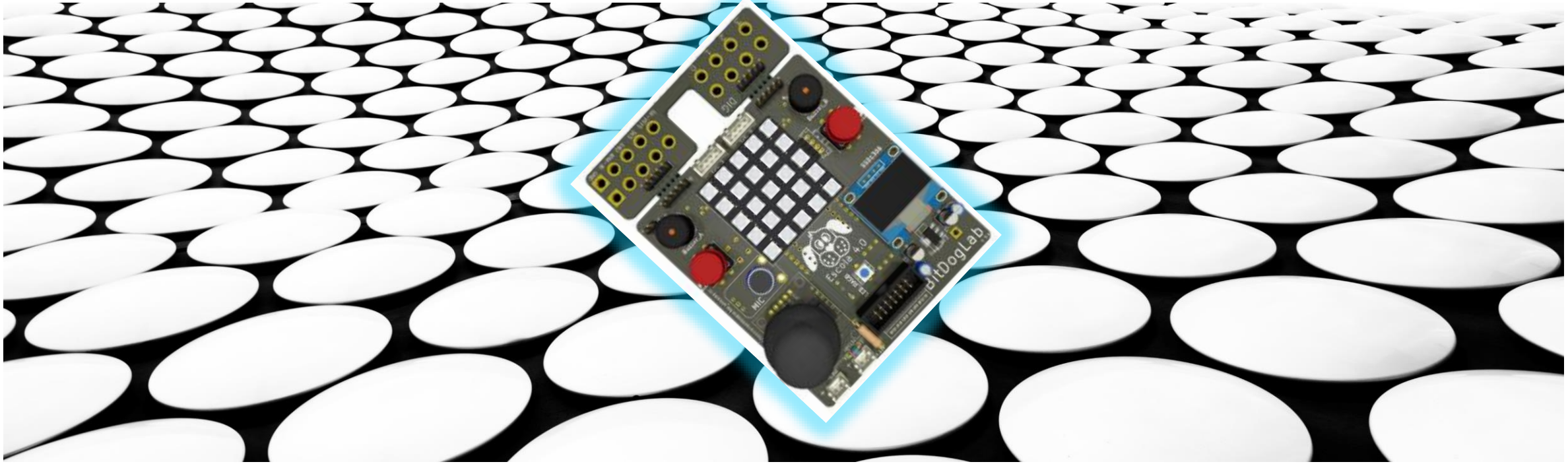
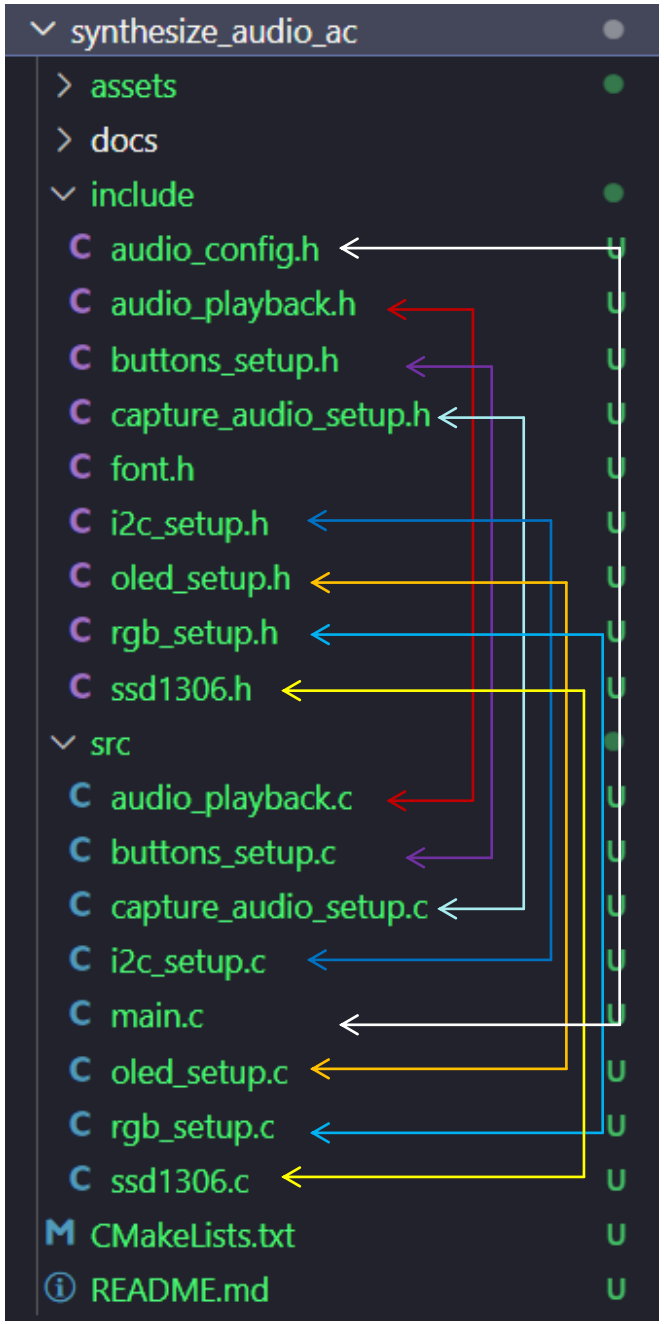
O Sintetizador de Áudio é um projeto desafiador que conecta conceitos fundamentais de eletrônica e acústica de maneira prática e criativa. Ao gerar e manipular ondas sonoras, este dispositivo demonstra como é possível produzir timbres variados por meio de ajustes nos parâmetros de frequência, forma de onda, filtros e efeitos. Neste projeto, vamos desenvolver um sintetizador de áudio utilizando sistemas embarcados.

PROJETO SINTETIZADOR DE ÁUDIO

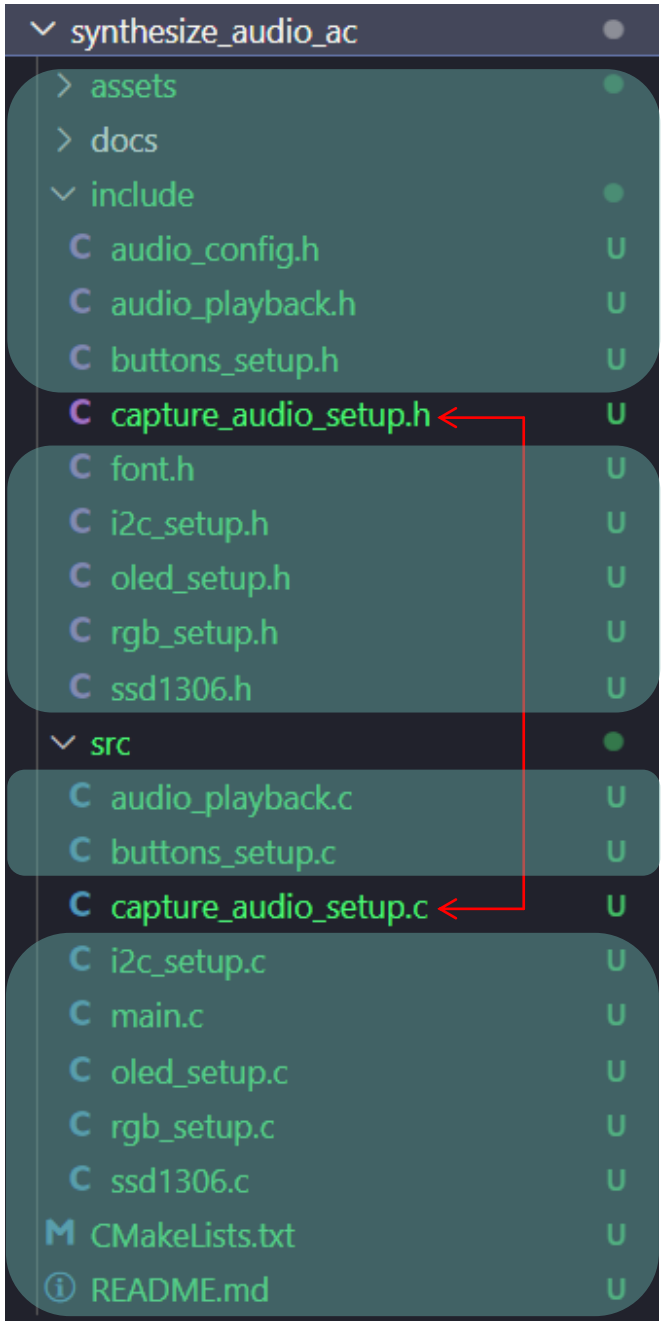
O TEM FOCO NOS SEGUINTES CONCEITOS – RELACIONADOS A ÁUDIO DIGITAL: ○ CAPTURA ○ ARMAZENAMENTO ○ PROCESSAMENTO ○ REPRODUÇÃO





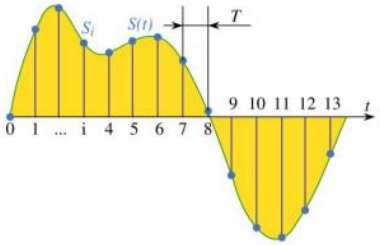
Visão Geral das Funções

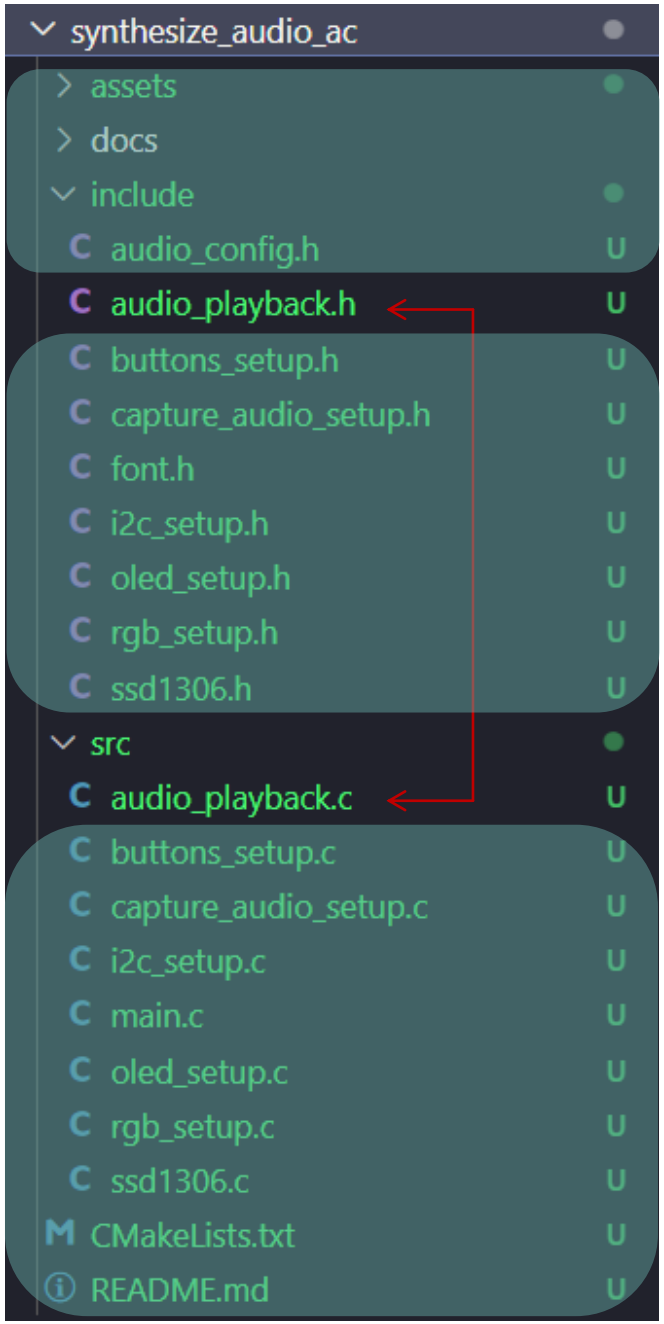
Este projeto implementa um **sintetizador de áudio completo** usando a BitDogLab, com foco em captura, armazenamento, processamento e reprodução de sinais sonoros. Cada unidade (.c e .h) do sistema contribui com uma funcionalidade essencial para alcançar esse objetivo...

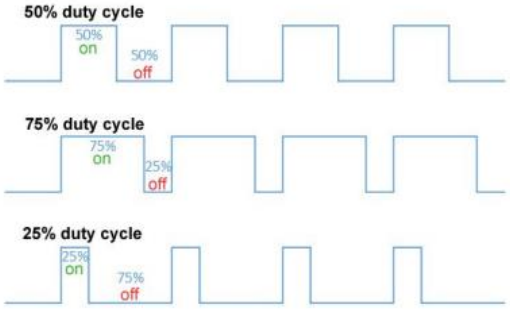


capture_audio_setup.c e .h

Realiza a **captura de áudio** analógico via ADC, com auxílio de DMA para transferir dados diretamente para o buffer. Esta etapa transforma sinais analógicos em digitais, armazenando as amostras na RAM...

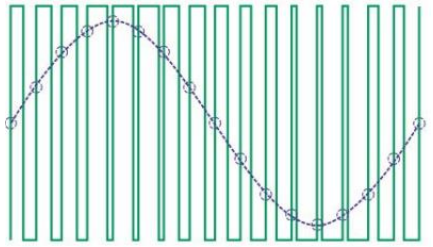


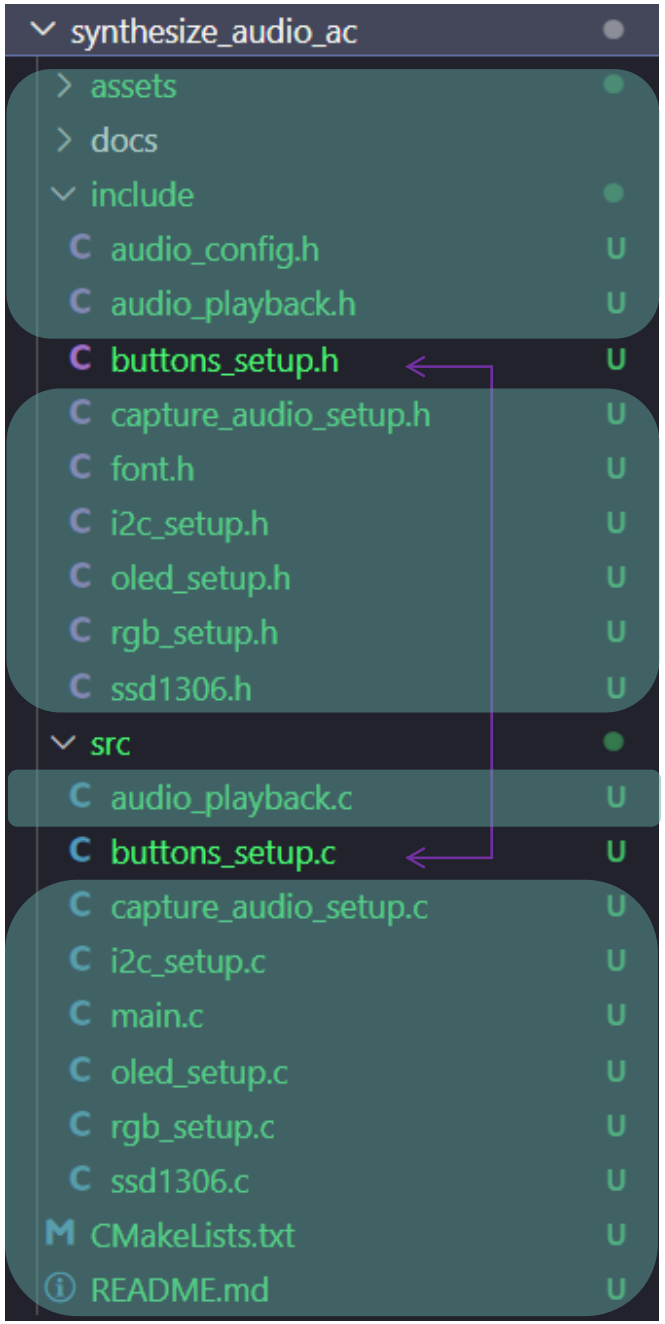


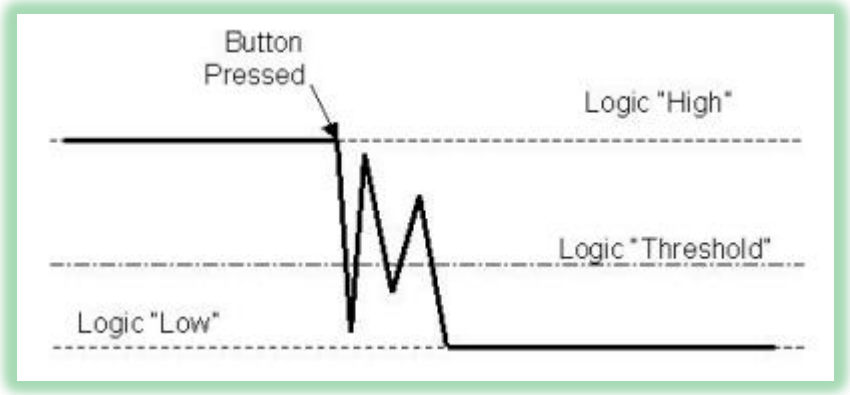


audio_playback.c e .h

Cuida da **reprodução do áudio digital**, convertendo as amostras em pulsos de PWM com duty cycle proporcional à amplitude da onda. Esse sinal PWM pode ser usado para acionar um atuador (ex.: buzzer) e gerar som...

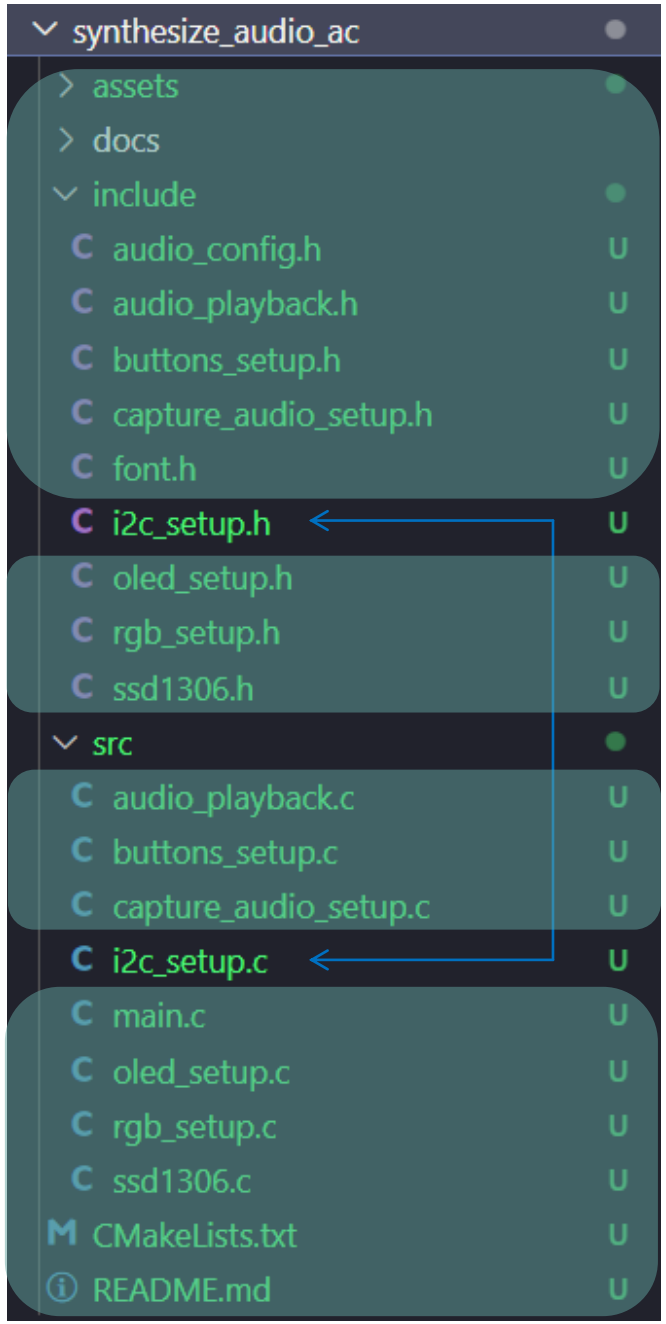


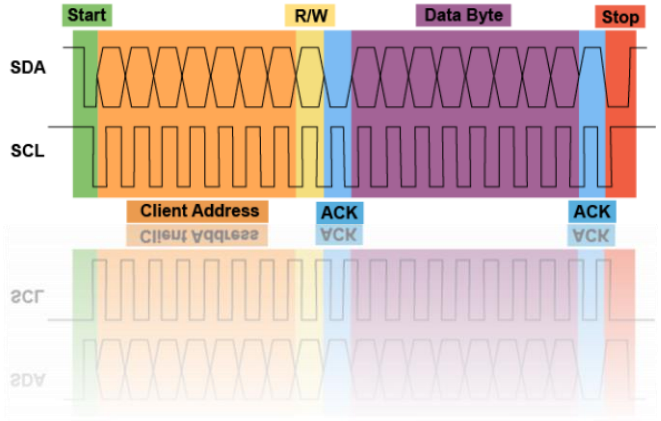




buttons_setup.c e .h

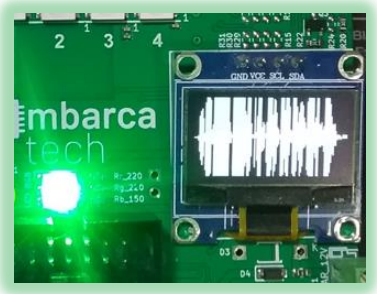
Inicializa os **botões físicos** usados para controlar a gravação e a reprodução. Define a lógica de polling dos botões A e B da BitDogLab...

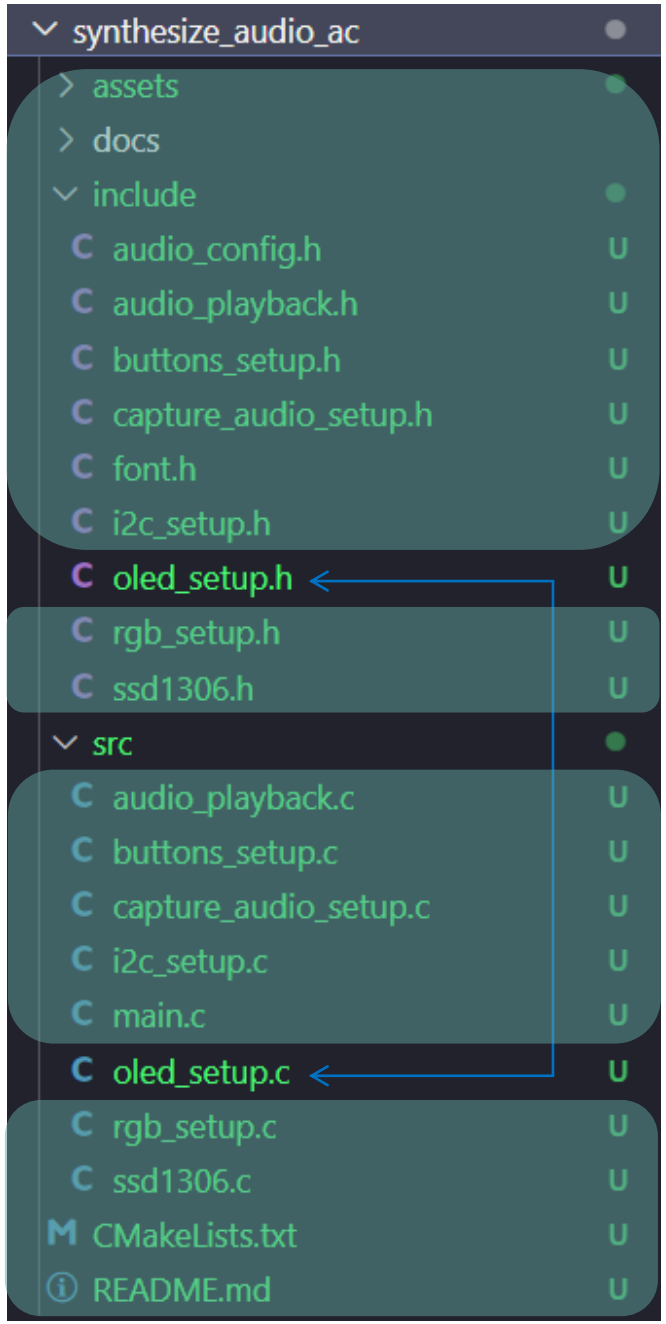




i2c_setup.c e .h

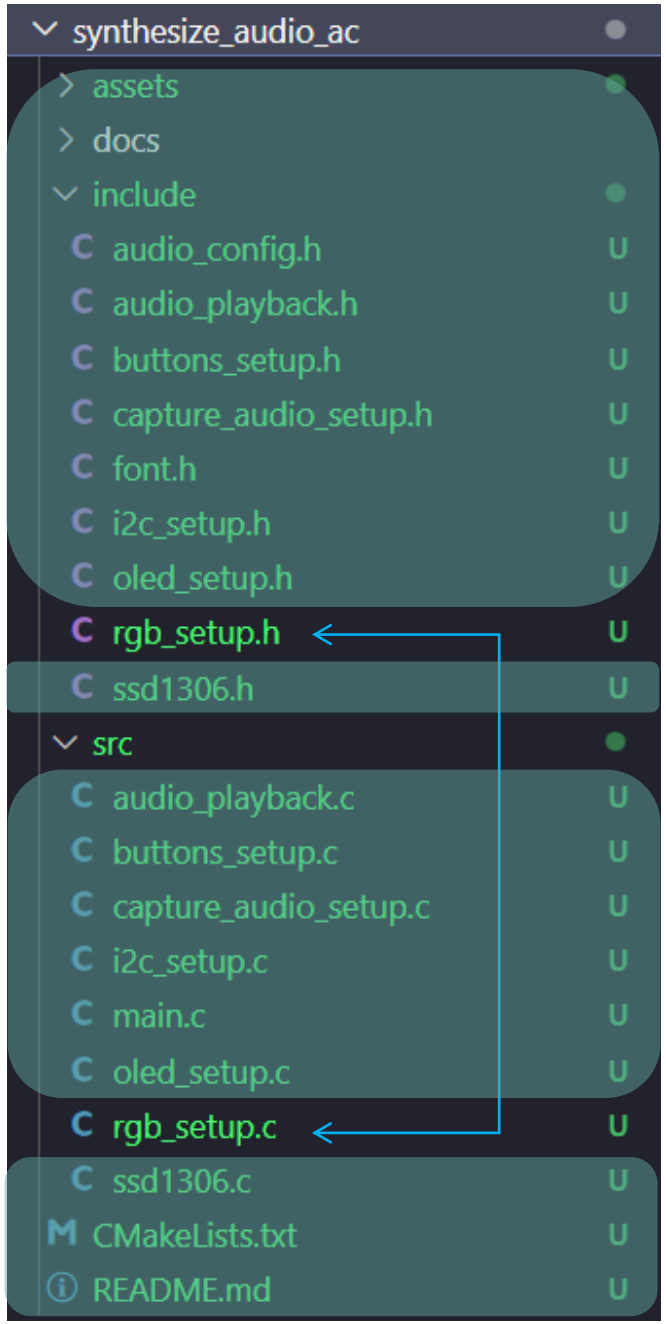
Configura o barramento **I2C** utilizado para comunicação com o display OLED, essencial para exibir a forma de onda do áudio...





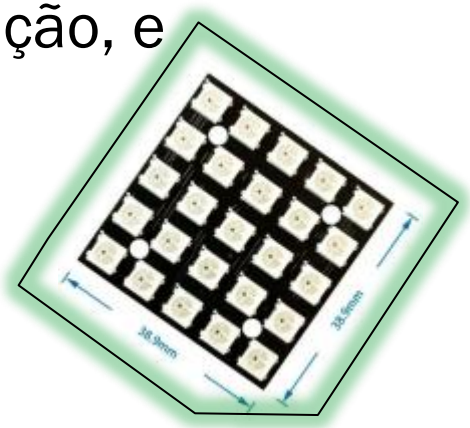
oled_setup.c e .h

Controla o **display OLED**. Mostra a forma de onda das amostras capturadas, convertendo os dados do buffer em uma representação gráfica da amplitude ao longo do tempo...

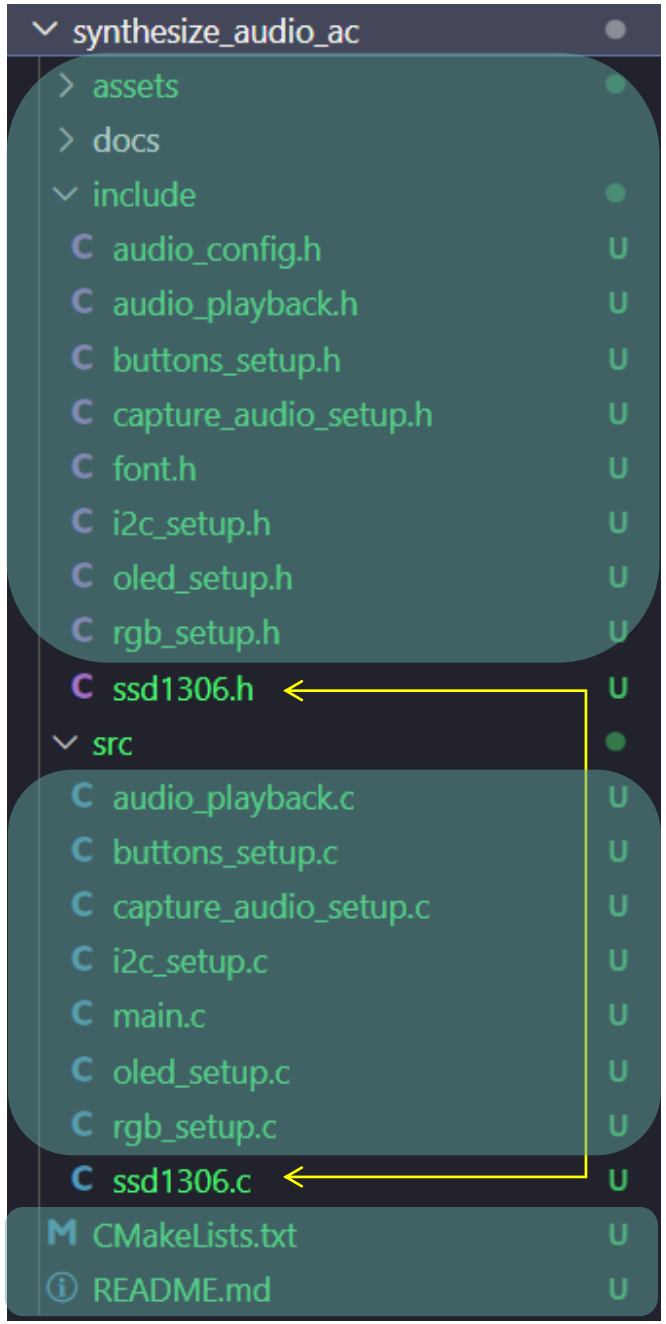


rgb_setup.c e .h

Gerencia os **LEDs RGB**, fornecendo **feedback visual:** vermelho durante a gravação, verde na reprodução, e desligado quando o sistema está inativo...

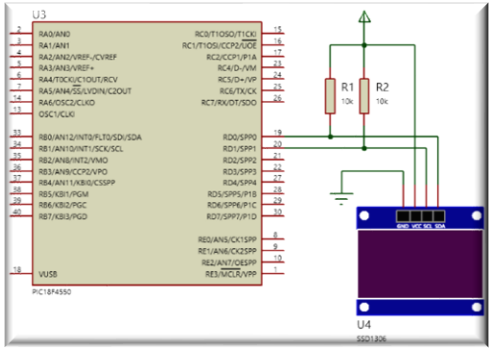


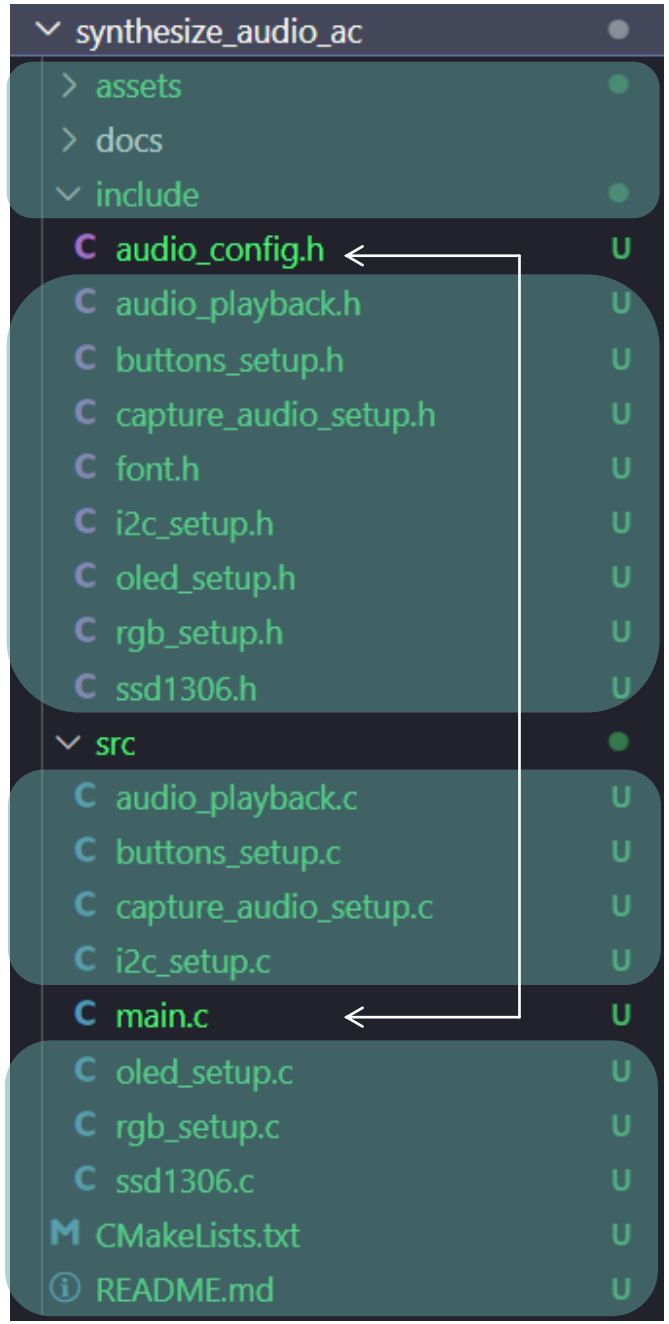
https://github.com/BitDogLab/BitDogLab https://hardware.org.br/ https://github.com/alfecjo



ssd1306.c e .h

Biblioteca gráfica usada internamente por oled_setup.c para enviar comandos e desenhar no display OLED compatível com o controlador SSD1306...





main.c juntamente com **audio_config.h**

Coordena o sistema como um todo. Inicializa os periféricos (ADC, PWM, I2C, botões, LEDs, display OLED) e define o loop principal de controle, escutando os botões para iniciar gravação e reprodução...

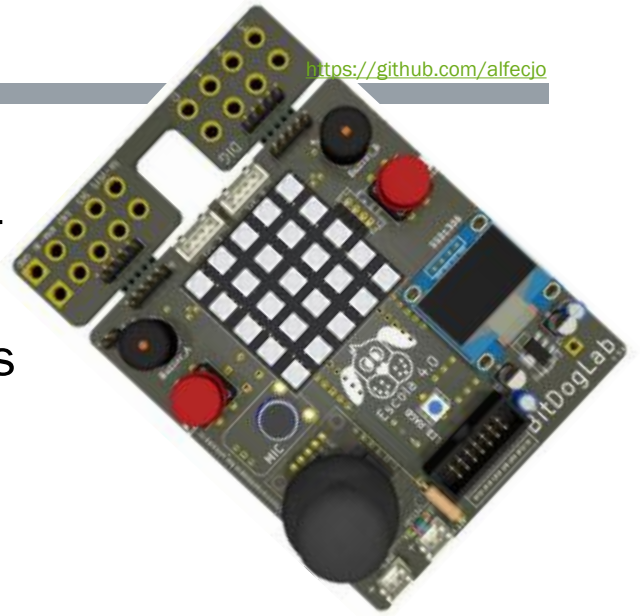
Arquitetura Modular: Clareza, Reusabilidade e Eficiência no Desenvolvimento

A estrutura do projeto foi cuidadosamente dividida em **unidades modulares compostas por arquivos** .c (implementação) e .h (interface), respeitando **princípios fundamentais da engenharia de software embarcado**, como **alta coesão**, **baixo acoplamento** e **reusabilidade**.

Essa abordagem traz diversas vantagens:

🔁 1. Reusabilidade

Cada módulo trata de uma responsabilidade clara (como controle de LED, gravação de áudio, reprodução, visualização OLED) e pode ser facilmente reaproveitado em outros projetos com finalidades diferentes, desde que compartilhem da mesma função básica (ex: um projeto futuro que use o mesmo display OLED ou uma rotina de DMA similar).



🧩 2. Alta Coesão

O código de cada arquivo .c lida **exclusivamente com uma tarefa específica** — por exemplo, o audio_playback.c trata somente da reprodução PWM. Isso facilita entender, testar e ajustar funcionalidades individualmente, sem efeitos colaterais em outras partes do sistema.

🔒 3. Baixo Acoplamento

A separação em módulos permite que cada parte dependa **apenas do que precisa saber.** O main.c, por exemplo, apenas orquestra os módulos sem conhecer os detalhes internos de como o PWM, DMA ou display funcionam. Essa independência entre as partes torna o sistema mais **flexível a mudanças**.

🛠️ 4. Portabilidade

Caso a aplicação precise ser portada para outra placa (de outro fabricante, por exemplo), os módulos de interface direta com o hardware podem ser **trocados ou adaptados isoladamente**, mantendo intacta a lógica geral da aplicação. Basta reescrever o módulo de baixo nível (ex: substituindo o driver de ADC ou PWM), sem impactar o restante do código.

👥 5. Facilidade de Manutenção e Entendimento

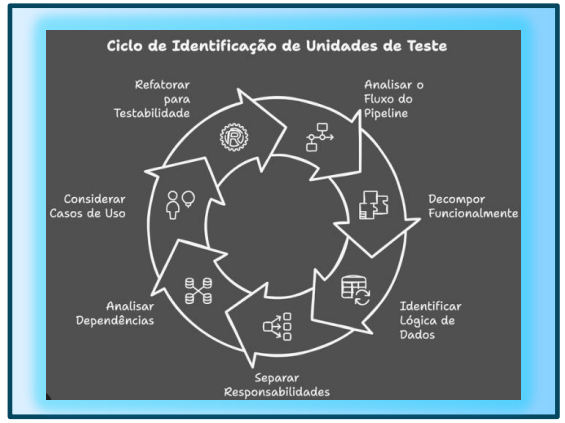
Ao seguir esse estilo de organização, um novo desenvolvedor pode rapidamente compreender a arquitetura do sistema. Ao abrir os arquivos, a relação entre os nomes e as funcionalidades é direta, o que evita a necessidade de "decifrar a mente" do programador original. Isso acelera a manutenção, depuração e extensão do código.

🧪 6. Testabilidade

Cada módulo pode ser testado individualmente, seja por simulação, por testes de unidade ou testes em hardware. Isso é essencial para isolar bugs e verificar o comportamento de partes específicas do sistema com mais precisão.

🔧 7. Evolução com Baixo Impacto

Adicionar novos recursos ou alterar funcionalidades existentes é seguro e controlado. Por exemplo, é possível trocar o modo de visualização do display sem afetar a gravação ou reprodução de áudio, pois cada parte atua de forma **independente e bem definida**.



🔄 8. Modularidade e Escalabilidade

O estilo adotado promove a **evolução progressiva da aplicação**. Novos módulos podem ser adicionados facilmente (ex: um analisador de frequência ou um encoder de áudio), aproveitando a base já existente, sem reescrever código antigo.

🎯 Conclusão

A separação por responsabilidades em arquivos .c/.h torna o código **mais limpo, compreensível, portátil e sustentável.** Este modelo modular é um diferencial profissional, que prepara o projeto não apenas para funcionar bem **hoje**, mas para **crescer, ser reutilizado e mantido no futuro com segurança e eficiência**.