notas Musicais via Microfone

O sistema também incorpora um microfone, responsável por capturar frequências sonoras emitidas pelo ambiente. A detecção das notas ocorre por meio de:

1. **Transformada Rápida de Fourier (FFT)** – Analisa o espectro do sinal de áudio e extrai as componentes de frequência.
2. **Correlação de Frequências** – Identifica a **frequência predominante** no sinal capturado, determinando assim a nota musical correspondente.

Justificativa

O desenvolvimento deste projeto busca atender à necessidade de um sistema interativo e acessível para a detecção e reprodução de notas musicais, unindo aprendizado, tecnologia e entretenimento em um único dispositivo.

A música está presente em diversas áreas do conhecimento e do cotidiano, sendo essencial para o desenvolvimento de habilidades auditivas e cognitivas. No entanto, muitos dispositivos de reconhecimento e reprodução de notas musicais disponíveis no mercado são complexos, de difícil acesso ou exigem conhecimento técnico avançado. Esse projeto preenche essa lacuna ao oferecer uma solução didática, acessível e interativa, garantindo uma experiência intuitiva para diversos perfis de usuários.

Além disso, este projeto apresenta uma abordagem prática e inovadora para a identificação e reprodução de notas musicais, combinando teoria musical com tecnologia de forma acessível. Com isso, possibilita novas formas de aprendizado e exploração sonora, atendendo tanto interessados em música quanto entusiastas da tecnologia e engenharia.

Principais Razões para o Desenvolvimento do Projeto

- **Aprendizado e Educação Musical**: A detecção e reprodução de notas musicais podem ser aplicadas como uma ferramenta educacional, auxiliando no treinamento auditivo, no reconhecimento de frequências e na afinação de instrumentos musicais.

- **Integração de Tecnologias**: O projeto explora a aplicação de diversos componentes, como microfone, matriz de leds, display, buzzers e processamento digital de sinais (FFT), incentivando o estudo e o uso de tecnologias embarcadas em aplicações reais.

- **Aplicações Diversificadas:** Além de auxiliar estudantes e entusiastas da música, o sistema pode ser útil para músicos, professores e desenvolvedores, servindo como uma ferramenta para reconhecimento sonoro, síntese de áudio e aprendizado musical de forma acessível.

- **Exploração da Plataforma BitdogLab**: O projeto aproveita ao máximo os periféricos disponíveis na BitdogLab, demonstrando seu potencial para aplicações didáticas, interativas e experimentais.

Com essa abordagem, o sistema não apenas facilita o aprendizado musical, mas também demonstra a integração entre eletrônica e tecnologia aplicada à música, abrindo possibilidades para futuras melhorias e expansões.

Originalidade

Após uma análise detalhada de projetos existentes, constatou-se que há uma escassez de trabalhos que abordem a **detecção e reprodução de notas musicais** utilizando exclusivamente os periféricos integrados na **placa BitDogLab**. A maioria dos projetos similares encontrados emprega diferentes plataformas de hardware ou combinações de dispositivos externos para alcançar funcionalidades semelhantes.

Por exemplo, o trabalho de **Guilherme de Nez Silvano**, intitulado "**Sistema de Reconhecimento de Escalas Musicais Utilizando Processamento Digital de Sinais**" , apresenta um sistema capaz de identificar notas musicais individuais e classificá-las dentro de uma escala. Embora compartilhe o objetivo de reconhecimento de notas, este projeto difere significativamente desse em termos de hardware utilizado e abordagem de implementação.

O projeto de Silvano não faz uso da placa BitDogLab; em vez disso, utiliza outros componentes de hardware e técnicas específicas de processamento de sinais. Em contraste, este projeto destaca-se por explorar de maneira inovadora e criativa os recursos nativos da BitDogLab, como o microfone embutido, o display OLED, a matriz de LEDs e o buzzer com PWM.

Portanto, a originalidade deste projeto reside na utilização exclusiva dos periféricos da BitDogLab para a detecção e reprodução de notas musicais, preenchendo uma lacuna na literatura e nos projetos existentes ao demonstrar uma solução completa e autônoma baseada nesta plataforma específica.

Diagrama de Blocos

Os componentes físicos do sistema seguem o seguinte esquema de ligações entre si:

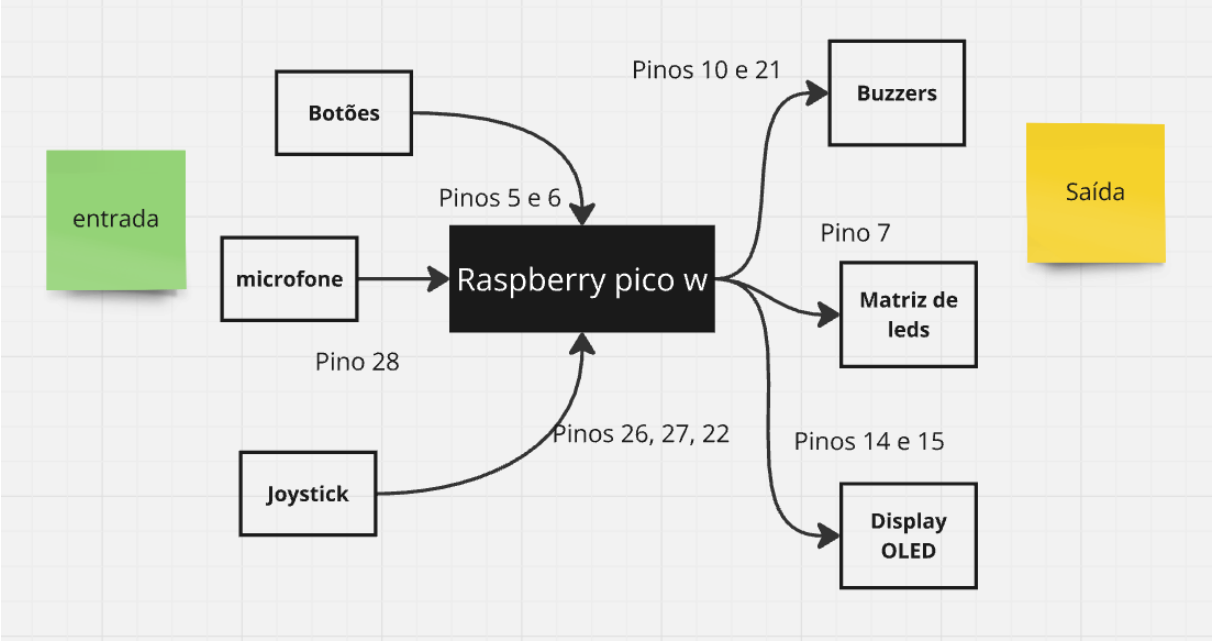

Figura 1 - Diagrama de Blocos

1. Microfone

O microfone integrado da placa BitdogLab será responsável por capturar as frequências sonoras das notas musicais emitidas no ambiente. O sinal capturado será enviado para a Raspberry Pico W, onde será processado utilizando algoritmos de análise de frequência, permitindo a identificação precisa das notas musicais.

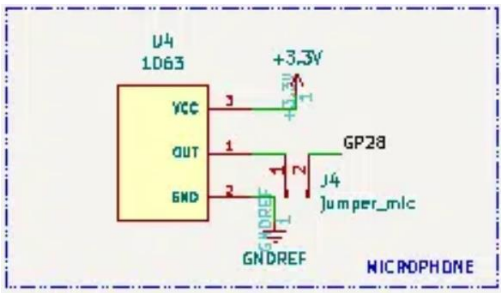

Figura 2 - Hardware do Microfone

Para garantir o funcionamento correto do microfone, foram utilizadas as seguintes funções na configuração do ADC (Conversor Analógico-Digital):

- ***adc_init()*** – Inicializa o módulo ADC da Raspberry Pico W, permitindo a leitura de sinais analógicos.
- ***adc_gpio_init(MIC_PIN)*** – Configura o pino correspondente ao microfone como entrada analógica.
- ***adc_select_input(2)*** – Define o canal ADC que será utilizado para capturar o sinal do microfone.

- ***adc_read()*** – Realiza a leitura do sinal analógico convertido, permitindo a extração dos dados de áudio para processamento.

2. Joystick

O joystick será utilizado como principal dispositivo de navegação na interface do sistema exibida no display. Com ele, o usuário poderá percorrer o menu interativo e selecionar as opções desejadas de forma intuitiva.

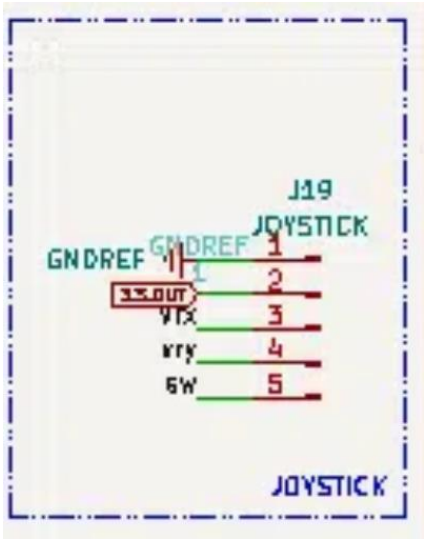

Figura 3 - Hadware do Joystick

Para garantir o funcionamento correto do **joystick**, foram utilizadas as seguintes funções na configuração do **ADC (Conversor Analógico-Digital)**:

- ***adc_init()*** – Inicializa o módulo ADC da Raspberry Pico W, permitindo a leitura de sinais analógicos do joystick.
- ***adc_gpio_init(VX_PIN)*** – Configura o pino correspondente ao eixo X do joystick como entrada analógica.
- ***adc_select_input(1)*** – Define o canal **ADC 1** como a fonte de leitura do joystick, garantindo que os dados capturados correspondam ao movimento desejado.
- ***adc_read()*** – Realiza a leitura do sinal analógico convertido, retornando um valor entre 0 e 4095, que representa a posição atual do joystick:
    - ***Valores baixos (<1500)*** → Indicam que o joystick foi movido para a esquerda.
    - ***Valores médios (~2048)*** → Indicam que o joystick está em repouso no centro.
    - ***Valores altos (>2500)*** → Indicam que o joystick foi movido para a direita.

3. **Botões**

Os botões físicos terão a função de confirmar seleções e interações do usuário com o sistema. Cada acionamento gera uma interrupção nos pinos correspondentes da Raspberry Pico W, garantindo uma resposta rápida e precisa às ações do usuário.

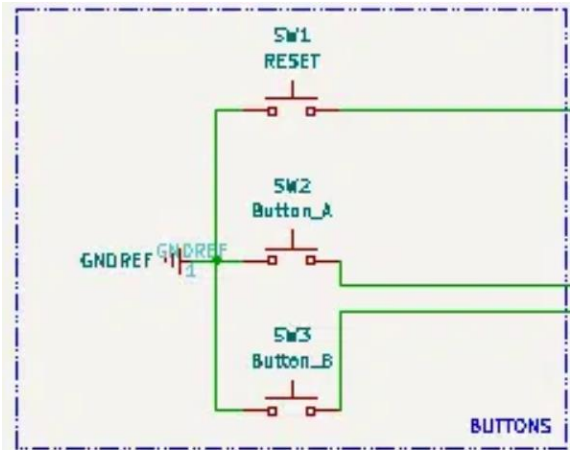

Figura 4 - Hadware dos Botões

Para garantir o funcionamento correto dos **botões**, foram utilizadas as seguintes funções na configuração dos **GPIOs (General Purpose Input/Output)**:

- ***gpio_init(BUTTON_A)*** e ***gpio_init(BUTTON_B)*** – Inicializam os pinos correspondentes aos botões A e B, preparando-os para serem utilizados como entradas digitais.
- ***gpio_set_dir(BUTTON_A, GPIO_IN)*** e ***gpio_set_dir(BUTTON_B, GPIO_IN)*** – Configuram os pinos dos botões como **entradas**, permitindo a detecção do estado de cada botão.
- ***gpio_pull_up(BUTTON_A)*** e ***gpio_pull_up(BUTTON_B)*** – Ativam resistores **pull-up internos**, garantindo que os botões permaneçam em estado **alto (1)** quando não pressionados, evitando leituras erradas.
- ***gpio_set_irq_enabled_with_callback(BUTTON_A, GPIO_IRQ_EDGE_FALL, true, &gpio_irq_handler);***
- Habilita uma **interrupção** no botão A para detectar a **transição de alto (1) para baixo (0)**, ou seja, quando o botão é pressionado.
- Chama a função gpio_irq_handler() para processar a ação correspondente.
- ***gpio_set_irq_enabled(BUTTON_B, GPIO_IRQ_EDGE_FALL, true);*** – Ativa a interrupção para o botão B, garantindo que ele seja detectado corretamente quando pressionado.
- ***gpio_irq_handler(uint gpio, uint32_t events)*** – Função que processa os eventos dos botões e muda o estado do menu conforme necessário.

4. **Buzzers**

   Os buzzers serão responsáveis pela reprodução sonora das notas musicais detectadas ou selecionadas no sistema. A geração dos sons será feita por **modulação por largura de pulso (PWM)**, garantindo que cada nota seja reproduzida com a frequência correta.

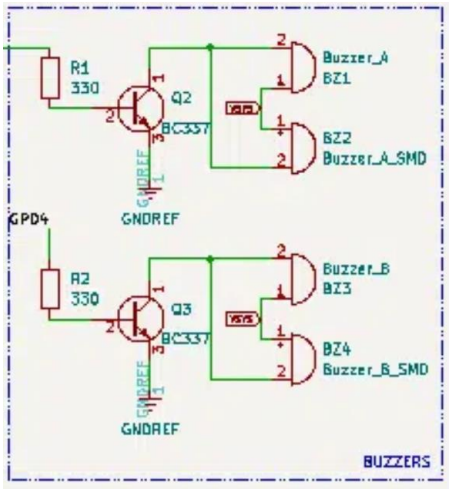

Figura 5 - Hadware dos buzzers

Para garantir o funcionamento correto dos **buzzers**, foram utilizadas as seguintes funções na configuração do **PWM (Pulse Width Modulation - Modulação por Largura de Pulso)**:

- ***initialization_buzzers(BUZZER_A, BUZZER_B)*** – Inicializa os pinos correspondentes aos buzzers, configurando-os para operar com **sinal PWM**, que permite a geração de frequências sonoras.
- ***buzzer_pwm(uint gpio, uint16_t frequency, uint16_t duration_ms)*** – Função responsável por tocar uma nota musical nos buzzers:
  - ***gpio*** – Define o pino onde o buzzer está conectado.
  - ***frequency*** – Define a frequência da nota musical (em Hertz), permitindo a reprodução do som correto.
  - **duration_ms** – Define a duração do som emitido (em milissegundos).
- ***pwm_set_gpio_level(gpio, duty_cycle)*** – Ajusta o **ciclo de trabalho (duty cycle)** do PWM, controlando a intensidade do som gerado.
- ***pwm_set_wrap(slice, clock_get_hz(clk_sys) / frequency)*** – Configura o **período do PWM** com base na frequência da nota musical desejada.
- ***sleep_ms(duration_ms)*** – Mantém o som ativo pelo tempo especificado antes de desativá-lo.
- ***pwm_set_enabled(slice, false)*** – Desativa o PWM após a reprodução da nota para evitar sons contínuos indesejados.

5. **Matriz de leds**

A matriz de leds será utilizada para **exibição visual das notas musicais**, tornando a experiência mais imersiva e didática. Durante a interação, a matriz será atualizada dinamicamente para representar graficamente as notas detectadas ou reproduzidas, auxiliando no reconhecimento visual das mesmas.

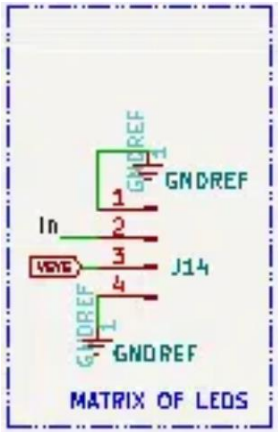

Figura 6- Hadware da matriz de leds

Para garantir o funcionamento correto da **matriz de LEDs**, foram utilizadas as seguintes funções na configuração e controle do **WS2812B (NeoPixel)**:

Configuração e Controle da Matriz de LEDs

- ***matrix_rgb(uint r, uint g, uint b, float intensity)*** – Converte valores de **RGB** para um formato adequado ao controlador WS2812B, ajustando a **intensidade** da cor.
  - **r, g, b** → Definem a intensidade das cores vermelha, verde e azul.
  - **intensity** → Define o brilho da cor, variando de 0 (apagado) a 1 (brilho total).
  - Retorna um valor de **32 bits** no formato adequado para o envio via **PIO**.
- ***draw_pio(pixel *draw, PIO pio, uint sm, float intensity)*** – Envia os valores de cor da matriz para o controlador **PIO** (Periférico de Entrada/Saída Programável).
  - **draw** → Array de pixels que contém as cores que serão exibidas.
  - **pio** → Instância do periférico PIO usada para controlar os LEDs.
  - **sm** → Máquina de estado utilizada na programação da PIO.
  - **intensity** → Controla o brilho da matriz.

draw_note(PIO pio, uint sm, const char *note) – Exibe **a nota musical correspondente** na matriz de LEDs.

- **note** → String contendo a nota musical a ser representada (ex.: "C", "D#", "G").
- Cada nota é representada por um padrão de LEDs específico.

- Se a nota for um **sustenido (#)**, um **LED azul** é ativado na extremidade direita da matriz.

Configuração da PIO para o Controle da Matriz de LEDs
A função responsável por configurar a **PIO** para controlar os LEDs é:
void PIO_setup(PIO *pio, uint *sm)
Essa função inicializa a **PIO** e configura um **programa PIO personalizado** para manipular os LEDs WS2812B.
Explicação das funções utilizadas

- ***pio = pio0;**
  - Define a instância da PIO que será utilizada (**pio0** neste caso).
  - A Raspberry Pico possui duas PIOs disponíveis (**pio0** e **pio1**).
- **uint offset = pio_add_program(*pio, &pio_matrix_program);**
  - **Carrega** o programa PIO que controla a matriz de LEDs na memória da PIO.
  - O programa pio_matrix_program foi definido anteriormente e contém as instruções específicas para comunicação com os LEDs WS2812B.
  - offset armazena a posição do programa na memória da PIO.
- ***sm = pio_claim_unused_sm(*pio, true);**
  - **Reserva** uma **máquina de estado livre** dentro da PIO para executar o programa carregado.
  - A PIO possui **4 máquinas de estado** por instância, e cada uma pode operar independentemente.
- **pio_matrix_program_init(*pio, *sm, offset, LED_PIN);**
  - **Inicializa** a máquina de estado (*sm) para controlar os LEDs.
  - Configura os parâmetros necessários para comunicação com os LEDs WS2812B.
  - **LED_PIN** representa o pino GPIO ao qual a matriz de LEDs está conectada.

6. Display OLED

O display OLED servirá como a principal interface gráfica do sistema, apresentando ao usuário as notas musicais identificadas, informações sobre a navegação no menu e demais mensagens interativas. Ele garantirá uma visualização clara e objetiva das opções disponíveis.

Figura 7- Hadware do display OLED

Para garantir o funcionamento correto do **display OLED SSD1306**, foram utilizadas as seguintes funções para inicialização, configuração, envio de comandos e exibição de gráficos e texto.

---

1. Inicialização do Display

- **ssd1306_init(ssd1306_t *ssd, uint8_t width, uint8_t height, bool external_vcc, uint8_t address, i2c_inst_t *i2c)**
  - Inicializa a estrutura de dados do display.
  - Define **largura (width)** e **altura (height)** do display.
  - Configura o **endereço I2C (address)** para comunicação com o display.
  - Aloca memória para o **buffer de RAM** que armazena os pixels antes de serem enviados ao display.

---

2. Configuração do Display

- **ssd1306_config(ssd1306_t *ssd)**
  - Envia uma sequência de **comandos de configuração** ao display para ativá-lo corretamente.
  - Configura a **memória gráfica**, **modo de exibição**, **contraste**, **orientação** e **tensão de operação**.

Comandos importantes:

- **SET_DISP | 0x00** → Desliga temporariamente o display.
- **SET_MEM_ADDR e 0x01** → Define o modo de endereçamento da memória (horizontal).
- **SET_DISP_START_LINE | 0x00** → Define a linha inicial de exibição.
- **SET_SEG_REMAP | 0x01** → Ajusta a orientação horizontal dos pixels.
- **SET_COM_OUT_DIR | 0x08** → Ajusta a orientação vertical.
- **SET_CONTRAST | 0xFF** → Define o contraste máximo.
- **SET_CHARGE_PUMP | 0x14** → Habilita a bomba de carga para alimentar o display.
- **SET_DISP | 0x01** → Liga o display após a configuração.

---

3. Envio de Comandos

- **ssd1306_command(ssd1306_t *ssd, uint8_t command)**
  - Envia um **comando individual** via I2C para controlar o display.
  - Utiliza a função i2c_write_blocking() para comunicar-se com o display via barramento **I2C**.

---

4. Atualização da Tela

- **ssd1306_send_data(ssd1306_t *ssd)**
  - Atualiza a tela enviando **o conteúdo do buffer de RAM** para o display.
  - Define os **endereços de coluna e página**, garantindo que os dados sejam desenhados corretamente.

---

5. Manipulação de Pixels

- **ssd1306_pixel(ssd1306_t *ssd, uint8_t x, uint8_t y, bool value)**
  - Define o estado de um **pixel individual** no buffer de memória do display.
  - O pixel pode ser **ligado (1)** ou **desligado (0)**.
  - Utiliza manipulação de bits para ajustar os dados na memória do display.

---

6. Preenchimento da Tela

- **ssd1306_fill(ssd1306_t *ssd, bool value)**
  - Preenche a tela inteira com **pixels ativados (1)** ou **desativados (0)**.
  - Itera por todas as posições do display e chama ssd1306_pixel().

---

7. Desenho de Formas Gráficas

- **ssd1306_rect(ssd1306_t *ssd, uint8_t top, uint8_t left, uint8_t width, uint8_t height, bool value, bool fill)**
  - Desenha um **retângulo** na tela, com ou sem preenchimento.
  - Utiliza ssd1306_pixel() para definir os contornos e o preenchimento interno.
- **ssd1306_line(ssd1306_t *ssd, uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1, bool value)**
  - Desenha uma **linha** entre dois pontos utilizando o **algoritmo de Bresenham**.
- **ssd1306_hline(ssd1306_t *ssd, uint8_t x0, uint8_t x1, uint8_t y, bool value)**
  - Desenha uma **linha horizontal** entre dois pontos.
- **ssd1306_vline(ssd1306_t *ssd, uint8_t x, uint8_t y0, uint8_t y1, bool value)**

- Desenha uma **linha vertical** entre dois pontos.

8. Exibição de Texto

- **ssd1306_draw_char(ssd1306_t *ssd, char c, uint8_t x, uint8_t y)**
  - Renderiza um **caractere** na tela, utilizando uma matriz de fontes armazenada na memória (font.h).
  - Cada caractere é representado por um **conjunto de 8x8 pixels**.
  - **ssd1306_draw_string(ssd1306_t *ssd, const char *str, uint8_t x, uint8_t y)**
  - Escreve uma **string completa** na tela, chamando ssd1306_draw_char() para cada caractere.
  - Ajusta automaticamente a posição para evitar que o texto ultrapasse os limites do display.

9. Integração com o Sistema

- **setup_display()**
  - Configura o **barramento I2C**, inicializa o display e exibe a tela inicial.
- **menu()**
  - Controla as telas do sistema e exibe as opções disponíveis no **display OLED**.
  - Alterna entre os modos **MENU**, **DETECTAR NOTA** e **TOCAR NOTA**, exibindo as informações na tela.

7. Raspberry Pico W

A Raspberry Pico W será o núcleo do sistema, responsável por processar os sinais recebidos do microfone, gerenciar a interface de usuário e controlar os periféricos, como buzzers, matriz de LEDs e display OLED. Seu poder de processamento e conectividade tornam possível a implementação eficiente das funcionalidades propostas.

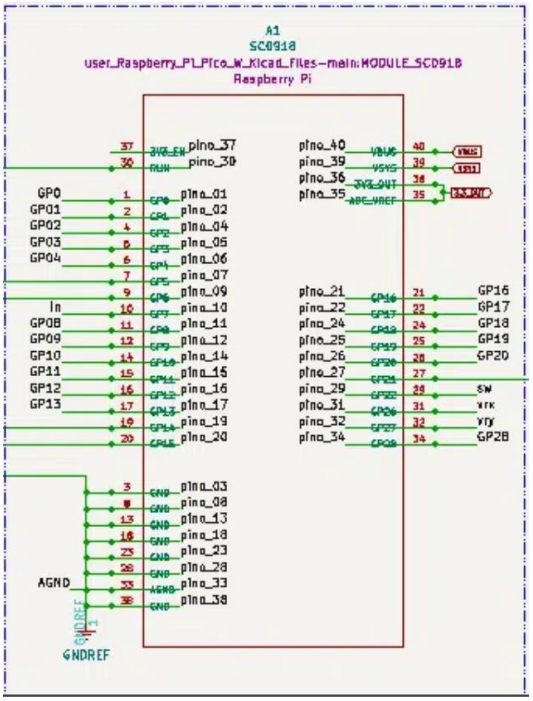

Figura 8-Raspberry pico w

8. Pinagem

O projeto utiliza os seguintes pinos da Raspberry Pico W para conectar e controlar os periféricos:

Componente	Pino	Função
Matriz de LEDs	7	Controle via PIO
Pushbutton A	5	Interação – Alterna modos (MENU/DETECTAR)
Pushbutton B	6	Ativa/desativa buzzer
Microfone	28	Captura som – Entrada ADC
Joystick	26, 27, 22	Controle de notas – Entrada ADC e botão
Display OLED	14 (SDA), 15 (SCL)	Comunicação I2C
Buzzer A	10	Geração de som via PWM
Buzzer B	21	Geração de som via PWM

9. Circuito completo do hardware

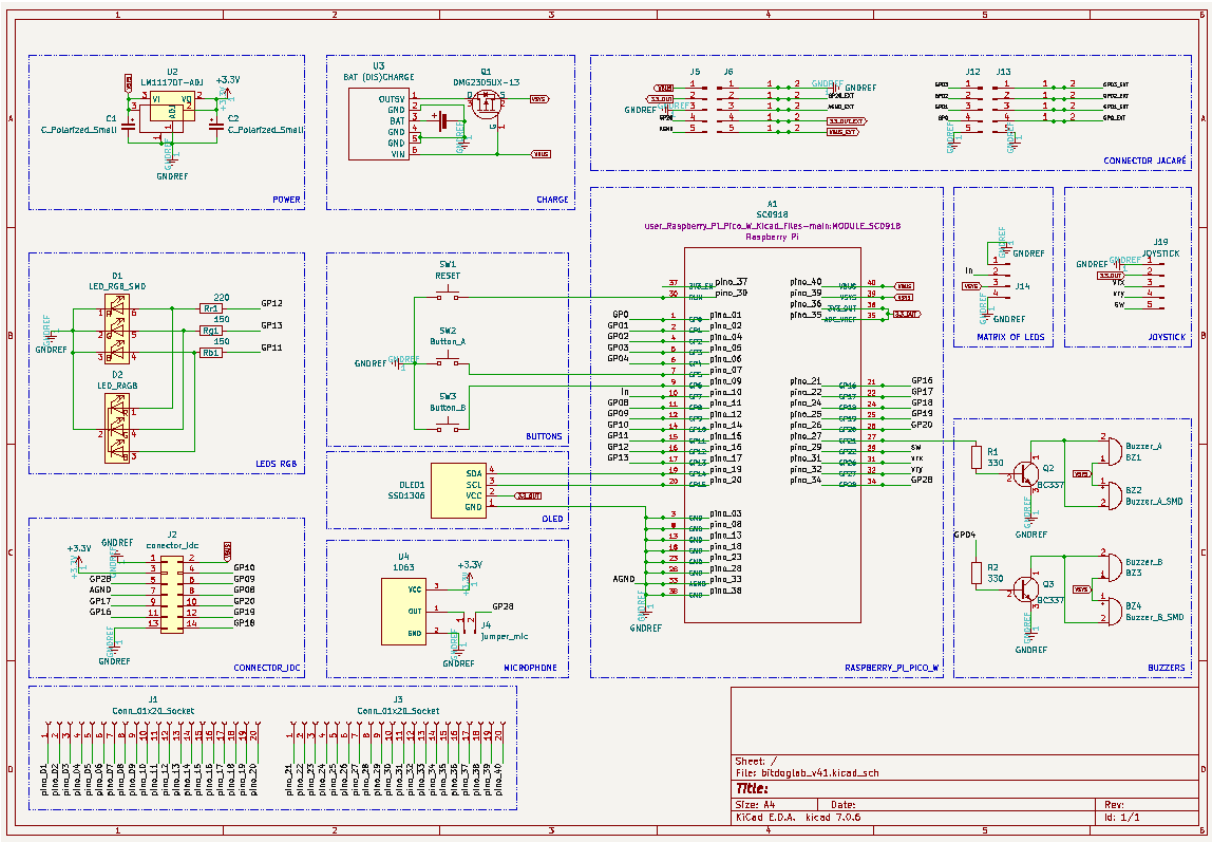

Figura 9 - Circuito da bitDogLab

10. Diagrama de Camadas

Figura 10- Diagrama de Camadas

1. FFT (Transformada Rápida de Fourier)

A Transformada Rápida de Fourier (FFT) é utilizada para converter os sinais de áudio captados pelo microfone do domínio do tempo para o domínio da frequência. No seu código, essa funcionalidade está implementada na função **fft()**, que reorganiza os dados e aplica a transformada para identificar

os componentes espectrais do sinal. Isso permite a análise das frequências dominantes presentes no áudio captado.

2. Interpolação

A interpolação é usada para melhorar a precisão na identificação da frequência dominante, refinando a posição do pico da FFT. A função **interpolar_pico()** ajusta a frequência detectada considerando os valores ao redor do maior pico identificado na FFT, reduzindo erros de quantização e aumentando a precisão na detecção da nota musical.

3. Gerenciamento das Interrupções

O gerenciamento das interrupções permite que o software responda rapidamente a eventos externos, como o pressionamento dos botões. A função **gpio_irq_handler()** trata as interrupções dos botões, alternando entre diferentes modos do sistema (menu, detecção de notas e geração de som). O sistema de debounce é implementado para evitar leituras errôneas devido ao ruído mecânico dos botões.

4. Identificação da Nota Musical

A detecção da nota musical ocorre a partir da frequência dominante extraída da FFT. A função **detectar_nota()** compara a frequência detectada com uma tabela de frequências base armazenada no arquivo notas.h, determinando qual nota musical está sendo tocada. Esse processo permite que o software exiba a nota correspondente no display e ative os dispositivos apropriados.

5. Atualização do Menu

A interface gráfica do sistema é controlada por meio do display OLED, que exibe informações ao usuário. A função **menu()** gerencia a exibição de diferentes telas, permitindo alternar entre a detecção de notas e a geração de sons. Para isso, a biblioteca ssd1306 é utilizada para renderizar elementos gráficos e atualizar dinamicamente os conteúdos do display.

6. Geração de Sinal PWM

A geração de sinais PWM (Modulação por Largura de Pulso) é utilizada para controlar o buzzer e gerar sons correspondentes às notas musicais. A função **buzzer_pwm()** configura um pino GPIO no modo PWM, ajustando a frequência e a duração do sinal para gerar a onda sonora correspondente. Isso permite que o software toque notas musicais baseadas na entrada do usuário.

7. Configuração da Matriz de LEDs

A matriz de LEDs é utilizada para fornecer uma representação visual das notas musicais detectadas ou selecionadas. A função **draw_note()** recebe uma nota musical e exibe um padrão específico na matriz de LEDs. Além disso, a função **draw_number()** permite a exibição de números na matriz, sendo útil para indicar seleções no menu.

5. **Principais variáveis utilizadas**

float magnitudes[SAMPLES / 2];

Descrição: Armazena as magnitudes dos componentes de frequência após a execução da Transformada Rápida de Fourier (FFT). Uso: Essencial para identificar a frequência dominante do sinal de áudio captado pelo microfone.

ssd1306_t ssd;

Descrição: Estrutura usada para armazenar o estado do display OLED SSD1306. Uso: Utilizada para controlar e atualizar o conteúdo exibido no display.

uint last_interrupt_a = 0; e uint last_interrupt_b = 0;

Descrição: Armazena o timestamp da última interrupção dos botões A e B para implementar um sistema de debounce. Uso: Evita leituras errôneas devido ao ruído mecânico dos botões.

uint DEBOUNCE_MS = 200;

Descrição: Define um tempo mínimo (em milissegundos) entre duas leituras consecutivas de um botão, prevenindo múltiplos acionamentos indesejados. Uso: Utilizado no tratamento de interrupções para os botões.

bool buzzer_enabled = false;

Descrição: Variável de controle que define se o buzzer deve tocar ou não. Uso: Evita a reprodução contínua do som e permite a ativação/desativação conforme a interação do usuário.

enum TELAS { MENU, DETECTAR, TOCAR };

Descrição: Enumeração que define os diferentes estados do menu do sistema. Uso: Controla qual tela está sendo exibida no display OLED.

uint tela_atual = MENU;

Descrição: Armazena o estado atual da interface do usuário.

Uso: Determina se o software está no menu principal, no modo de detecção de notas ou no modo de reprodução.

float freq_dominante = 0;

Descrição: Guarda a frequência dominante detectada após o processamento do sinal de áudio.

Uso: Utilizada para determinar a nota musical correspondente.

char *nota = "##";

Descrição: Ponteiro para armazenar o nome da nota musical correspondente à frequência detectada.

Uso: Exibida no display OLED e utilizada para tocar a nota correta no buzzer.

volatile uint nota_selecionada = 0;

Descrição: Variável compartilhada entre a lógica principal e as interrupções para armazenar a nota musical atualmente selecionada pelo usuário.

Uso: Controla qual nota será tocada pelo buzzer quando o usuário interage com os botões ou joystick.

struct repeating_timer timer;

Descrição: Estrutura utilizada para configurar um temporizador de execução periódica.

Uso: Chamado regularmente para verificar a posição do joystick e atualizar a seleção de notas.

PIO pio; e uint sm;

Descrição: Variáveis associadas ao uso do periférico PIO (Programável Input/Output) do Raspberry Pi Pico.

Uso: Controla a comunicação com a matriz de LEDs e outras operações que utilizam PIO.

const char *notas[] = {"C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"};

Descrição: Array de strings contendo os nomes das notas musicais.

Uso: Utilizado para mapear frequências detectadas às notas musicais correspondentes.

const float frequencias_base[] = { ... };

Descrição: Tabela contendo as frequências fundamentais das notas musicais em diferentes oitavas. Uso: Utilizada na função detectar_nota() para determinar qual nota musical corresponde à frequência detectada.

6. **Fluxograma**

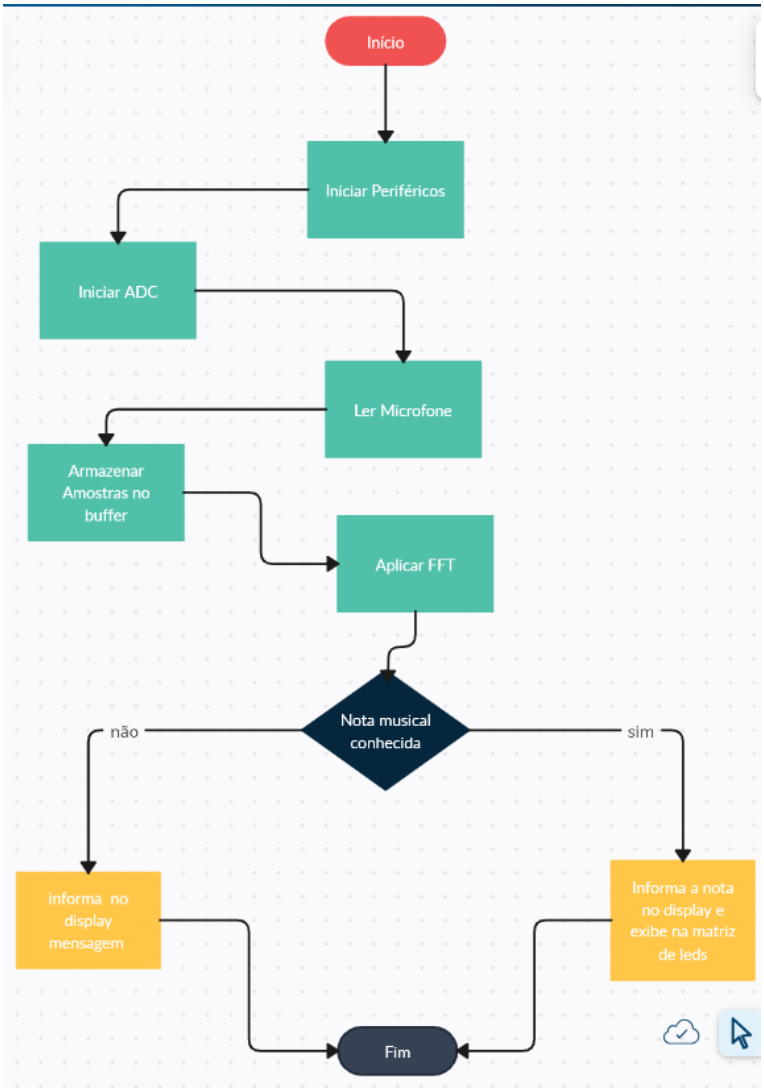

Figura 11 - rotina de detecção de nota musical

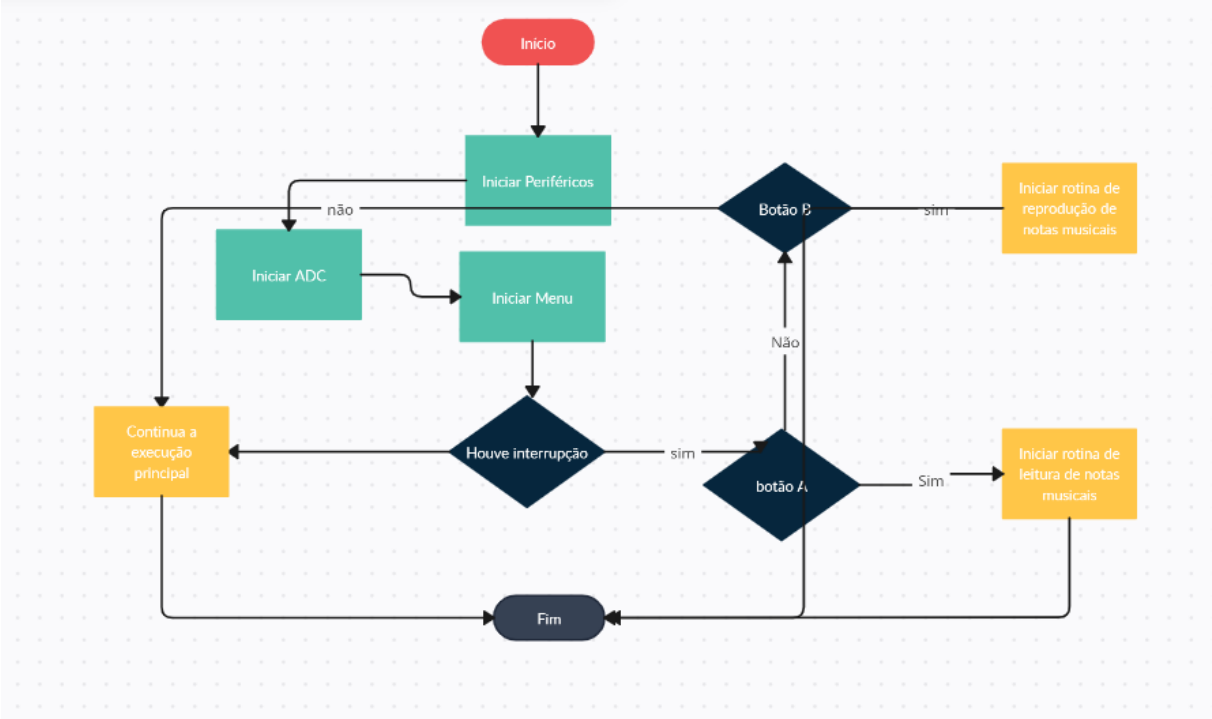

Figura 12 - rotina de interrupção

Inicialização do Software

O processo de inicialização do software segue uma sequência de etapas para configurar os componentes de hardware e preparar o sistema para funcionamento. Abaixo está a descrição detalhada das etapas envolvidas:

7. **Inicialização do Sistema e Periféricos**

A função **stdio_init_all();** configura a interface de entrada e saída padrão, permitindo a comunicação com o terminal.

A função **adc_init();** inicializa o módulo ADC (Conversor Analógico-Digital), que será usado para captar o sinal do microfone e do joystick.

As funções **adc_gpio_init(MIC_PIN);** e **adc_gpio_init(VX_PIN);** configuram os pinos GPIO correspondentes ao microfone e ao joystick como entradas analógicas.

8. **Configuração do Display OLED**

A função **setup_display();** realiza a inicialização do display OLED SSD1306 via I2C.

A função **i2c_recovery();** é chamada para garantir que o barramento I2C esteja em um estado funcional antes de iniciar a comunicação.

O display é configurado para exibir as informações de menu e das notas detectadas.

9. **Configuração da Máquina de Estado Programável (PIO)**

A função **PIO_setup(&pio, &sm);** configura o PIO (Programável Input/Output) para controlar a matriz de LEDs.

O programa correspondente é carregado na PIO e vinculado a um estado de máquina (sm) que será usado para atualizar os LEDs.

10. **Configuração dos Botões de Controle**

A função **setup_button(BUTTON_A);** e **setup_button(BUTTON_B);** inicializam os pinos GPIO dos botões A e B como entradas com pull-up ativado.

As interrupções são habilitadas por meio da função **enable_interrupt();,** permitindo a detecção de eventos de pressionamento dos botões.

11. **Configuração do Timer para o Joystick**

A função **add_repeating_timer_ms(100, joystick_callback, NULL, &timer);** configura um timer periódico que executa a função **joystick_callback()** a cada 100ms.

Esse callback é responsável por monitorar a posição do joystick e atualizar a nota selecionada.

12. **Configuração dos Buzzers**

A função **initialization_buzzers(BUZZER_A, BUZZER_B);** inicializa os buzzers conectados ao sistema.

Os pinos GPIO associados aos buzzers são configurados como saídas e inicialmente desligados.

13. **Configuração de Buffers para Processamento de Áudio**

Arrays **real[SAMPLES]** e **imag[SAMPLES]** são declarados para armazenar os valores amostrados do sinal do microfone e os resultados da FFT.

Essas variáveis são essenciais para a análise espectral do sinal de entrada.

14. **Loop Principal do Sistema**

Após a inicialização, o sistema entra em um loop infinito (**while (true)**) onde:

O menu é atualizado e exibido na tela (**menu**();).

O sinal do microfone é lido e processado via ADC.

A FFT é aplicada para extrair a frequência dominante.

A nota correspondente é identificada e exibida no display.

Caso o usuário interaja com os botões, a ação apropriada (tocar nota ou alternar entre menus) é executada.

O sistema aguarda 500ms **(sleep_ms(500);)** entre as iterações para evitar processamento excessivo.

Estrutura e formato dos dados

1. Dados para Processamento de Áudio

O software utiliza buffers para armazenar os valores do sinal de áudio capturado pelo microfone e processá-los por meio da FFT.

Arrays de Amostragem de Áudio

```
#define SAMPLES 1024
float real[SAMPLES], imag[SAMPLES];
```

- **real[SAMPLES]** → Armazena os valores do sinal de entrada (domínio do tempo) convertidos pelo ADC.
- **imag[SAMPLES]** → Inicialmente zerado, usado na FFT para armazenar a parte imaginária do espectro de frequência.

2. Dados para Identificação da Nota Musical

O software utiliza tabelas de frequências para mapear a frequência detectada em uma nota musical.

Lista de Notas Musicais

```
const char *notas[] = {"C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"};
```

Estados do Menu

```
enum TELAS { MENU, DETECTAR, TOCAR };
uint tela_atual = MENU;
```

- **Define os diferentes modos do sistema (Menu Principal, Detecção de Notas, Reprodução de Notas).**
- **Controla qual tela está sendo exibida no display.**

5. Dados para Controle da Matriz de LEDs

A matriz de LEDs exibe informações visuais associadas às notas musicais.

Estrutura de Pixels

```
typedef struct {
    uint8_t red;
    uint8_t green;
    uint8_t blue;
} pixel;
```

- **Armazena a cor de um pixel da matriz de LEDs (valores de R, G e B).**
- **Usado para representar notas musicais com cores específicas.**

Frame de Exibição

```
typedef pixel frame[25];
```

- **Array de 25 pixels usado para desenhar números e notas musicais na matriz de LEDs.**

15. **Configuração dos registros de hardware**

1. Configuração do ADC (Conversor Analógico-Digital)

O ADC é utilizado para capturar o sinal do microfone e ler o estado do joystick. As configurações são realizadas na inicialização do sistema:

```
c
CopiarEditar
adc_init();
adc_gpio_init(MIC_PIN);
adc_gpio_init(VX_PIN);
```

- adc_init(); → Inicializa o módulo ADC.
- adc_gpio_init(MIC_PIN); → Configura o pino do microfone como entrada analógica.
- adc_gpio_init(VX_PIN); → Configura o pino do joystick como entrada analógica.
- adc_select_input(2); → Seleciona a entrada do ADC a ser lida durante a execução.

2. Configuração dos GPIOs para Botões

Os botões de controle (A e B) são configurados como entradas com pull-up ativado para detecção de pressionamento.

```
c
CopiarEditar
void setup_button(uint gpio) {
    gpio_init(gpio);
    gpio_set_dir(gpio, GPIO_IN);
```

```
    gpio_pull_up(gpio);
}
```

- gpio_init(gpio); → Inicializa o pino GPIO especificado.
- gpio_set_dir(gpio, GPIO_IN); → Define o pino como entrada.
- gpio_pull_up(gpio); → Ativa o resistor de pull-up interno, garantindo um estado alto quando o botão não está pressionado.

As interrupções para os botões são configuradas com:

```
gpio_set_irq_enabled_with_callback(BUTTON_A, GPIO_IRQ_EDGE_FALL, true, &gpio_irq_handler);
gpio_set_irq_enabled(BUTTON_B, GPIO_IRQ_EDGE_FALL, true);
```

- gpio_set_irq_enabled_with_callback() → Habilita interrupções para o botão A, chamando gpio_irq_handler() ao detectar um pressionamento.
- gpio_set_irq_enabled() → Habilita interrupções para o botão B.

3. Configuração do PWM para o Buzzer

O buzzer utiliza o módulo PWM para gerar sons com diferentes frequências.

```
c
void buzzer_pwm(uint gpio, uint16_t frequency, uint16_t duration_ms) {
    if (frequency == 0) return;

    gpio_set_function(gpio, GPIO_FUNC_PWM);
    uint slice = pwm_gpio_to_slice_num(gpio);
    uint channel = pwm_gpio_to_channel(gpio);

    float clock_div = 4.0f;
    pwm_set_clkdiv(slice, clock_div);

    uint32_t wrap_value = (125000000 / (clock_div * frequency)) - 1;
    pwm_set_wrap(slice, wrap_value);
    pwm_set_chan_level(slice, channel, wrap_value / 2);

    pwm_set_enabled(slice, true);
    sleep_ms(duration_ms);

    pwm_set_enabled(slice, false);
    gpio_set_function(gpio, GPIO_FUNC_SIO);
    gpio_set_dir(gpio, GPIO_OUT);
    gpio_put(gpio, 0);
}
```

- gpio_set_function(gpio, GPIO_FUNC_PWM); → Configura o pino GPIO como saída de PWM.
- uint slice = pwm_gpio_to_slice_num(gpio); → Obtém o número do slice PWM associado ao pino.

- pwm_set_clkdiv(slice, clock_div); → Define o divisor de clock do PWM.
- pwm_set_wrap(slice, wrap_value); → Define o período do PWM.
- pwm_set_chan_level(slice, channel, wrap_value / 2); → Define o duty cycle do PWM.
- pwm_set_enabled(slice, true); → Habilita o PWM no pino especificado.

4. Configuração do Display OLED via I2C

A comunicação com o display OLED SSD1306 é realizada via protocolo I2C.

```
void setup_display() {
    i2c_recovery();
    i2c_init(I2C_PORT, 400 * 1000);

    gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
    gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
    gpio_pull_up(I2C_SDA);
    gpio_pull_up(I2C_SCL);

    ssd1306_init(&ssd, WIDTH, HEIGHT, false, endereco, I2C_PORT);
    ssd1306_config(&ssd);
    ssd1306_send_data(&ssd);
}
```

- i2c_init(I2C_PORT, 400 * 1000); → Inicializa o barramento I2C com velocidade de 400kHz.
- gpio_set_function(I2C_SDA, GPIO_FUNC_I2C); e gpio_set_function(I2C_SCL, GPIO_FUNC_I2C); → Configuram os pinos SDA e SCL como pinos de comunicação I2C.
- gpio_pull_up(I2C_SDA); e gpio_pull_up(I2C_SCL); → Ativam resistores de pull-up nos pinos de comunicação.

5. Configuração da Matriz de LEDs

A matriz de LEDs é controlada via PIO (Programável Input/Output).

```
void PIO_setup(PIO *pio, uint *sm) {
    *pio = pio0;
    uint offset = pio_add_program(*pio, &pio_matrix_program);
    *sm = pio_claim_unused_sm(*pio, true);
    pio_matrix_program_init(*pio, *sm, offset, LED_PIN);
}
```

- pio_add_program(*pio, &pio_matrix_program); → Carrega o programa PIO na memória do microcontrolador.
- *sm = pio_claim_unused_sm(*pio, true); → Obtém um estado de máquina disponível para execução do programa PIO.
- pio_matrix_program_init(*pio, *sm, offset, LED_PIN); → Configura a execução do programa PIO nos LEDs.

A função draw_pio() é usada para enviar comandos à matriz de LEDs:

```
void draw_pio(pixel *draw, PIO pio, uint sm, float intensity) {
   for (int16_t i = 0; i < PIXELS; i++) {
      uint32_t valor_led = matrix_rgb(draw[i].red, draw[i].green, draw[i].blue, intensity);
      pio_sm_put_blocking(pio, sm, valor_led);
   }
}
```

- pio_sm_put_blocking(pio, sm, valor_led); → Envia dados para o estado de máquina da PIO, atualizando os LEDs.

Protocolo de comunicação

Protocolo I²C (Inter-Integrated Circuit)
O **I²C** é um protocolo de comunicação serial síncrono que permite a comunicação entre múltiplos dispositivos usando apenas duas linhas:

SDA (Serial Data Line) – Linha de dados.

SCL (Serial Clock Line) – Linha de clock.

No software, o **microcontrolador (RP2040)** atua como **mestre** e o **display OLED SSD1306** como **escravo**, trocando comandos e dados via I²C.
Configuração do I²C

Metodologia

O desenvolvimento do projeto foi realizado de forma iterativa, seguindo um fluxo de **pesquisa, teste, implementação e otimização** dos componentes de hardware e software. As etapas do projeto foram estruturadas para garantir uma abordagem sistemática na construção do sistema de detecção e exibição de notas musicais.

1. Pesquisa e Fundamentação Teórica

O projeto teve início com uma pesquisa sobre métodos de detecção de notas musicais via hardware, abordando diferentes estratégias de captação e análise de sinais acústicos. Foram estudadas abordagens comuns na literatura, incluindo:

- **Transformada Rápida de Fourier (FFT)** para análise espectral do sinal.
- **Interpolação parabólica** para refinar a detecção da frequência dominante.

2. Estudo e Testes dos Periféricos da Placa BitdogLab

Após a pesquisa inicial, iniciou-se o estudo detalhado dos periféricos disponíveis na BitdogLab. Os módulos foram testados individualmente, garantindo a compreensão de suas capacidades e limitações. Entre os periféricos avaliados, destacam-se:

- **Módulo ADC (Conversor Analógico-Digital)**:
  - Configuração da taxa de amostragem para captação eficiente do áudio do microfone.
- **Buzzer com PWM (Modulação por Largura de Pulso)**:
  - Avaliação da capacidade de gerar diferentes frequências para reprodução de notas musicais.
  - Definição do duty cycle ideal para emissão sonora clara e audível.
- **Display OLED (SSD1306, via I²C)**:
  - Configuração e inicialização do barramento I²C.
  - Testes com diferentes métodos de exibição de caracteres e gráficos.
- **Matriz de LEDs (Controlada via PIO – Programável Input/Output)**:
  - Testes de diferentes padrões de exibição para representação visual das notas.
  - Configuração da intensidade luminosa e sincronização com as demais saídas do sistema.

Cada módulo foi implementado e validado separadamente, garantindo que suas funcionalidades fossem compreendidas e integradas de forma eficiente.

3. Implementação do Processamento de Sinais

Após a validação dos periféricos, iniciou-se a implementação das ferramentas matemáticas para análise do áudio captado pelo microfone. O pipeline de processamento de sinais foi estruturado em três fases principais:

3.1. Captura do Sinal de Áudio

- A taxa de amostragem foi definida em **44.1 kHz**, garantindo precisão suficiente para análise de áudio.
- O sinal foi armazenado em um buffer de **1024 amostras**, permitindo a aplicação da FFT sobre um intervalo adequado para a identificação de frequências musicais.

3.2. Aplicação da Transformada Rápida de Fourier (FFT)

- O algoritmo de **FFT** foi implementado para converter o sinal captado do domínio do tempo para o domínio da frequência.
- As componentes espectrais do sinal foram extraídas, e os valores de magnitude foram calculados para identificar os picos de frequência.

3.3. Interpolação para Detecção Precisa da Frequência Dominante

- Como a FFT fornece apenas valores discretos de frequência, foi aplicada uma interpolação parabólica para refinar a localização exata do pico.
- Esse refinamento aumentou a precisão da correspondência entre frequência detectada e nota musical.

4. Integração dos Módulos e Expansão das Funcionalidades

Conforme o projeto avançou, novos módulos foram adicionados para enriquecer a experiência do usuário e tornar o sistema mais interativo e intuitivo. As principais expansões foram:

4.1. Implementação do Buzzer para Emissão de Notas

- Um buzzer foi integrado ao sistema para **emitir as notas correspondentes às frequências detectadas**.
- O PWM foi configurado para gerar diferentes frequências, com um duty cycle otimizado para melhor resposta sonora.
- Um sistema de interrupções foi adicionado para ativação do buzzer mediante entrada do usuário.

4.2. Feedback Visual no Display OLED

- O display OLED foi configurado para exibir a frequência detectada e a nota correspondente, proporcionando feedback imediato ao usuário.
- Foram implementados menus interativos para alternar entre os modos detecção e emissão de notas.

4.3. Representação Gráfica com Matriz de LEDs

- Para melhorar a visualização, uma matriz de LEDs foi utilizada para representar a nota musical detectada.
- Cada nota foi associada a um padrão de cores, facilitando a identificação visual das frequências processadas.
- A comunicação com a matriz foi feita via PIO (Programável Input/Output), permitindo atualização rápida dos LEDs.

Resultados

O projeto resultou em um sistema funcional de detecção e reprodução de notas musicais, cumprindo todas as exigências do trabalho final do Embarcatech. Utilizando apenas os periféricos presentes na placa BitdogLab, a implementação demonstrou um uso inovador e criativo dos recursos disponíveis, explorando técnicas avançadas de processamento de sinais e controle de hardware embarcado.

O sistema desenvolvido foi capaz de captar e analisar sinais sonoros do microfone, aplicando a Transformada Rápida de Fourier (FFT) e interpolação para identificar a frequência dominante. Com essa abordagem, foi possível detectar notas musicais com precisão e fornecer feedback visual imediato no

display OLED e na matriz de LEDs, garantindo maior interatividade para o usuário.

Além da detecção, o software também permitiu a reprodução de notas musicais através do buzzer, utilizando um sinal PWM controlado por software para gerar frequências correspondentes às notas detectadas. Isso possibilitou não apenas a análise do som ambiente, mas também a interação direta do usuário com a emissão sonora.

Desafios e Ajustes na Detecção de Notas

Durante os testes, foi observada a interferência de ruídos externos no ambiente de leitura do microfone. Esses ruídos afetaram a precisão da detecção, principalmente em frequências mais baixas. No entanto, a implementação da interpolação parabólica e a normalização dos dados do ADC contribuíram para minimizar esse efeito, permitindo a identificação mais estável das notas musicais.

Outro ajuste importante foi a limitação das frequências reproduzidas pelo buzzer. Observou-se que as notas emitidas na 6ª oitava apresentavam a melhor correspondência sonora em relação às frequências esperadas. Isso pode estar relacionado às características acústicas do buzzer ou à resposta do sistema PWM utilizado. Dessa forma, a reprodução de notas foi restrita à 6ª oitava, garantindo uma melhor experiência auditiva para o usuário.

Destaques do Projeto

Os principais resultados obtidos incluem:

- Detecção eficiente de notas musicais a partir da captação de áudio e análise espectral via FFT.
- Representação visual interativa da nota detectada no display OLED e na matriz de LEDs.
- Capacidade de reprodução de notas musicais através do buzzer, permitindo que o usuário interaja diretamente com o sistema.
- Uso otimizado dos periféricos da placa BitdogLab, demonstrando a viabilidade do projeto sem a necessidade de hardware externo adicional.
- Aprimoramento da precisão da detecção com técnicas de interpolação e filtragem, minimizando os impactos de ruídos externos.

Conclusão

O sistema desenvolvido demonstrou um uso criativo e eficiente dos recursos embarcados, explorando técnicas de processamento digital de sinais e controle de hardware para a construção de um analisador e reprodutor de notas musicais. Apesar dos desafios encontrados, os ajustes realizados garantiram um desempenho satisfatório, permitindo a captação e reprodução de sons com precisão adequada para o propósito do projeto.

Este trabalho representa uma solução inovadora para a análise de frequências musicais em tempo real, integrando hardware e software de forma otimizada e proporcionando uma experiência didática e interativa para os usuários.

Referências

BITDOGLAB – Uma Jornada Educativa com Eletrônica, Embarcados e IA EMBARCADOS. **BitDogLab – Uma Jornada Educativa com Eletrônica, Embarcados e IA**. 2023. Disponível em: https://embarcados.com.br/bitdoglab-uma-jornada-educativa-com-eletronica-embarcados-e-ia/. Acesso em: 26 fev. 2025.

APRESENTANDO a placa de desenvolvimento BitDogLab YOUTUBE. **Apresentando a placa de desenvolvimento BitDogLab**. 2023. Disponível em: https://www.youtube.com/watch?v=aS0tE-y4iuQ. Acesso em: 26 fev. 2025.

ANÁLISE de Fourier UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. **Análise de Fourier**. UFRGS, 2023. Disponível em: https://www.ufrgs.br/reamat/TransformadasIntegrais/livro-af/livro.pdf. Acesso em: 26 fev. 2025.

DIAS, Bruno Rafael Rodrigues **Sistema Inteligente para Reconhecimento de Timbres**. Universidade Federal do Piauí, 2013. Disponível em: https://ufpi.br/arquivos_download/arquivos/PICOS/Not%C3%ADcias/PICOS_2022/Biblioteca/2013/Sistemas_da_Informa%C3%A7%C3%A3o_2013/Bruno_Rafael_Rodrigues_Dias.pdf. Acesso em: 26 fev. 2025.

SOUZA, H. A. S.; OLIVEIRA, H. M. **Desenvolvimento de um Sistema Computacional de Transcrição de Melodias Monofônicas para Partitura**. ResearchGate, 2007. Disponível em: https://www.researchgate.net/publication/237399879_Desenvolvimento_de_um_sistema_computacional_de_transcricao_de_melodias_monofonicas_para_partitura. Acesso em: 26 fev. 2025.

SILVANO, Guilherme de Nez. **Sistema de Reconhecimento de Escalas Musicais Utilizando Processamento Digital de Sinais.** 2019. Trabalho de Conclusão de Curso (Graduação em Engenharia

da Computação) – Universidade do Extremo Sul Catarinense, Criciúma, 2019. Disponível em: http://repositorio.unesc.net/bitstream/1/8858/1/Guilherme%20de%20Nez%20Silvano.pdf. Acesso em: 26 fev. 2025.