Documentação da Biblioteca `BuzzerPi`

Documentação da Biblioteca `BuzzerPi`

Introdução

A biblioteca **BuzzerPi** foi desenvolvida para facilitar o controle de um buzzer ou alto-falante piezoelétrico no Raspberry Pi Pico. Ela utiliza o módulo PWM (Pulse Width Modulation) do Pico para gerar tons em diferentes frequências, permitindo a reprodução de notas musicais, melodias e efeitos sonoros. A biblioteca oferece funções para tocar tons individuais, melodias completas e até mesmo "beeps" repetidos, com controle sobre a frequência, duração e divisão do clock.

Essa biblioteca é ideal para projetos que envolvem feedback sonoro, como alarmes, notificações, jogos simples ou qualquer aplicação que necessite de geração de áudio básica.

Propósito da Biblioteca

A biblioteca **BuzzerPi** foi criada para:

1. **Gerar Tons Audíveis:**
   - Utilizar o PWM para gerar tons em diferentes frequências, permitindo a reprodução de notas musicais.
2. **Tocar Melodias:**
   - Reproduzir sequências de notas (melodias) com controle sobre a duração de cada nota.
3. **Efeitos Sonoros Simples:**
   - Criar efeitos sonoros como "beeps" repetidos para notificações ou alertas.
4. **Controle de Frequência e Duração:**
   - Permitir o ajuste fino da frequência do tom e da duração de cada nota ou efeito sonoro.

Funcionalidades Principais

A biblioteca **BuzzerPi** oferece as seguintes funcionalidades:

1. **Inicialização do PWM:**
   - Configura um pino GPIO para funcionar como saída PWM, permitindo a geração de tons.
2. **Cálculo de Parâmetros PWM:**

- Calcula automaticamente os valores de "wrap" e divisão do clock para gerar a frequência desejada.

3. **Reprodução de Tons:**
   - Toca um tom em uma frequência específica por uma duração determinada.
4. **Reprodução de Melodias:**
   - Toca uma sequência de notas (melodia) com durações específicas.
5. **Efeitos Sonoros:**
   - Reproduz "beeps" repetidos, útil para notificações ou alertas.

Documentação Detalhada das Funções

Aqui está uma explicação detalhada de todas as funções da biblioteca, com foco em seu propósito e comportamento.

Função: `void initialize_pwm(uint pin)`

Propósito:
Configura um pino GPIO para funcionar como saída PWM, permitindo a geração de tons.

Parâmetros:

- `uint pin`: O número do pino GPIO que será configurado como saída PWM.

Comportamento:

1. Configura o pino GPIO para usar a função PWM usando `gpio_set_function()`.

Exemplo de Uso:

```
initialize_pwm(15); // Configura o pino 15 como saída PWM
```

Função: `uint16_t calculate_wrap(uint32_t target_frequency, float clkdiv)`

Propósito:
Calcula o valor de "wrap" necessário para gerar uma frequência específica com um determinado divisor de clock.

Parâmetros:

- `uint32_t target_frequency`: A frequência desejada do tom (em Hz).
- `float clkdiv`: O divisor de clock a ser usado.

Valor de Retorno:

- O valor de "wrap" calculado, que é usado para configurar o PWM.

Comportamento:

1. Obtém a frequência do clock do sistema usando `clock_get_hz(clk_sys)`.
2. Calcula o valor de "wrap" com base na frequência desejada e no divisor de clock.
3. Retorna o valor de "wrap", limitado a 65535 (o valor máximo suportado pelo PWM).

Exemplo de Uso:

```
uint16_t wrap = calculate_wrap(440, 1.0); // Calcula o wrap para 440 Hz (nota Lá)
```

Função: `void play_tone(uint pin, uint32_t freq, uint duration_ms)`

Propósito:
Toca um tom em uma frequência específica por uma duração determinada.

Parâmetros:

- `uint pin`: O pino GPIO onde o buzzer está conectado.
- `uint32_t freq`: A frequência do tom (em Hz).
- `uint duration_ms`: A duração do tom (em milissegundos).

Comportamento:

1. Obtém o número do slice PWM associado ao pino usando `pwm_gpio_to_slice_num()`.
2. Calcula o valor de "wrap" necessário para a frequência desejada.
3. Configura o valor de "wrap" e o divisor de clock no slice PWM.
4. Define o nível do PWM para 50% do valor de "wrap" (duty cycle de 50%).
5. Habilita o PWM e toca o tom pelo tempo especificado.
6. Desliga o PWM após a duração do tom.

Exemplo de Uso:

```
play_tone(15, 440, 1000); // Toca um Lá (440 Hz) por 1 segundo no pino 15
```

Função: `void play_tone_clkdiv(uint pin, int freq, int duration_ms, float clkdiv)`

Propósito:
Toca um tom em uma frequência específica por uma duração determinada, com controle sobre o divisor de clock.

Parâmetros:

- `uint pin`: O pino GPIO onde o buzzer está conectado.
- `int freq`: A frequência do tom (em Hz).
- `int duration_ms`: A duração do tom (em milissegundos).
- `float clkdiv`: O divisor de clock a ser usado.

Comportamento:

1. Obtém o número do slice PWM associado ao pino usando `pwm_gpio_to_slice_num()`.
2. Calcula o valor de "wrap" necessário para a frequência desejada e o divisor de clock especificado.
3. Configura o valor de "wrap" e o divisor de clock no slice PWM.
4. Define o nível do PWM para 50% do valor de "wrap" (duty cycle de 50%).
5. Habilita o PWM e toca o tom pelo tempo especificado.
6. Desliga o PWM após a duração do tom.

Exemplo de Uso:

```
play_tone_clkdiv(15, 440, 1000, 2.0); // Toca um Lá (440 Hz) por 1 segundo com divisor de clock 2.0
```

Função: `void play_melody(uint pin, int *melody, int *durations, float clkdiv, int length)`

Propósito:
Toca uma sequência de notas (melodia) com durações específicas.

Parâmetros:

- `uint pin`: O pino GPIO onde o buzzer está conectado.
- `int *melody`: Um array de frequências das notas da melodia.
- `int *durations`: Um array de durações (em milissegundos) para cada nota.
- `float clkdiv`: O divisor de clock a ser usado.
- `int length`: O número de notas na melodia.

Comportamento:

1. Itera sobre o array de notas e durações.
2. Para cada nota, chama `play_tone_clkdiv()` para tocar a nota com a duração especificada.
3. Se a nota for 0 (silêncio), pausa pelo tempo especificado.

Exemplo de Uso:

```
int melody[] = {440, 494, 523}; // Notas Lá, Si, Dó
int durations[] = {500, 500, 500}; // Durações de 500 ms cada
```

```
play_melody(15, melody, durations, 1.0, 3); // Toca a melodia no pino 15
```

Função: `void beep(uint pin, int freq, int duration, int repetition)`

Propósito:
Reproduz "beeps" repetidos, útil para notificações ou alertas.

Parâmetros:

- `uint pin`: O pino GPIO onde o buzzer está conectado.
- `int freq`: A frequência do "beep" (em Hz).
- `int duration`: A duração de cada "beep" (em milissegundos).
- `int repetition`: O número de repetições do "beep".

Comportamento:

1. Repete o "beep" pelo número de vezes especificado.
2. Toca o "beep" usando `play_tone()` e pausa por 500 ms entre cada "beep".

Exemplo de Uso:

```
beep(15, 440, 200, 3); // Toca 3 "beeps" de 440 Hz com duração de 200 ms no pino 15
```

Conclusão

A biblioteca **BuzzerPi** fornece uma maneira simples e eficiente de controlar um buzzer ou alto-falante piezoelétrico no Raspberry Pi Pico. Com funções para tocar tons individuais, melodias completas e efeitos sonoros repetidos, essa biblioteca é ideal para projetos que necessitam de feedback sonoro. Com essa documentação detalhada, você está pronto para integrar a biblioteca em seus projetos e aproveitar ao máximo suas funcionalidades. Para mais detalhes, consulte o arquivo `BuzzerPi.c`.