PROTOCOLOS: UART - I2C - SPI - Serial communications

Sergio Lucas e Vinícius Souza

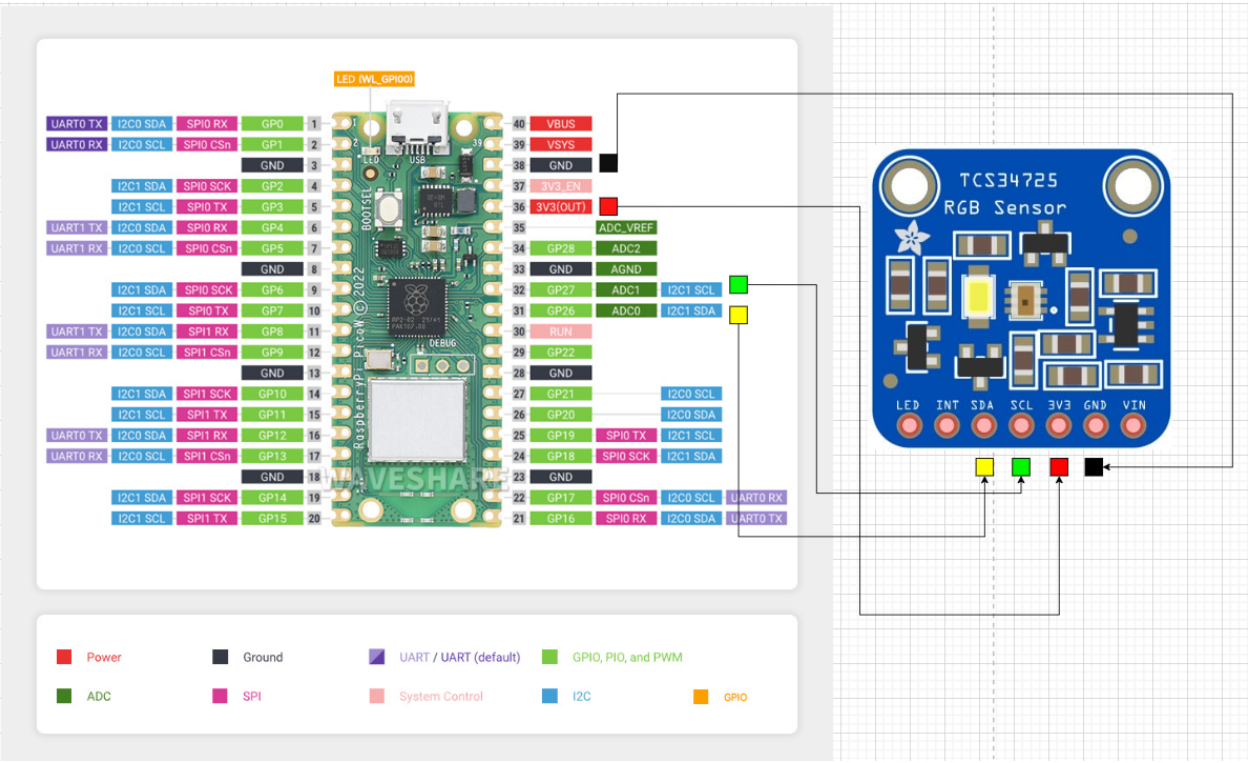
Introdução

Neste trabalho, foi executado um código em MicroPython em um Raspberry Pi Pico W, onde se foi testado um sensor RGB TCS34725 utilizando a comunicação I2C.

Materiais e Métodos

- Raspberry Pi Pico W
- Sensor RGB TCS34725
- Plataforma de desenvolvimento Thonny

Diagrama



No diagrama acima, é fizemos as seguintes ligações:

- Pino 38 Raspberry Pi Pico W (GND) -> Pino GND TCS34725
- Pino 36 Raspberry Pi Pico W (3V3) -> Pino 3V3 TCS34725

- Pino 32 Raspberry Pi Pico W (I2C1 SCL) -> Pino SCL TCS34725
- Pino 31 Raspberry Pi Pico W (I2C1 SDA) -> Pino SDA TCS34725

Representado pelas cores Preto (GND), Vermelho (3V3), Verde (I2C1 SCL) e Amarelo (I2C1 SDA), como foi feito na montagem física (Ver imagem em resultados)

Na explicação dos pinos, temos o pino terra (GND) e o pino de alimentação (3V3), que servem para alimentar o dispositivo e o manter ligado.

Os pinos específicos do I2C são o I2C1 SCL e I2C1 SDA, SCL quer dizer “Serial Clock” e SDA “Serial Data”, eles são pinos de entrada/saída bidirecional.

O Raspberry Pi Pico W tem diversas entradas I2C, porém apenas 2 controladores, sendo estes o I2C0 e I2C1. Neste caso, estamos utilizando o controlador I2C1.

Fora isso, também notamos que há 3 pinos não utilizados pelo sensor TCS34725, que são os pinos LED, INT e VIN. A funcionalidade desses pinos é:

- LED: Pode ser conectado a um LED RGB e mostrar, por exemplo, a cor obtida.
- INT: Pino de interrupção usado para alertar o microcontrolador quando ocorrem eventos específicos.
- VIN: Pino de alimentação externa para fornecer energia ao sensor, possível alternativa ao 3V3.

Descritivo

A comunicação I2C entre o microcontrolador Raspberry Pi Pico W e o sensor TCS34725 é feita pela conexão entre o SDA e o SCL, que ocorre da seguinte forma:

O SDA carrega os dados e o SCL carrega o sinal de relógio para sincronização da comunicação I2C, eles são essenciais para o funcionamento da comunicação I2C pois garantem que tanto o dispositivo principal quanto o dispositivo secundário (no caso, o microcontrolador e o sensor) estão sincronizados em relação ao momento em que os dados são enviados e recebidos. Sem essa sincronização, poderiam ocorrer erros de comunicação com os dispositivos e potencialmente lendo dados incorretos ou em momentos inadequados.

Em termos práticos, primeiro o microcontrolador envia um sinal de início, seguido pelo endereço do sensor com um bit indicando se a operação é de leitura ou escrita. Se o

sensor reconhece o endereço, ele responde com um sinal de reconhecimento (ACK), e a transferência de dados começa, realizando todo o processo de SDA/SCL descrito acima.

Já a comunicação serial entre o microcontrolador e um PC é feita normalmente através de uma conexão USB ou UART (Universal Asynchronous Receiver/Transmitter), o processo consiste de 2 passos; a transmissão de dados (TX) e a recepção de dados (RX)

Em ambos os casos, os dados são enviados um bit por vez, começando pelo bit menos significativo (LSB). Essas comunicações são assíncronas, ou seja, não há um sinal de relógio compartilhado entre o microcontrolador e o PC, como na comunicação entre o microcontrolador e o sensor. Ao invés disso, ambos os dispositivos devem concordar com uma taxa de transmissão (baud rate) antes da comunicação.

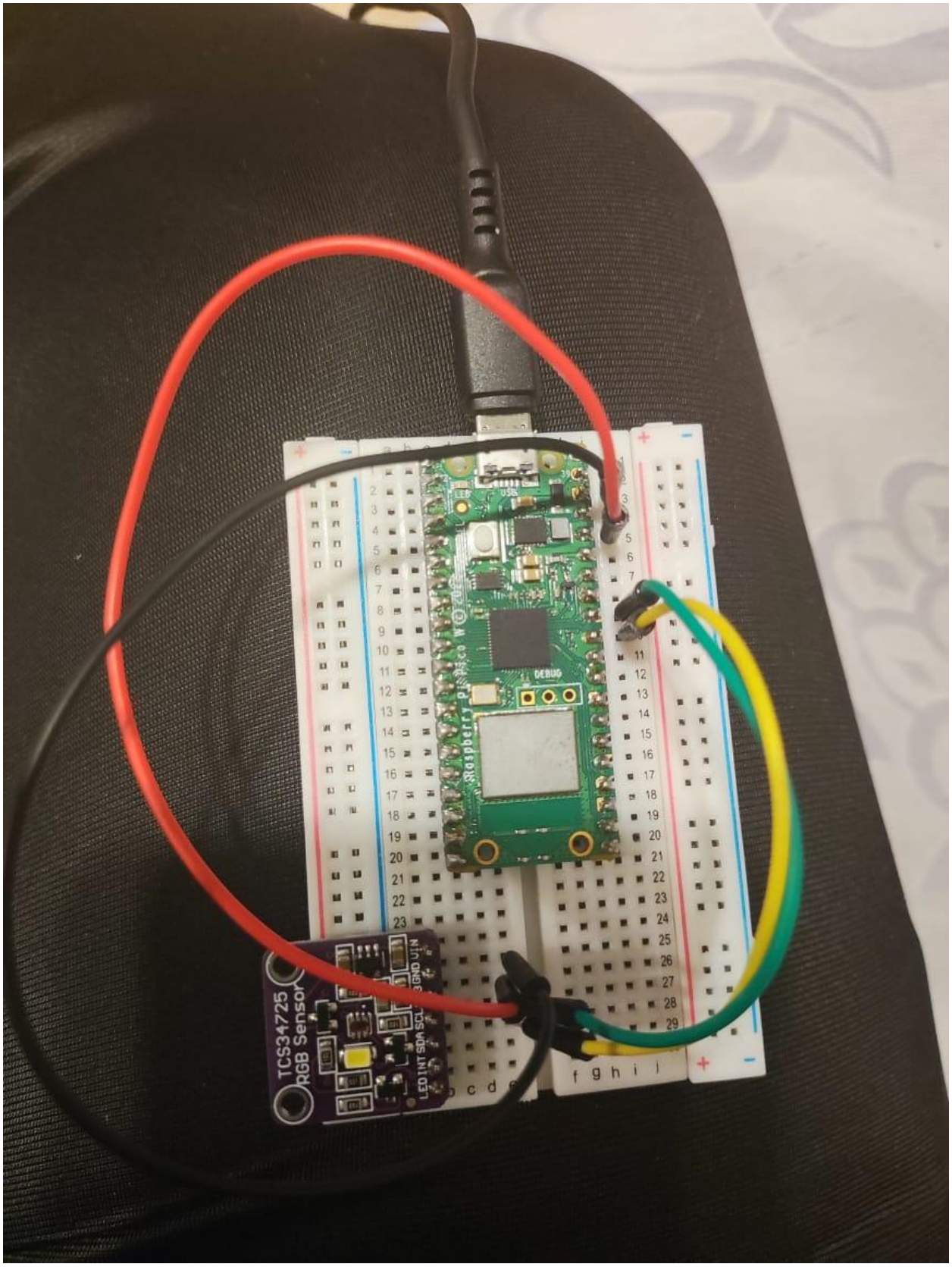
No nosso caso, estamos utilizando a comunicação USB, então quando o Raspberry Pi Pico W precisa enviar seus dados, ele utiliza a UART interna para converter os dados em serial e os envia ao pc, que tem um driver de dispositivo virtual COM que utiliza para ler esses dados.

Código

```
import machine
import utime

# Utiliza a entrada I2C, pinos 32 e 31
i2c = machine.I2C(1, scl=machine.Pin(27), sda=machine.Pin(26))
endereco_sensor = 0x29  # Endereço padrão do sensor RGB TCS34725
bit_comando = 0x80  # Bit de comando para registrar endereços

i2c.writeto_mem(endereco_sensor, 0x00 | bit_comando, bytearray([0x01]))  # Ligar o
Sensor
utime.sleep_ms(3)
i2c.writeto_mem(endereco_sensor, 0x00 | bit_comando, bytearray([0x01 | 0x02]))  #
Habilitar o sensor

def ler_cor(register):
    return int.from_bytes(i2c.readfrom_mem(endereco_sensor, register | bit_comando,
2), 'little') # Devolve os valores lidos em little endian
```

```
while True:
    r = ler_cor(0x16)   # Cor vermelha (Red)
    g = ler_cor(0x18) # Cor verde (Green)
    b = ler_cor(0x1A)  # Cor azul (Blue)
    c = ler_cor(0x14) # Intensidade (Clear)
    print(f"Vermelho: {r}, Verde: {g}, Azul: {b}, Intensidade: {c}")
    utime.sleep(1)
```

Resultados


TCS34725
RGB Sensor
LED INT SDA SCL 3GND VIN
DEBUG

Thonny - <untitled> @ 6 : 19

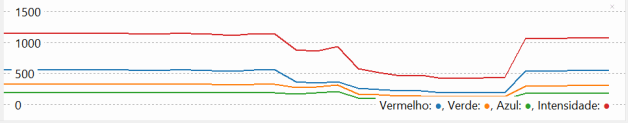
File Edit View Run Tools Help

<untitled> *

```python
import machine
import utime

# Utiliza a entrada I2C, pinos 32 e 31
i2c = machine.I2C(1, scl=machine.Pin(27), sda=machine.Pin(26))
endereco_sensor = 0x29  # Endereço padrão do sensor RGB TCS34725
bit_comando = 0x80  # Bit de comando para registrar endereços

i2c.writeto_mem(endereco_sensor, 0x00 | bit_comando, bytearray([0x01]))  # Ligar o Sensor
utime.sleep_ms(3)
i2c.writeto_mem(endereco_sensor, 0x00 | bit_comando, bytearray([0x01 | 0x02]))  # Habilitar o sensor

def ler_cor(register):
    return int.from_bytes(i2c.readfrom_mem(endereco_sensor, register | bit_comando, 2), 'little') # Devolve os valores lidos em little end

while True:
    r = ler_cor(0x16)   # Cor vermelha (Red)
    g = ler_cor(0x18) # Cor verde (Green)
    b = ler_cor(0x1A)  # Cor azul (Blue)
    c = ler_cor(0x14) # Intensidade (Clear)
```

Shell

```
Vermelho: 191, Verde: 133, Azul: 85, Intensidade: 432
Vermelho: 192, Verde: 130, Azul: 82, Intensidade: 425
Vermelho: 535, Verde: 296, Azul: 179, Intensidade: 1059
Vermelho: 537, Verde: 298, Azul: 180, Intensidade: 1064
Vermelho: 542, Verde: 301, Azul: 182, Intensidade: 1075
Vermelho: 542, Verde: 301, Azul: 182, Intensidade: 1074
Vermelho: 542, Verde: 301, Azul: 182, Intensidade: 1074
```



MicroPython (Raspberry Pi Pico) • Board CDC @ COM4