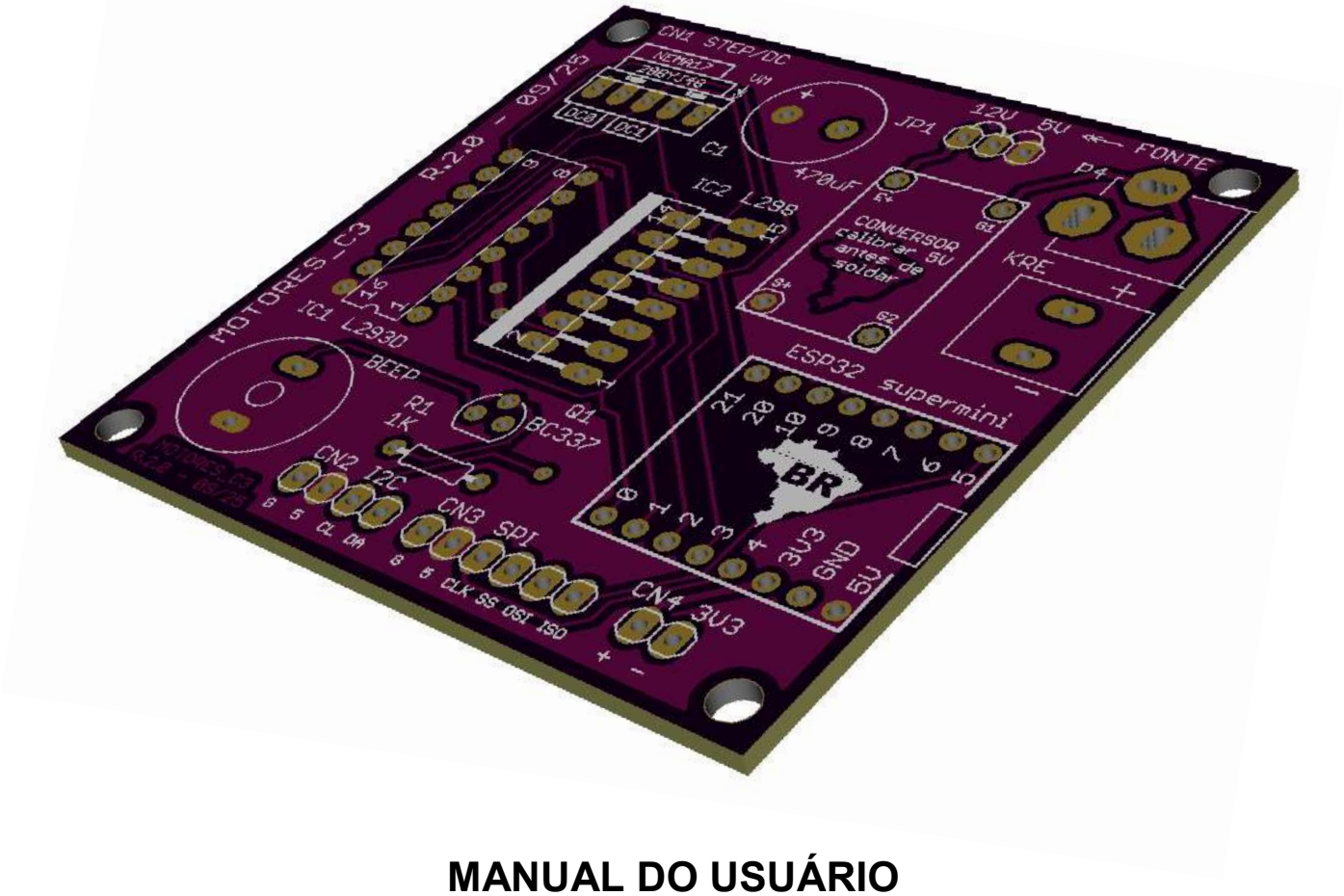
PCI
MOTORES_C3



MANUAL DO USUÁRIO

Aplicável.as placas:

MOTORES_C3 versão 2.0 e posteriores
com o módulo microcontrolador ESP32 C3 supermini de 16 pinos

Manual rev. 1.0 ─ 09/2025
Eventuais atualizações desse manual podem ser encontradas no diretório \docs da biblioteca motores_C3.h, disponível em: https://github.com/izyino/motores_c3.h

INTRODUÇÃO:

A presente placa de circuito impresso intitulada MOTORES_C3 foi concebida para facilitar a construção de dispositivos para controle de motores de passo ou até dois motores DC, via WiFi, contemplando em um reduzido espaço de apenas 6 x 6 cm, o moderno microcontrolador ESP32 C3 em sua versão supermini de 16 pinos, os drivers do motor de passo e dos motores DC, e todos os demais componentes, substituindo assim os antigos, obsoletos e saudosos Arduinos e seus “shields” sobrepostos em forma de sanduíche.

A versão 2.0 da placa de circuito impresso MOTORES_C3 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 motor de passo modelos Nema-17 ou 28BYJ48 ou até dois motores DC de baixa ou média corrente, de forma bidirecional, com controle de velocidade PWM

- Utilização do driver ponte H dupla, L293D, dip 16 ou L298

- Beep sonoro passivo

- Conectores para acesso aos sinais I2C, SPI e 3V3

- Completa biblioteca para controle dos motores de passo, dos 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/motores_c3.h

CONECTORES EXISTENTES NA PLACA MOTORES_C3:

CN1 – motor STEP ou motores DC:

1 – motor DC 0 ou Step L1 (fio azul, se 28BYJ48)
2 – motor DC 0 ou Step L1 (fio rosa, se 28BYJ48)
3 – motor DC 1 ou Step L2 (fio amarelo, se 28BYJ48)
4 – motor DC 1 ou Step L2 (fio laranja, se 28BYJ48)
5 – sem conexão - Step comum (fio vermelho, se 28BYJ48)

CN2 – I2C:

1 – Gnd
2 – Vcc (+5V)
3 – SCL
4 – SDA

CN3 – SPI:

1 – Gnd
2 – Vcc (+5V)
3 – CLK
4 – SS
5 – MOSI
6 – MISO

CN4 – 3V3:

1 – Gnd
2 – +3V3

RELAÇÃO DE COMPONENTES:

Placa de circuito impresso Rev.1.1
Módulo ESP32 C3 supermini, 16 pinos
CI driver L293D, dip 16 ou L298 (ou um ou outro, nunca os dois)
Conector molex 5 pinos com polarizador (para os motores em CN1)
Conector P4 fêmea, 90 graus, solda placa
Módulo conversor 12V–>5V MP2307 (montar ou não dependendo da configuração)
Borne KRE de 2 pinos (montagem opcional)
Beep passivo (montagem opcional)
Transistor BC337 (montagem opcional)
Resistor de 1K montagem opcional)
Conector KRE de 2 pinos (montagem opcional)
Conector 4 pinos macho (CN2 – I2C, montagem opcional)
Conector 6 pinos macho (CN3 – SPI, montagem opcional)
Conector 2 pinos macho (CN4 – 3V3, montagem opcional)

POSSIVEIS CONFIGURAÇÕES:

1) Para aplicações que utilizem até dois motores DC de baixa potência ou um motor de passo modelo 28BYJ48 ou similar, deve-se montar a PCI com o driver L293D apenas (L298 e conversor não montados), com um jumper entre o pino central e o pino mais a direita (5V) de JP1. Nessa configuração deve-se utilizar uma fonte de 5Vcc em P4 ou no borne KRE. Observação: esta aplicação poderia também utilizar o driver L298, que apesar de ser mais caro que o L293, o substitui perfeitamente

2) Para aplicações que utilizem até dois motores DC de média potência ou um motor de passo modelo Nema17 ou similar, deve-se montar a PCI com o driver L298 e o módulo conversor, com um jumper entre o pino central e o pino mais a esquerda (12V) de JP1. Nessa configuração deve-se utilizar uma fonte de 12Vcc em P4 ou no borne KRE.

DIAGRAMA ELÉTRICO:

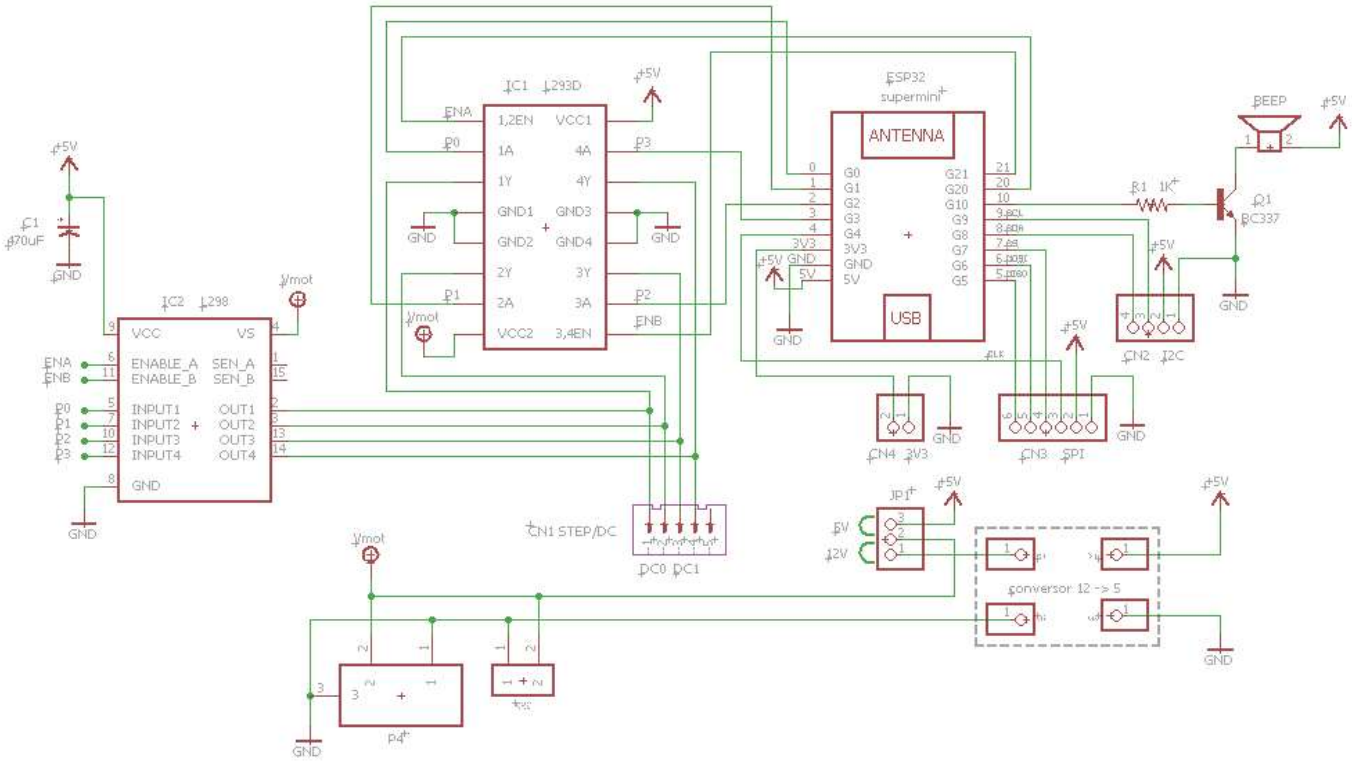


Fig. 1 – Diagrama elétrico da placa MOTORES_C3 Rev. 2.0

ASPÉCTO DA PLACA MOTORES_C3:

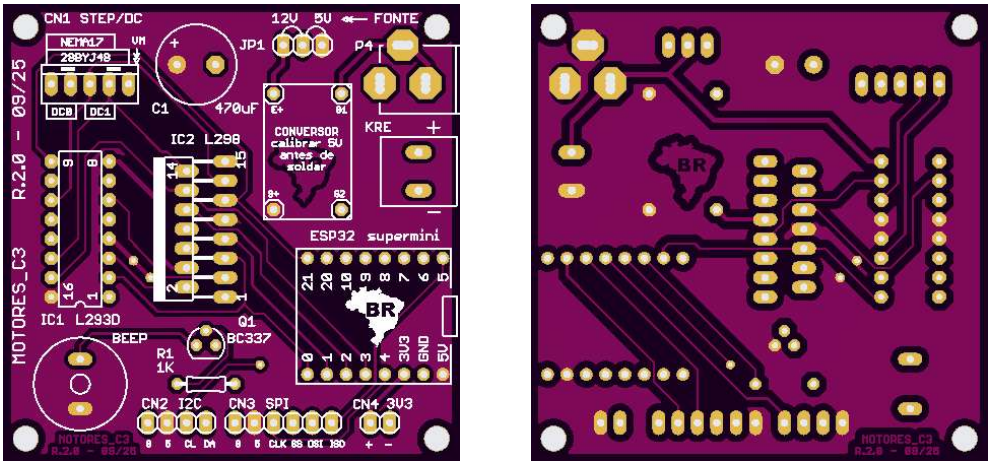


Fig. 2 – Placa MOTORES_C3 rev. 2.0 (em tamanho real)

BIBLIOTECA:

Visando facilitar o desenvolvimento de aplicações a serem hospedadas nas placas MOTORES_C3, foi criada uma biblioteca de nome motores_c3.h, disponível em https://github.com/izyino/motores_c3.h a qual dispõe das seguintes funções:

#include <motores_c3.h>

para incluir a biblioteca ao programa. Dependendo de onde a biblioteca estiver gravada, pode-se usar alternativamente o formato #include "**motores_c3.h**"

motores_c3 x(t);

comando construtor que deve ser informado logo após o include, sendo t uma variável do tipo uint8_t que define o tipo e o modo de operação do(s) motor(es) conectado(s) a CN1, sendo possível os seguintes valores:

0 – Para um ou dois motores 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 Nema-17, 200 passos por volta

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(steps, velstep, cwstep);

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

steps – variável uint32_t contendo o número de passos a movimentar

velstep – variável unint8_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();

esta função retorna no formato uint32_t o número de passos ainda restantes para que o motor chegue ao seu destino. Zero significa que o motor já chegou ao seu último destino e já se encontra parado. Antes de comandar qualquer movimentação deve-se consultar esta função para ter certeza que o motor se encontra parado

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

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

n – variável uint8_t com número do motor DC que será movimentado (0 ou 1):
Se n=0, um motor DC deverá estar conectado aos pinos 1 e 2 do CN1
Se n=1, um motor DC deverá estar conectado aos pinos 3 e 4 do CN1

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

velDC – variável unint8_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 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

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();

esta função interrompe o movimento do motor de passo

x.stopDC(n);

esta função interrompe o movimento do motor DC n (n=0 ou 1)

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 < motores_c3.h>
motores_c3 x(2);
```

na sessão do setup:

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

//movimenta o motor de passo (conectado em CN1), tipo 28BYJ-48,
//velocidade 3 RPM, sentido horário, 2048 passos:

```
//função principal:
x.runStep(2048, 3, true);
//o motor começa a se movimentar imediatamente após a função runStep ser chamada

//para saber se o motor de passo já chegou ao destino, fazer
if (x.stepstogo()>0) {ainda não chegou ao destino. Está em movimento...};

//a qualquer momento o movimento do motor de passo pode ser interrompido
x.stopStep();
```

//movimenta o motor DC nº0 (conectado aos pinos 1 e 2 do CN1),
//velocidade 75%, sentido anti-horário, durante 15segundos:

```
// função principal:
x.runDC(0, 15000, 75, false);
//o motor começa a se movimentar imediatamente após a função runDC ser executada

//para saber se o motor DC nº0 ainda está girando ou já esta parado, fazer
if (x.timetogo(0)>0) {ainda não terminou o último comando runDC. Está em movimento...};

//a qualquer momento o movimento do motor DC nº0 pode ser interrompido
x.stopDC(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 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