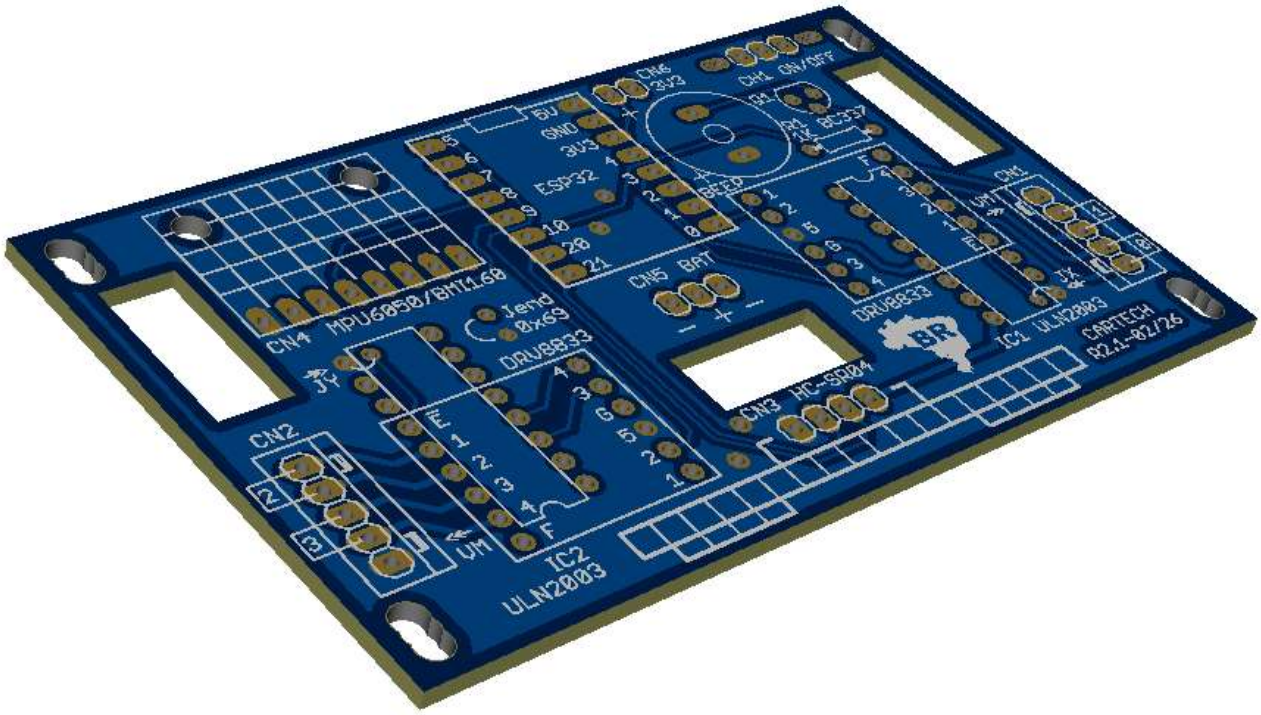
PCI
CARTECH



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 motores de passo do tipo 28BYJ48 e/ou em motores DC, 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 e todos os demais componentes, incluindo previsão para sensor de proximidade por ultrasom e sensor acelerômetro e giroscópio de três eixos.

A versão 2.1 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 2.4GhHz, 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 ou até quatro motores DC ou ainda combinações de ambos os tipos

- Utilização dos drivers ULN2003, dip 16 e/ou módulos drv8833

- Suporta motores Nema17, alimentados com 5V, em ciclos de no máximo 15 segundos rodando com no mínimo 45 segundos em repouso se usado drivers ULN2003 ou ciclos de 30 segundos com 60 segundos de repouso se utilizados módulos drv8833

- 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

- Arquitetura aberta, permitindo o seu uso em uma infinidade de aplicações

- Completa biblioteca para controle dos motores de passo, motores DC, do beep sonoro e do Led azul, bem como um controle de tempo, não blocante, 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 ou motores DC n. 0 e 1

1 – L1 fio azul (GPIO 3 do ESP32C3 → driver) - motor DC n° 0
2 – L1 fio rosa (GPIO 2 do ESP32C3 → driver) - motor DC n° 0
3 – L2 fio amarel/o (GPIO 1 do ESP32C3 → driver) - motor DC n° 1
4 – L2 fio laranja (GPIO 0 do ESP32C3 → driver) - motor DC n° 1
5 – 5vcc - fio vermelho

CN2 – motor de passo n° 1 ou motores DC n. 2 e 3

1 – L1 fio azul (GPIO 7 do ESP32C3 → driver) - motor DC n° 2
2 – L1 fio rosa (GPIO 6 do ESP32C3 → driver) - motor DC n° 2
3 – L2 fio amarel/o (GPIO 5 do ESP32C3 → driver) - motor DC n° 3
4 – L2 fio laranja (GPIO 4 do ESP32C3 → driver) - motor DC n° 4
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 – Gnd

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 e/ou 2 x módulos drv8833
1 x Sensor acelerômetro e giroscópio MPU6050 ou BMI610 (opcional)
1 x Sensor de proximidade HC-SR04 (opcional)
1 x Chave liga/desliga
1 x Beep passivo
1 x Transistor BC337
1 x Resistor de 1KΩ
2 x Conector molex 5 pinos com polarizador (CN1 e CN2 para os motores)
1 x Conector 4 pinos fêmea (CN3 – ultrasom 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 são 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

OBSERVAÇÕES: 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

As possíveis configurações em termos de motores são as seguintes:

* 1 ou 2 motores de passo
* 1 motor de passo e 1 ou 2 motores DC
* até 4 motores DC

Essas escolhas são feitas pela instalação do driver ULN2003 ou do módulo drv8833 nas posições de IC1 e de IC2. Por exemplo, ULN2003 instalado em IC1. Nesse caso, o conector CN1 estará apto a receber **apenas um motor de passo**. Se ao invés disso for instalado um módulo drv8833 em IC1, o conector CN1 estará apto a receber **um motor de passo ou até 2 (dois) motores DC**. Situação análoga ocorre em relação a IC2 / CN2.

Lembrando que sempre que for utilizado o módulo drv8833 ao invés do ULN2003, deve-se fechar o jumper correspondente (JX e/ou JY) os quais colocam o sinal sleep/enable do módulo na condição “enable” (nível alto).

Na placa versão 2.1 foram adicionados os seguintes dispositivos: Chave liga/desliga e a possibilidade de montagem dos módulos drv8833 como alternativa aos ULN2003, juntamente com os jumpers JX e JY. Os demais dispositivos permaneceram inalterados em relação a versão 1.0.

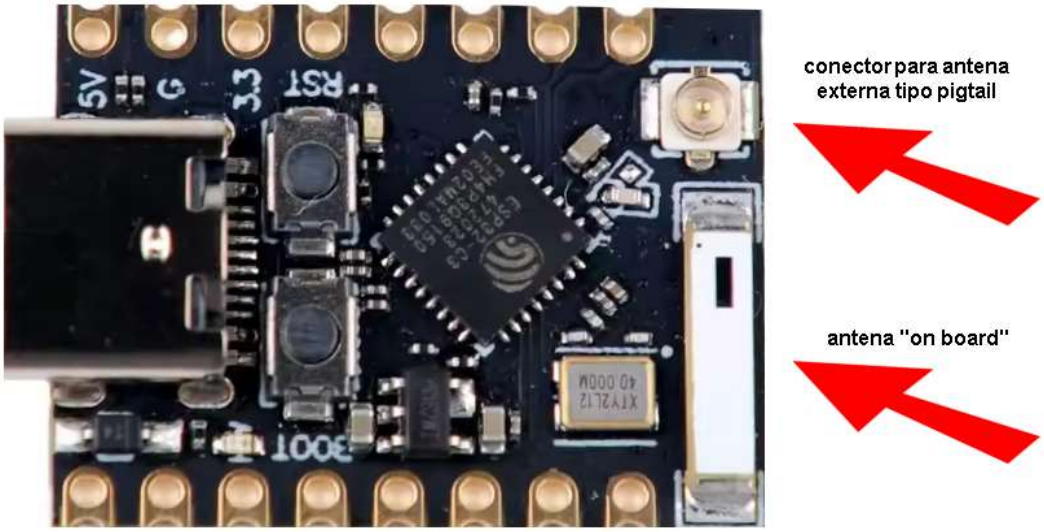


Fig. 1 – Modelo de ESP32 C3 recomendado

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.

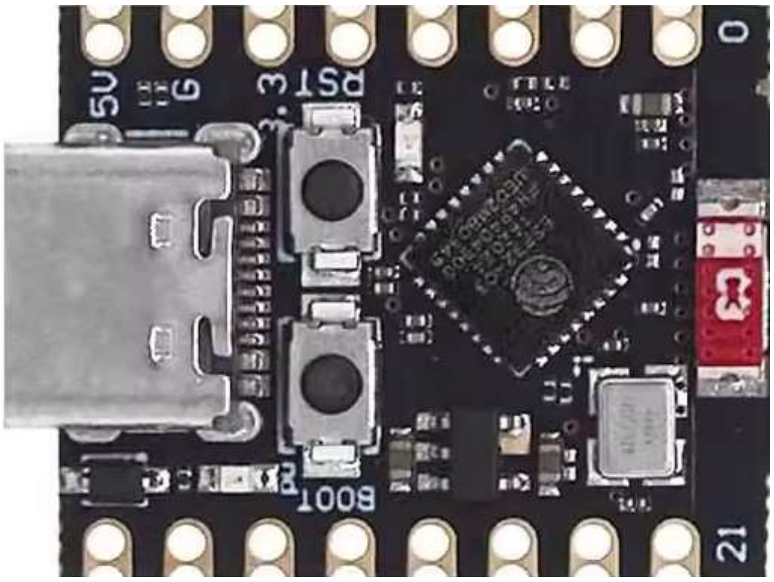


Fig. 2 – Modelo de ESP32 C3 **-NÃO-** recomendado

DIAGRAMA ELÉTRICO:

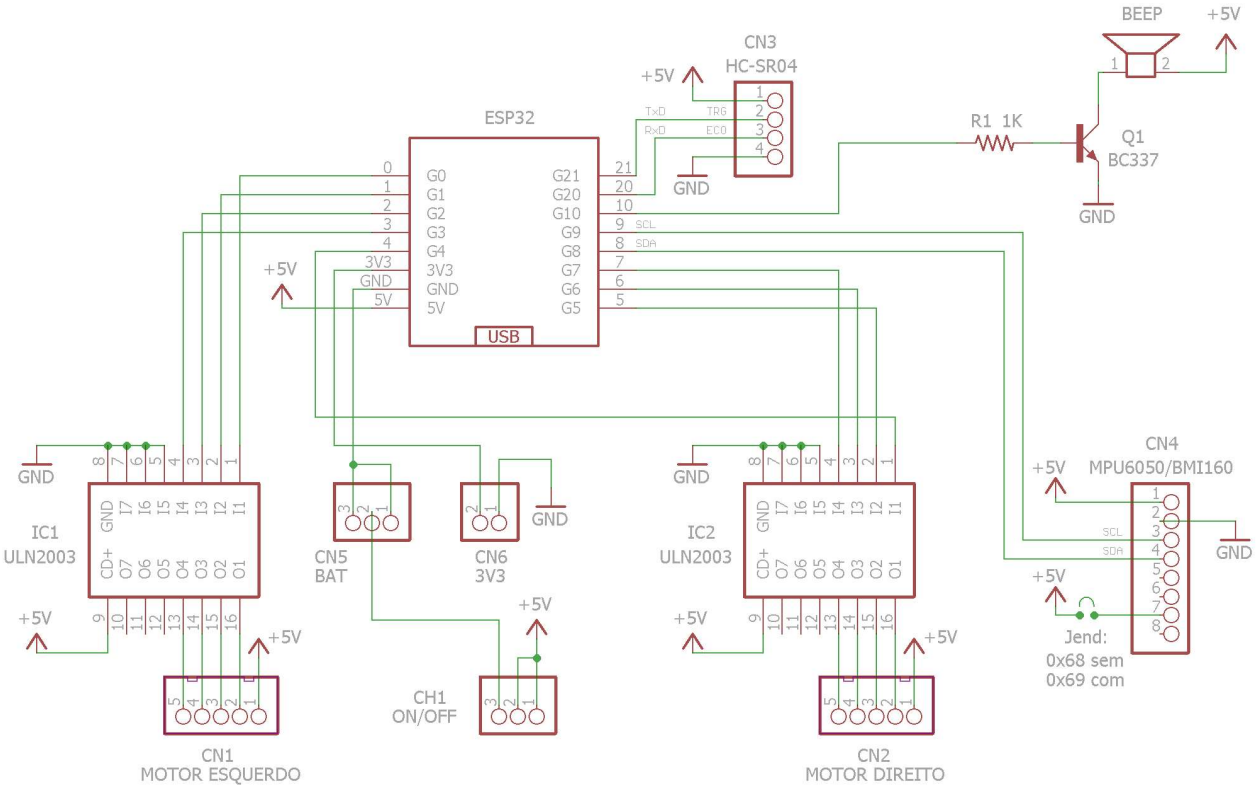


Fig. 3 – Diagrama elétrico da placa CARTECH Rev. 2.1

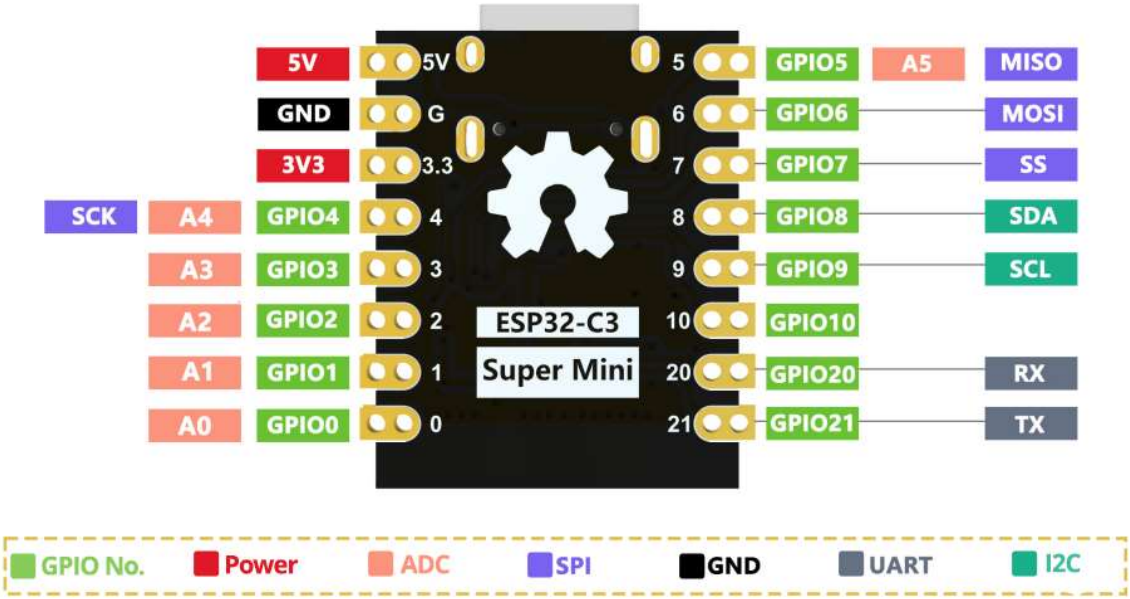


Fig. 4 – Pinagem e sinais disponíveis no módulo ESP32 C3 supermini

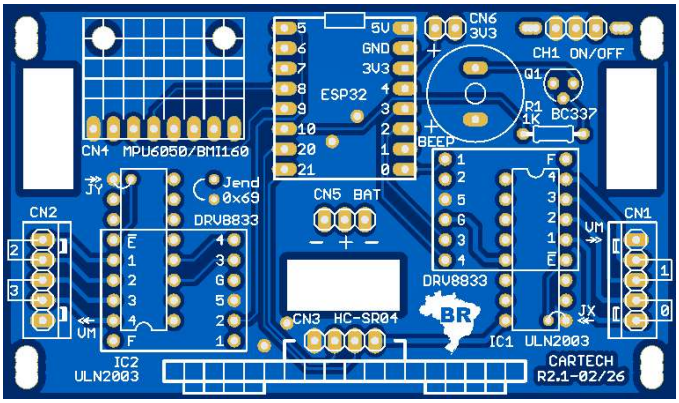


Fig. 5 – Placa CARTECH rev. 2.1 (lado dos componentes)

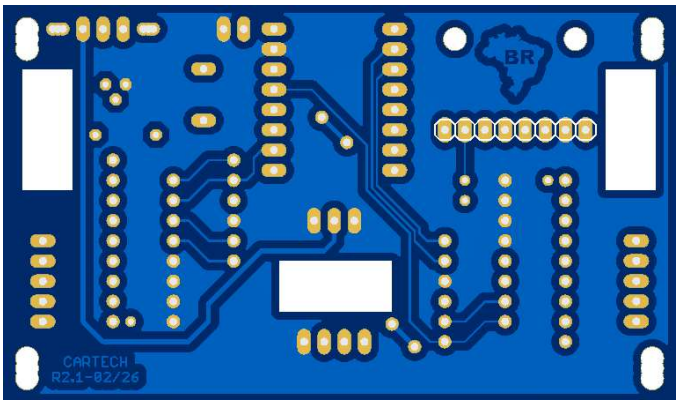


Fig. 6 – Placa CARTECH rev. 2.1 (lado da solda)

OBSERVAÇÕES:

1) As ranhuras retangulares existentes na placa destinam-se a passagem dos fios dos dois motores e da bateria. Os furos oblongos são para o ajuste fino do centro de massa do conjunto.

2) 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). Já o módulo drv8833 oferece corrente nominal de 1500mA, podendo portanto controlar motores Nema17 com ciclos bem maiores que os ULN2003, porém não de forma contínua.

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 motbedled.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(cn1, cn2);

comando construtor que deve ser informado logo após o include, sendo cn1 e cn2 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:

0 – Para motor DC
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 n° 0, **caso exista**, 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(7, 6, 5, 4, -1, -1);

comando que informa os pinos do microcontrolador ESP32C3 associados ao motor de passo n° 1, **caso exista**, 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 (7, 6, 5, 4, -1, -1); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin()

x.pinsDC0(2, 3, -1);

comando que informa os pinos do microcontrolador ESP associados ao motor DC n.0, **caso exista**, sendo os dois primeiros do motor e o terceiro da velocidade PWM. 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 pinsDC0 exatamente como mostrado acima, ou seja: x.pinsDC0 (2, 3, -1); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin(). Nestes casos, quando não houver pino específico de enable (-1), o PWM para controle da velocidade é aplicado aos pinos 2 ou 3 do microcontrolador, dependendo do sentido de rotação do motor

x.pinsDC1(0, 1, -1);

comando que informa os pinos do microcontrolador ESP associados ao motor DC n.1, **caso exista**, sendo os dois primeiros do motor e o terceiro da velocidade PWM. 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 pinsDC1 exatamente como mostrado acima, ou seja: x.pinsDC1 (0, 1, -1); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin().Nestes casos, quando não houver pino específico de enable (-1), o PWM para controle da velocidade é aplicado aos pinos 0 ou 1 do microcontrolador, dependendo do sentido de rotação do motor

x.pinsDC2(6, 7, -1);

comando que informa os pinos do microcontrolador ESP associados ao motor DC n.2, **caso exista**, sendo os dois primeiros do motor e o terceiro da velocidade PWM. 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 pinsDC2 exatamente como mostrado acima, ou seja: x.pinsDC2 (6, 7, -1); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin().Nestes casos, quando não houver pino específico de enable (-1), o PWM para controle da velocidade é aplicado aos pinos 6 ou 7 do microcontrolador, dependendo do sentido de rotação do motor

x.pinsDC3(4, 5, -1);

comando que informa os pinos do microcontrolador ESP associados ao motor DC n.3, **caso exista**, sendo os dois primeiros do motor e o terceiro da velocidade PWM. 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 pinsDC3 exatamente como mostrado acima, ou seja: x.pinsDC3 (4, 5, -1); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin().Nestes casos, quando não houver pino específico de enable (-1), o PWM para controle da velocidade é aplicado aos pinos 4 ou 5 do microcontrolador, dependendo do sentido de rotação do motor

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 uint16_t que define a velocidade da movimentação em RPM (rotações por minuto)

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.runDC(n, time, veldc, cwdc);

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

n – variável uint8_t com número do motor DC que será movimentado (0 a 3):

time – variável uint32_t contendo o tempo em milisegundos que o motor DC ficará ativado

velDC – variável uint8_t que define a velocidade da movimentação, em termos de porcentagem entre 0 e 100. Sendo 0=0% motor parado, 100=100% motor com velocidade máxima.

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

x.timetogo(n);

esta função retorna no formato uint32_t, em milisegundos, o tempo ainda restante para que o motor DC n (n=0 a 3) complete o último comando runDC. Se retornar zero significa que o motor DC n já está parado. Antes de comandar qualquer movimentação do motor DC n deve-se consultar esta função para ter certeza que o mesmo se encontra parado. A variável n é 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 freqüência dos beeps, em Hertz (Hz). Os beeps passivos comuns respondem bem freqüências 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

CONSIDERAÇÕES SOBRE A BIBLIOTECA MOTBEPLED:

A base funcional da biblioteca motbepled consiste em um núcleo de rotinas que são executadas a cada 100 microsegundos a partir de interrupções que ocorrem nesse mesmo intervalo de tempo (10KHz). Esse período define portanto a resolução de todos os eventos temporais controlados pela biblioteca.

Sobre motores de passo

A velocidade de giro dos motores de passo é dada pela fórmula: **t = npv / 600000 / rpm**, onde "npv" é o número de passos por volta do motor; "rpm" é a rotação por minuto desejada e "t" é o tempo de espera entre cada passo do motor, dado em centenas de microsegundos, ou seja, o número de interrupções que deve ocorrer entre cada passo do motor. Assim, a exatidão da RPM do motor de passo será quanto mais precisa quanto mais baixa for a velocidade desejada.

Sobre motores DC

A velocidade de giro dos motores DC é dada pela porcentagem de tempo em nível alto do sinal PWM gerado para este fim, variando de 0% (motor parado) a 100% (motor girando na velocidade máxima).

O tempo que o motor permanecerá girando é dado em milisegundos, o qual é mantido com precisão absoluta, uma vez que o controle temporal é feito pela rotina de interrupção, a qual ocorre a cada 100 microsegundos.

Sobre o beep sonoro e o led

O tempo de duração de cada beep e o tempo de pausa entre eles, assim como o tempo de duração de cada piscada (led aceso) e o tempo de pausa entre as piscadas (led apagado) são informados em milisegundos, os quais são igualmente mantidos com precisão absoluta, uma vez que o controle temporal é feito pela rotina de interrupção, a qual ocorre a cada 100 microsegundos.

Sobre o acesso a contagem de tempo

O acesso a contagem de tempo de forma assíncrona (não blocante) é feita pelas funções setms e getms, usadas como alternativa para a função “delay”. As funções setms e getms utilizam milisegundos como unidade, sendo portanto igualmente precisas pela mesma razão já citada.

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 n° 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 n° 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 n° 0 pode ser interrompido
x.stopStep(0);
```

//movimenta o motor DC n.1,
//velocidade 75%, sentido anti-horário, durante 15segundos:

```
//função principal:
x.runDC(1, 15000, 75, false);
//o motor começa a se movimentar imediatamente após a função runDC ser executada
//para saber se o motor DC nº1 ainda está girando ou já esta parado, fazer
if (x.timetogo(1)>0) {ainda não terminou o último comando runDC. Está em movimento...};
//a qualquer momento o movimento do motor DC n.1 pode ser interrompido
x.stopDC(1);
```

//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 piscarimediatamente 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

---