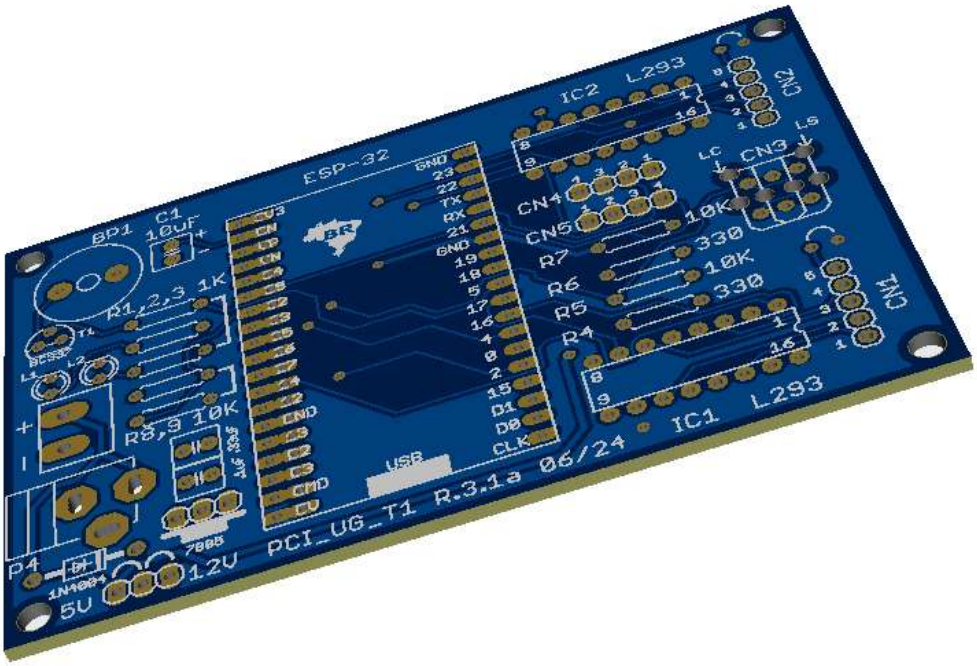
# PCI DE USO GERAL

## TIPO 1

Para controle de motores de passo
combinados com motores DC e outros dispositivos




## DOCUMENTAÇÃO BÁSICA PRELIMINAR

Placas versão 2.1 e posteriores

Manual rev. 2.1 – 06/2024
Eventuais atualizações desse manual podem ser encontradas em: https://github.com/izyino/manuais

# 1 - INTRODUÇÃO

A presente placa de circuito impresso PCI UG T1 foi concebida para permitir uma melhor organização na montagem de circuitos envolvendo motores de passo, motores DC, sensores reflexivos, sensores de distância e outros dispositivos, concentrando em uma placa de circuito impresso de dimensões reduzidas todo o hardware destinado a desempenhar as mencionadas funções, incluindo controle via Wi-Fi através de acesso a redes já existentes e como também gerenciamento de sua própria rede Wi-Fi, criada e administrada pelo poderoso microcontrolador ESP-32, tudo devidamente acomodado em uma placa de circuito impresso de apenas 60 x 90mm.

Dentre as aplicações para a placa PCI UG T1 destacam-se: alimentadores para animais domésticos, automação de portas, janelas, cortinas e persianas, robótica em geral, traçadores e registradores gráficos, ferremodelismo, furadeiras, frezadeiras, projetos educacionais em mecatrônica e uma infinidade de outras aplicações. As principais características da PCI UG T1 são as seguintes:

- Placa de circuito impresso dupla face para suporte dos módulos utilizados, medindo apenas 60 x 90 mm
- Utilização do microcontrolador Tensilica Xtensa 32-bit LX6 dual-core ESP32 em sua versão WROOM com PCI de 38 pinos, com 448Kbytes de ROM, 520Kbytes de SRAM, 8+8Kbytes de SRAM, RTC, 1Kbit de eFuses, clock de 240MHz
- Suporte para motores de passo do tipo 28BYJ-48 e motores DC de até 1,5A, através de circuitos ponte H com a utilização dos CIs L293D, em diversas combinações
- Suporte para até dois motores de passo do tipo Nema17 (disponível na versão de placa PCI_UG_T2, com extensões para drivers L298N)
- Suporte para até dois sensores reflexivos do tipo TCRT5000 ou quaisquer outros
- Suporte para sensor de distância do tipo VL53LX, display Oled ou quaisquer outros dispositivos com interface I2C
- Possibilidade de implementação de Wi-Fi como ponto de acesso e/ou estação, permitindo atualização de seu firmware via internet, automaticamente
- Beep e Led para sinalização sonora e visual
- Biblioteca auxiliar para controle de forma assíncrona, dos motores, beep e Led, disponível em https://github.com/izyino/motbepled
- Hardware flexível e aberto, servindo para uma infinidade de outras aplicações

## 2 - COMPONENTES UTILIZADOS (nem todos ao mesmo tempo)

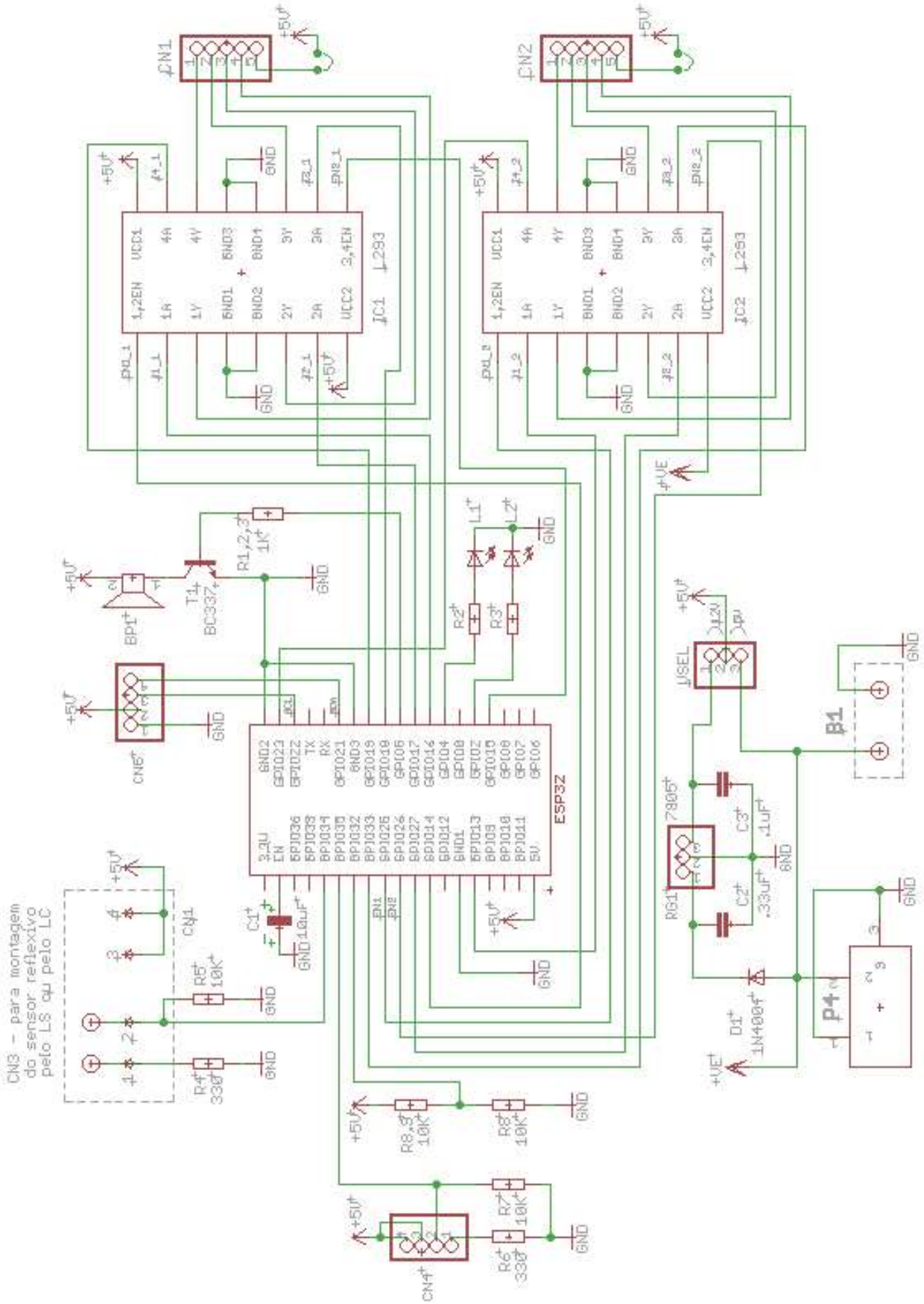
- 1 x Módulo microcontrolador ESP-32 WROOM de 38 pinos
- 2 x CI ponte H, L293D
- 2 x Soquete slim para CI de 16 pinos
- 1 x Regulador 7805
- 1 x capacitor tatalum 0.1uf
- 1 x capacitor tatalum 0.33uf
- 2 x Sensor reflexivo TCRT5000
- 1 x Sensor de distância VL53LX ou qualquer outro dispositivo I2C
- 1 x Beep passive TMB12A05, $\phi$12mm, alt 9,6mm, esp 7,8mm
- 1 x Transistor BC337
- 2 x Led colorido 3mm
- 4 x Resistores de 10K, 1/8W
- 3 x Resistor de 1K, 1/8W
- 2 x Resistor 330, 1/8W
- 1 x Capacitor eletrolítico de 10uF, 25V ou mais
- 1 x Conector P4 fêmea, solda placa
- 1 x Borne KRE de 2 pinos
- 2 x Barra de 5 pinos (conectores CN1, CN2)
- 2 x Barra de 4 pinos (CN4 e CN5)
- 2 x Barra de 3 pinos (jumper vsel)
- 1 x Jumper plástico (para vsel)
- 1 x Placa de circuito impresso PCI_UG_T1

## 3 – PINOS E CONEXÕES

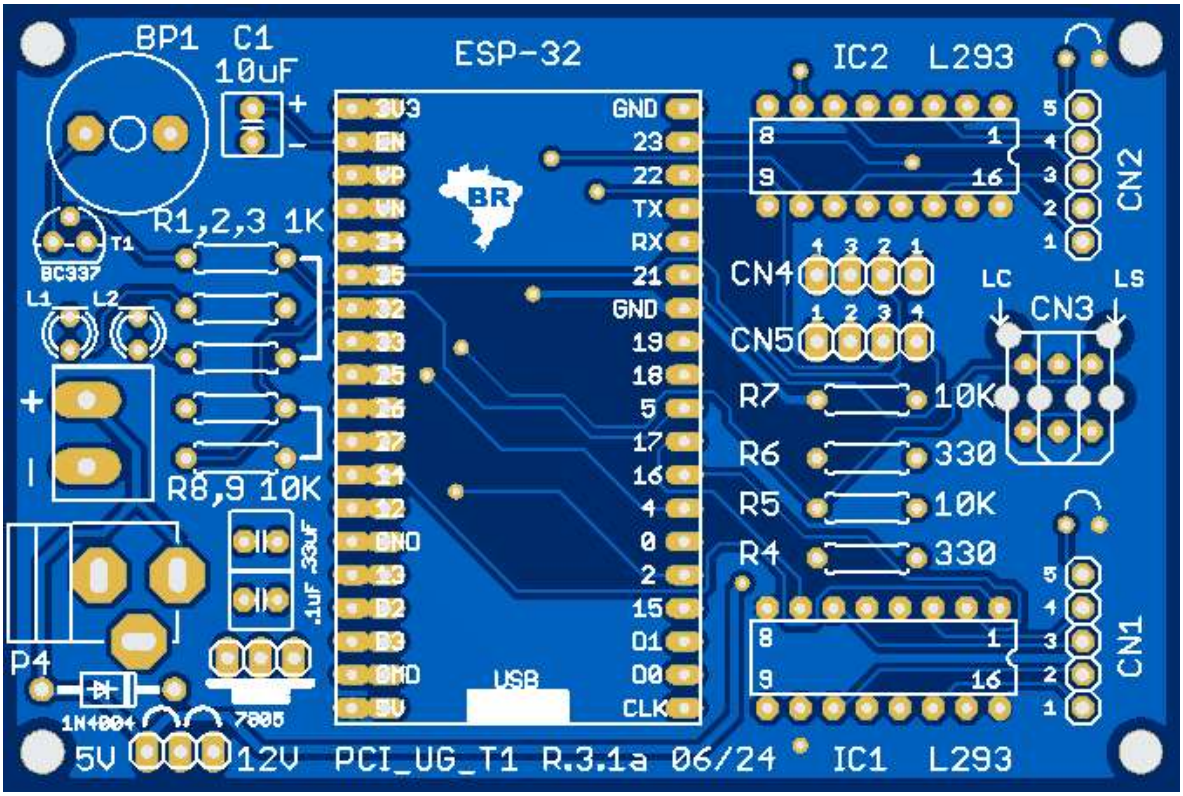
GPIO 16, 17, 18, 19 – Motor de passo n.0 / motor DC n.0 e n.1
GPIO 13, 27, 33, 23 – Motor de passo n.1 / motor DC n.2 e n.3
GPIO 14 – Enable do motor de passo n.0 / PWM motor DC n.0
GPIO 15 – Enable do motor de passo n.0 / PWM motor DC n.1
GPIO 25 – Enable do motor de passo n.1 / PWM motor DC n.2
GPIO 26 – Enable do motor de passo n.1 / PWM motor DC n.3
GPIO 5 – Beep de uso geral
GPIO 4 – Led de uso geral
GPIO 32 – Divisor de tensão para leitura do nível do Vcc
GPIO 34 – Sensor reflexivo n.1
GPIO 35 – Sensor reflexivo n.2
GPIO 21, 22 –SDA/SCL - Interface I2C

Obs: Na PCI versão 2.0 os sinais enable dos motores de passo 0 e 1 e dos motores DC 0, 1, 2 e 3 estão conectados diretamente a Vcc. Esse problema foi corrigido nas versões 2.1a e posteriores

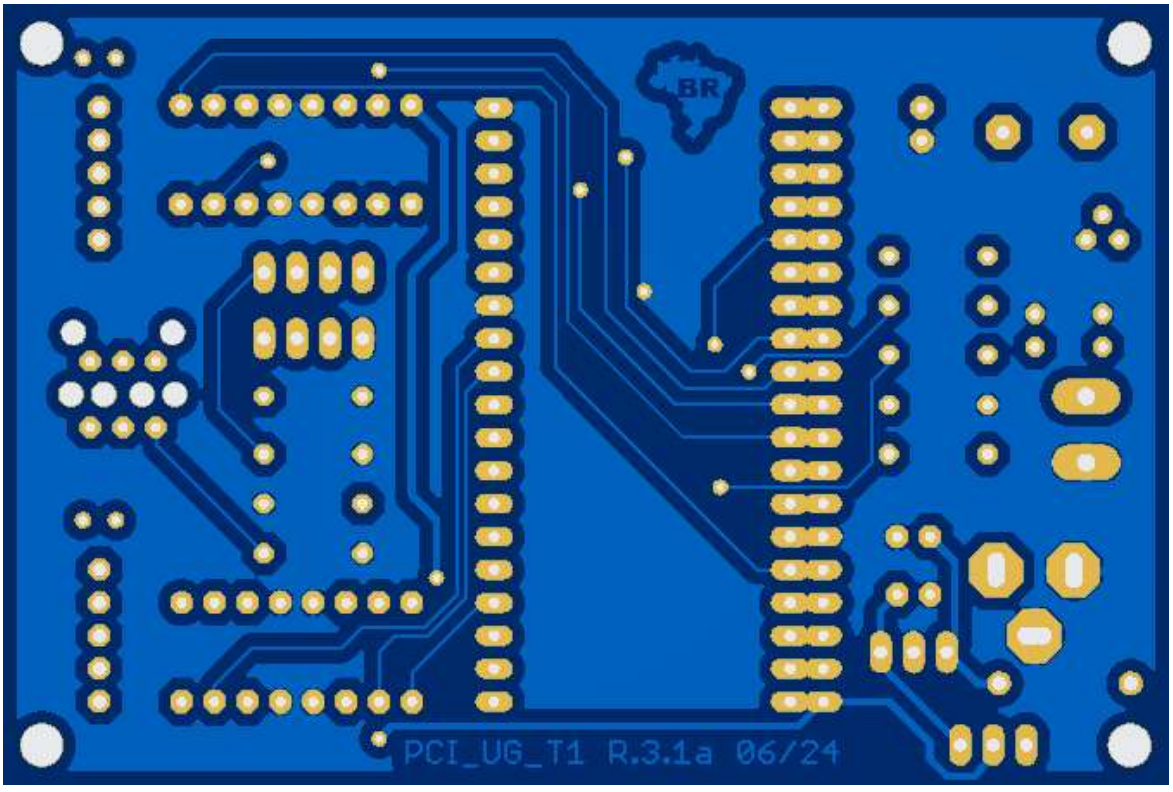
## 5 - DIAGRAMA ESQUEMÁTICO



# 6 - ASPÉCTO DA PLACA

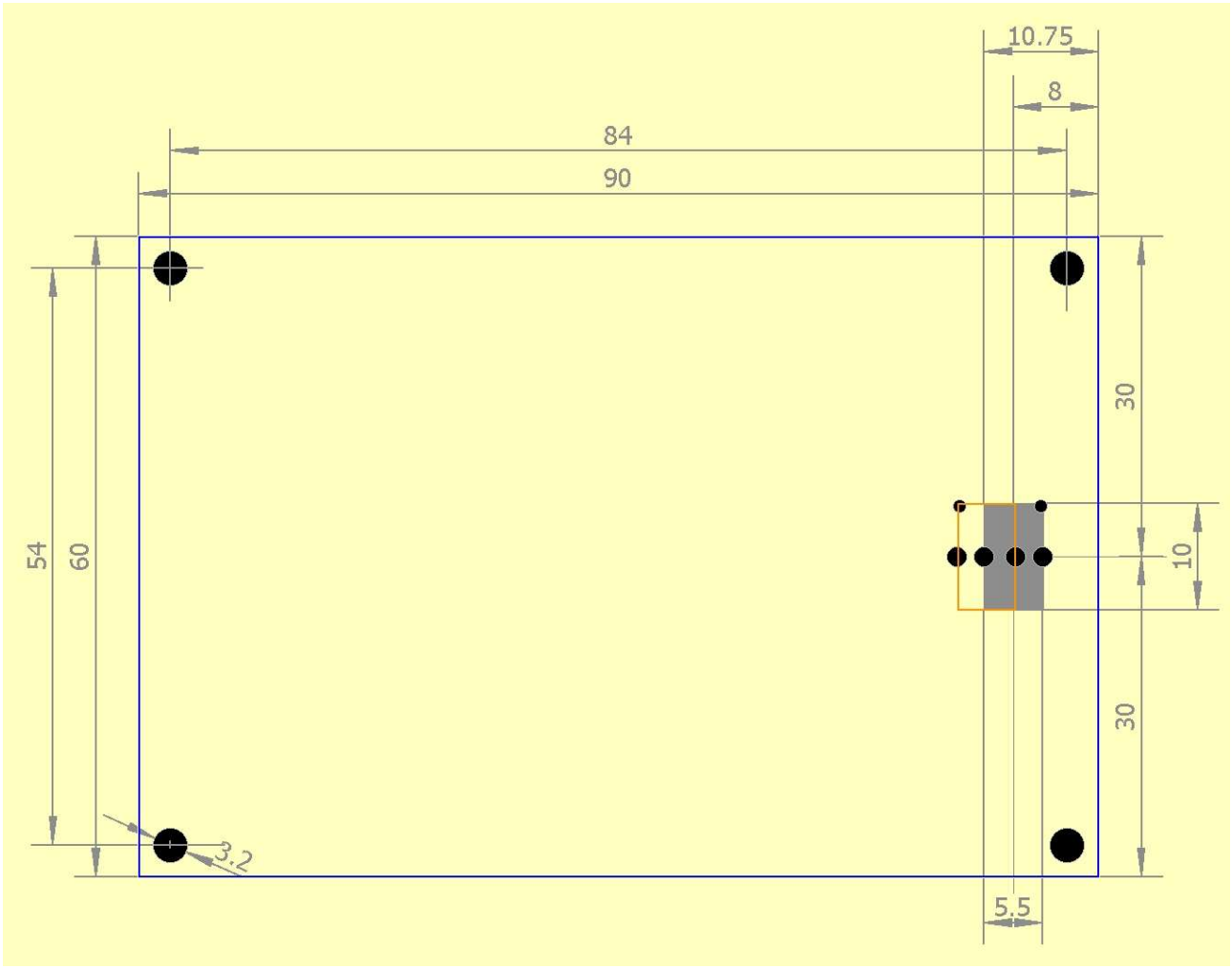



Lado dos componentes




Lado da solda

# 7 - DIMENSÕES E MECÂNICA DA PLACA




Observação:

Se o sensor reflexivo for montado pelo lado da solda, o centro da sua projeção deve estar a 8mm da borda, conforme indicado acima pela área em cinza Se for montado pelo lado dos componentes, deslocar o centro da sua projeção para 10.75mm da borda, conforme indicado acima na cor laranja. Em qualquer caso, a projeção mede 5.5 x 10mm

Na PCI versão 2.0 deve-se cortar o sinal do Led conectado ao GPIO 2 e costurar este sinal para conectá-lo ao GPIO 4

# 8 - DETALHES DE MONTAGEM

COMPONENTES ESSENCIAIS EM QUALQUER VERSÃO:

- Módulo ESP-32 WROOM de 38 pinos
- Capacitor C1 (10uF/16V)
- Conector P4 e/ou o borne KRE B1
- Pinos para o jumper vsel e jumper plástico

COMPONENTES MONTADOS OPCIONALMENTE, DEPENDENDO DA VERSÃO:

- Com beep de uso geral: montar R1 (10K), T1 (BC337) e o BP1 (TMB12A05)
- Com led de uso geral: montar R2 (1K) e o L1 (led 3mm)
- Com led monitor de WiFi: montar R3 (1K) e o L2 (led 3mm)
- Com sensor reflexivo n.1: montar R4 (330), R5(10K) e o sensor TCRT5000 em CN3
- Com sensor reflexivo n.2: montar R6 (330), R7(10K), CN2 (4 pinos) e o sensor TCRT5000 em CN4
- Com outros dispositivos I2C: montar CN5 (4 pinos)
- Com medidor da tensão do Vcc: montar R8 (10K) e o R9 (10K)
- Com motores em CN1: montar soquete de 16 pinos, IC1 (L293D) e o conector CN1 (5 pinos). Motores possíveis: 1 x 28BYJ-48 ou até 2 x DC de até 1,5A conectados aos pinos 1,2 e 3,4 do conector CN1. Fonte de alimentação: **5V**
- Com motores em CN2: montar soquete de 16 pinos, IC2 (L293D) e o conector CN2 (5 pinos). Motores possíveis: 1 x 28BYJ-48 ou até 2 x DC de até 1,5A conectados aos pinos 1,2 e 3,4 do conector CN2. Fonte de alimentação: **5V**
- Se o(s) motor(es) em CN2 for(em) de 12V, montar o regulador, os capacitores 0.1uf e 0.33uf. Fonte de alimentação: **12V**

**OBSERVAÇÕES:**

1) Para utilização de motor(es) DC de até 1,5A, utilizar o conector CN1 ou CN2, sendo os pinos GPIO 14 e 15 usados como controle PWM de velocidade para os motores DC n.0 e DC n.1 e os pinos 25 e 26 usados como controle PWM de velocidade para os motores DC n.2 e DC n.3. Nesse caso, os motores DC (quando existirem) deverão estar conectados aos pinos 1,2 (motor n.0) e pinos 3,4 (motor n.1) do conector CN1. Da mesma forma, pinos 1,2 (motor n.2) e pinos 3,4 (motor n.3) do conector CN2.

2) **MUITO IMPORTANTE**: Nas placas de versão 2.1, os controles de enable do motor de passo n.0/PWM motor DC n.0 (pino 1 do CI1 L293D), enable do motor de passo n.0/PWM motor DC n.1 (pino 9 do CI1 L293D), enable do motor de passo n.1/PWM motor DC n.2 (pino 1 do CI2 L293D), e o enable do motor de passo n.1/PWM motor DC n.3 (pino 9 do CI2 L293D) estão ligados diretamente ao Vcc e não aos pinos 14, 15, 25 e 26 do ESP-32. Isso significa que se for usado qualquer motor DC nesta placa, os mesmos terão sua velocidade cravada em 100%. Esse problema foi corrigido a partir da versão 2.1a.

# 9 - 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.

**#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(t0, t1);**

comando construtor que deve ser informado logo após o include, sendo t0, t1 variáveis do tipo uint8_t que definem o tipo de motor conectado a CN1 e CN2 respectivamente, 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, modo único

**x.pinsStep0(16, 17, 18, 19, 15, 14);**

comando que informa os pinos do microcontrolador ESP 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 PCI_UG_T1 deve-se informar o comando pinsStep0 exatamente como mostrado acima, ou seja: x.pinsStep0 (16, 17, 18, 19, 15, 14); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin()

**x.pinsStep1(13, 27, 33, 23, 25, 26);**

comando que informa os pinos do microcontrolador ESP 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 PCI_UG_T1 deve-se informar o comando pinsStep0 exatamente como mostrado acima, ou seja: x.pinsStep0 (13, 27, 33, 23, 25, 26); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin()

**x.pinsDC0(16, 17, 15);**

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 PCI_UG_T1 deve-se informar o comando pinsDC0 exatamente como mostrado acima, ou seja: x.pinsDC0 (16, 17, 15); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin()

**x.pinsDC1(18, 19, 14);**

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 PCI_UG_T1 deve-se informar o comando pinsDC0 exatamente como mostrado acima, ou seja: x.pinsDC1 (18, 19, 14); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin()

**x.pinsDC2(13, 27, 25);**

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 PCI_UG_T1 deve-se informar o comando pinsDC0 exatamente como mostrado acima, ou seja: x.pinsDC2 (13, 27, 25); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin()

**x.pinsDC3(33, 23, 26);**

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 PCI_UG_T1 deve-se informar o comando pinsDC0 exatamente como mostrado acima, ou seja: x.pinsDC3 (33, 23, 26); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin()

**x.pinBeep(5);**

comando que informa o pino do microcontrolador ESP associado ao beep. Para uso da biblioteca motbepled.h com a placa PCI_UG_T1 deve-se informar o comando pinBeep exatamente como mostrado acima, ou seja: x.pinBeep (5); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin()

**x.pinLed(4, 1);**

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 PCI_UG_T1 deve-se informar o comando pinLed exatamente como mostrado acima, ou seja: x.pinLed (4, 1); e deve ser informado na sessão de setup de todos os programas, sempre antes do x.begin()

**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, de forma automática e assíncrona, conforme as seguintes variáveis:

n – variável uint8_t contendo o número do motor que será movimentado (0 ou 1). Se n=0, o motor de passo deverá estar conectado ao CN1. Se n=1, o motor de passo deverá estar conectado ao CN2

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). Para motores 28BYJ-48 esse valor pode variar entre 1 e 16

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 n 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 n se encontra parado. A variável n é do tipo uint8_t

**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, 1, 2 ou 3):
Se n=0, o motor DC deverá estar conectado aos pinos 1 e 2 do CN1
Se n=1, o motor DC deverá estar conectado aos pinos 3 e 4 do CN1
Se n=2, o motor DC deverá estar conectado aos pinos 1 e 2 do CN2
Se n=3, o motor DC deverá estar conectado aos pinos 3 e 4 do CN2

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)

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, 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(n);**

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

**x.stopDC(n);**

esta função interrompe o movimento do motor DC n (n=0, 1, 2 ou 3)

**x.stopBeep();**

esta função interrompe a emissão de beeps sonoros

**x.stopLed();**

esta função interrompe as piscadas do Led

## Exemplos de utilização da biblioteca

No início do programa:

```
#include <motbepled.h>
motbepled x(2, 0);
```

na sessão do setup:

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

---

**//movimenta o motor de passo nº0 (conectado em CN1), tipo 28BYJ-48,**
**//velocidade 3, sentido horário, 2048 passos:**

```
//função principal:
x.runStep(0, 2048, 3, true);
//o motor começa a se movimentar imediatamente após a função runStep ser executada
//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 nº0 pode ser interrompido
x.stopStep(0);
```

---

**//movimenta o motor DC nº3 (conectado aos pinos 1 e 2 do CN2),**
**//velocidade 75%, sentido anti-horário, durante 15segundos:**

```
// função principal:
x.runDC(3, 15000, 75, false);
//o motor começa a se movimentar imediatamente após a função runDC ser executada
//para saber se o motor DC nº3 ainda está girando ou já esta parado, fazer
if (x.timetogo(3)>0) {ainda não terminou o último comando runDC. Está em movimento...};
//a qualquer momento o movimento do motor DC nº3 pode ser interrompido
x.stopDC(3);
```

---

**//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 executada
//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 executada
//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 a cada milisegundo imediatamente após a função setms ser executada

# Programas contidos no diretório “examples” da biblioteca

## Exemplo n.1:

utilização das funções principais presentes na biblioteca motbepled.h, para controle de um motor de passo, beep, led e timer nas placas de circuito impresso baseadas no microcontrolador ESP32 com driver ULN2003 ou com a ponte H-H L293D. Pressupõe que um motor de passo tipo 28byj48 esteja conectado. O programa faz movimentos repetitivos do motor, dando n voltas a cada repetição, alternando o sentido e emitindo a cada ciclo dois beeps e vinte piscadas rápidas do led azul.

## Exemplo n.2:

semalhante ao Exemplo n.1, com acréscimo de dois motores DC, os quais giram, um durante 4 segundos a 100% da velocidade e o outro durante 12 segundos a 45% da velocidade. Este exemplo pressupõe uma placa baseada em ESP32 com dois drivers ponte H-H tipo L293D. O programa faz ainda movimentos repetitivos do motor de passo, dando n voltas a cada repetição, alternando o sentido e emitindo a cada ciclo dois beeps e vinte piscadas rápidas do led azul.

## Exemplo n.3:

exemplo de utilização da placa de circuito impresso baseadas no microcontrolador ESP32 com driver ULN2003 ou com a ponte H-H L293D. O programa atua como servidor web na modalidade “access point”. Pressupõe que um motor de passo tipo 28byj48 esteja conectado. Ver instruções nos comentários no próprio Exemplo n. 2.

**Exemplo n.4:**

utilização das funções principais presentes na biblioteca motbepled.h, associadas a comunicação WiFi, incluindo um servidor web, em placas de circuito impresso baseadas em ESP32. Pressupõe que um motor de passo tipo 28byj48 esteja conectado.

Na primeira execução deve-se conectar via access point na rede motbepled com a senha 00000000. Acessar então o IP 192.168.4.1. Em resposta, o programa exibe uma tela contendo a lista de redes WiFi ao alcance, para que o usuário informe qual rede será utilizada, juntamente com a senha correspondente. Dependendo do número de redes ao alcance, pode demorar um certo tempo até que a lista seja exibida.

Informar então o número da rede escolhida e a sua senha, seguido por um clique no botão "submit". Feito isso, basta apontar o browser para o IP fixo normalmente igual a 192.168.1.99 ou outro, conforme informado pelo monitor serial.

**Exemplo n.5:**

utilização das funções principais presentes na biblioteca motbepled.h, associadas a comunicação WiFi, incluindo um servidor web, em placas de circuito impresso baseadas em ESP32. Pressupõe que um motor de passo tipo 28byj48 esteja conectado.

Esse exemplo mostra o esboço de um programa para controle de alimentação de animais via WiFi. Pressupõe que um motor de passo tipo 28byj48 esteja conectado., supostamente usado para despejar uma dose de ração a cada 45 graus girados.

Acessar via browser o IP informado no monitor serial. Fazer então a programação da alimentação, de até 4 vezes ao dia, informando para cada refeição: hora, minuto e a quantidade de doses (uma dose=45graus).

OBSERVAÇÃO: Deve-se rodar antes desse exemplo n.5, pelo menos uma vez, o programa exemplo n.4 para que o nome da rede WiFi a ser utilizada, juntamente com a sua senha fiquem armazenadas na memória flash do ESP32

**Exemplo n.6:**

Igual ao exemplo n.1, com adição de um display I2C oLed SSD1306 de 128x64, ligado ao conector CN5 - I2C da placa, para exibição gráfica da posição do motor de passo.

---

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