

Raspberry Pi Pico W / RP2040: Interfaces de Comunicação UART, SPI e I2C

Unidade 4 | Capítulo 6

Executores:



Coordenação:



Iniciativa:



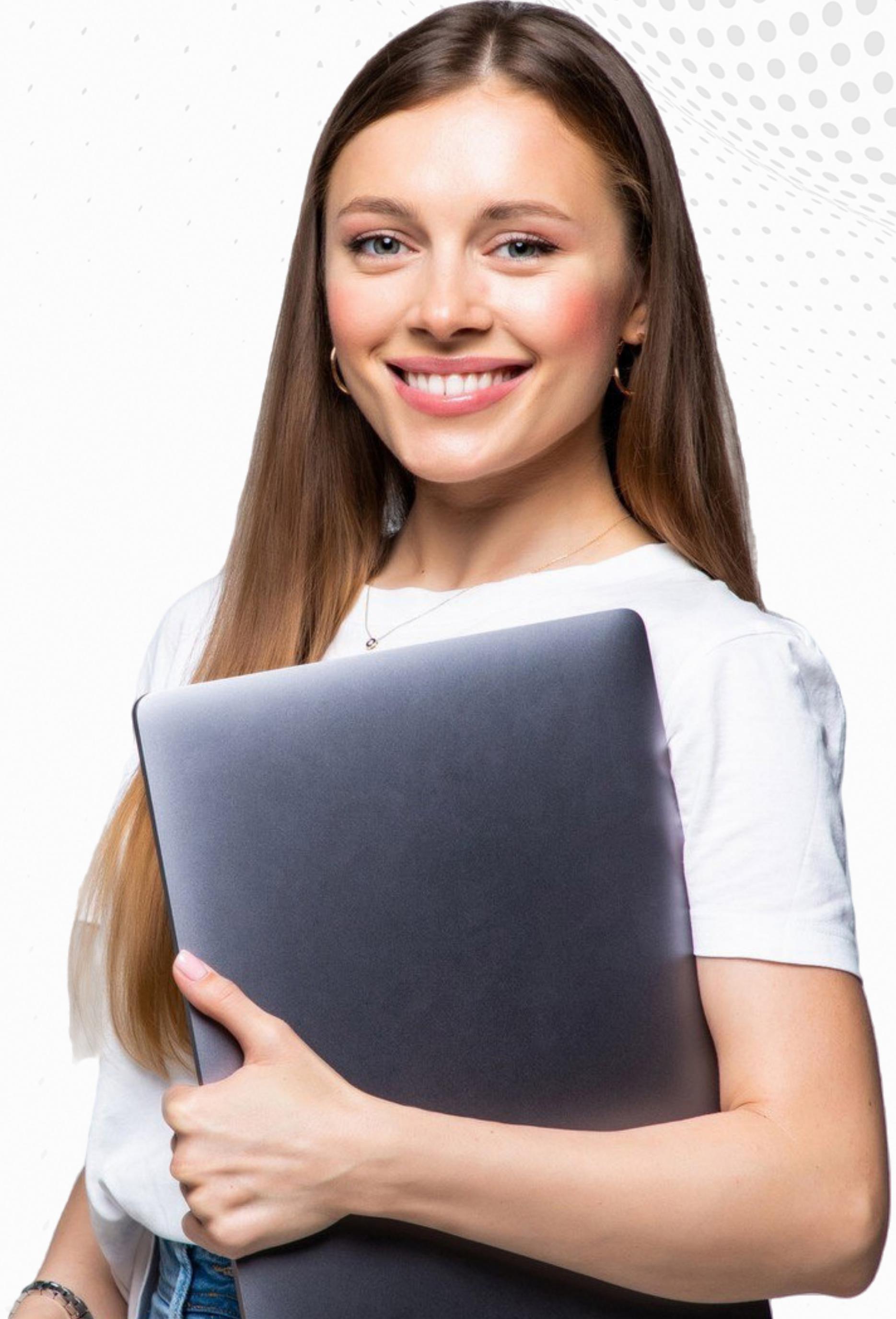
Sumário

- Objetivos
- RaspberryPi Pico W / RP2040
- Contexto de Interfaces Seriais
- Revisão Capítulo 5
- Interfaces Seriais RP2040 /Raspberry Pi Pico W
- SPI
- I2C
- UART
- Conclusão



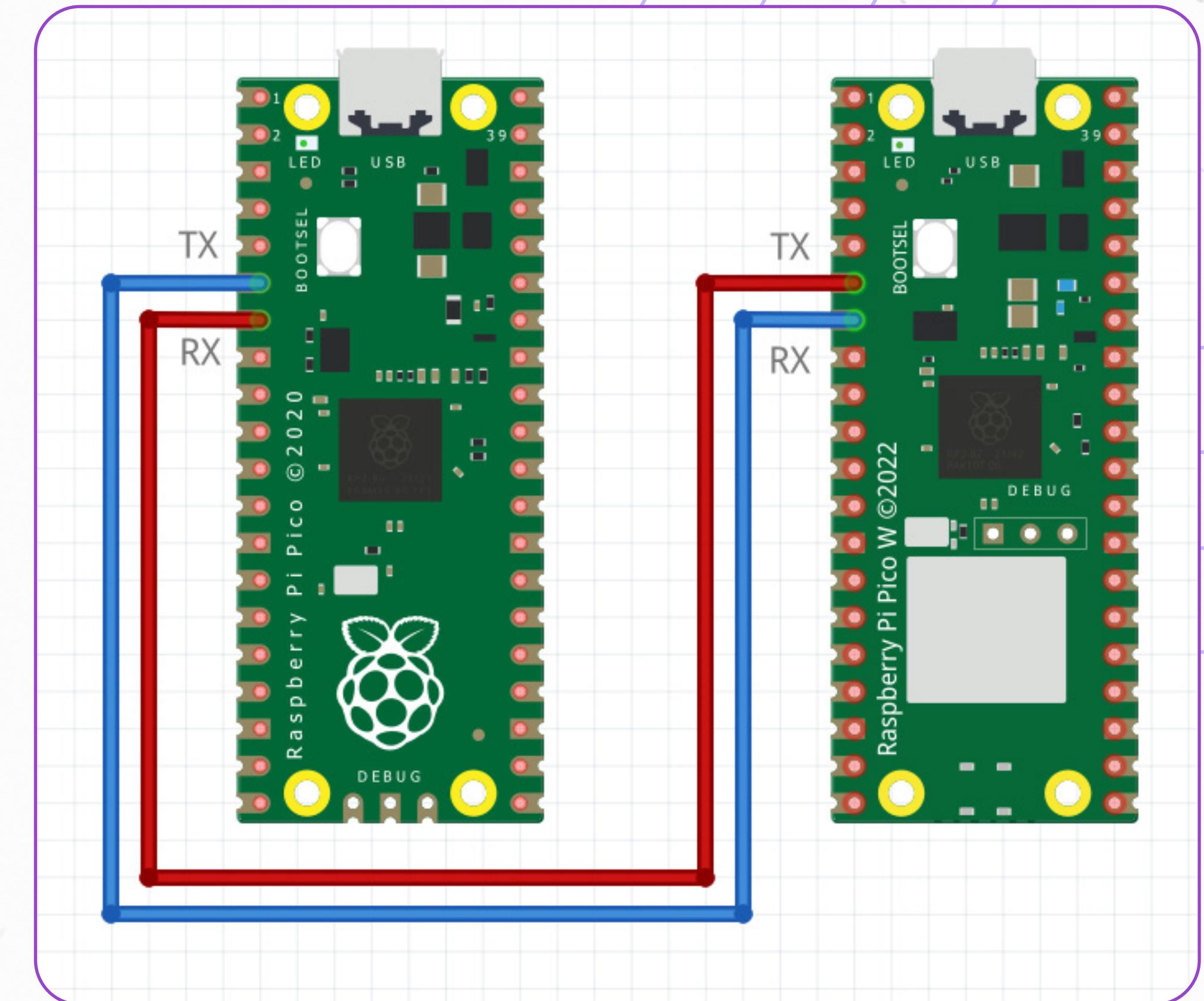
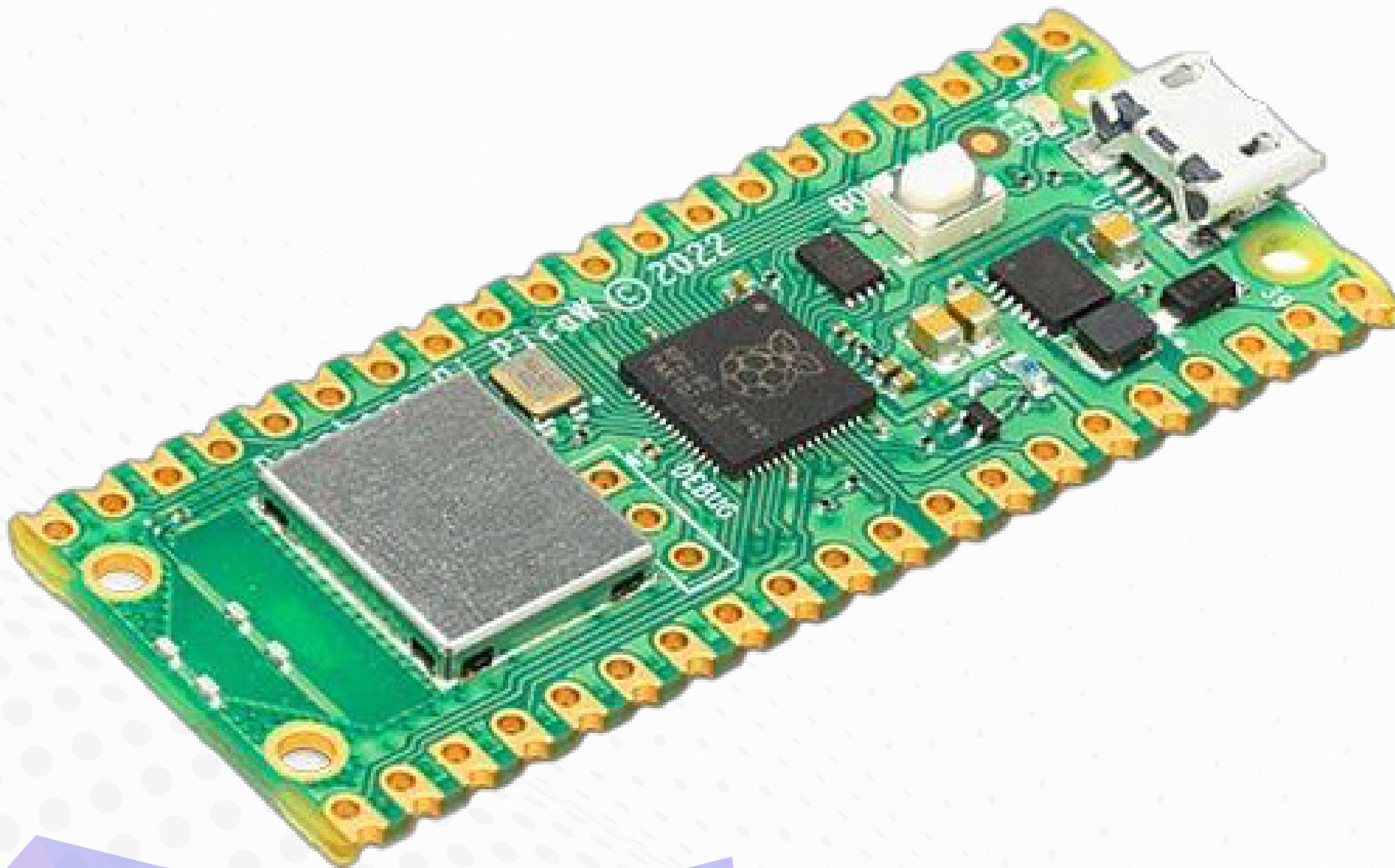
Objetivos

- Programar e conectar, corretamente, um sistema embarcado com outros sistemas embarcados ou outros dispositivos com esses protocolos de comunicação.



RaspberryPi Pico W / RP2040

Interfaces de Comunicação UART, SPI e I2C



Fonte: <https://www.makerhero.com/produto/raspberry-pi-pico-w/>

Contexto de Interfaces Seriais

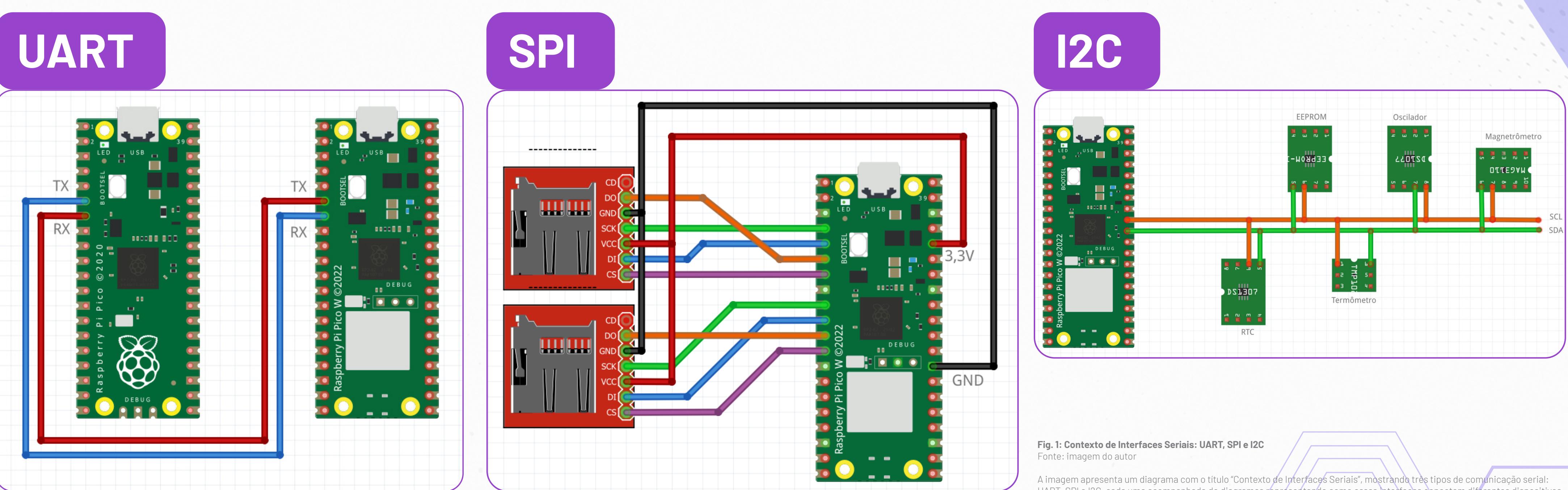


Fig. 1: Contexto de Interfaces Seriais: UART, SPI e I2C

Fonte: imagem do autor

A imagem apresenta um diagrama com o título "Contexto de Interfaces Seriais", mostrando três tipos de comunicação serial: UART, SPI e I2C, cada uma acompanhada de diagramas representando como essas interfaces conectam diferentes dispositivos.

UART:

Na seção esquerda, duas placas Raspberry Pi Pico estão conectadas através de seus pinos TX (transmissão) e RX (recepção). O TX da primeira placa está conectado ao RX da segunda placa, e o RX da primeira está conectado ao TX da segunda. Os fios de conexão são mostrados nas cores vermelha e azul.

SPI:

Na seção do meio, há uma placa Raspberry Pi Pico conectada a dois dispositivos que parecem ser leitores de cartão SD. As conexões são mais numerosas, com vários fios coloridos ligando pinos da Pico a esses dispositivos. As linhas conectam os pinos SPI, como MISO, MOSI, SCK e CS, além de fornecer alimentação (3,3V) e terra (GND).

I2C:

Na seção direita, a Raspberry Pi Pico está conectada a quatro dispositivos diferentes, representando sensores ou módulos. Esses dispositivos estão conectados em uma topologia de barramento, todos usando os mesmos dois fios (um verde e um laranja) que correspondem aos pinos de comunicação serial I2C (SCL para clock e SDA para dados). Os dispositivos conectados incluem uma EEPROM, um acelerômetro e outros módulos, todos compartilhando a mesma linha de dados e clock.

Revisão Capítulo 5

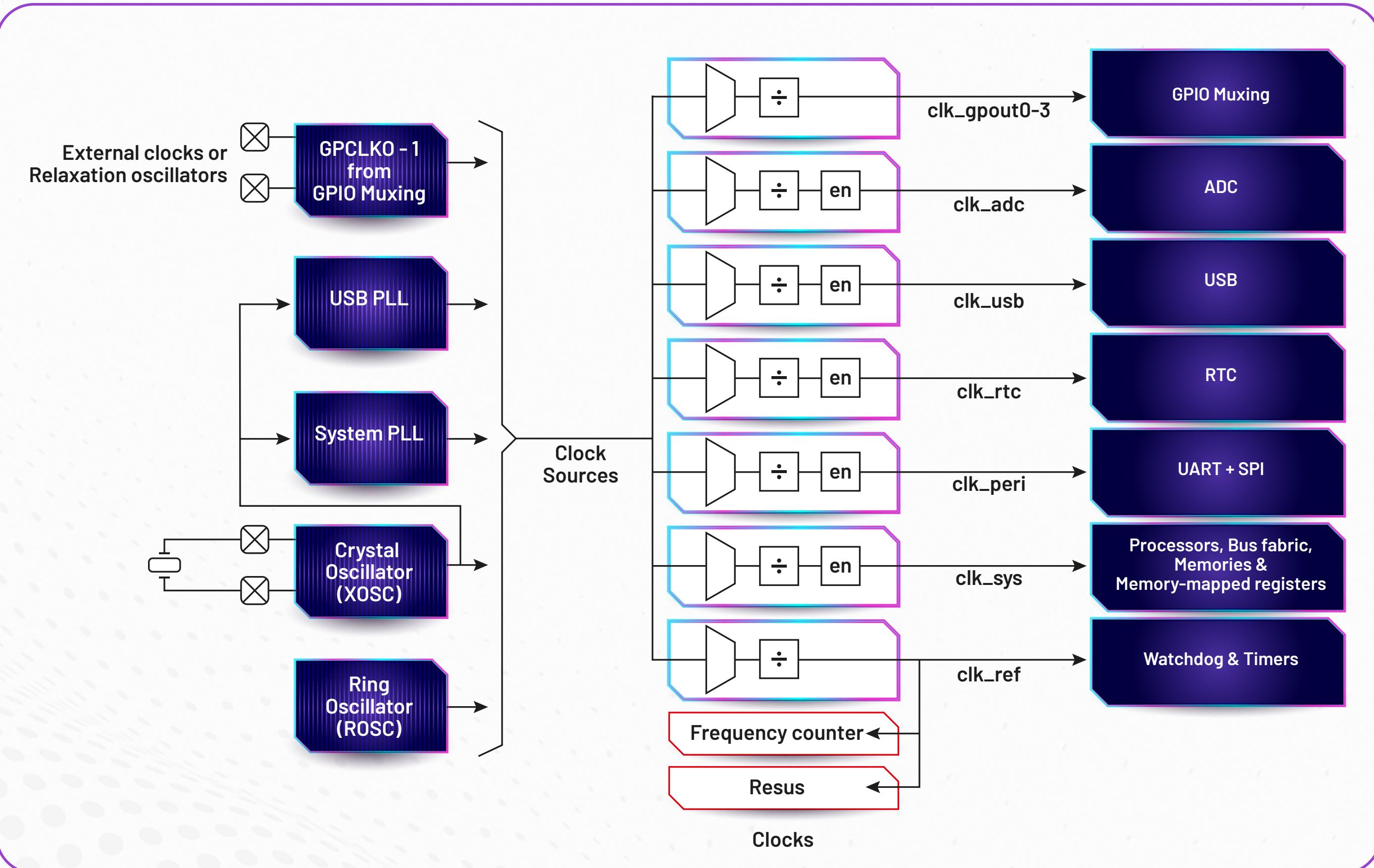


Fig. 2. Visão Geral do Clock.

Fonte: <https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf>

A imagem mostra um diagrama de blocos que descreve o sistema de clock de um dispositivo. O diagrama mostra as diferentes fontes de clock disponíveis, incluindo osciladores externos, USB PLL, System PLL, Crystal Oscillator (XOSC), Ring Oscillator (ROSC) e Frequency Counter.

Os clocks são então distribuídos para diferentes blocos do dispositivo, incluindo GPIO Multiplexing, ADC, USB, RTC, UART+SPI, Processors, Bus Fabric, Memories & Memory-mapped registers e Watchdog & Timers.

O diagrama usa caixas e linhas para representar os diferentes blocos e conexões. As caixas estão rotuladas com os nomes dos blocos e as linhas representam os sinais de clock. Cada bloco de clock, incluindo osciladores externos, USB PLL,

System PLL, Crystal Oscillator (XOSC), Ring Oscillator (ROSC) e Frequency Counter, é conectado a um conjunto de portas "não-e" que habilitam ou desabilitam o clock para os diferentes blocos do dispositivo. O diagrama mostra como os clocks são gerados, distribuídos e gerenciados no dispositivo.

Pico Examples: Lines 13 - 21

```
static uint64_t get_time(void) {
// Reading low latches the high value
uint32_t lo = timer_hw->timelr;
uint32_t hi = timer_hw->timehr;
return ((uint64_t) hi << 32u) | lo;
}
```

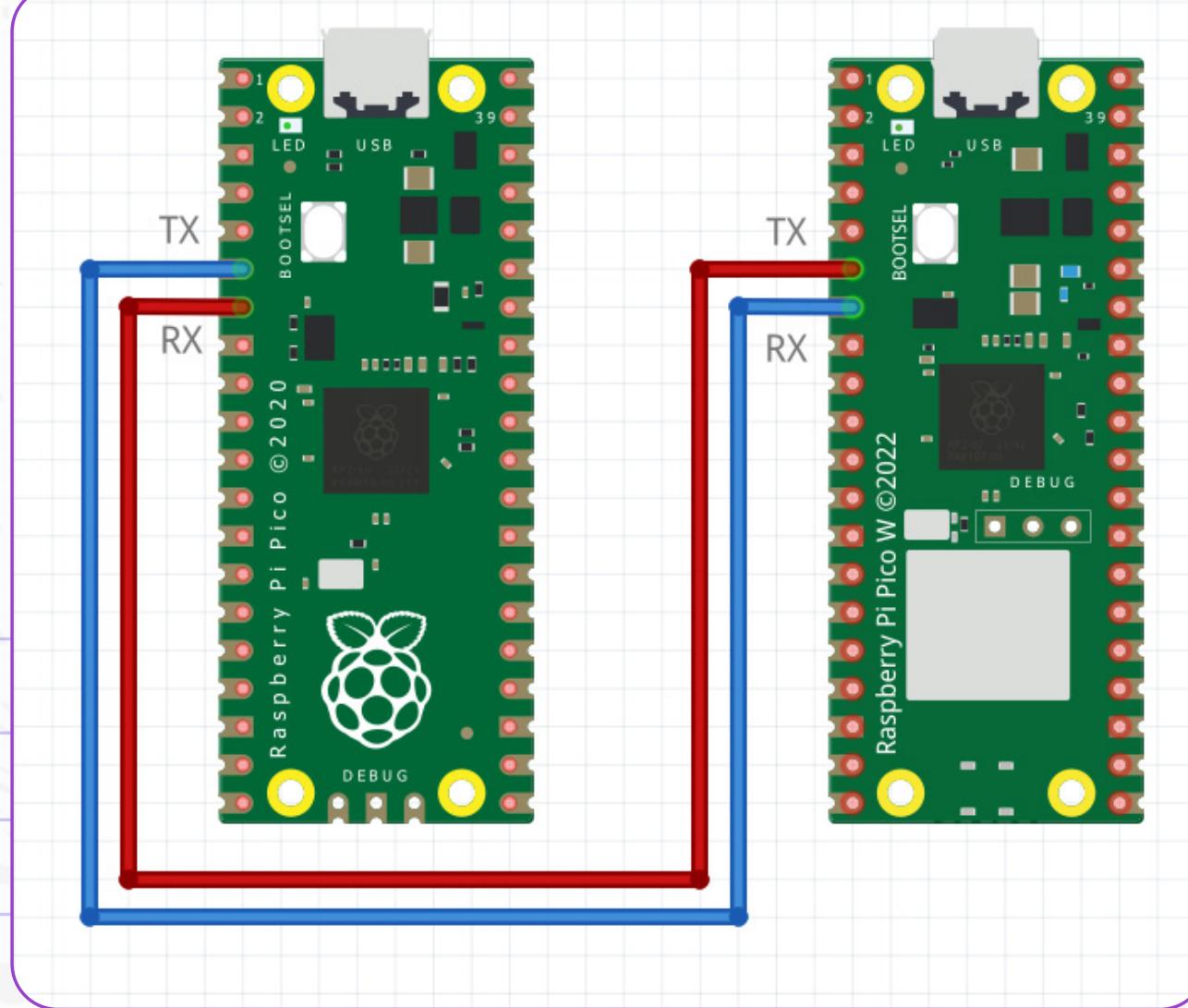
Fig. 3. Pico Examples: Lines 13 - 21

Fonte: https://github.com/raspberrypi/pico-examples/blob/master/timer/timer_lowlevel/timer_lowlevel.c

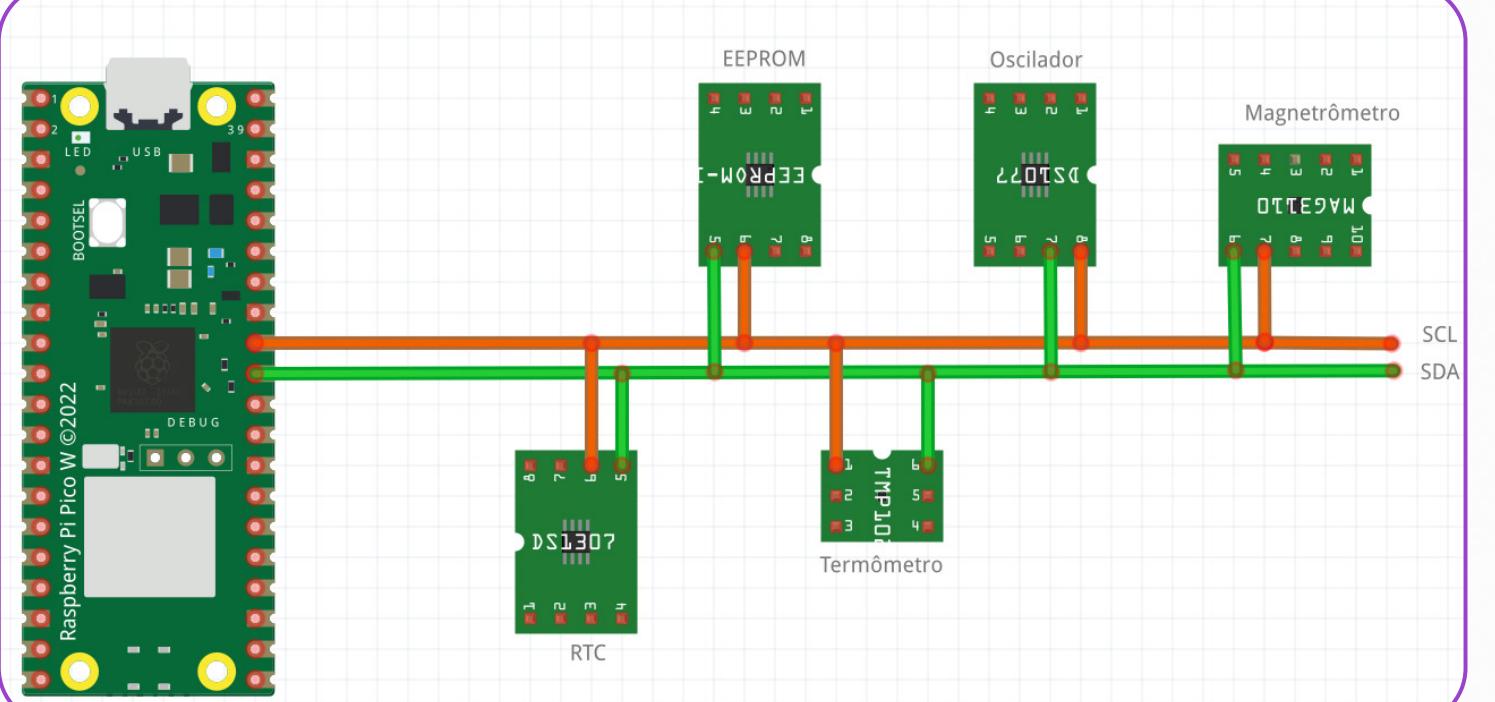
```
static uint64_t get_time(void){
// Reading low latches the high value
uint32_t lo = timer_hw->timelr;
uint32_t hi = timer_hw->timehr;
return ((uint64_t) hi << 32u) | lo;
}
```

Interfaces Seriais RP2040 / Raspberry Pi Pico W

UART



I2C



SPI

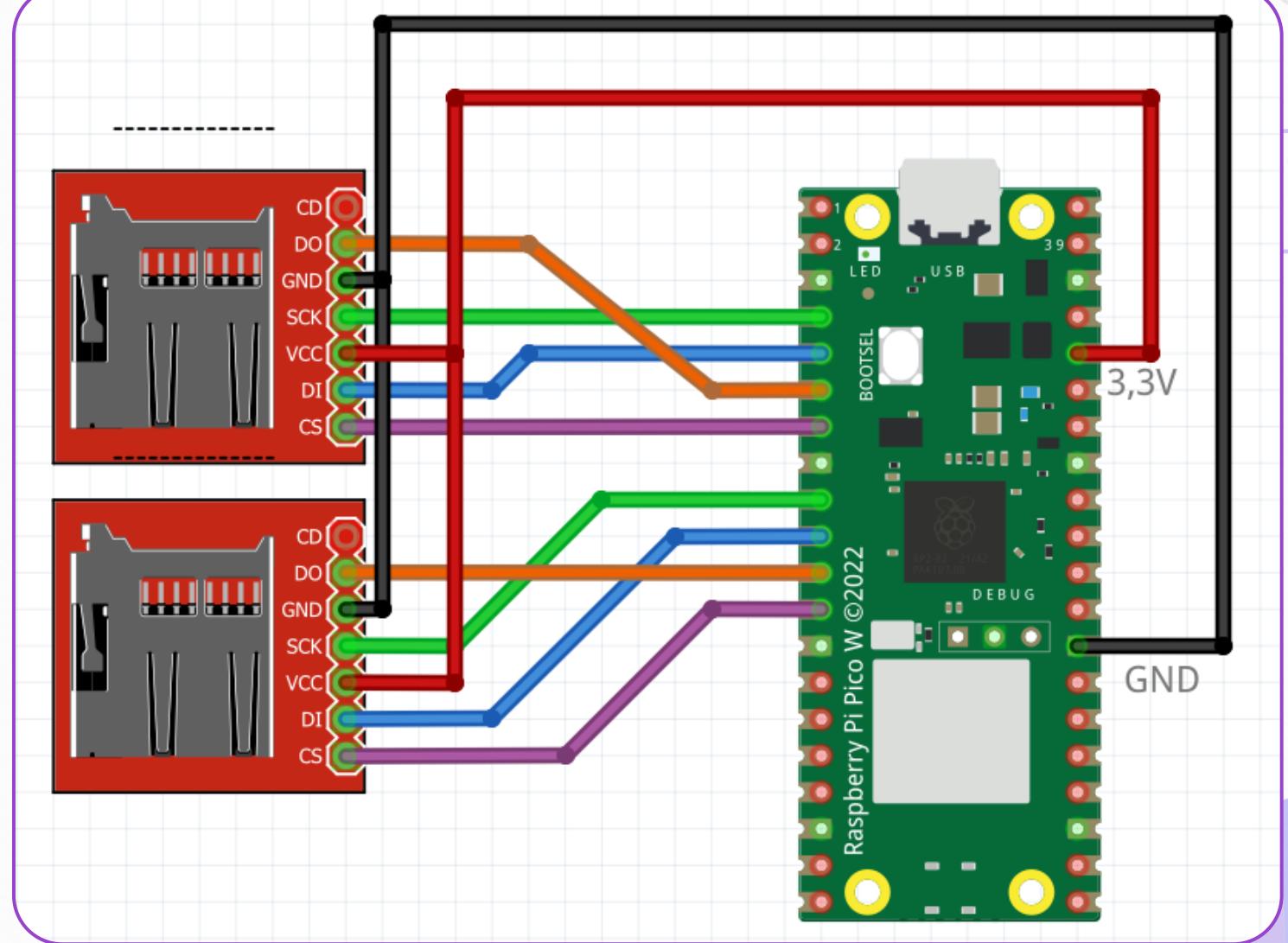
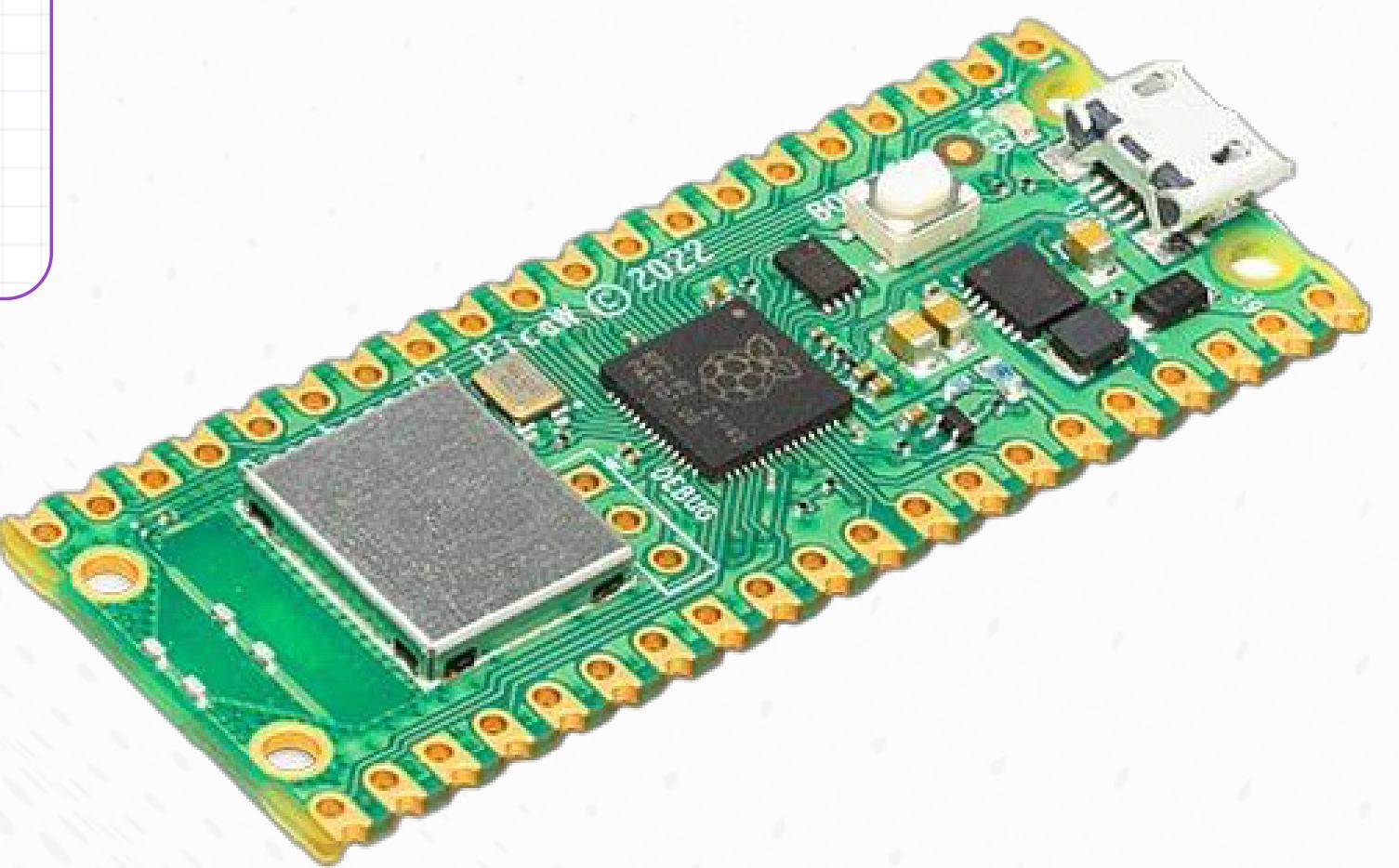


Fig. 1: Contexto de Interfaces Seriais: UART, SPI e I2C
Fonte: imagem do autor



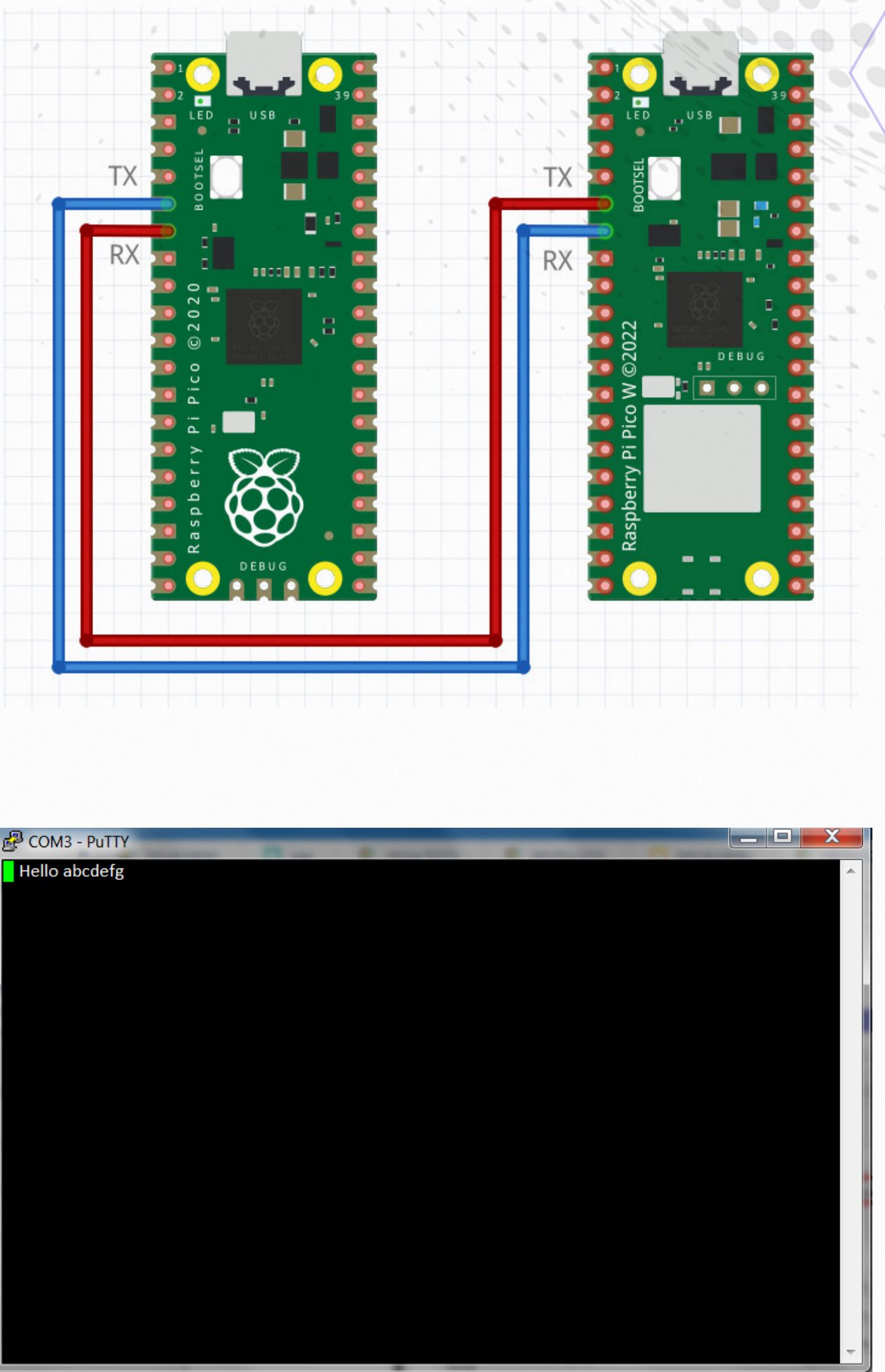
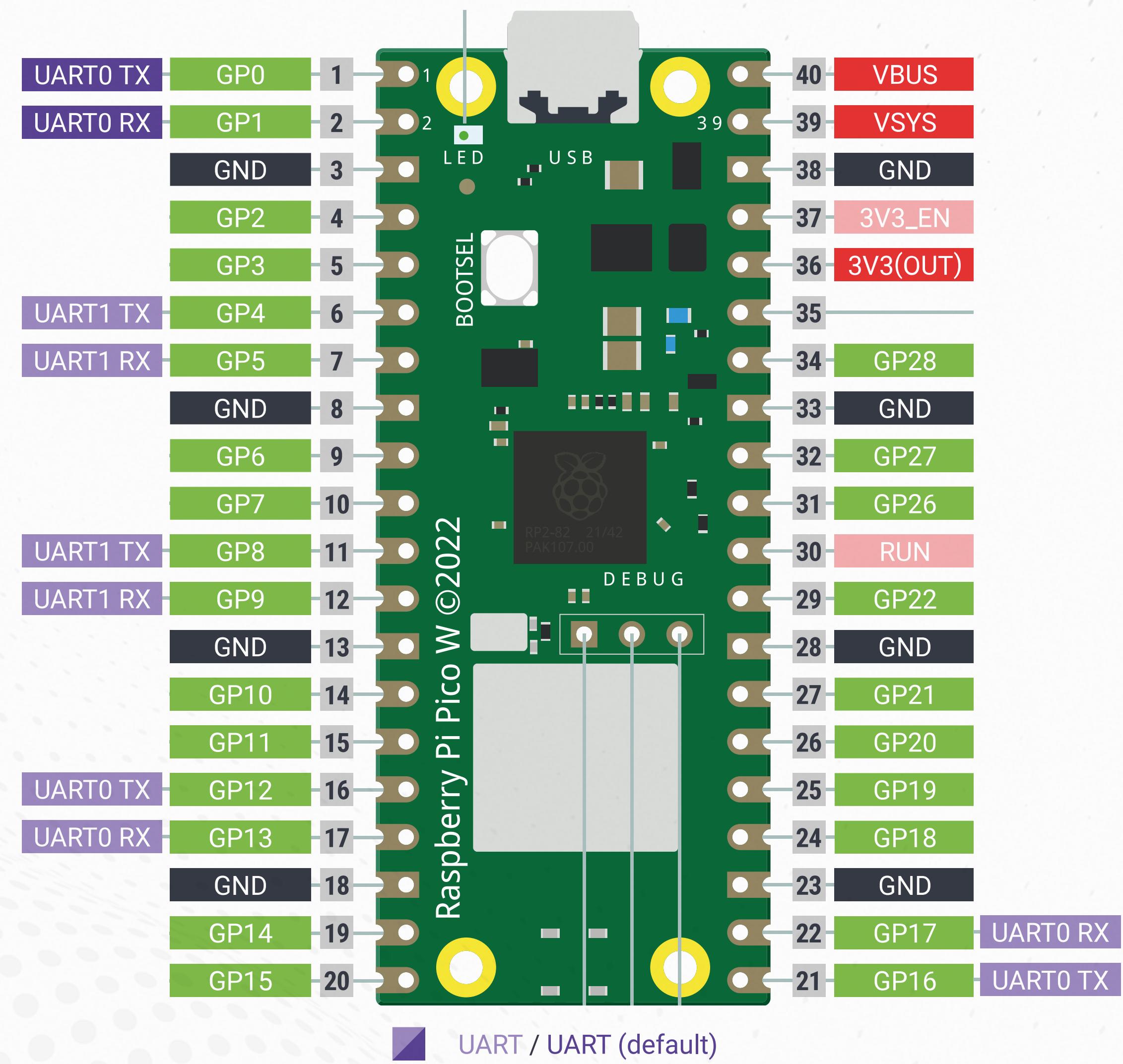


Fig. 4.
Fonte: imagem do autor

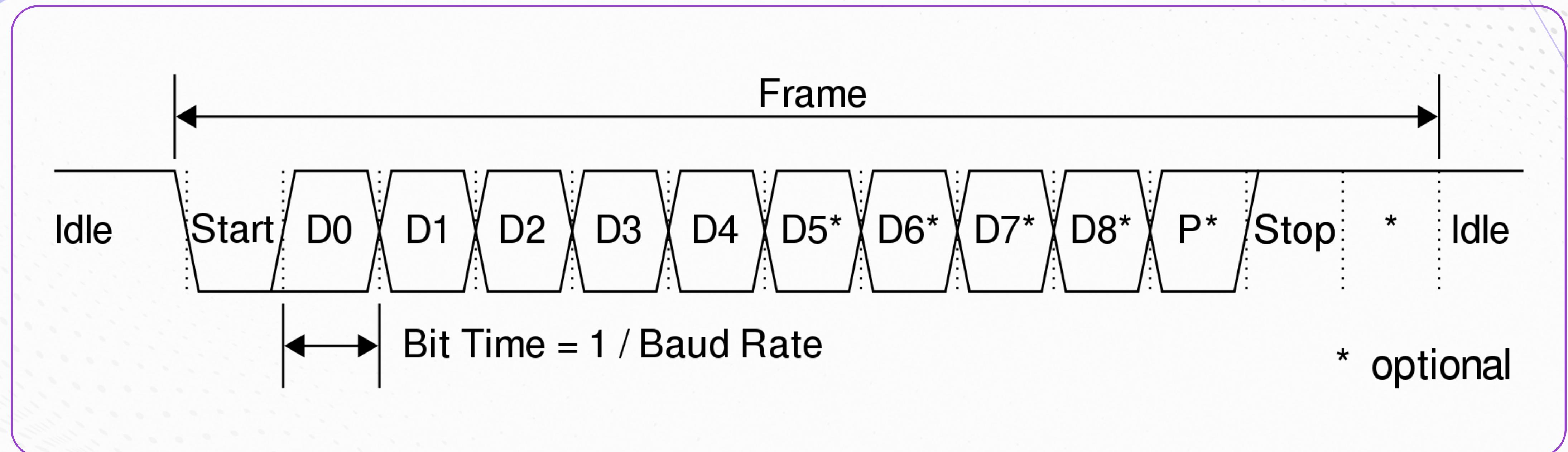


Fig. 5.
Fonte: <https://digilent.com/blog/uart-explained/>

SPI

SPI / SPI (default)

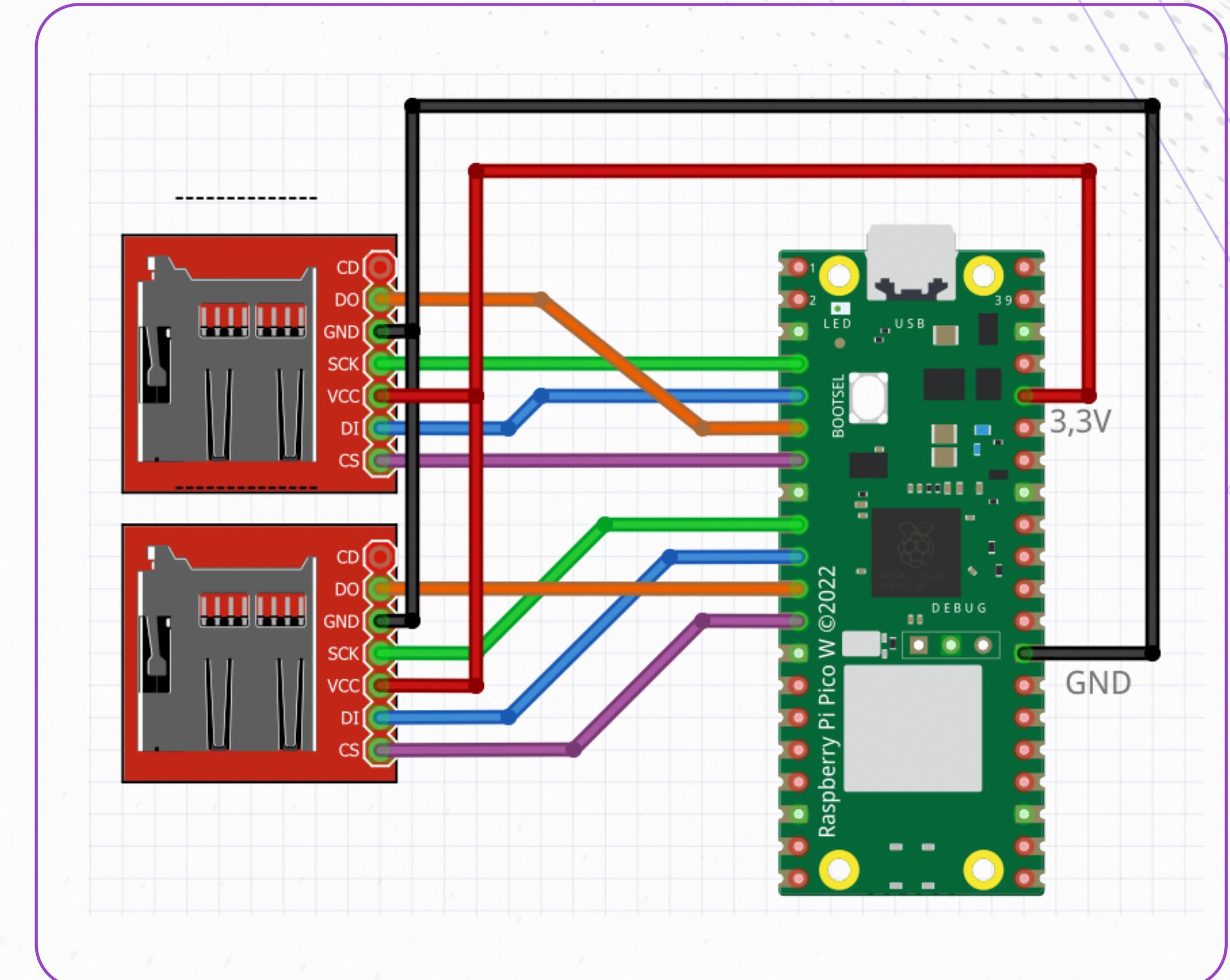
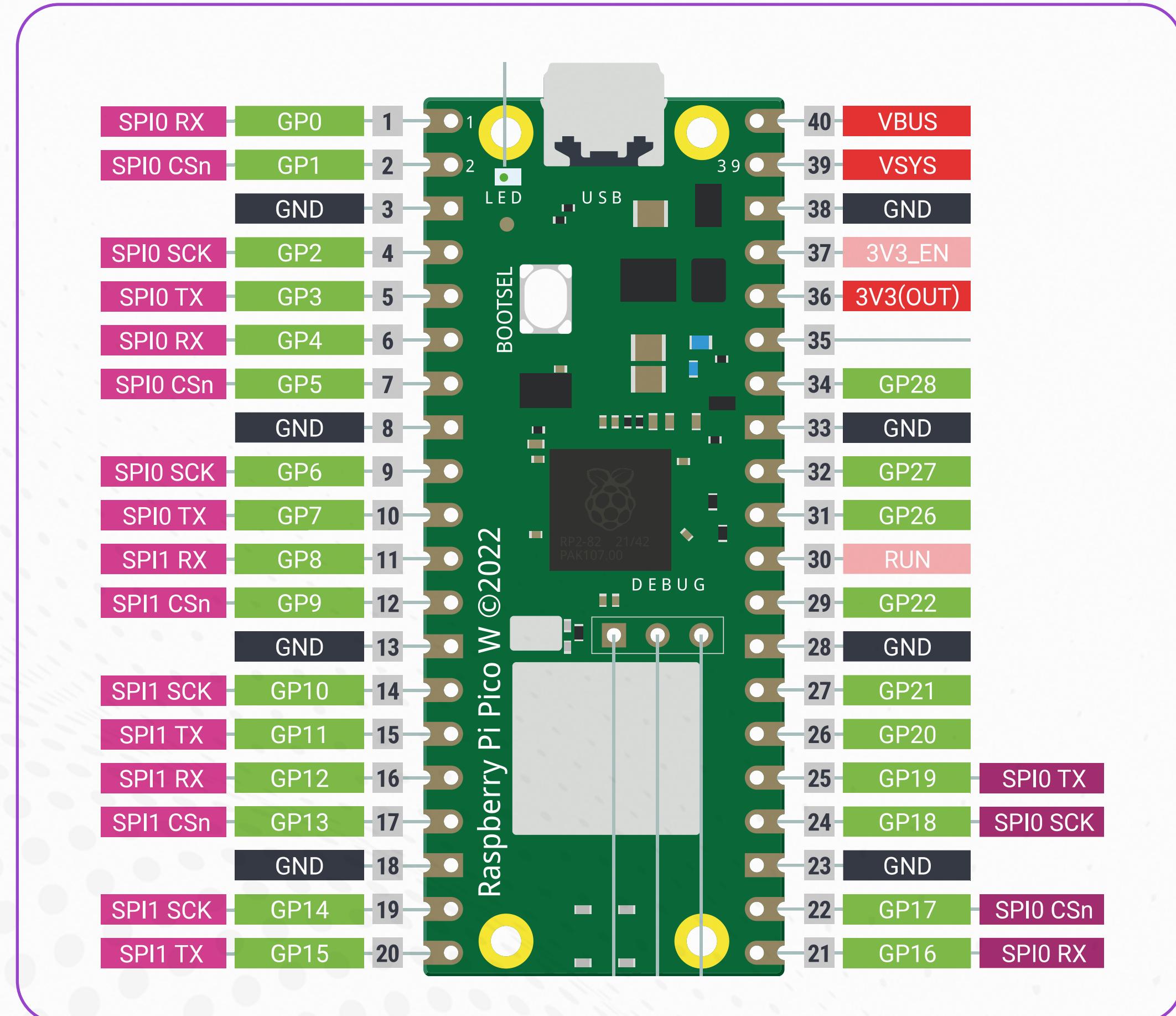


Fig. 6.
Fonte: imagem do autor

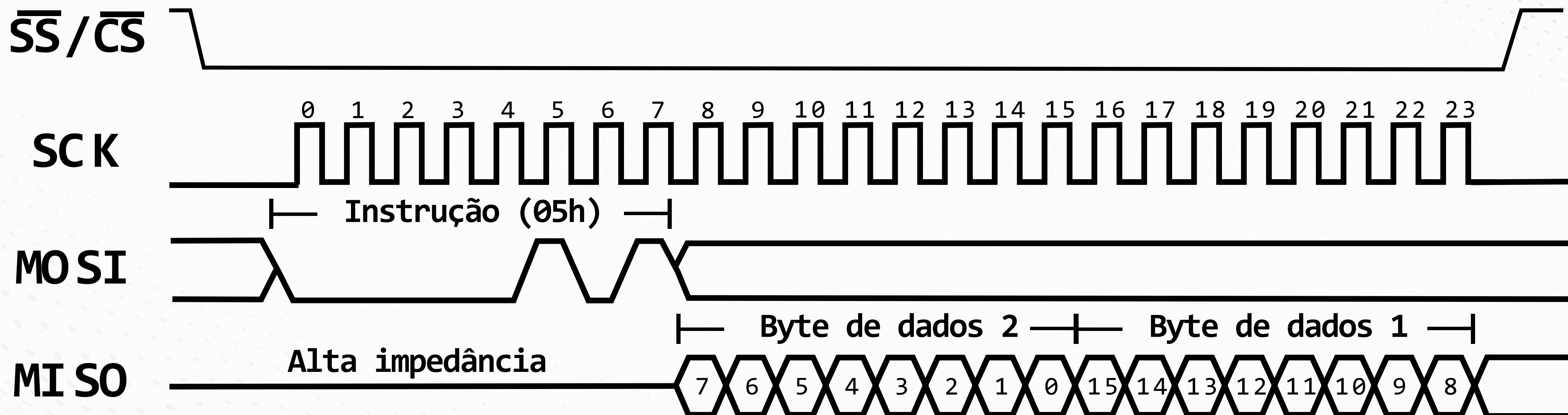


Fig. 7.
Fonte: imagem do autor

I²C

I²C / I²C (default)

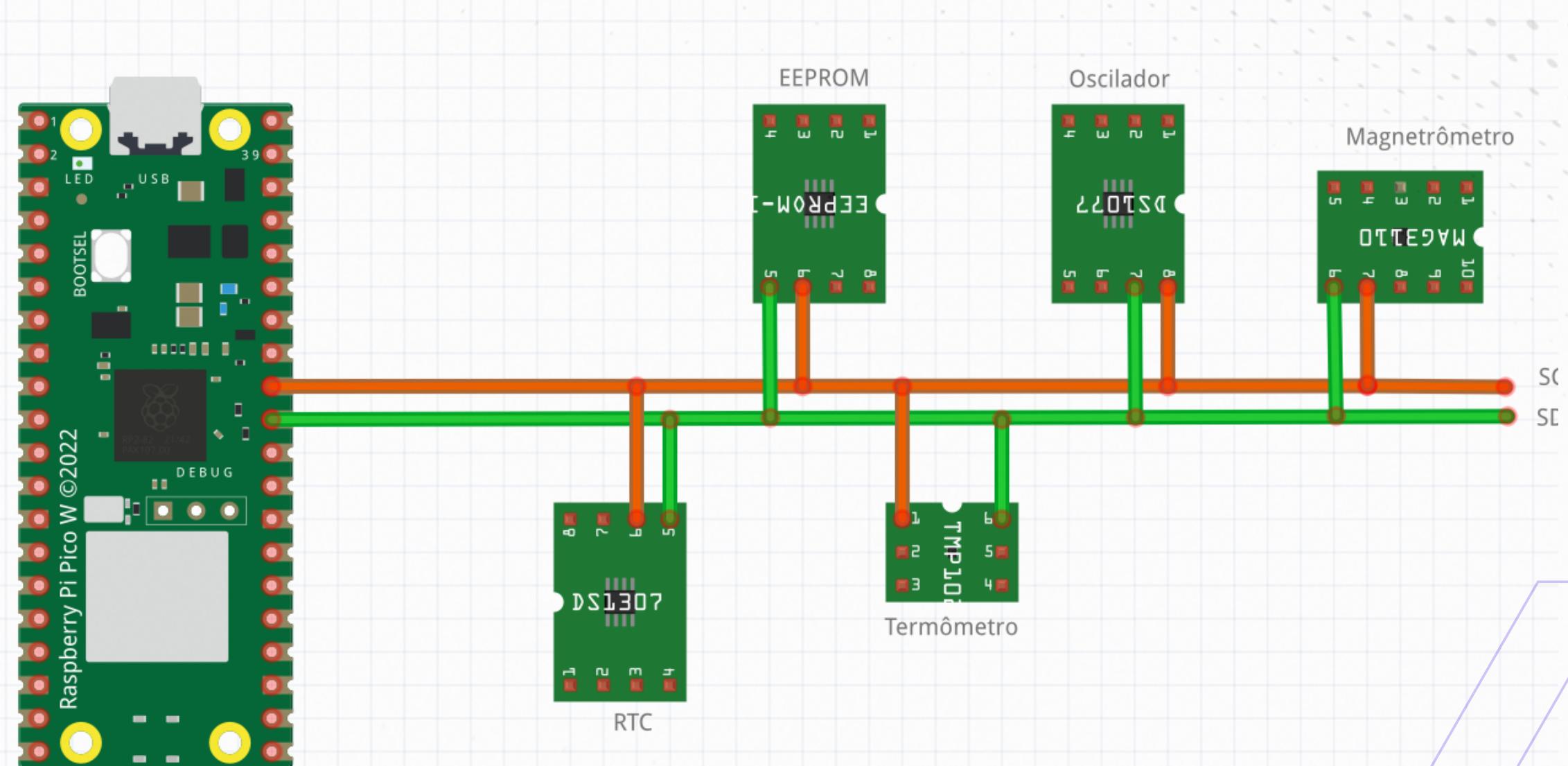
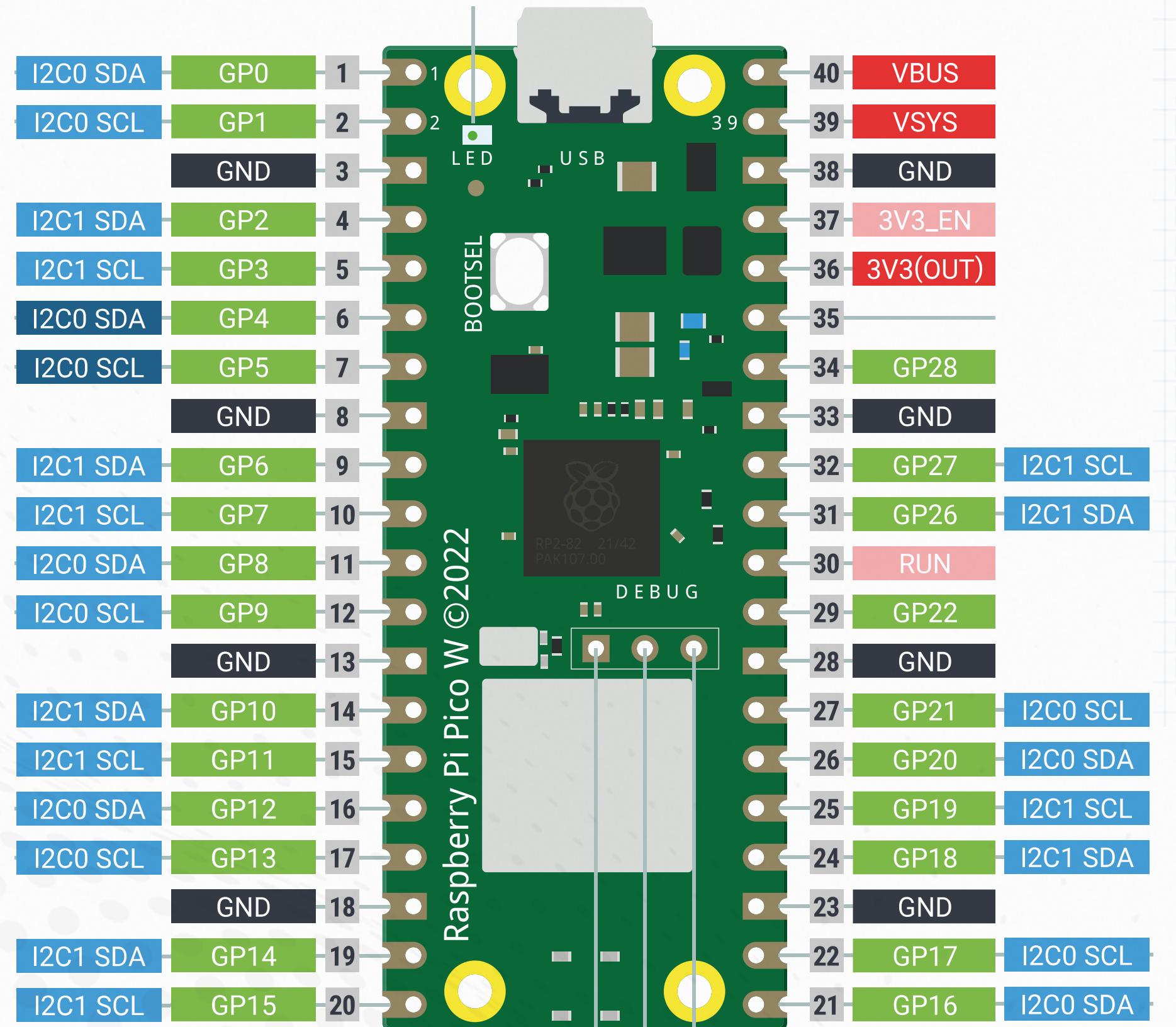


Fig. 8.
Fonte: <https://leanpub.com/rpitandt/read>

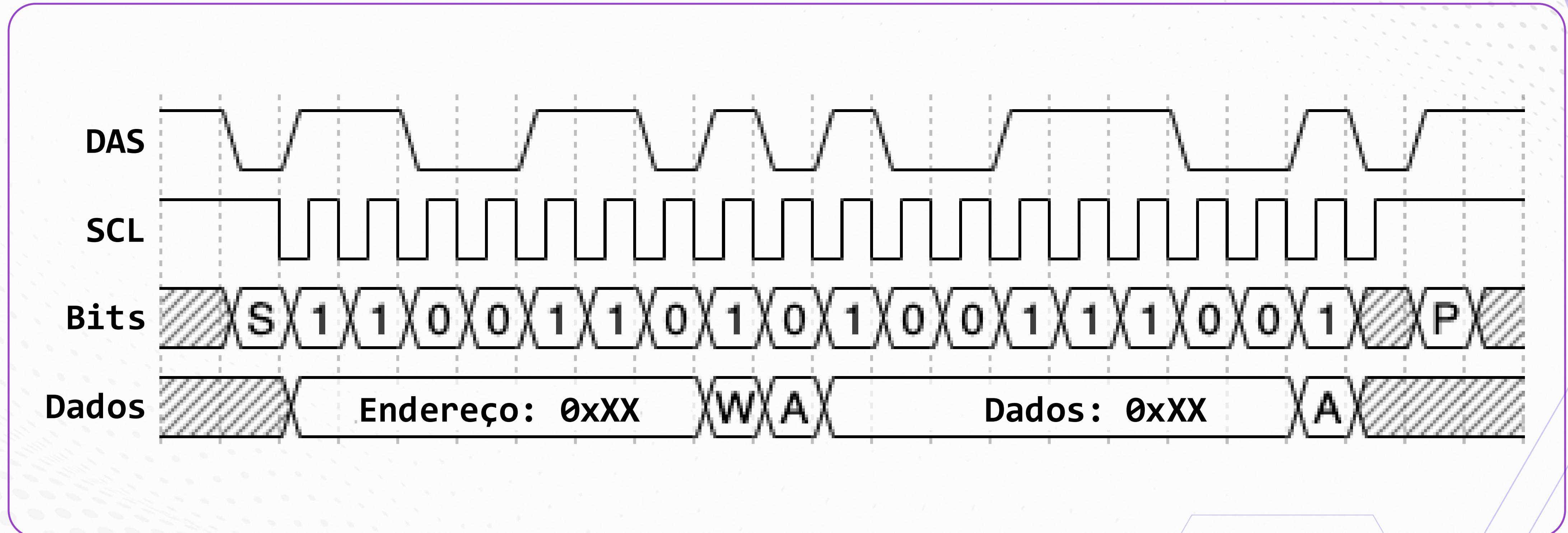


Fig. 9.
Fonte: Modificado de: <https://interrupt.memfault.com/blog/i2c-ina-nutshell>

Interface	UART	I2C	SPI
Complexidade	Simples	Fácil de comandar vários dispositivos	Complexidade aumenta com o número de dispositivos.
Velocidade	Slowest	Mais rápida que UART	Mais rápida das três
Bits por segundo (bps)	Até 115.2k	Até 400k	Até 10M
Número de dispositivos	Up to 2 devices	Up to 127	Alguns
Número de fios	3	3	4+1 por dispositivo
Nº de Masters e Slaves	Um para um	Múltiplos Masters e Slaves	1 Master, múltiplos Slaves

Fig. 10. Quadro resumo das interfaces
Fonte: imagem do autor

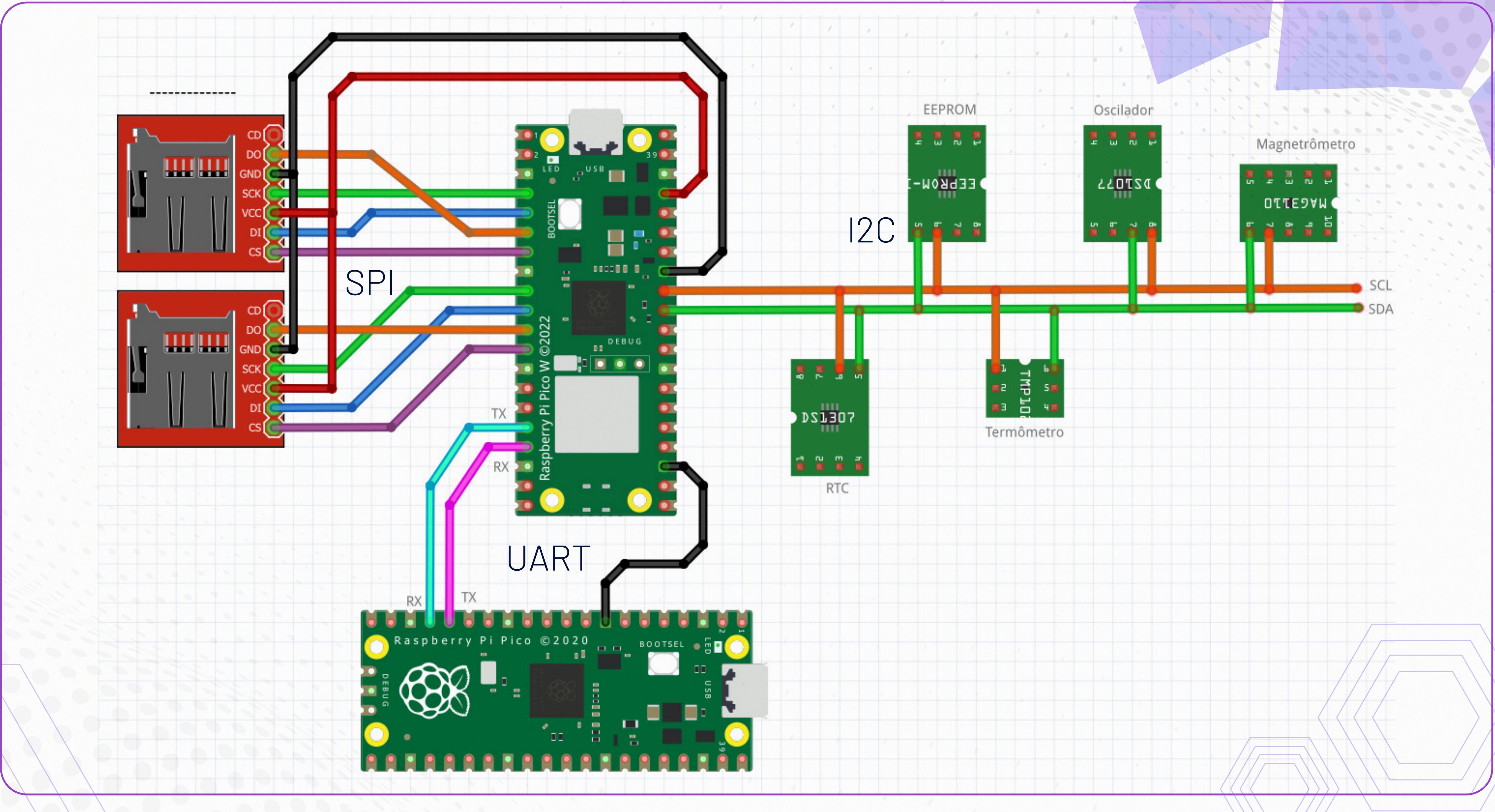


Fig. 11. Interfaces

```
#include "pico/stdlib.h"

int main() {
    stdio_init_all();
    // Setup do código

    while (true) {
        // Super Loop
        printf("Olá Mundo!\n");
        sleep_ms(1000);
    }
}
```

Fig. 12. Código em Linguagem C
Fonte: imagem do autor

A imagem exibe um trecho de código em linguagem C para ser utilizado com uma placa Raspberry Pi Pico. O código parece ser um exemplo simples de como configurar e usar a saída serial para imprimir "Olá Mundo!" repetidamente, com um intervalo de um segundo.
Aqui está uma descrição detalhada para pessoas com deficiência visual:
O código começa com a inclusão de uma biblioteca específica da Raspberry Pi Pico, indicada pela linha:
`#include "pico/stdlib.h"`
Na função `main()`, o código realiza a inicialização padrão usando a função `stdio_init_all()`, que configura o sistema de entrada e saída.
Em seguida, dentro de um loop infinito (`while (true)`), o código imprime "Olá Mundo!" na saída serial utilizando a função `printf()`. O texto está formatado com uma quebra de linha no final (`\n`).
Após cada impressão, o código pausa por 1000 milissegundos (1 segundo), usando a função `sleep_ms(1000)`.
A estrutura do código é simples, projetada para imprimir continuamente "Olá Mundo!" a cada segundo na saída serial.

UART

```
#include "pico/stdlib.h"
#include "hardware/uart.h"

int main() {

    // Configurações da UART
    uart_init uart0, 115200); // Inicializa a UART0 com baud rate de 115200
    gpio_set_function(0, GPIO_FUNC_UART); // Configura pino 0 como TX
    gpio_set_function(1, GPIO_FUNC_UART); // Configura pino 1 como RX

    while (true) {
        uart_puts(uart0, "Hello, UART!\n"); // Envia uma mensagem via UART
        sleep_ms(1000); // Pausa de 1 segundo
    }
}
```

Fig. 13. Código em Linguagem C – UART
Fonte: imagem do autor

A imagem apresenta um código de computador escrito na linguagem de programação C. Esse código é utilizado para configurar e utilizar uma interface de comunicação serial chamada UART.

Elementos principais do código:
Inclusão de bibliotecas: As linhas iniciais `#include "pico/stdlib.h"` e `#include "hardware/uart.h"` incluem bibliotecas necessárias para o funcionamento do código. Essas bibliotecas fornecem funções e estruturas de dados para trabalhar com o hardware do microcontrolador e a comunicação serial.

Função principal `main()`: Esta é a função principal do programa, onde o código é executado.

Configuração da UART: Dentro da função `main()`, há um bloco de código que configura a UART. Isso inclui:

Inicialização da UART: A função `uart_init(uart0, 115200)` inicializa a UART com uma taxa de transmissão de 115200 bits por segundo.

Configuração dos pinos: As funções `gpio_set_function(0, GPIO_FUNC_UART)` e `gpio_set_function(1, GPIO_FUNC_UART)` configuram os pinos 0 e 1 do microcontrolador como pinos de transmissão (TX) e recepção (RX) da UART, respectivamente.

Loop infinito: O `while(true){ ... }` cria um loop infinito, fazendo com que o código dentro dele seja executado repetidamente.

Envio de dados: Dentro do loop, a função `uart_puts(uart0, "Hello, UART!\n")` envia a mensagem "Hello, UART!" pela UART.

Pausa: A função `sleep_ms(1000)` faz com que o programa pausar por 1000 milissegundos (1 segundo) antes de enviar a próxima mensagem.

SPI

```
#include "pico/stdlib.h"
#include "hardware/spi.h"

int main() {

    // Configurações do SPI
    spi_init(spi0, 500000); // Inicializa SPI0 com velocidade de 500kHz
    gpio_set_function(2, GPIO_FUNC_SPI); // SCLK
    gpio_set_function(3, GPIO_FUNC_SPI); // MOSI
    gpio_set_function(4, GPIO_FUNC_SPI); // MISO
    gpio_set_function(5, GPIO_FUNC_SPI); // Chip Select (CS)

    uint8_t data_out = 0x55; // Dados para enviar
    uint8_t data_in = 0;     // Dados recebidos

    while (true) {
        gpio_put(5, 0); // Seleciona o dispositivo (CS baixo)
        spi_write_read_blocking(spi0, &data_out, &data_in, 1); // Envia e recebe 1 byte
        gpio_put(5, 1); // Libera o dispositivo (CS alto)
        sleep_ms(1000); // Pausa de 1 segundo
    }
}
```

Fig. 14. Código em Linguagem C – SPI
Fonte: imagem do autor

O código em linguagem C apresentado é para interface serial SPI e realiza os seguintes passos principais:
Inclusão de Bibliotecas: As bibliotecas pico/stdlib.h e hardware/spi.h são incluídas para acesso a funções de hardware e SPI.
Função principal (main): Define o ponto de execução do código.
Configuração do SPI: Inicializa o SPI a 500kHz e configura os pinos 2, 3, 4 e 5 como SCLK, MOSI, MISO e CS, respectivamente.
Declaração de Variáveis: data_out e data_in são declaradas para armazenar dados de envio e recebimento.
Loop Infinito: O código entra em loop contínuo.
Transmissão e Recepção: spi_write_read_blocking envia e recebe dados entre data_out e data_in.
Controle do Chip: Usa gpio_put para ativar e desativar o dispositivo SPI.
Pausa: sleep_ms(1000) insere uma pausa de 1 segundo entre transmissões.
O programa configura e executa comunicação SPI de forma contínua, enviando e recebendo dados em intervalos regulares.

I2C

```
#include "pico/stdlib.h"
#include "hardware/i2c.h"

#define I2C_ADDR 0x48 // Endereço do dispositivo I2C

int main() {

    // Configurações do I2C
    i2c_init(i2c0, 100 * 1000); // Inicializa I2C0 a 100kHz
    gpio_set_function(4, GPIO_FUNC_I2C); // SDA
    gpio_set_function(5, GPIO_FUNC_I2C); // SCL
    gpio_pull_up(4); // Habilita pull-up para SDA
    gpio_pull_up(5); // Habilita pull-up para SCL

    uint8_t buffer[2];

    while (true) {
        i2c_read_blocking(i2c0, I2C_ADDR, buffer, 2, false); // Lê 2 bytes do sensor
        uint16_t valor = (buffer[0] << 8) | buffer[1]; // Combina os dois bytes
        printf("Valor lido: %d\n", valor); // Exibe o valor lido
        sleep_ms(1000); // Pausa de 1 segundo
    }
}
```

Fig. 15. Código em Linguagem C - I2C
Fonte: imagem do autor

O código apresentado estabelece uma comunicação serial entre um microcontrolador Raspberry Pi Pico e um dispositivo externo utilizando o protocolo I2C.
As principais etapas do código são:
1 - Inclusão de Bibliotecas: As bibliotecas necessárias para a comunicação I2C são incluídas no código.
2 - Definição do Endereço I2C: O endereço específico do dispositivo a ser comunicado é definido.
3 - Configuração da Interface I2C: Os pinos do microcontrolador são configurados para a comunicação I2C e a interface é inicializada com uma velocidade determinada.
4 - Leitura de Dados: O código lê dados do dispositivo I2C, armazenando-os em um buffer.
5 - Processamento dos Dados: Os dados lidos são combinados e formatados para serem exibidos.
Loop Infinito: O processo de leitura e exibição dos dados é repetido continuamente.

Conclusão

