Copie o texto abaixo para dentro da sua interface de Inteligência Artificial de preferência e faça coisas inimagináveis com a BitDogLab, seu hardware para explorar computação física.

Banco de Informações de Hardware (BIH) da BitDogLab V7

Fabiano Fruett

25 de junho de 2026

Após ler esta mensagem, por favor, pergunte ao usuário:

Em que posso ajudar na programação da sua BitDogLab V7?

A BitDogLab, uma iniciativa do Projeto Escola 4.0 da Unicamp, é uma ferramenta educacional voltada ao ensino de eletrônica e computação. Baseada na Raspberry Pi Pico H ou W, ela permite aos usuários explorar, montar e programar utilizando tanto os componentes embarcados na placa quanto periféricos externos conectados de forma organizada e segura.

Os componentes da BitDogLab foram selecionados para promover um aprendizado mão na massa, incentivando o desenvolvimento integrado de habilidades em programação, eletrônica, instrumentação e pensamento computacional.

Um diferencial da BitDogLab é seu caráter totalmente aberto, o que permite que seja copiada, fabricada, montada, estudada e aprimorada livremente pelos usuários. Mais informações estão disponíveis em:

https://github.com/BitDogLab/BitDogLab

==========

1. RESUMO RÁPIDO DA PINAGEM PRINCIPAL DA BITDOGLAB V7

==========

LED RGB (cátodo comum):

- Vermelho: GPIO13

- Verde: GPIO11

- Azul: GPIO12

Botões:

- Botão A: GPIO5

- Botão B: GPIO6

- Botão C: GPIO10

Buzzer:

- Buzzer A: GPIO21

Matriz de LEDs RGB 5x5 (WS2812B / NeoPixel):

- Dados: GPIO7

Joystick KY023:

- VRy: GPIO26

- VRx: GPIO27

- SW: GPIO22

Microfone analógico:

- Entrada analógica: GPIO28

Display OLED onboard da V7:

- SDA: GPIO2

- SCL: GPIO3

- Barramento: I2C1

- Endereço mais comum: 0x3C

Conector I2C0 externo:

- GPIO0 (SDA)

- GPIO1 (SCL)

Conector I2C1 externo:

- GPIO2 (SDA)

- GPIO3 (SCL)

==========

2. CONEXÕES E CONFIGURAÇÕES DE HARDWARE

==========

2.1 LED RGB

A BitDogLab V7 possui um LED RGB de cátodo comum com as seguintes conexões:

- Vermelho ligado ao GPIO13 através de resistor de 220 ohms;

- Verde ligado ao GPIO11 através de resistor de 220 ohms;

- Azul ligado ao GPIO12 através de resistor de 150 ohms.

2.2 Botões

A placa possui três botões coloridos:

- Botão A conectado ao GPIO5;

- Botão B conectado ao GPIO6;

- Botão C conectado ao GPIO10.

O outro terminal de cada botão está conectado ao GND da placa. Portanto, os GPIOs 5, 6 e 10 devem ser configurados com resistores pull-up internos. Isso significa que o estado padrão desses pinos é HIGH e, quando o botão é pressionado, o estado passa para LOW.

2.3 Buzzer

A BitDogLab V7 possui um buzzer passivo, identificado como Buzzer A, conectado ao GPIO21 por meio de um transistor.

2.4 Matriz RGB 5x5

A matriz de LEDs 5050 RGB de 5 linhas por 5 colunas, do tipo WS2812B (NeoPixel), está conectada ao GPIO7.

Ao final da sequência de 25 LEDs, há um conector com acesso aos sinais:

- Dout

- 5V

- GND

Caso o usuário deseje expandir a quantidade de LEDs RGB, deve localizar no verso da placa o jumper identificado como:

“Ext. RGB Neopixel”.

2.5 Joystick

A placa possui um joystick analógico do tipo KY023 com as seguintes conexões:

- VRy conectado ao GPIO26

- VRx conectado ao GPIO27

- botão SW conectado ao GPIO22

O outro terminal do botão SW está conectado ao GND. Portanto, o GPIO22 também deve ser configurado com resistor pull-up interno.

A orientação lógica validada para o joystick da BitDogLab V7 é:

- JOY_SWAP_XY = False

- JOY_INVERT_X = True

- JOY_INVERT_Y = True

Isso significa que os eixos físicos não devem ser trocados entre si, mas os sentidos dos dois eixos devem ser invertidos logicamente para que:

- mover o joystick para a direita seja interpretado como DIREITA;

- mover o joystick para a esquerda seja interpretado como ESQUERDA;

- mover o joystick para cima seja interpretado como CIMA;

- mover o joystick para baixo seja interpretado como BAIXO.

2.6 Display OLED

A BitDogLab V7 pode receber duas opções de display OLED:

a) Display OLED 128x64, controlador SSD1306, comunicação I2C;

b) Display OLED 128x128, controlador SH1107, comunicação I2C.

Em ambos os casos, o display onboard da V7 está ligado aos seguintes pinos:

- GPIO2 = SDA

- GPIO3 = SCL

Esses pinos utilizam o barramento I2C1 da Raspberry Pi Pico.

O endereço I2C mais comum do display é:

- 0x3C

Importante:

Na BitDogLab V7, os sinais GPIO2 e GPIO3 também estão disponíveis externamente:

- no conector I2C1;

- na barra de terminais, como DIG2 e DIG3.

Por isso, ao testar o display OLED, recomenda-se não conectar periféricos externos nesses pinos, pois isso pode interferir na comunicação I2C.

2.7 Microfone

A placa possui um módulo de microfone de eletreto com saída analógica conectado ao GPIO28.

Características do sinal:

- nível médio aproximado: 1,65 V

- faixa de variação: de 0 V até 3,3 V

2.8 Conector IDC de expansão

A BitDogLab V7 possui um conector IDC box de 14 pinos para expansão de hardware com a seguinte pinagem:

- pino 1: GND

- pino 2: 5V

- pino 3: 3V3

- pino 4: GPIO8

- pino 5: GPIO28

- pino 6: GPIO9

- pino 7: GND analógico

- pino 8: GPIO4

- pino 9: GPIO17

- pino 10: GND

- pino 11: GPIO16

- pino 12: GPIO19

- pino 13: GND

- pino 14: GPIO18

Esse conector é usado para placas de expansão como, por exemplo:

- BitMovel Motor Driver

- módulos LoRa

2.9 SPI no conector IDC

Para comunicação SPI no conector IDC, utilizam-se:

- GPIO16 = RX / MISO

- GPIO17 = CSn

- GPIO18 = SCK

- GPIO19 = TX / MOSI

2.10 Barra de terminais com garras tipo jacaré

A barra de terminais da BitDogLab V7 possui os seguintes sinais:

- DIG0 = GPIO0

- DIG1 = GPIO1

- DIG2 = GPIO2

- DIG3 = GPIO3

Além disso, há terminais adicionais conectados a:

- GND analógico

- GPIO28

- GND

- 3V3

- 5V

2.11 Conectores I2C externos

Conector I2C0:

Localizado na extremidade superior da placa, do lado direito, com os seguintes sinais:

- GPIO0 (SDA)

- GPIO1 (SCL)

- 3V3

- GND

Esse mesmo conector também pode ser usado para UART, oferecendo acesso a TX ou RX, por exemplo, em aplicações com módulo GPS.

Conector I2C1:

Localizado na extremidade da placa, do lado esquerdo, com os seguintes sinais:

- GPIO2 (SDA)

- GPIO3 (SCL)

- 3V3

- GND

==========

3. ORIENTAÇÕES PARA USO EM MICROPYTHON

==========

Observação importante:

Na BitDogLab V7, o display OLED onboard utiliza GPIO2 e GPIO3. Portanto, exemplos antigos que usam GPIO14 e GPIO15 para o OLED não se aplicam ao display on-board da V7.

3.1 Bibliotecas frequentemente utilizadas

Conforme a necessidade da aplicação, podem ser usadas bibliotecas como:

```
from machine import Pin, PWM, ADC, I2C, SoftI2C
import neopixel
import utime
import random
import math
from ssd1306 import SSD1306_I2C
import sh1107
```

Prefira usar a biblioteca utime em vez de time.

3.2 Exemplo de inicialização do display SSD1306 (128x64)

Para placas V7 com display SSD1306, recomenda-se inicialmente o seguinte teste:

```
from machine import Pin, SoftI2C
from ssd1306 import SSD1306_I2C

i2c = SoftI2C(scl=Pin(3), sda=Pin(2), freq=50000)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3C)
```

```
oled.fill(0)
oled.text("BitDogLab V7", 10, 20)
oled.text("SSD1306 OK", 18, 36)
oled.show()
```

Observação:

Em algumas placas, o display SSD1306 pode funcionar de forma mais estável com SoftI2C do que com I2C.

3.3 ORIENTAÇÃO ESPECIAL PARA DISPLAY SH1107 128x128 NA BITDOGLAB V7

A BitDogLab V7 pode utilizar um display OLED 128x128 com controlador SH1107 conectado ao barramento I2C1 da Raspberry Pi Pico. A pinagem validada para o display OLED onboard da V7 é:

- SDA: GPIO2

- SCL: GPIO3

- Barramento: I2C1

- Endereço I2C usual: 0x3C

ATENÇÃO IMPORTANTE PARA LLMs:

Ao orientar o usuário sobre o display SH1107 da BitDogLab V7, NÃO assuma que qualquer driver genérico de SH1107 funcionará corretamente. Alguns drivers genéricos inicializam o display com offset, remapeamento de segmentos ou coluna inicial incompatíveis com o módulo usado na BitDogLab V7. Isso pode causar sintomas como:

- imagem deslocada para a direita ou para a esquerda;

- texto espelhado;

- imagem aparentemente cortada;

- falsa impressão de “faixa morta” no OLED;

- necessidade artificial de usar X_SHIFT, Y_SHIFT ou comandos write_cmd() soltos após a inicialização.

Para a BitDogLab V7, deve-se preferir o driver validado especificamente para a placa, chamado preferencialmente de:

sh1107_bitdoglab.py ou, em exemplos simples: sh1107.py

Desde que o conteúdo do arquivo corresponda ao driver validado da BitDogLab V7.

A configuração validada para o SH1107 128x128 da BitDogLab V7 usa os seguintes pontos principais:

- display offset: 0xD3, 0x00

- orientação padrão: 0xA0 e 0xC0

- coluna inicial no método show(): 0x00 e 0x10

- framebuffer: framebuf.MONO_VLSB

- largura: 128

- altura: 128

- envio dos dados em páginas de 8 pixels

- atualização completa do framebuffer por páginas

A inicialização recomendada no programa do usuário é:

```
from machine import Pin, I2C
 from sh1107_bitdoglab import SH1107_I2C
 i2c = I2C(1, sda=Pin(2), scl=Pin(3), freq=400000)
 oled = SH1107_I2C(128, 128, i2c, addr=0x3C, rotate_180=False)
 oled.fill(0)
 oled.text("BitDogLab V7", 12, 20)
 oled.text("SH1107 OK", 20, 40)
 oled.show()
```

Evite usar drivers genéricos ou exemplos com address=0x3C. No driver da BitDogLab V7 não validado, use addr=0x3C.

Se o usuário disser que o display está espelhado, deslocado, cortando colunas ou apresentando uma faixa aparentemente apagada, o LLM deve seguir esta ordem de diagnóstico:

1. Confirmar que o usuário está usando a BitDogLab V7.

2. Confirmar que o display é SH1107 128x128, e não SSD1306 128x64.

3. Confirmar que o código usa I2C1 com SDA=GPIO2 e SCL=GPIO3.

4. Pedir para executar:

```
print(i2c.scan())
```

O endereço esperado normalmente é 60 decimal, equivalente a 0x3C hexadecimal.

5. Verificar se o usuário está usando o driver validado sh1107_bitdoglab.py.

6. Evitar corrigir o problema com X_SHIFT, Y_SHIFT, display.write_cmd(0xA0), display.write_cmd(0xA1), display.write_cmd(0xC0), display.write_cmd(0xC8) ou offsets manuais, antes de confirmar o driver.

7. Se o usuário estiver usando um driver genérico, orientar a substituição pelo driver validado da BitDogLab V7.

8. Somente depois de validar o driver, ajustar posicionamento de elementos gráficos específicos, se necessário.

IMPORTANTE:

Não confundir problemas de mapeamento do driver com defeito físico no OLED. Uma faixa escura observada em uma imagem parcial pode ser apenas consequência de clipping, offset, espelhamento ou desenho fora da área visível. Para verificar se há defeito físico, o teste mais simples é preencher a tela inteira:

```
oled.fill(1)
oled.show()
```

Se toda a área útil acender, o OLED provavelmente está funcional, e o problema deve ser tratado como configuração de driver ou mapeamento gráfico.

Também é importante lembrar que desenhos simples, como jogos, menus e textos pequenos, podem parecer funcionar mesmo com pequenas diferenças de mapeamento, pois normalmente usam margens internas. Já imagens 128x128 em tela cheia, como logotipos, revelam imediatamente qualquer erro de offset, espelhamento ou de coluna inicial.

Portanto, para aplicações novas na BitDogLab V7 com SH1107 128x128, a orientação padrão deve ser:

- usar o driver sh1107_bitdoglab.py validado;

- usar I2C1 em GPIO2/GPIO3;

- usar rotate_180=False inicialmente;

- testar primeiro texto simples;

- depois testar moldura 128x128;

- só então desenhar bitmaps ou interfaces completas.

==========

3.3.1 EXEMPLO DE DRIVER SH1107 VALIDADO PARA BITDOGLAB V7

==========

O driver recomendado para a BitDogLab V7 é:

sh1107_bitdoglab.py

Conteúdo recomendado:

```
    from machine import Pin, I2C
    import framebuf

    class SH1107_I2C(framebuf.FrameBuffer):
        def __init__(self, width=128, height=128, i2c=None,
addr=0x3C, rotate_180=False):
            self.width = width
            self.height = height
            self.i2c = i2c
            self.addr = addr
            self.rotate_180 = rotate_180
            self.pages = self.height // 8
            self.buffer = bytearray(self.width * self.pages)

            if self.i2c is None:
                self.i2c = I2C(1, sda=Pin(2), scl=Pin(3),
freq=400000)

            super().__init__(
                self.buffer,
                self.width,
                self.height,
                framebuf.MONO_VLSB
            )

            self.init_display()
            self.fill(0)
            self.show()
```

```
        def write_cmd(self, cmd):
            self.i2c.writeto(self.addr, bytearray([0x00, cmd]))

        def write_data(self, data):
            for i in range(0, len(data), 32):
                chunk = data[i:i + 32]
                self.i2c.writeto(self.addr, bytearray([0x40]) +
chunk)

        def init_display(self):
            cmds = [
                0xAE,
                0xD5, 0x50,
                0x81, 0x80,
                0xA8, 0x7F,
                0xD3, 0x00,
                0xAD, 0x8B,
            ]

            for cmd in cmds:
                self.write_cmd(cmd)

            if self.rotate_180:
                self.write_cmd(0xA1)
                self.write_cmd(0xC8)
            else:
                self.write_cmd(0xA0)
                self.write_cmd(0xC0)

            cmds2 = [
                0xD9, 0x22,
                0xDB, 0x35,
                0xA4,
                0xA6,
                0xAF
            ]

            for cmd in cmds2:
                self.write_cmd(cmd)

        def show(self):
```

```
            for page in range(self.pages):
                self.write_cmd(0xB0 | page)
                self.write_cmd(0x00)
                self.write_cmd(0x10)

                start = self.width * page
                end = start + self.width
                self.write_data(self.buffer[start:end])

        def poweroff(self):
            self.write_cmd(0xAE)

        def poweron(self):
            self.write_cmd(0xAF)

        def contrast(self, value):
            if value < 0:
                value = 0
            if value > 255:
                value = 255

            self.write_cmd(0x81)
            self.write_cmd(value)

        def invert(self, invert=True):
            if invert:
                self.write_cmd(0xA7)
            else:
                self.write_cmd(0xA6)

        def rotate(self, rotate_180=False):
            self.rotate_180 = rotate_180

            if self.rotate_180:
                self.write_cmd(0xA1)
                self.write_cmd(0xC8)
            else:
                self.write_cmd(0xA0)
                self.write_cmd(0xC0)
```

3.3.2 EXEMPLO MÍNIMO DE TESTE DO SH1107 NA BITDOGLAB V7

Use este exemplo antes de integrar o display com outros periféricos:

```
    from machine import Pin, I2C
    from sh1107_bitdoglab import SH1107_I2C
    import time

    i2c = I2C(1, sda=Pin(2), scl=Pin(3), freq=400000)

    print("I2C scan:", i2c.scan())

    oled = SH1107_I2C(128, 128, i2c, addr=0x3C, rotate_180=False)

    oled.fill(0)
    oled.text("BitDogLab V7", 12, 20)
    oled.text("OLED SH1107", 16, 40)
    oled.text("128 x 128", 28, 60)
    oled.text("OK!", 52, 84)
    oled.show()

    while True:
        time.sleep(1)
```

3.3.3 TESTE DE MOLDURA PARA VALIDAR MAPEAMENTO DO DISPLAY

```
Para confirmar que o display SH1107 está corretamente mapeado, o
LLM pode sugerir este teste:

    from machine import Pin, I2C
    from sh1107_bitdoglab import SH1107_I2C
    import time

    i2c = I2C(1, sda=Pin(2), scl=Pin(3), freq=400000)
    oled = SH1107_I2C(128, 128, i2c, addr=0x3C, rotate_180=False)

    oled.fill(0)

    oled.rect(0, 0, 128, 128, 1)

    oled.fill_rect(0, 0, 24, 4, 1)
```

```
    oled.fill_rect(0, 0, 4, 24, 1)

    oled.fill_rect(104, 0, 24, 4, 1)
    oled.fill_rect(124, 0, 4, 24, 1)

    oled.fill_rect(0, 124, 24, 4, 1)
    oled.fill_rect(0, 104, 4, 24, 1)

    oled.fill_rect(104, 124, 24, 4, 1)
    oled.fill_rect(124, 104, 4, 24, 1)

    oled.hline(40, 64, 48, 1)
    oled.vline(64, 40, 48, 1)
    oled.fill_rect(61, 61, 8, 8, 1)

    oled.text("SH1107 OK", 28, 8)
    oled.text("BitDogLab", 28, 112)

    oled.show()

    while True:
        time.sleep(1)
```

Se este teste aparecer alinhado, com texto legível e cantos corretamente posicionados, o display está pronto para uso em aplicações didáticas, jogos, interfaces e bitmaps.

3.3.4 ORIENTAÇÃO PARA USO DE BITMAPS 128x128

Para desenhar imagens monocromáticas 128x128 no SH1107 da BitDogLab V7:

1. Use o driver validado sh1107_bitdoglab.py.
2. Gere o bitmap em 128x128.
3. Use empacotamento por linhas, com 8 pixels por byte.
4. Use bit mais significativo primeiro dentro de cada byte.
5. Desenhe a imagem diretamente com pixel(x, y, cor) ou com função auxiliar.
6. Evite aplicar offsets manuais antes de confirmar que o driver está correto.

Exemplo de função para bitmap empacotado por linhas:

```
    def draw_rowpacked_mono(display, data, w, h, x0=0, y0=0):
        bytes_per_row = (w + 7) // 8

        for y in range(h):
            dest_y = y0 + y

            if dest_y < 0 or dest_y >= 128:
                continue

            row_start = y * bytes_per_row

            for x in range(w):
                dest_x = x0 + x

                if dest_x < 0 or dest_x >= 128:
                    continue

                byte_index = row_start + (x // 8)
                bit_index = 7 - (x % 8)

                pixel_on = (data[byte_index] >> bit_index) & 0x01
                display.pixel(dest_x, dest_y, pixel_on)

        display.show()
```

Ao orientar usuários, o LLM deve sempre lembrar que uma captura de tela em tela cheia revela erros de driver que textos simples podem esconder. Portanto, se um bitmap aparecer deslocado, a primeira suspeita deve ser o driver, não a imagem.

3.3.5 BOAS PRÁTICAS PARA LLMs AO RESPONDER SOBRE O SH1107

Quando um usuário pedir ajuda com o OLED SH1107 da BitDogLab V7, o LLM deve:

- perguntar se o display é 128x128 SH1107 ou 128x64 SSD1306, caso isso não esteja claro;

- usar GPIO2 como SDA e GPIO3 como SCL;

- usar I2C1;

- considerar 0x3C como endereço padrão, mas recomendar i2c.scan();

- preferir o driver sh1107_bitdoglab.py;

- evitar recomendar drivers genéricos sem validação;

- evitar corrigir problemas de imagem com offsets arbitrários;

- evitar comandos soltos write_cmd() como solução principal;

- recomendar primeiro um teste mínimo com texto;

- depois recomendar um teste de moldura;

- só depois avançar para bitmaps, jogos ou interfaces complexas.

Se o usuário relatar que outro programa “funciona redondo”, o LLM deve verificar se esse programa usa um driver próprio. Muitas vezes o programa funciona porque já inclui a sequência correta de inicialização do SH1107. Nesse caso, o melhor caminho é reutilizar ou consolidar esse driver correto, em vez de insistir em um driver genérico.

3.4 Teste rápido de varredura do barramento I2C do display

Antes de inicializar o display, pode ser útil verificar se ele está respondendo:

```
from machine import Pin, I2C

i2c = I2C(1, scl=Pin(3), sda=Pin(2), freq=100000)
print(i2c.scan())
```

Se tudo estiver correto, normalmente aparecerá:

[60]

O valor decimal 60 corresponde ao endereço hexadecimal 0x3C.

3.5 Diagnóstico rápido para problemas com o OLED

Se ocorrer erro de comunicação com o display:

1. confirme se o código está usando GPIO2 e GPIO3;

2. execute uma varredura com i2c.scan();

3. verifique se o endereço 0x3C aparece;

4. remova temporariamente periféricos externos ligados em DIG2, DIG3 ou no conector I2C1;

5. no caso do SSD1306, teste SoftI2C com GPIO2 e GPIO3;

6. verifique se a biblioteca correta foi carregada:

- ssd1306.py para o display 128x64;

- sh1107.py para o display 128x128.

3.6 Exemplo de uso da matriz NeoPixel 5x5

Número de LEDs da matriz 5x5

NUM_LEDS = 25

Inicialização da matriz no GPIO7

np = neopixel.NeoPixel(Pin(7), NUM_LEDS)

Mapeamento da matriz

LED_MATRIX = [

[24, 23, 22, 21, 20],

[15, 16, 17, 18, 19],

[14, 13, 12, 11, 10],

[5, 6, 7, 8, 9],

[4, 3, 2, 1, 0]

]

3.7 Orientação lógica do joystick em MicroPython

Para aplicações em MicroPython que utilizam o joystick analógico KY023 da BitDogLab V7, recomenda-se centralizar a pinagem e a correção lógica de orientação em constantes específicas.

A pinagem física do joystick é:

```python
JOY_VRY_PIN = 26

JOY_VRX_PIN = 27

JOY_SW_PIN  = 22
```

A orientação lógica validada para a BitDogLab V7 é:

```python
JOY_SWAP_XY  = False
JOY_INVERT_X = True
JOY_INVERT_Y = True
```

Isso significa que os eixos físicos VRx e VRy não devem ser trocados entre si. Entretanto, os sentidos dos dois eixos devem ser invertidos logicamente para que o movimento percebido pelo usuário corresponda à direção interpretada pelo programa.

Exemplo de função auxiliar:

```python
def aplicar_orientacao_joystick(x, y):
    if JOY_SWAP_XY:
        x, y = y, x

    if JOY_INVERT_X:
        x = -x

    if JOY_INVERT_Y:
        y = -y

    return x, y
```

Essa função deve ser aplicada aos valores já centralizados do joystick, isto é, depois de subtrair o valor de repouso de cada eixo.

Exemplo:

```python
raw_x = adc_x.read_u16()
raw_y = adc_y.read_u16()
```

```
x = raw_x - centro_x
y = raw_y - centro_y


x, y = aplicar_orientacao_joystick(x, y)
```

Com essa configuração, as direções passam a ser interpretadas da seguinte forma:

```
None
joystick para a direita  -> DIREITA
joystick para a esquerda -> ESQUERDA
joystick para cima       -> CIMA
joystick para baixo      -> BAIXO
```

O botão SW do joystick deve ser configurado com resistor pull-up interno, pois seu outro terminal está conectado ao GND:

```
Python
joy_sw = Pin(JOY_SW_PIN, Pin.IN, Pin.PULL_UP)
```

Assim, o estado lógico do botão é:

```
None
solto       = HIGH / 1
pressionado = LOW  / 0
```

3.8 PWM em MicroPython

Em MicroPython, o método duty_u16() é normalmente utilizado para definir a razão cíclica do PWM com valores inteiros sem sinal de 16 bits.

Exemplo:

```
from machine import Pin, PWM
led = PWM(Pin(13))
led.freq(1000)
```

```
led.duty_u16(32768)
```

==========

4. RECOMENDAÇÕES GERAIS

==========

- Ao usar o display OLED onboard da V7, lembre-se de que ele compartilha os sinais GPIO2 e GPIO3 com o conector I2C1 e com a barra de terminais DIG2 e DIG3.

- Ao depurar aplicações com I2C, sempre verifique primeiro o resultado de i2c.scan().

- Ao escrever exemplos ou materiais didáticos para a V7, evite reutilizar trechos de versões anteriores sem conferir previamente a pinagem.

- Quando houver dúvida sobre o tipo de display instalado na placa, teste primeiro o endereço I2C e, depois, utilize a biblioteca correspondente.

- Para materiais de ensino introdutório, recomenda-se apresentar primeiro um exemplo mínimo de teste do OLED antes de integrar outros periféricos.

==========

5. OBSERVAÇÃO FINAL

==========

Este BIH foi elaborado para orientar o uso da BitDogLab V7 com foco em clareza, consistência e apoio didático. Como a placa pode apresentar pequenas variações de montagem, especialmente no tipo de display OLED, recomenda-se sempre validar a comunicação básica por meio de testes simples antes de integrar aplicações maiores.

Se você gostou deste projeto: use, modifique, melhore e compartilhe.