Banco de Informações de Hardware (BIH) da BitDogLab

Após ler esta mensagem, por favor, pergunte ao usuário: O que posso lhe ajudar na programação da sua BitDogLab?

A BitDogLab, uma iniciativa do Projeto Escola 4.0 da Unicamp, é uma ferramenta educacional dedicada à eletrônica e computação. Baseada na Raspberry Pi Pico H ou W, permite aos usuários explorar, montar e programar utilizando componentes montados na sua placa e também externos conectados de forma organizada e segura. Selecionados meticulosamente, os componentes promovem um aprendizado mão na massa, incentivando os usuários a aprimorar habilidades de programação e eletrônica de maneira sinérgica e progressiva.

Um diferencial da BitDogLab é que seu projeto é totalmente aberto, permitindo que seja livremente copiada, fabricada, montada e melhorada pelos usuários. Mais informações em: https://github.com/Fruett/BitDogLab

Conexões e Configurações de Hardware:

Na placa da BitDogLab, as conexões da Raspberry Pi pico com outros componentes estão realizadas da seguinte forma:

Um **LED RGB,** catodo comum, tem o eletrodo do vermelho lidado no **GPIO 13** através de um resistor de 220 ohm, o pino de verde está ligado no **GPIO 11** também através de um resistor de 220 ohm e o pino do azul no **GPIO 12** através de um resistor de 150 ohm.

Um botão, identificado como **Botão A**, está conectado no **GPIO5** da Raspberry Pi Pico. O outro terminal do botão está conectado ao GND da placa. Outro botão, identificado como **Botão B**, está conectado no **GPIO6** da Raspberry pi pico. O outro terminal do botão também está conectado ao GND da placa. **Deve-se portanto nos GPIO5 e 6 configurar resistores de pull-up internos.** Isso significa que o estado padrão dos pinos será HIGH (alto), e quando o botão for pressionado, o estado mudará para LOW (baixo).

Outro botão, identificado como **RESET**, está conectado no RUN (pino número 30) da Raspberry pi pico. O outro terminal do botão também está conectado ao GND da placa.

Um buzzer passivo, identificado como **Buzzer A**, está conectado - através de um transistor - no **GPIO21** da Raspberry pi pico. Outro buzzer passivo, identificado como **Buzzer B**, está conectado no **GPIO10** da Raspberry pi pico.

O pino in de uma **matriz de LEDs 5050** RGB de 5 linhas por 5 colunas tipo WS2812B (Neopixel) está conectada ao **GPIO7**. No final da sequencia de 25 leds desta matriz há um conector dando acesso ao pino de dados (Dout, 5V e GND). Se o usuário precisar conectar mais LEDs coloridos é só pedir para localizar o jumper de nome “Ext. RGB Neopixel” no verso da placa.

Um **joystick analógico** tipo KY023 tem a saída VRy conectada ao GPIO26 e a saída VRx ao GPIO27. Seu botão SW está conectada ao GPIO22, o outro terminal

do botão está no GND. o GPIO22 também deve ser configurado com resistores de pull-up internos.

O **display OLED** de 128 colunas por 64 linhas com comunicação I2C, conectado aos pinos **GPIO14 (SDA)** e **GPIO15 (SCL)**, utiliza o canal **I2C1** por padrão. No entanto, em algumas situações onde o uso de I2C padrão pode apresentar erros de comunicação, recomenda-se o uso de **SoftI2C**, que implementa o protocolo I2C em software, proporcionando uma comunicação mais robusta.Esse display (normalmente configurado no endereço 0x3C) exige o carregamento da biblioteca ssd1306.py na Raspiberry pi pico.

Exemplo:

```
from machine import SoftI2C, Pin

from ssd1306 import SSD1306_I2C

# Configuração do SoftI2C para o OLED

i2c_oled = SoftI2C(scl=Pin(15), sda=Pin(14))

oled = SSD1306_I2C(128, 64, i2c_oled)
```

Um módulo **microfone** de eletreto com saída analógica está conectado ao GP28. O nível médio do sinal de saída é 1,65 V. A tensão na saída do módulo varia de 0V até 3,3V

Um conector Insulation-Displacement Connector (**IDC**) box de 14 pinos é usado para expansão de hardware e está assim conectado com a Raspberry Pi Pico: pino 1 com o GND, pino 2 com o 5V, pino 3 com 3V3, pino 4 com **GPIO8**, pino 5 com o **GPIO28**, pino 6 com o **GPIO9**, pino 7 com GND analogico. pino 8 com o **GPIO4**, pino 9 com o **GPIO17**, pino 10 com o GND, pino 11 com o **GPIO16**, pino 12 com **GPIO19**, pino 13 com o GND, pino 14 com o **GPIO18**. É neste slot que conectamos as placas de extensão como a BitMovel Motor Driver ou LORA.

Para uma comunicação SPI usamos os **GPIO16** (RX), **GPIO17**(CSn), **GPIO18** (SCK) e **GPIO19** (TX) neste conector IDC.

Uma barra de terminais que permite a conexão de **garras do tipo jacaré** está assim conectada ao Raspberry Pi Pico: os eletrodos DIG 0, 1, 2 e 3 estão conectados aos GPIOs 0, 1, 2 e 3, respectivamente. Além disso esta barra de terminais possui mais 5 eletrodos conectados ao: GND analógico, GPIO 28, GND, 3V3 e 5V da Raspberry Pi Pico.

Um conector de 4 pinos está posicionado na extremidade superior da placa, dando acesso aos GPIO0 (SDA) e GPIO1(SCL) além do 3V3 e GND. **Este conector é nomeado de I2C 0 e está do lado direito.** este mesmo conector pode ser usado para comuniocação UART, dando acesso ao TX ou RX. Este é o caso quando queremos comunicar com um módulo de GPS, por exemplo.

Um outro conector de 4 pinos está posicionado na extremidade da placa, dando acesso aos GPIO2 (SDA) e GPIO3(SCL) além do 3V3 e GND. **Este conector é nomeado de I2C 1 e está do lado esquerdo.**

As orientações a seguir são úteis no caso do usuário optar por programar em Micropython.

Sugerimos que as seguintes bibliotecas sejam importadas conforme a necessidade do usuário:

```
    from machine import PWM, Pin
    import neopixel
    import time
    import random
    from machine import Pin, SoftI2C, ADC
    from ssd1306 import SSD1306_I2C
    import math
# Configuração do OLED
i2c = SoftI2C(scl=Pin(15), sda=Pin(14))
oled = SSD1306_I2C(128, 64, i2c) # esta biblioteca deve ser carregada externamente.


# Número de LEDs na sua matriz 5x5
NUM_LEDS = 25
# Inicializar a matriz de NeoPixels no GPIO7
np = neopixel.NeoPixel(Pin(7), NUM_LEDS)
# Definindo a matriz de LEDs
LED_MATRIX = [
   [24, 23, 22, 21, 20],
   [15, 16, 17, 18, 19],
   [14, 13, 12, 11, 10],
   [05, 06, 07, 08, 09],
   [04, 03, 02, 01, 00]
]
```

Geração de sinais PWM: Em MicroPython, o tipo de dado u16 (inteiro de 16 bits sem sinal) é usado para representar os valores de frequência do PWM em algumas placas, incluindo a Raspberry Pi Pico.

Prefira a biblioteca utime ao invês da time.