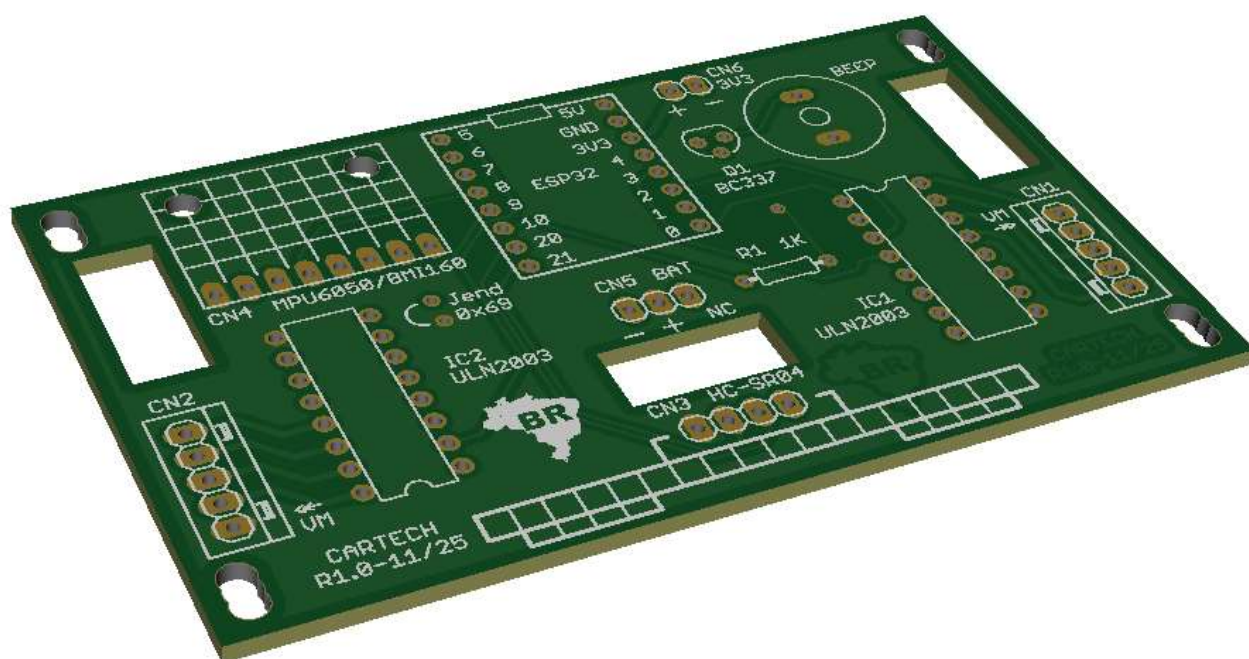


# PCI CARTECH



## MANUAL DO USUÁRIO

Aplicável as placas:

CARTECH versão 1.0 e posteriores  
com o módulo microcontrolador ESP32 C3 supermini de 16 pinos

Manual rev. 1.0 — 10/2025

Eventuais atualizações desse manual podem ser encontradas em: <https://github.com/izyino/manuais>

## **INTRODUÇÃO:**

A presente placa de circuito impresso intitulada CARTECH foi concebida para facilitar a construção de dispositivos baseados em até dois motores de passo do tipo 28BYJ48, via WiFi, contemplando em um reduzido espaço de apenas 50 x 85 mm, o moderno microcontrolador ESP32 C3 em sua versão supermini de 16 pinos, os drivers dos motores de passo e todos os demais componentes, incluindo conectores para um sensor de proximidade por ultrassom e um sensor acelerômetro e giroscópio de três eixos.

A versão 1.0 da placa de circuito impresso CARTECH possui as seguintes características básicas:

- Utilização do microcontrolador ESP-32 C3 na versão supermini de 16 pinos, com WiFi e Bluetooth, CPU de 32-bits RISC-V Single-core 160MHz, WiFi: 802.11b/g/n 2.4GHz, Bluetooth 5.0, Consumo ultra baixo de apenas 43uA, 400KB SRAM, 384KB ROM, 4Mb flash
- Controle de dispositivos com base em até dois motores de passo modelo 28BYJ48 ou similar
- Suporta motores Nema17, alimentados com 5V, em ciclos de no máximo 15 segundos rodando com no mínimo 45 segundos em repouso
- Utilização dos driveres ULN2003, dip 16
- Beep sonoro passivo
- Conector para acomodar diretamente o sensor acelerômetro e giroscópio MPU6050 ou BMI610
- Conector para acomodar diretamente o sensor de proximidade HC-SR04
- Conector para fornecimento de +3V3 de baixa corrente
- Completa biblioteca para controle dos motores de passo, do beep sonoro e do Led azul, bem como um controle de tempo, não bloqueante, em alternativa às funções millis() e delay(), disponível em <https://github.com/izyino/motbepled>

## **CONECTORES EXISTENTES NA PLACA CARTECH:**

### **CN1 – motor de passo n° 0**

- 1 – L1 fio azul (GPIO 3 do ESP32C3 → driver)
- 2 – L1 fio rosa (GPIO 2 do ESP32C3 → driver)
- 3 – L2 fio amarelo (GPIO 1 do ESP32C3 → driver)
- 4 – L2 fio laranja (GPIO 0 do ESP32C3 → driver)
- 5 – 5vcc - fio vermelho

### **CN2 – motor de passo n° 1**

- 1 – L1 fio azul (GPIO 7 do ESP32C3 → driver)
- 2 – L1 fio rosa (GPIO 6 do ESP32C3 → driver)
- 3 – L2 fio amarelo (GPIO 5 do ESP32C3 → driver)
- 4 – L2 fio laranja (GPIO 4 do ESP32C3 → driver)
- 5 – 5vcc - fio vermelho

### **CN3 – sensor ultrasom HC-SR04:**

- 1 – Vcc (+5V)
- 2 – Trigger (GPIO 21 do ESP32C3)
- 3 – Eco (GPIO 20 do ESP32C3)
- 4 – Gnd

### **CN4 – sensor acelerômetro e giroscópio MPU6050 ou BMI610:**

- 1 – Vcc (+5V)
- 2 – Gnd
- 3 – SCL (GPIO 9 do ESP32C3)
- 4 – SDA (GPIO 8 do ESP32C3)
- 5,6,7,8 – Sem conexão

### **CN5 – bateria:**

- 1 – Gnd
- 2 – Vcc (+5V)
- 3 – Sem conexão (polarizador)

### **CN6 – +3V3:**

- 1 – Vcc (+3V3 do ESP32C3)
- 2 – Gnd

**RELAÇÃO DE COMPONENTES:**

1 x Placa de circuito impresso Rev.1.0  
1 x Módulo ESP32 C3 supermini, 16 pinos  
2 x CI driver ULN2003, dip 16  
2 x Conector molex 5 pinos com polarizador, CN1 e CN2 para os motores de passo)  
1 x Beep passivo  
1 x Transistor BC337  
1 x Resistor de 1K $\Omega$   
1 x Conector 4 pinos fêmea (CN3 – ultrassom HC-SR04) \*\*  
1 x Conector 8 pinos fêmea (CN4 – acelerômetro e giroscópio MPU6050 ou BMI610) \*\*  
1 x Conector 3 pinos macho (CN5 – bateria)  
1 x Conector 2 pinos macho (CN6 – 3V3)

\*\* Esse conectores não necessários se os sensores forem soldados diretamente na placa. A soldagem dos sensores diretamente na placa é altamente recomendada, para que os sensores sejam mantidos firmemente sempre na mesma posição

**OBSERVAÇÃO:** O sensor acelerômetro e giroscópio MPU6050 ou BMI610 instalado na placa CARTECH poderá ter dois possíveis endereços I2C, definidos pelo jumper Jend: sem jumper = 0x68, com jumper = 0x69

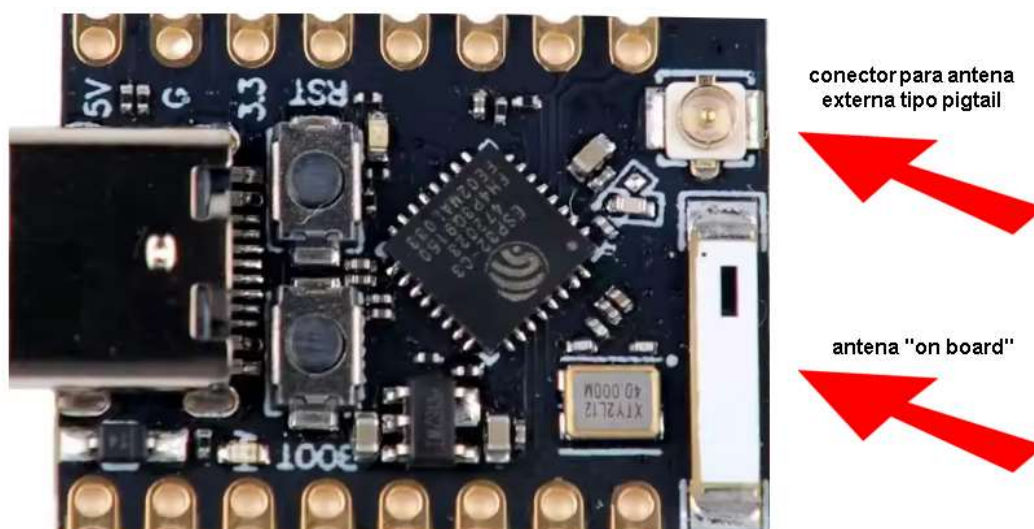


Fig. 1 – Modelo de ESP32 C3 recomendado

Existe diversos modelos diferentes de ESP32 C3 disponíveis no mercado. O modelo que mostrou mais facilidade de conexão e melhor desempenho, especialmente no tocante ao alcance WiFi é o mostrado na figura 1 acima.

O modelo mais comum, mostrado na figura 2 abaixo, **deve ser evitado** uma vez que esse modelo possui um alcance muito menor em virtude da sua antena "on board" ser visivelmente menos elaborada, sem contar com a ausência do conector para antena externa.

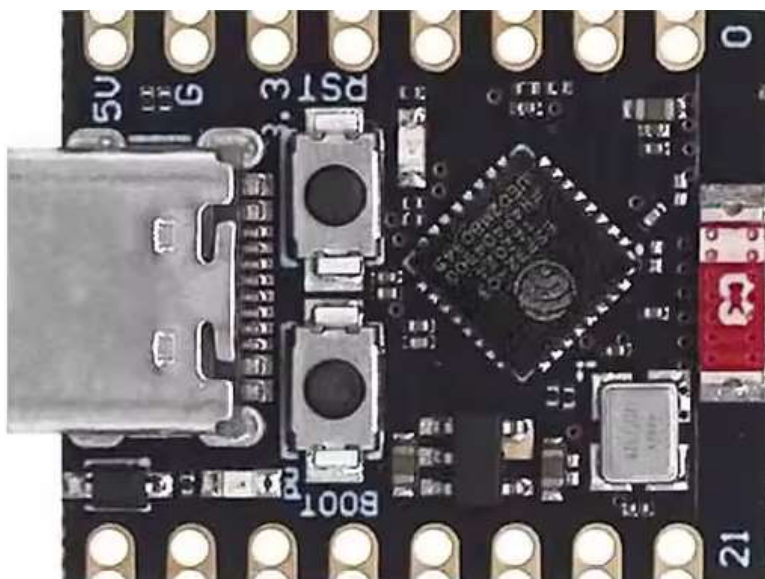


Fig. 2 – Modelo de ESP32 C3 **-NÃO-** recomendado

**DIAGRAMA ELÉTRICO:**

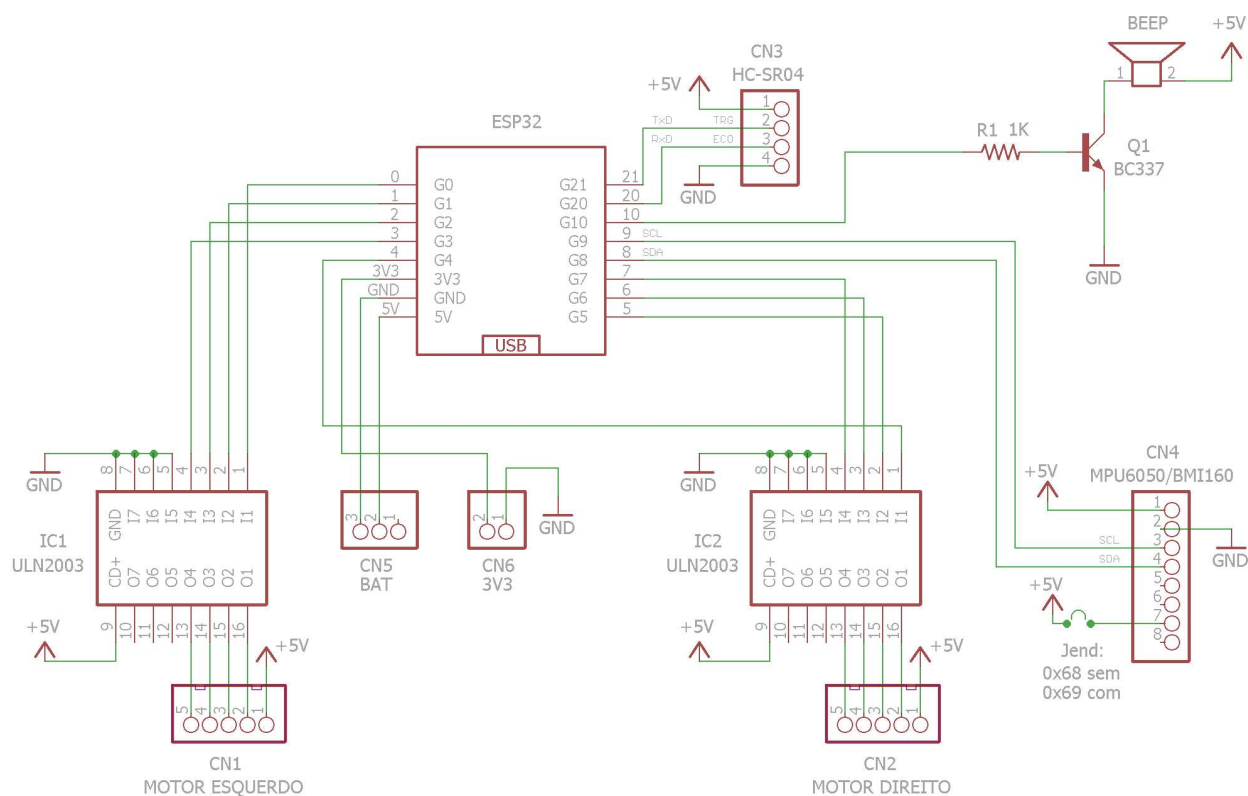


Fig. 3 – Diagrama elétrico da placa CARTECH Rev. 1.0

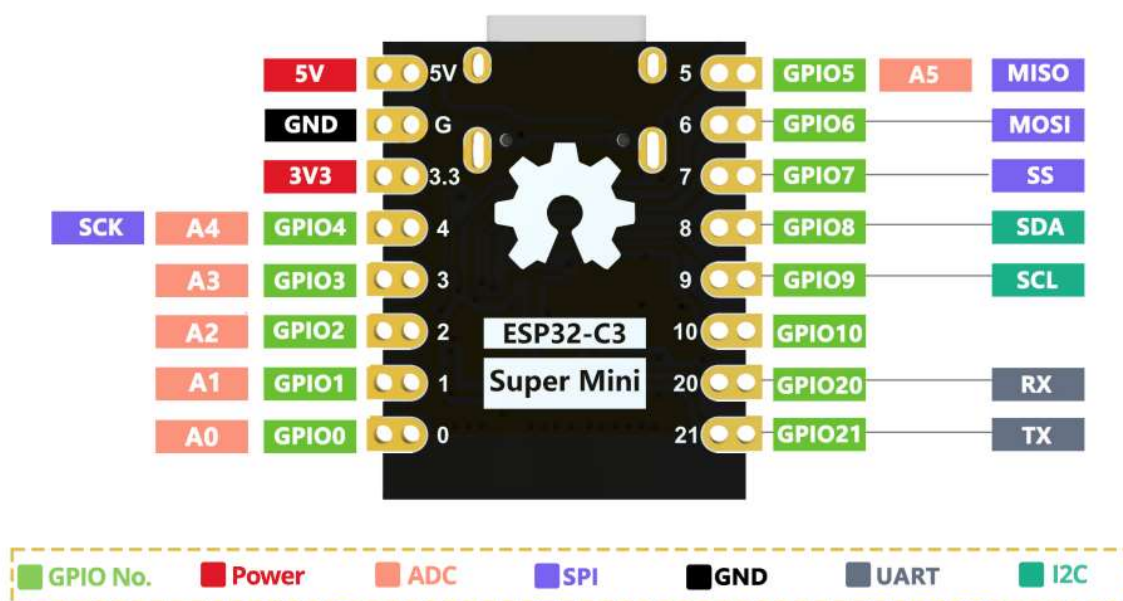


Fig. 4 – Pinagem e sinais disponíveis no módulo ESP32 C3 supermini



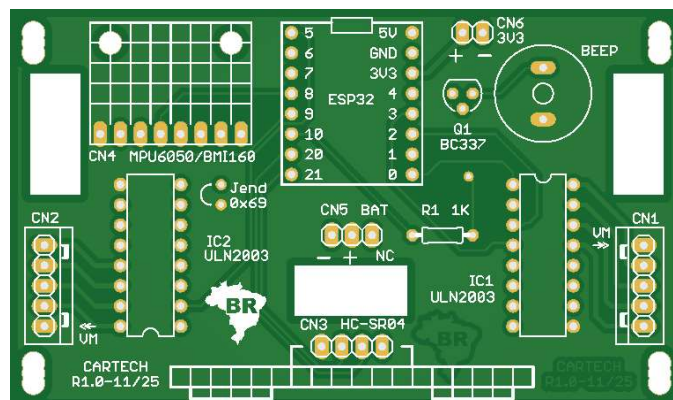


Fig. 5 – Placa CARTECH rev. 1.0 em tamanho real (lado dos componentes)

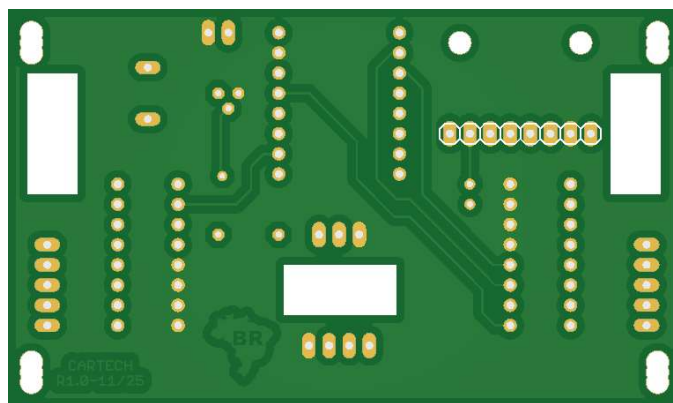


Fig. 6 – Placa CARTECH rev. 1.0 em tamanho real (lado da solda)

**OBSERVAÇÃO:** O driver dos motores ULN2003A possui corrente nominal de 500mA para cada um dos pinos de saída, suportando picos de curta duração de até 600mA cada pino. Assim, é possível se utilizar motor Nema17 em aplicações que demandem um breve período de operação do motor (máximo de 15 segundos) intercalado com períodos de repouso para resfriamento (mínimo de 45 segundos). Isso porque o Nema17 usualmente consome uma corrente máxima 1700mA a 12volts. No presente caso, o motor Nema17 é alimentado por 5 volts, ou seja, irá consumir apenas 835mA, corrente essa suportada pelo driver ULN2003A durante curtos períodos de tempo (dois pinos de 500mA cada conectados a cada uma das duas bobinas do Nema17).

## **BIBLIOTECA:**

Para facilitar o desenvolvimento de aplicações a serem hospedadas em placas baseadas nos microcontroladores ESP32, foi criada uma biblioteca de nome `motbepled.h`, disponível em <https://github.com/izyino/motbepled> a qual deve ser utilizada conforme as instruções seguintes.

As funções descritas a seguir referem-se apenas àquelas relevantes para a placa CARTECH. Registre-se que a biblioteca `motbepled.h` possui muitas outras funções que fazem sentido apenas para placas mais complexas que a CARTECH.

### **#include < motbepled.h>**

para incluir a biblioteca ao programa. Dependendo de onde a biblioteca estiver gravada, pode-se usar alternativamente o formato **#include "motbepled.h"**

### **motbepled x(t1, t2);**

comando construtor que deve ser informado logo após o include, sendo `t1` e `t2` variáveis do tipo `int8_t` que definem o tipo e o modo de operação dos motores eventualmente conectados a CN1 e CN2, sendo possível os seguintes valores:

- 1 – Para motor 28byj-48, 2048 passos por volta, baixo torque, baixo consumo
- 2 – Para motor 28byj-48, 2048 passos por volta, alto torque, alto consumo
- 3 – Para motor 28byj-48, 4096 passos por volta, médio torque, médio consumo
- 4 – Para motor Nema17, 200 passos por volta

### **x.pinsStep0(0, 1, 2, 3, -1, -1);**

comando que informa os pinos do microcontrolador ESP32C3 associados ao motor de passo nº 0, sendo os quatro primeiros das bobinas principais e os dois últimos dos sinais enable. O valor -1 significa que os sinais correspondentes não se aplicam no presente caso. Para uso da biblioteca `motbepled.h` com a placa CARTECH deve-se informar o comando `pinsStep0` exatamente como mostrado acima, ou seja: `x.pinsStep0 (0, 1, 2, 3, -1, -1)`; e deve ser informado na sessão de setup de todos os programas, sempre antes do `x.begin()`

### **x.pinsStep1(4, 5, 6, 7, -1, -1);**

comando que informa os pinos do microcontrolador ESP32C3 associados ao motor de passo nº 1, sendo os quatro primeiros das bobinas principais e os dois últimos dos sinais enable. O valor -1 significa que os sinais correspondentes não se



aplicam no presente caso. Para uso da biblioteca `motbepled.h` com a placa CARTECH deve-se informar o comando `pinsStep0` exatamente como mostrado acima, ou seja: `x.pinsStep0 (4, 5, 6, 7, -1, -1)`; e deve ser informado na sessão de setup de todos os programas, sempre antes do `x.begin()`

### **`x.pinBeep(10);`**

comando que informa o pino do microcontrolador ESP associado ao beep. Para uso da biblioteca `motbepled.h` com a placa CARTECH deve-se informar o comando `pinBeep` exatamente como mostrado acima, ou seja: `x.pinBeep (10)`; e deve ser informado na sessão de setup de todos os programas, sempre antes do `x.begin()`

### **`x.pinLed(8, 0);`**

comando que informa o pino do microcontrolador ESP associado ao Led e o nível lógico para o led aceso. Para uso da biblioteca `motbepled.h` com a placa CARTECH deve-se informar o comando `pinLed` exatamente como mostrado acima, ou seja: `x.pinLed (8, 0)`; e deve ser informado na sessão de setup de todos os programas, sempre antes do `x.begin()`. **IMPORTANTE:** Caso a placa contenha o sensor acelerômetro e giroscópio MPU6050 ou BMI610 montado, o controle do led azul não poderá ser utilizado, pois o GPIO 8 do ESP32C3 é também utilizado pelo sinal SDA da interface I2C. Neste caso o comando `x.pinLed(-1, 0)` deve ser informado

### **`x.begin();`**

inicializa as diversas funções da biblioteca. Deve ser colocado na sessão de setup de todos os programas que se utilizem da biblioteca

### **`x.runStep(n, steps, velstep, cwstep);`**

comando que ativa o motor de passo `n`, de forma automática e assíncrona, conforme as seguintes variáveis:

`n` – variável `uint8_t` com número do motor que será movimentado (0 ou 1):

`steps` – variável `uint32_t` contendo o número de passos a movimentar

`velstep` – variável `uint8_t` que define a velocidade da movimentação em RPM (rotações por minuto). Este valor pode variar entre 1 e 16, dependendo do motor utilizado de da corrente disponível na fonte de alimentação

`cwstep` – variável booleana que define o sentido da movimentação, sendo “true” para sentido horário e “false” para sentido anti-horário

**x.stepstogo(n);**

esta função retorna no formato uint32\_t o número de passos ainda restantes para que o motor n (n=0 ou 1) chegue ao seu destino. Zero significa que o motor já chegou ao seu último destino e já encontra-se parado. Antes de comandar qualquer movimentação deve-se consultar esta função para ter certeza que o motor encontra-se parado. A variável n deve ser do tipo uint8\_t

**x.beep(bnum, bdur, bfreq, binter);**

comando que ativa a emissão de beeps sonoros, de forma automática e assíncrona, conforme as seguintes variáveis:

bnum – variável inteira que especifica o número de beeps a serem emitidos

bdur – variável inteira que especifica a duração de cada beep, em milisegundos

bfreq – variável inteira que especifica a frequência dos beeps, em Hertz (Hz). Os beeps passivos comuns respondem bem frequências entre 200Hz e 5000Hz

binter – variável inteira que especifica a duração da pausa entre os beeps, em milisegundos

**x.led(lnum, ldur, linter);**

comando que ativa piscadas do Led (conectado ao pino 8 do módulo ESP32) , de forma automática e assíncrona, conforme as seguintes variáveis:

lnum – variável inteira que especifica o número de piscadas a serem emitidas

ldur – variável inteira que especifica a duração do Led acesso em cada piscada, em milisegundos

linter – variável inteira que especifica a duração do Led apagado em cada piscada, em milisegundos

**x.setms(yms);**

comando para inicializar o contador de milisegundos com o valor informado pela variável yms do tipo uint32\_t. Imediatamente após inicializado, o contador começa ser subtraído de 1 a cada milisegundo

**x.getms();**

esta função retorna no formato uint32\_t o estado atual do contador de milisegundos previamente inicializado pelo comando x.setms. Serve como alternativa para a função delay(), de forma assíncrona

**x.stopStep(0);**

esta função interrompe o movimento do motor de passo

**x.stopLed();**

esta função interrompe as piscadas do Led eventualmente em andamento

**x.stopBeep();**

esta função interrompe a emissão de beeps sonoros eventualmente em andamento

## **Exemplos de utilização da biblioteca**

No início do programa:

```
#include < motbepled.h>  
motbepled x(2, 2);
```

na sessão do setup:

```
x.begin();
```

---

```
//movimenta o motor de passo nº 0 (conectado em CN1), tipo 28BYJ-48,  
//velocidade 3 RPM, sentido horário, 2048 passos:
```

```
//função principal:
```

```
x.runStep(0, 2048, 3, true);
```

```
//o motor começa a se movimentar imediatamente após a função runStep ser chamada
```

```
//para saber se o motor de passo nº 0 já chegou ao destino, fazer
```

```
if (x.stepstogo(0)>0) {ainda não chegou ao destino. Está em movimento...};
```

```
//a qualquer momento o movimento do motor de passo nº 0 pode ser interrompido
```

```
x.stopStep(0);
```

---

**//emite 10 beeps de 2KHz de 0,5s com pausa interbeeps de 0,25s:**

//função principal:

`x.beep(10, 500, 2000, 250);`

//os beeps começam a ser emitidos imediatamente após a função beep ser chamada

//a qualquer momento a emissão dos beeps sonoros pode ser interrompida

`x.stopBeep();`

---

**//pisca o Led 50 vezes com 0,25s aceso seguido de 0,10s apagado:**

//função principal:

`x.led(50, 250, 100);`

//o led começa a piscar imediatamente após a função led ser chamada

//a qualquer momento as piscadas do Led podem ser interrompidas

`x.stopLed();`

---

**//contagem de 4 segundos, de forma assíncrona:**

//função principal:

`x.setms(4000);`

`while (x.getms()>0){enquanto espera 4s, pode fazer coisas...}`

//a variável x.xms começa a ser decrementada imediatamente após ter sido inicializada

---

O diretório <https://github.com/izyino/motbepled/tree/main/examples> da biblioteca motbepled contém diversos exemplos de programas, das mais variadas aplicações, como: movimentação de motores, emissão de beeps, acesso a redes WiFi, servidor web, e muitas outras.

**IMPORTANTE:** Antes da execução de qualquer um dos exemplos, deve-se conferir e alterar quando necessário o tipo e a pinagem dos motores, beep e Led atribuída pelos comandos motbepled, pinsStep, pinBeep e pinLed, para que correspondam fielmente ao hardware utilizado

---