

🎓 Revisão módulo capacitação

Aula síncrona (15/04/2025)
Prof. Wilton Lacerda Silva

Executores:



Coordenação:



Iniciativa:



Sumário

- Objetivos
- Configuração do VS Code para o RP2040
- Kit de desenvolvimento BitDogLab
- Interrupções
- Interfaces de Comunicação serial do RP2040
- Conversão Analógico Digital no RP2040
- Tarefa
- Conclusão

Objetivos

- Fazer uma revisão de alguns temas abordados nas aulas do curso de capacitação.
- Definir um “padrão para entrega dos projetos”
- Realizar uma tarefa para relembrar conceitos importantes.



Placa de desenvolvimento Raspberry Pi Pico

A **Raspberry Pi Pico** é uma placa de desenvolvimento de microcontrolador criada pela **Raspberry Pi Foundation**.

Projetado especialmente para tarefas embarcadas e IoT (Internet das Coisas).

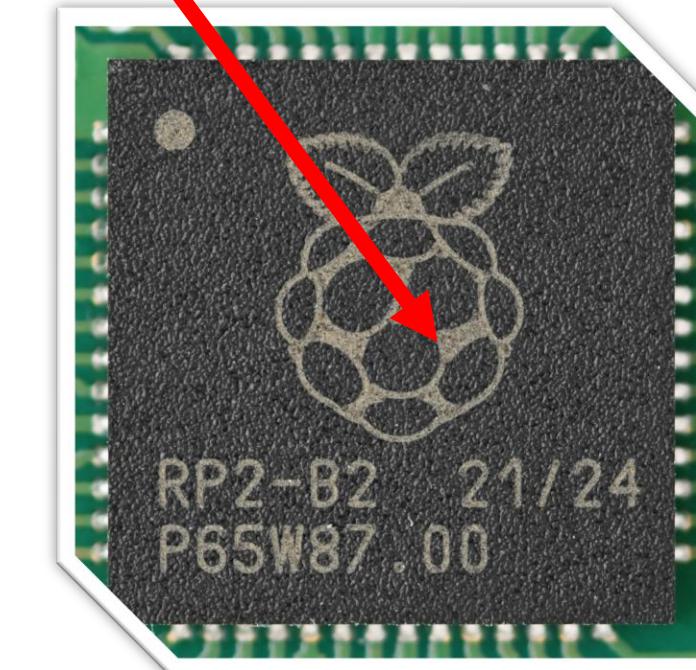
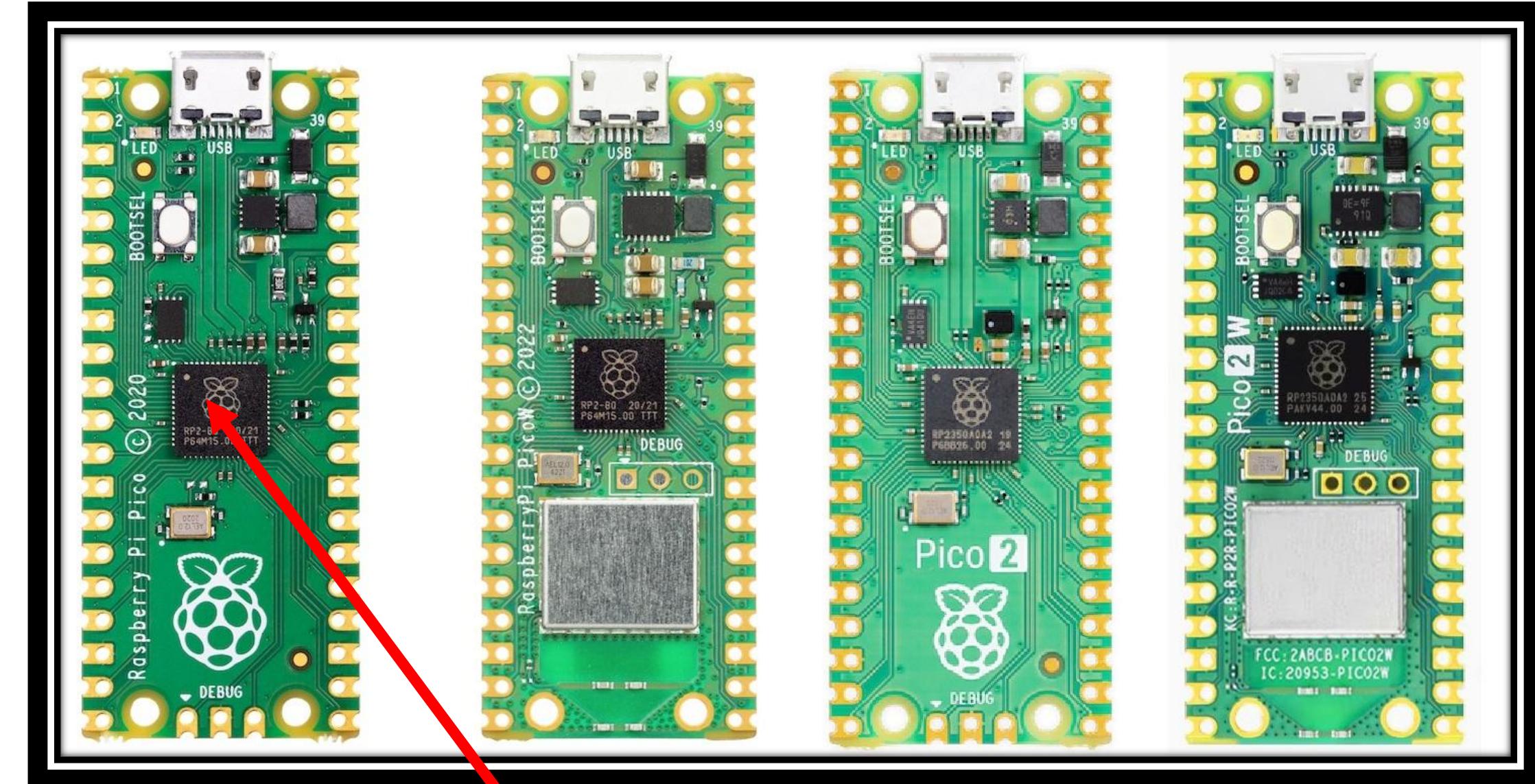
Microcontroladores

RP2040

- Raspberry Pi Pico
- Raspberry Pi Pico W

RP2350

- Raspberry Pi Pico 2
- Raspberry Pi Pico 2 W

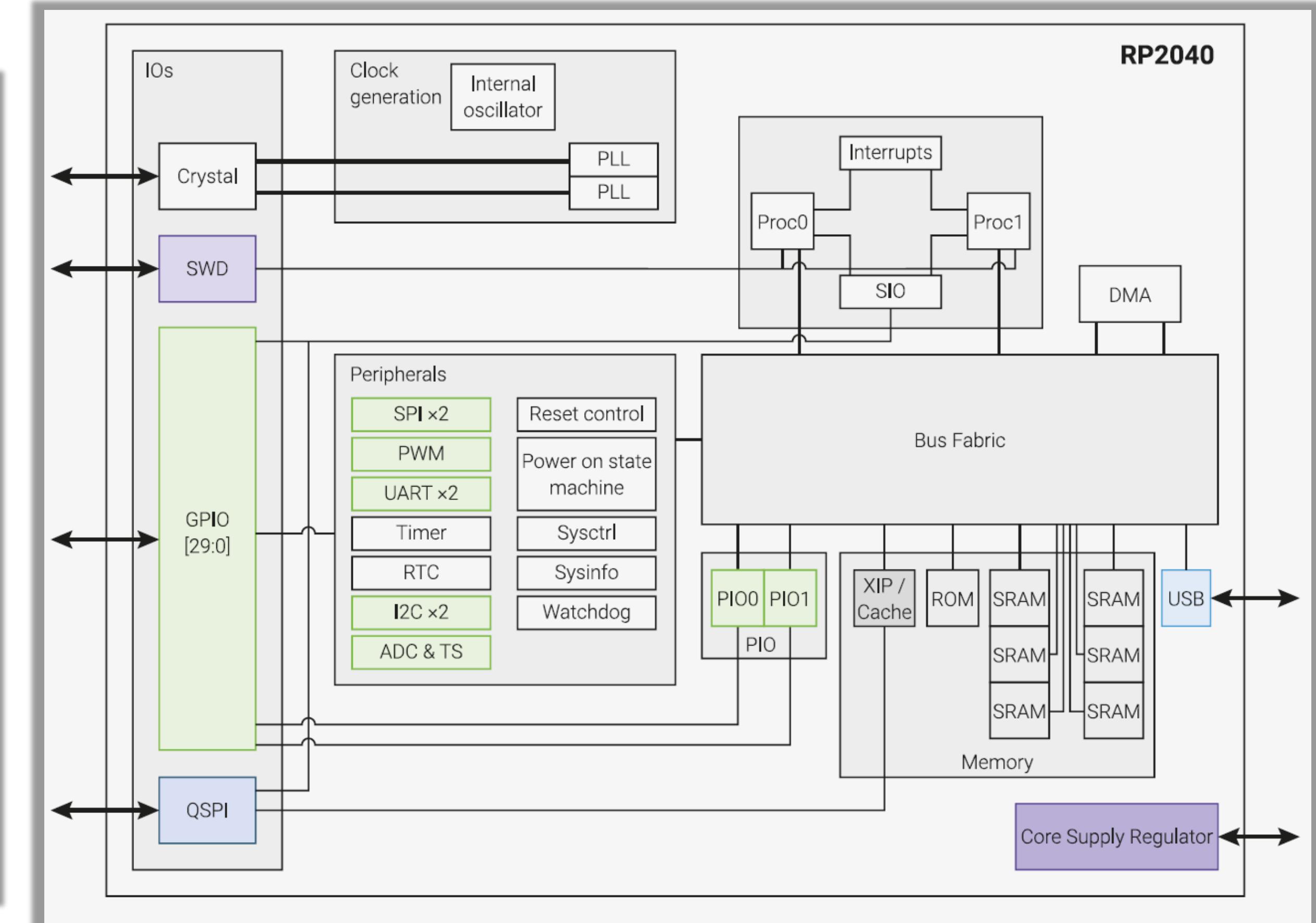




Microcontrolador RP2040

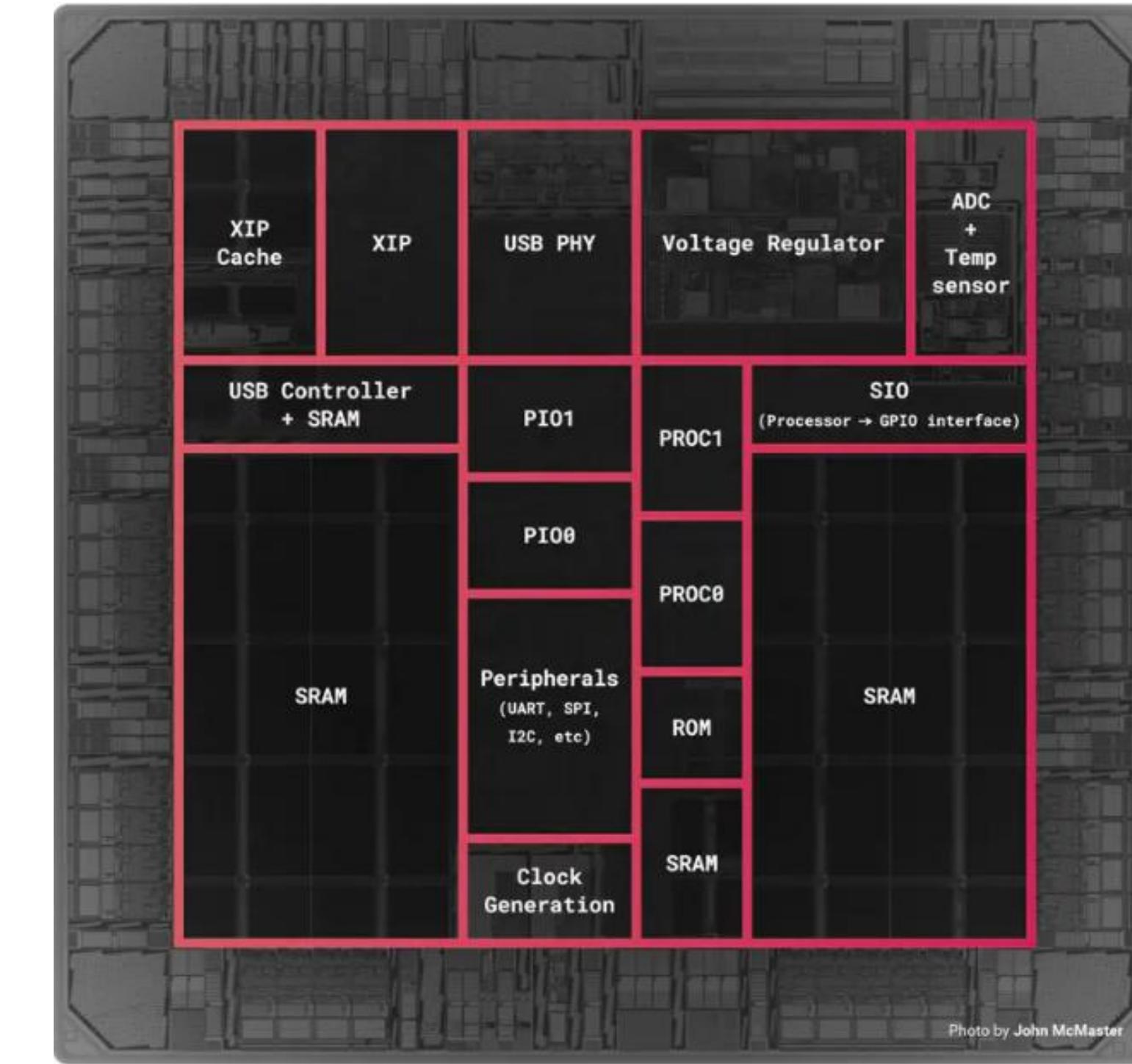
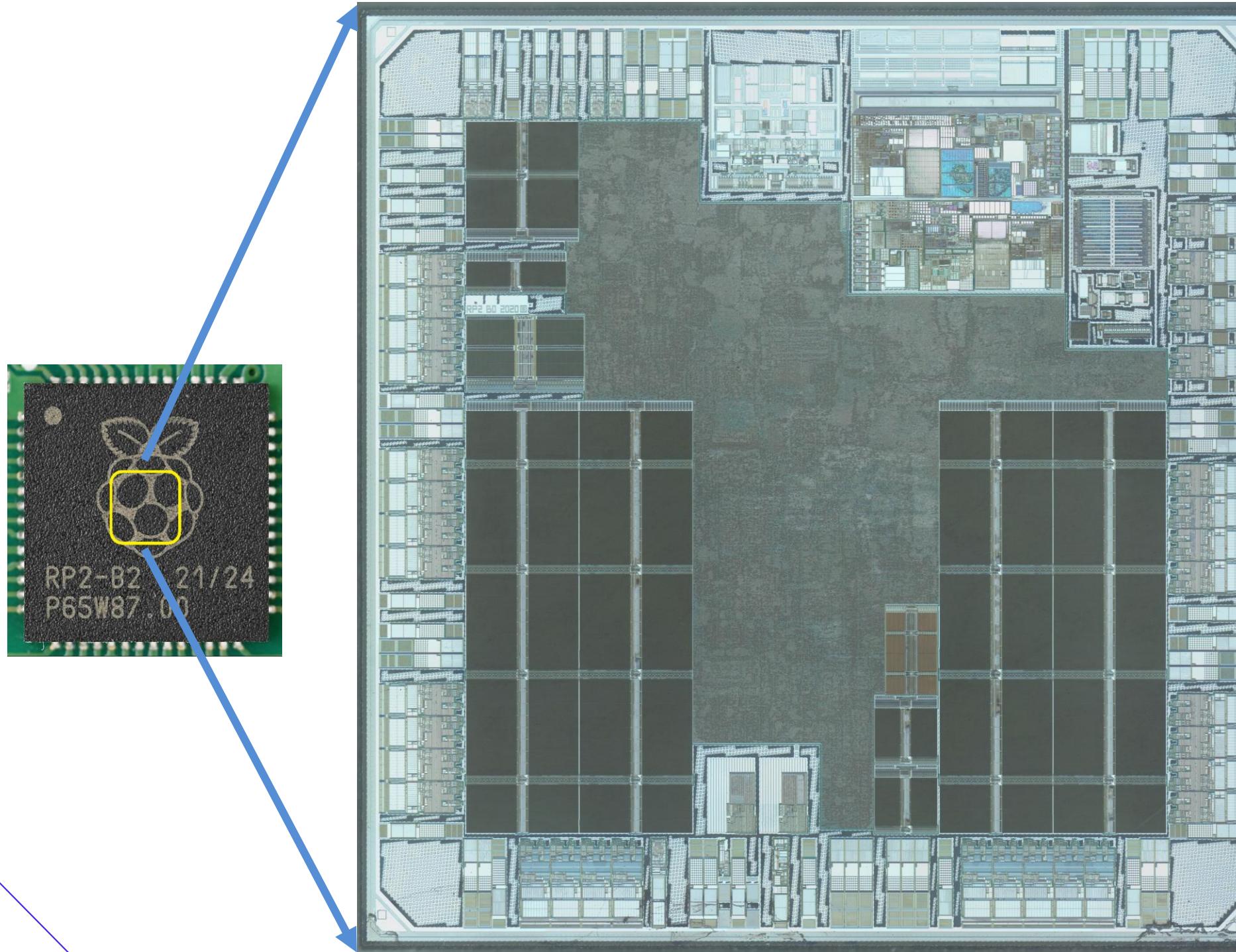
Características principais:

- Dual Cortex M0+ processors, up to 133MHz
- 264kB of embedded SRAM in 6 banks (4x64 + 2x4)
- Support for up to 16MB of off-chip Flash memory via dedicated QSPI bus
- 30 GPIO pins, 4 of which can be used as analogue inputs
- 6 dedicated I/O for SPI flash (supporting XIP)
- Dedicated hardware for commonly used peripherals
- Programmable I/O for extended peripheral support
- 4 channel ADC with internal temperature sensor, 500ksps, 12-bit conversion
- Peripherals
 - 2 UARTs
 - 2 SPI controllers
 - 2 I2C controllers
 - 16 PWM channels
 - USB 1.1 controller and PHY, with host and device support
 - 8 PIO state machines



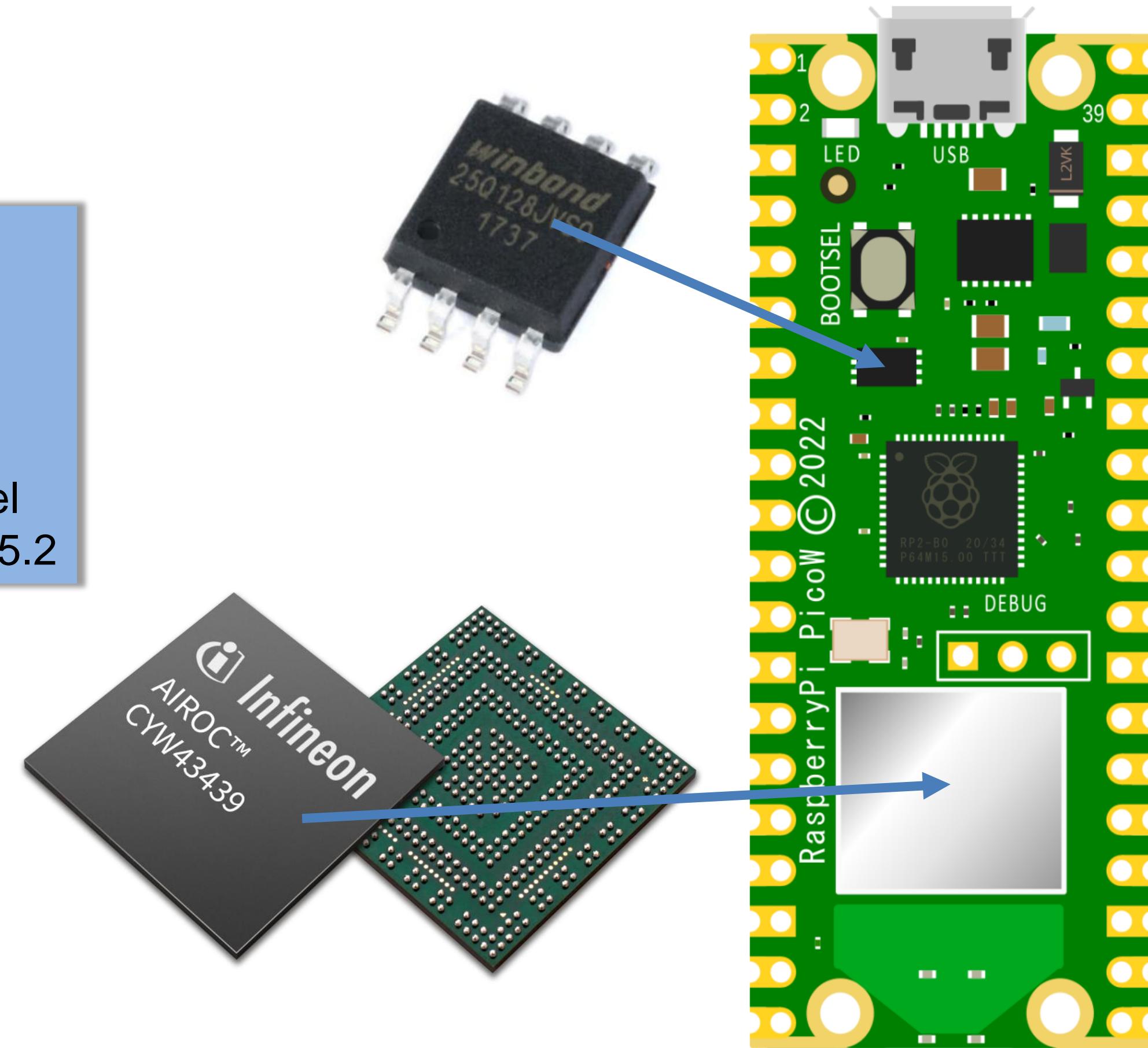
Die do RP2040

- O die é o pedaço de silício semicondutor onde os circuitos eletrônicos são fabricados.
- Construção à 40nm
- Estimativa de 264.000 transistores



Placa de desenvolvimento Raspberry Pi Pico W

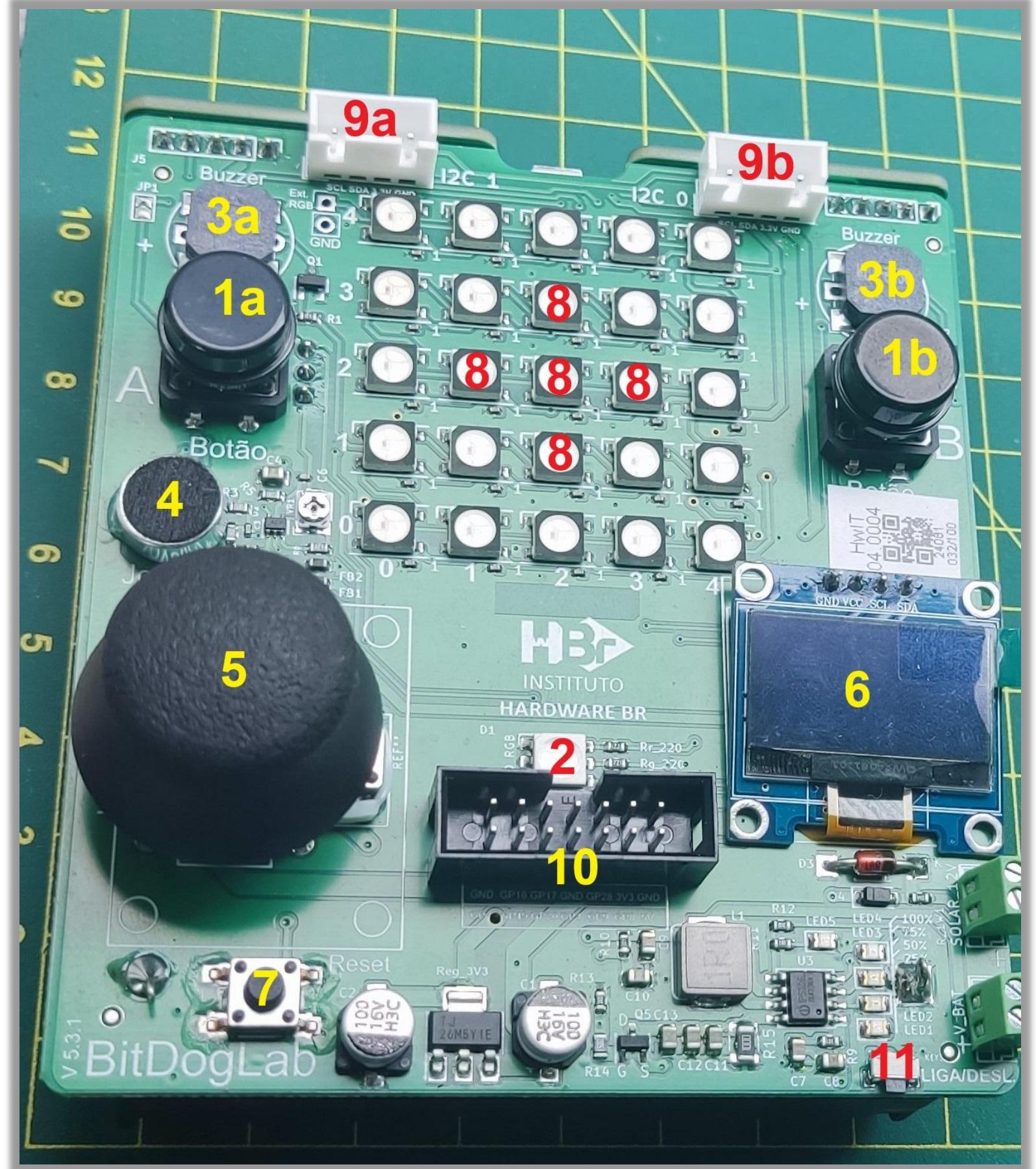
- Memória Flash
Original com 2MB
Máximo de 16MB
- Chip Infineon CYW43439 responsável
por 802.11b/g/n Wi-Fi 4 e Bluetooth 5.2



Kit BitDogLab

A BitDogLab é uma ferramenta educacional voltada para o ensino de sistemas embarcados, desenvolvida no contexto do projeto Escola 4.0 da Universidade de Campinas. Equipada com microcontrolador Raspberry Pi Pico W.

- 1a e 1b - Push-Buttons**
- 2 - Led RGB**
- 3a e 3b – Buzzer**
- 4 – Microfone**
- 5 – Joystick**
- 6 – Display (128 x 64 = 8192 pixels)**
- 7 – Chave de Reset**
- 8 – Matriz Led (5 x 5)**
- 9a e 9b – Conectores de expansão**
- 10 – Conector de expansão**
- 11 – Chave liga e desliga**



Configuração do VS Code para Raspberry Pi pico "RP2040"

Aqui é considerado que nada
está instalado no PC.



<https://code.visualstudio.com>

Instalação
do VS Code

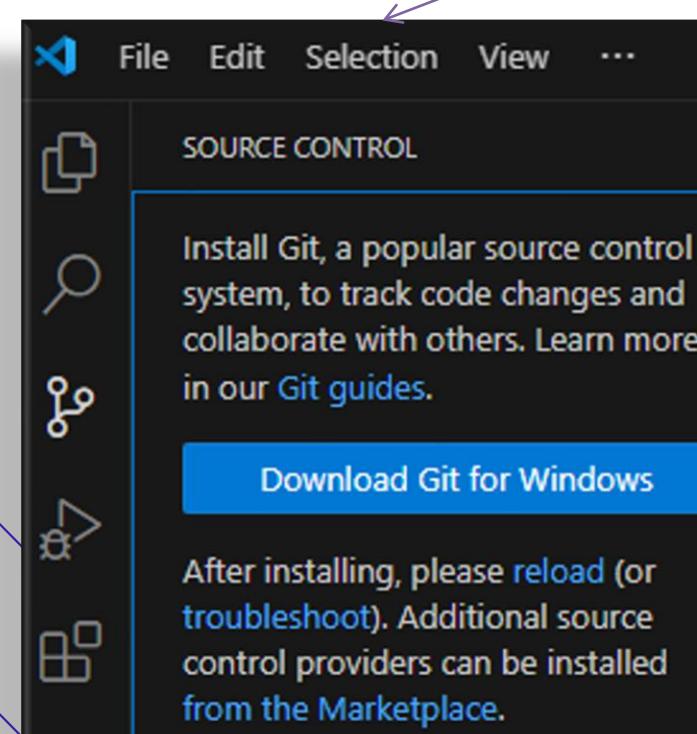
Instalação
do Git

Instalar
extensão:
Raspberry Pi
Pico

Instalação do SDK.
Criar um novo Projeto
(New C/C++ Project)

Compilar e
Carregar o
UF2 no RP2

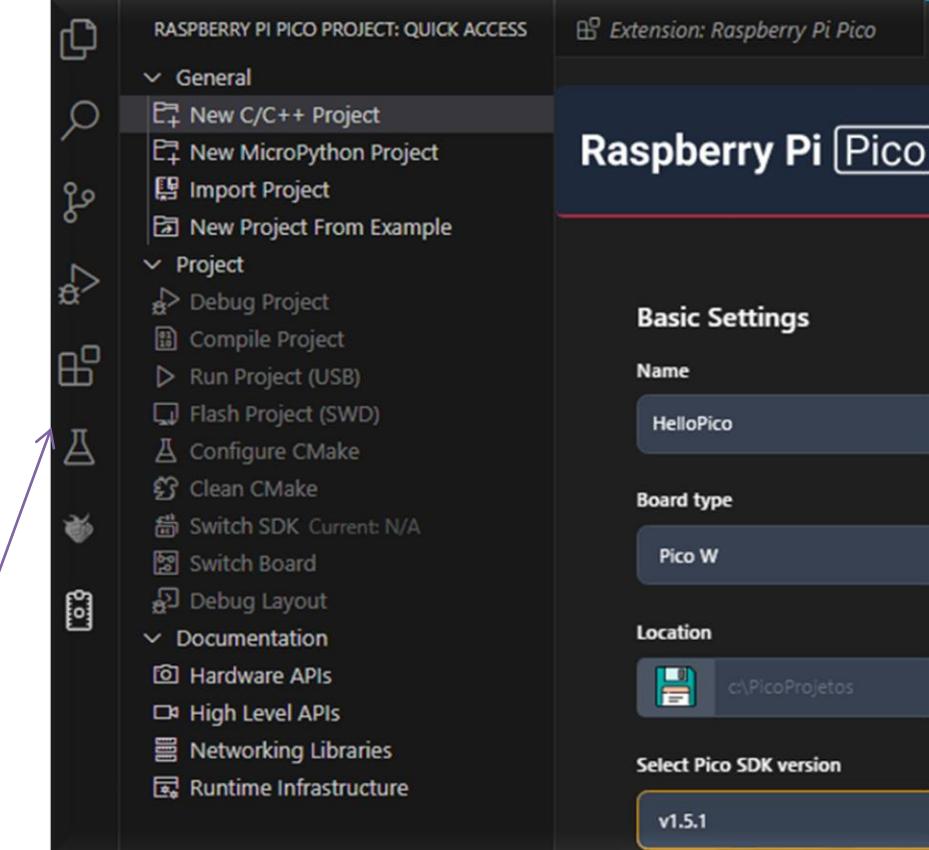
RUN
Requer o USB
driver ZADIG.



<https://github.com/raspberrypi/pico-sdk>

Serão instaladas 16 extensões, tais como:

Raspberry Pi Pico; Serial Monitor; RTOS Views;
Python; Peripheral Viewer; MicroPico; MemoryView;
debug-tracker-vscode; Cortex-Debug; Cmake Tools;
Cmake; C/C++ Themes; C/C++ Extension Pack; C/C++



9

VSCode: Codificando no SDK

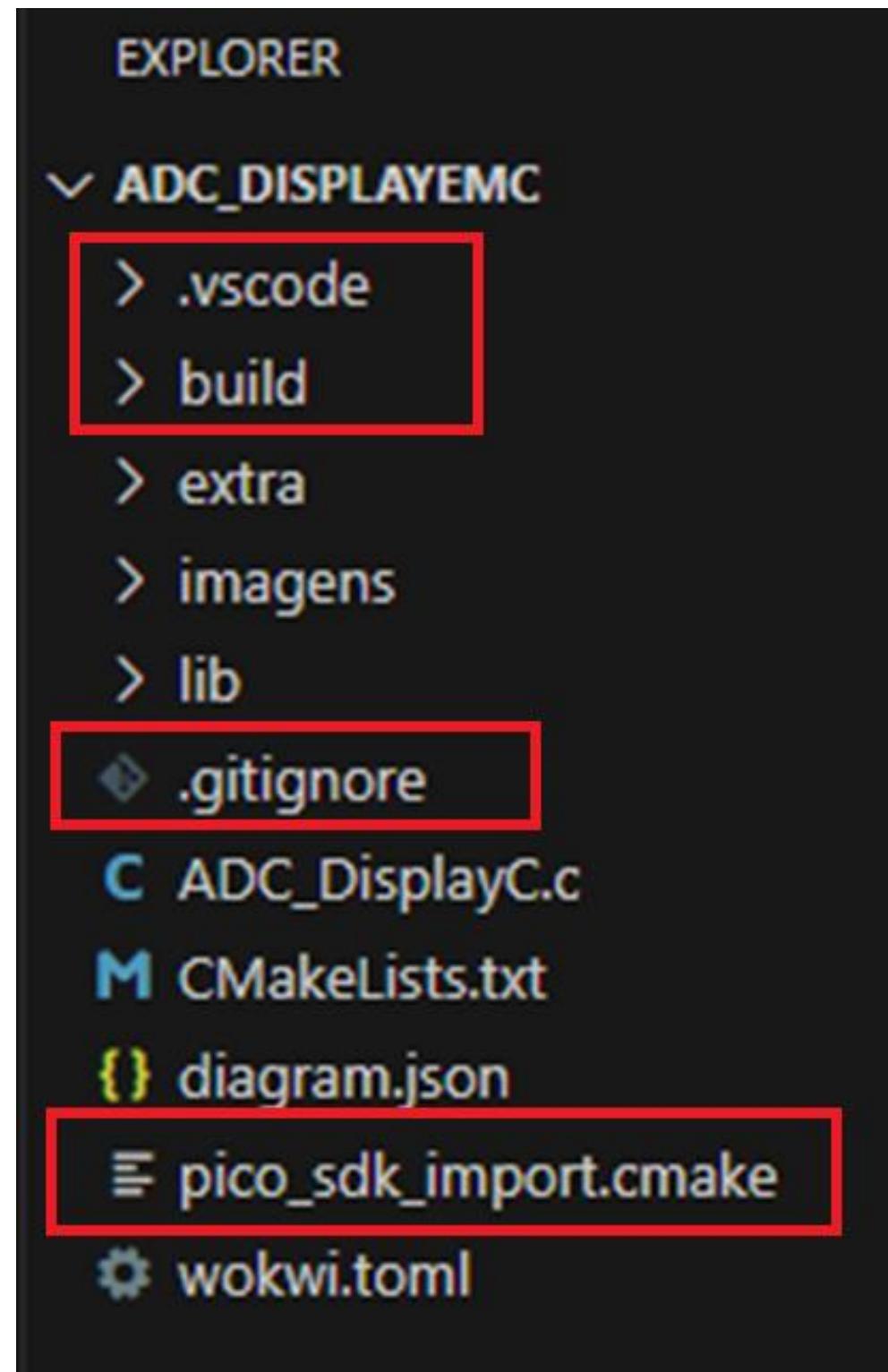
Trechos de código da biblioteca Pico SDK, utilizada para o Raspberry Pi Pico

```
#include <stdio.h> // Inclui a biblioteca padrão de entrada e saída
stdio_init_all(); // Inicializa a biblioteca de entrada e saída padrão para uso de print
printf("Olá mundo! \n"); // Imprime na tela

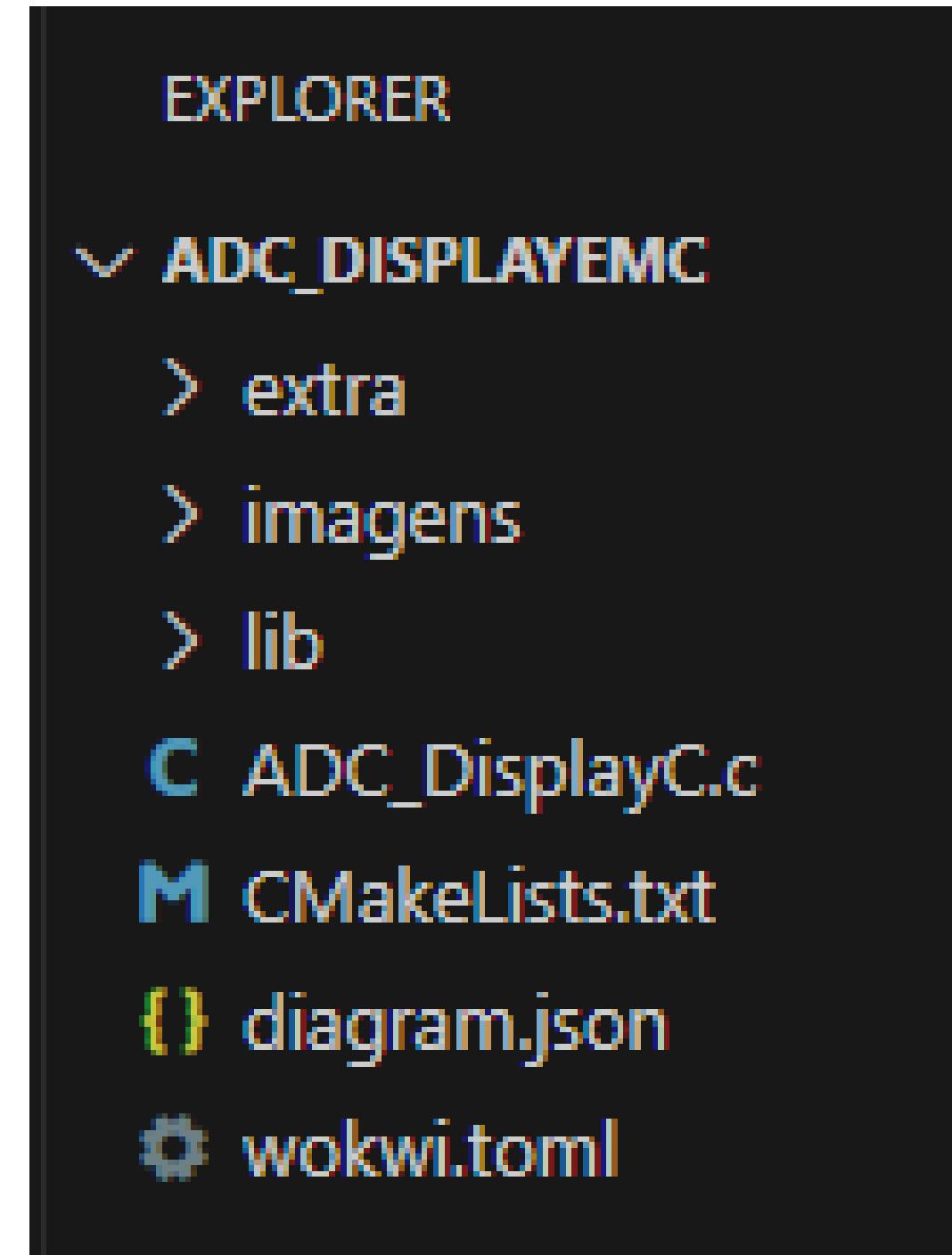
#include "pico/stdc.h" // Inclui a biblioteca de funções padrão
gpio_init(led_pin); // Inicializa o pino do led_pin
gpio_set_dir(led_pin, GPIO_OUT); // Define o pino do led_pin como saída
gpio_put(led_pin, true); // Liga o led
gpio_put(led_pin, 1); // Liga o led
gpio_put(led_pin, false); // Desliga o led
gpio_put(led_pin, 0); // Desliga o led
sleep_ms(duracao); // Aguarda um tempo "duracao" em milissegundos
```

VSCode: Estrutura de arquivos

Compilado



Compartilhado



Criação de um “padrão” para compartilhamento de projeto.

Aqui o intuito é facilitar e racionalizar os esforços para manutenção, modificação e correção de projetos.

Pode-se fazer isto no .gitignore quando for postar no github

VSCode: cmakelists.txt

Compilado

```
# == DO NOT EDIT THE FOLLOWING LINES for the Raspberry Pi Pico VS Code Extension
if(WIN32)
    set(USERHOME $ENV{USERPROFILE})
else()
    set(USERHOME $ENV{HOME})
endif()
set(sdkVersion 2.1.1)
set(toolchainVersion 14_2_Rel1)
set(picotoolVersion 2.1.1)
set(picoVscode ${USERHOME}/.pico-sdk/cmake/pico-vscode.cmake)
if (EXISTS ${picoVscode})
    include(${picoVscode})
endif()
# =====

cmake_minimum_required(VERSION 3.13)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
set(PICO_BOARD pico_w CACHE STRING "Board type")
include(pico_sdk_import.cmake)
```

Compartilhado

```
cmake_minimum_required(VERSION 3.13)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
set(PICO_BOARD pico_w CACHE STRING "Board type")
include(pico_sdk_import.cmake)

# Define o nome do projeto como Teste_ADC_Display,
project(Teste_ADC_Display C CXX ASM)
pico_sdk_init()

add_executable(${PROJECT_NAME}
    ADC_DisplayC.c # Código principal em C
    lib/ssd1306.c # Biblioteca para o display OLED
)

target_link_libraries(${PROJECT_NAME}
    pico_stdl�
    hardware_i2c
    hardware_adc
)

pico_enable_stdio_usb(${PROJECT_NAME} 1)
pico_enable_stdio_uart(${PROJECT_NAME} 0)

pico_add_extra_outputs(${PROJECT_NAME})
```

Polling e Interrupções

Comparação

- Polling: O MCU verifica repetidamente se um evento ocorreu, consumindo recursos do processador mesmo quando nada acontece.
- Interrupções: O MCU espera passivamente até receber um sinal de interrupção, o que é mais eficiente, pois libera o processador para executar outras tarefas enquanto aguarda eventos.

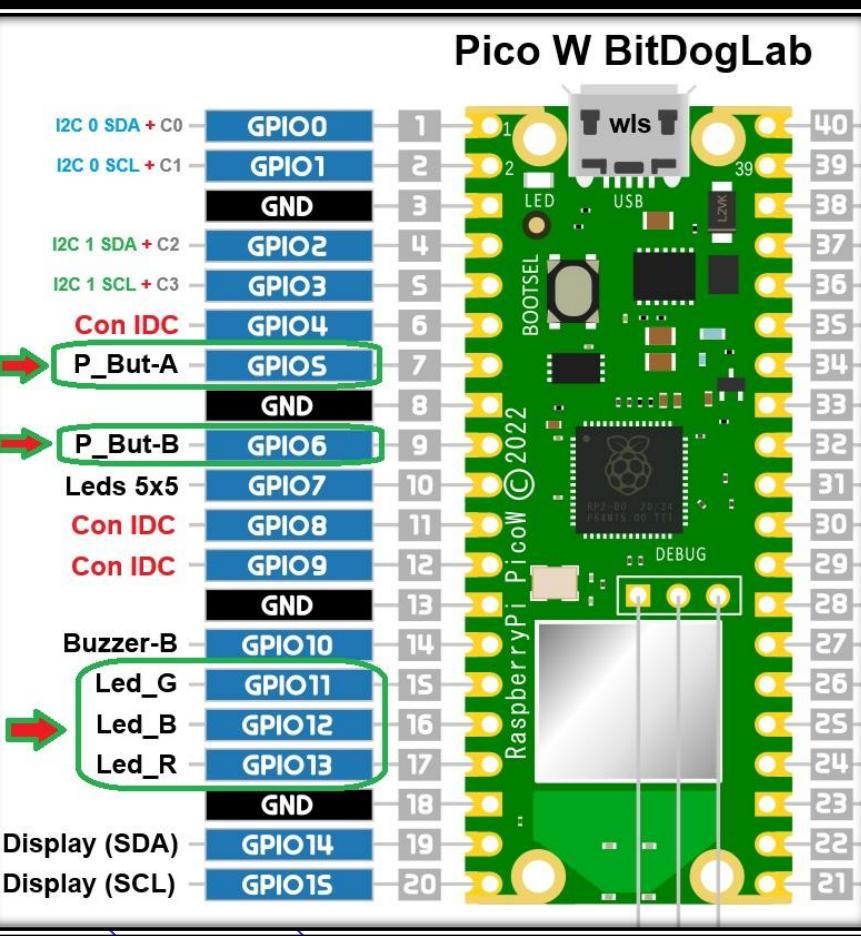
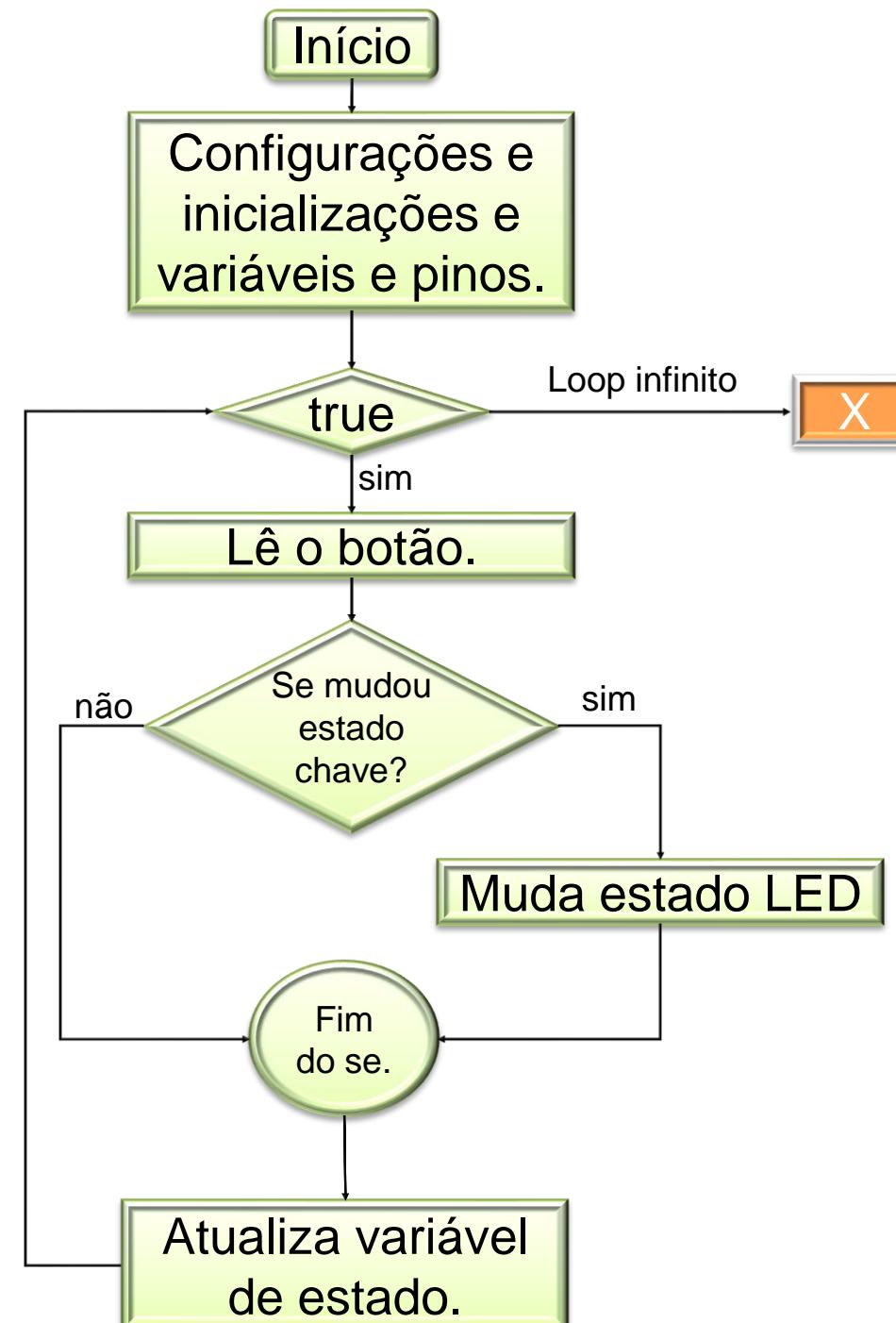
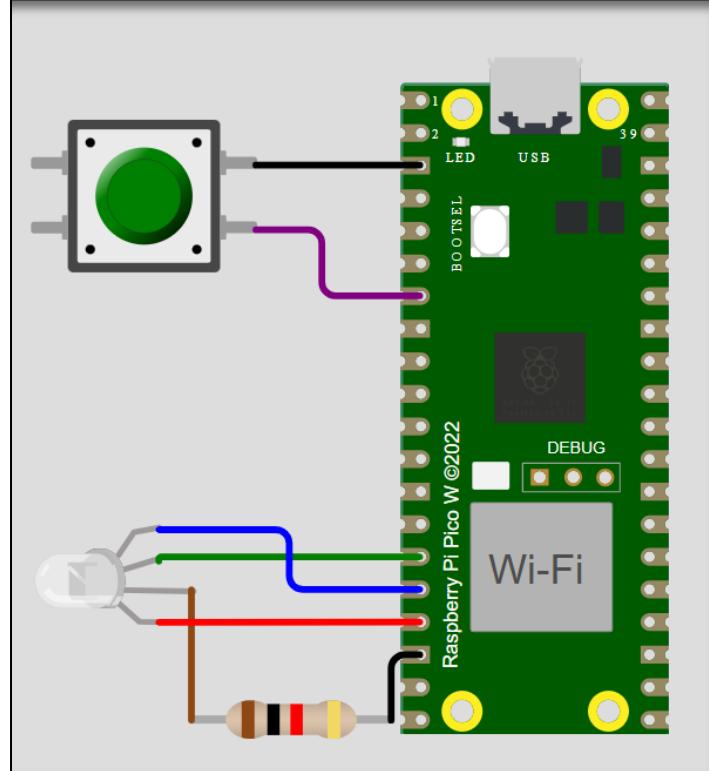


Exemplo prático de polling:

Um programa verifica constantemente se um botão foi pressionado.

Exemplo de polling no RP2040

Neste exemplo o usuário pressionará um botão e o LED VERDE mudará o seu estado.

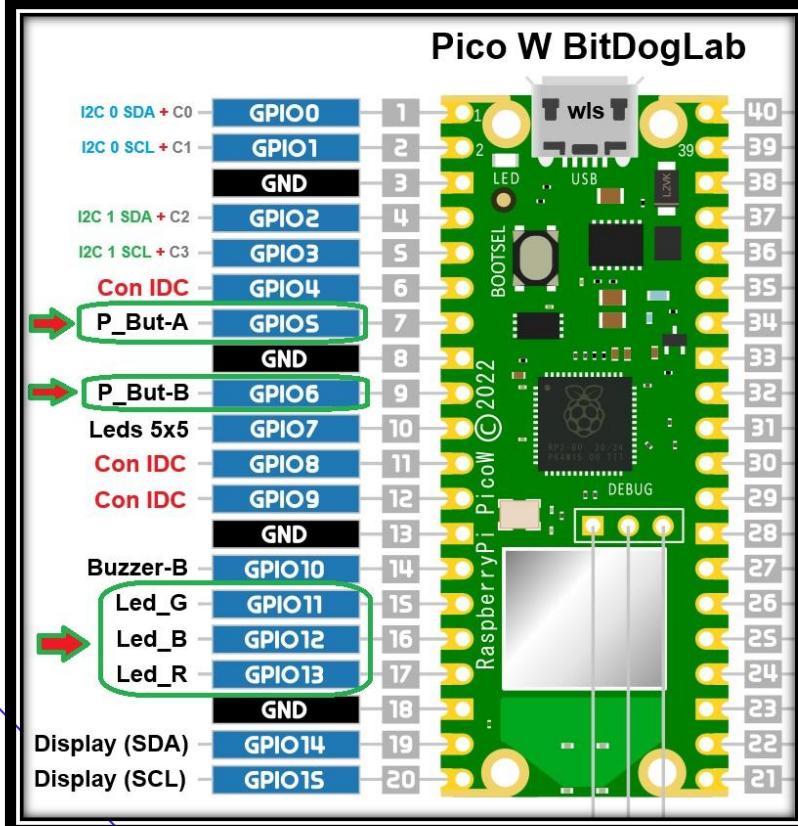
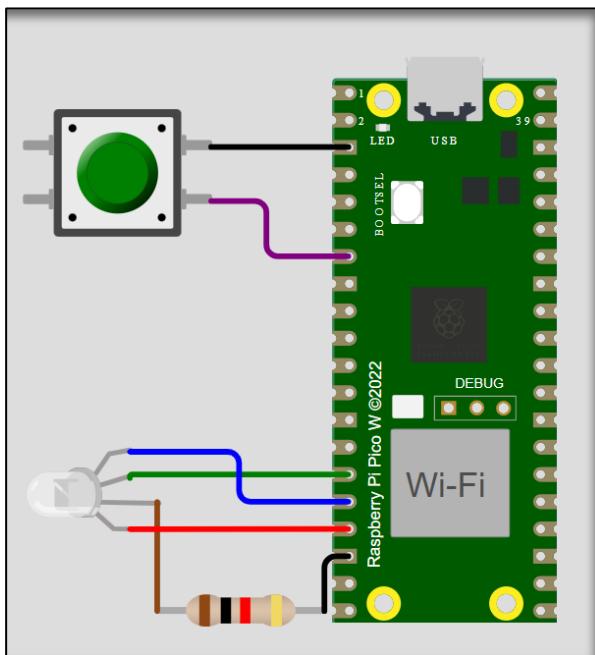


```
c SemInterrupt.c ...
1 #include "pico/stdlib.h"
2 // Configurações dos pinos
3 const uint led_pin = 11;      // Red=13, Blue=12, Green=11
4 const uint botao_pin = 5;     // Botão A = 5, Botão B = 6 , BotãoJoy = 22
5 int main()
6 {
7     // Inicializações
8     gpio_init(led_pin);          // Inicializa o pino do LED
9     gpio_set_dir(led_pin, GPIO_OUT); // Configura o pino como saída
10    gpio_put(led_pin, 0);         // Garante que o LED inicie apagado
11
12    gpio_init(botao_pin);        // Inicializa o botão
13    gpio_set_dir(botao_pin, GPIO_IN); // Configura o pino como entrada
14    gpio_pull_up(botao_pin);      // Habilita o pull-up interno
15    // Variáveis para o controle do estado do LED
16    bool led_estado = false;      // Estado inicial do LED (apagado)
17    bool ultimo_estado_botao = true; // Último estado do botão (inicialmente solto)
18    // Loop principal
19    while (true) {
20        // Lê o estado atual do botão
21        bool estado_atual_botao = gpio_get(botao_pin);
22        // Detecta uma transição do botão (pressionado e solto)
23        if (estado_atual_botao == false && ultimo_estado_botao == true) {
24            // Transição de solto para pressionado: alterna o estado do LED
25            led_estado = !led_estado;
26            gpio_put(led_pin, led_estado); // Atualiza o estado do LED
27        }
28        // Atualiza o estado anterior do botão
29        ultimo_estado_botao = estado_atual_botao;
30    }
31 }
```

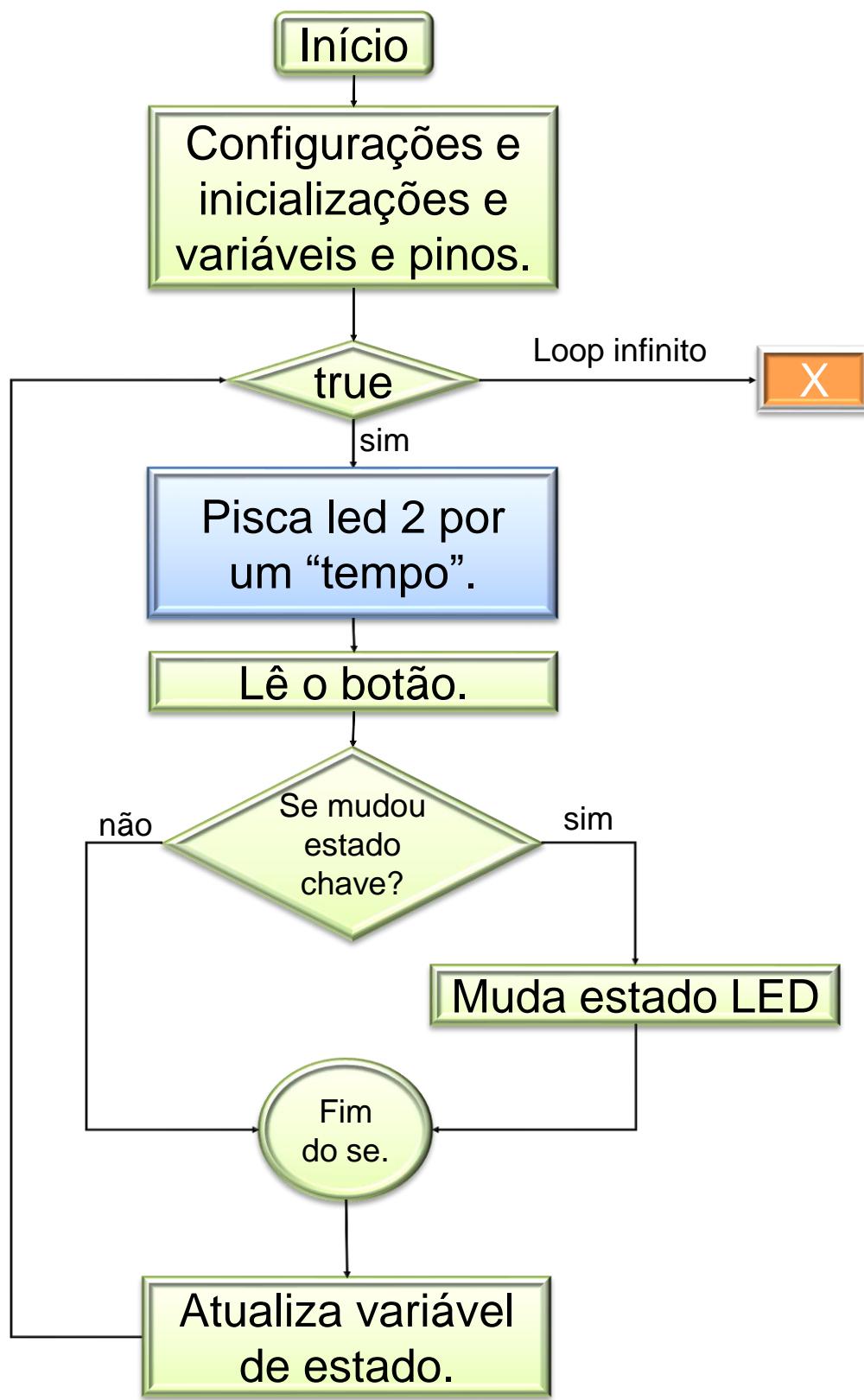
Vá no: github.com/wiltonlacerda
git clone <https://github.com/wiltonlacerda/EmbarcaTechU4C4.git>

Sem uso de Interrupções no RP2040

Neste exemplo o LED GREEN piscará continuamente, e quando o usuário pressionar o botão A o LED RED, eventualmente, deve mudar o seu estado.



Projeto: 01_PollingBlink



```
C SemInterruptBlink.c > main()
1 #include "pico/stdlib.h"
2 const uint led_pin = 11;           // Red=13, Blue=12, Green=11
3 const uint led_pin_pisca = 13;    // LED para piscar
4 const uint botao_pin = 5;         // Botão A = 5, Botão B = 6 , BotãoJoy = 22
5 #define tempo 1000
6 int main()
7 {
8     gpio_init(led_pin);           // Inicializa o pino do LED
9     gpio_set_dir(led_pin, GPIO_OUT); // Configura o pino como saída
10    gpio_put(led_pin, 0);          // Garante que o LED inicie apagado
11    gpio_init(led_pin_pisca);      // Inicializa o pino do LED para piscar
12    gpio_set_dir(led_pin_pisca, GPIO_OUT); // Configura o pino como saída
13    gpio_put(led_pin_pisca, 0);    // Garante que o LED inicie apagado
14    gpio_init(botao_pin);         // Inicializa o botão
15    gpio_set_dir(botao_pin, GPIO_IN); // Configura o pino como entrada
16    gpio_pull_up(botao_pin);      // Habilita o pull-up interno
17    bool led_estado = false;       // Estado inicial do LED (apagado)
18    bool ultimo_estado_botao = true; // Último estado do botão (inicialmente solto)
19    while (true)
20    {
21        gpio_put(led_pin_pisca, 1); // Liga o LED
22        sleep_ms(tempo);          // Mantém ligado por "tempo" ms
23        gpio_put(led_pin_pisca, 0); // Desliga o LED
24        sleep_ms(tempo);          // Mantém desligado por "tempo" ms
25        bool estado_atual_botao = gpio_get(botao_pin);
26        if (estado_atual_botao == false && ultimo_estado_botao == true)
27        {
28            led_estado = !led_estado;
29            gpio_put(led_pin, led_estado); // Atualiza o estado do LED
30        }
31        ultimo_estado_botao = estado_atual_botao;
32    }
33 }
```

Com uso de Interrupções no RP2040

Interrupções.

Uso do botão A LEDs RGB da placa BitDogLab para demonstração do funcionamento das interrupções.

O código faz o LED verde piscar a cada "tempo" ms.

Quando o botão A é pressionado, o LED azul alterna de estado.

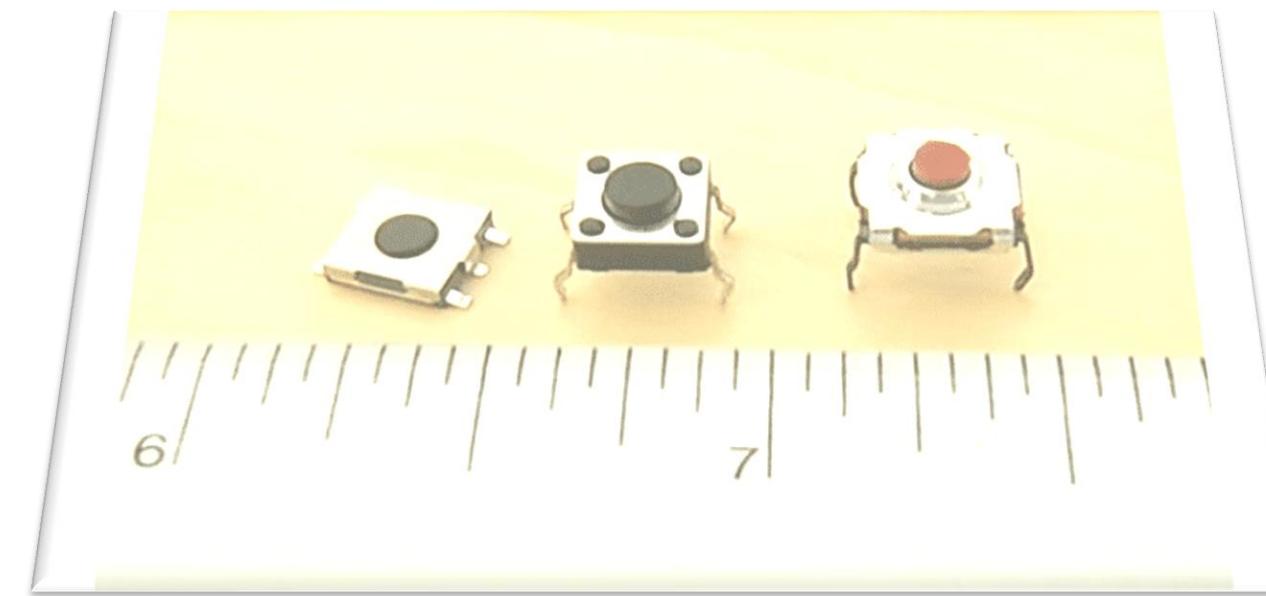
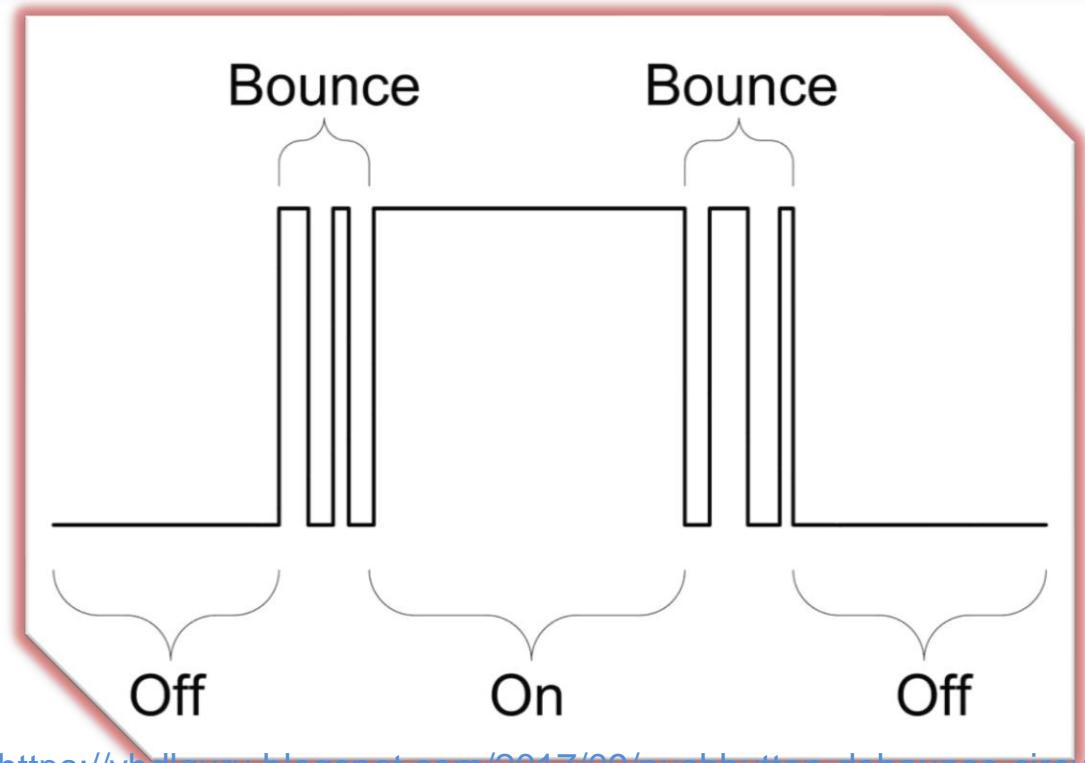
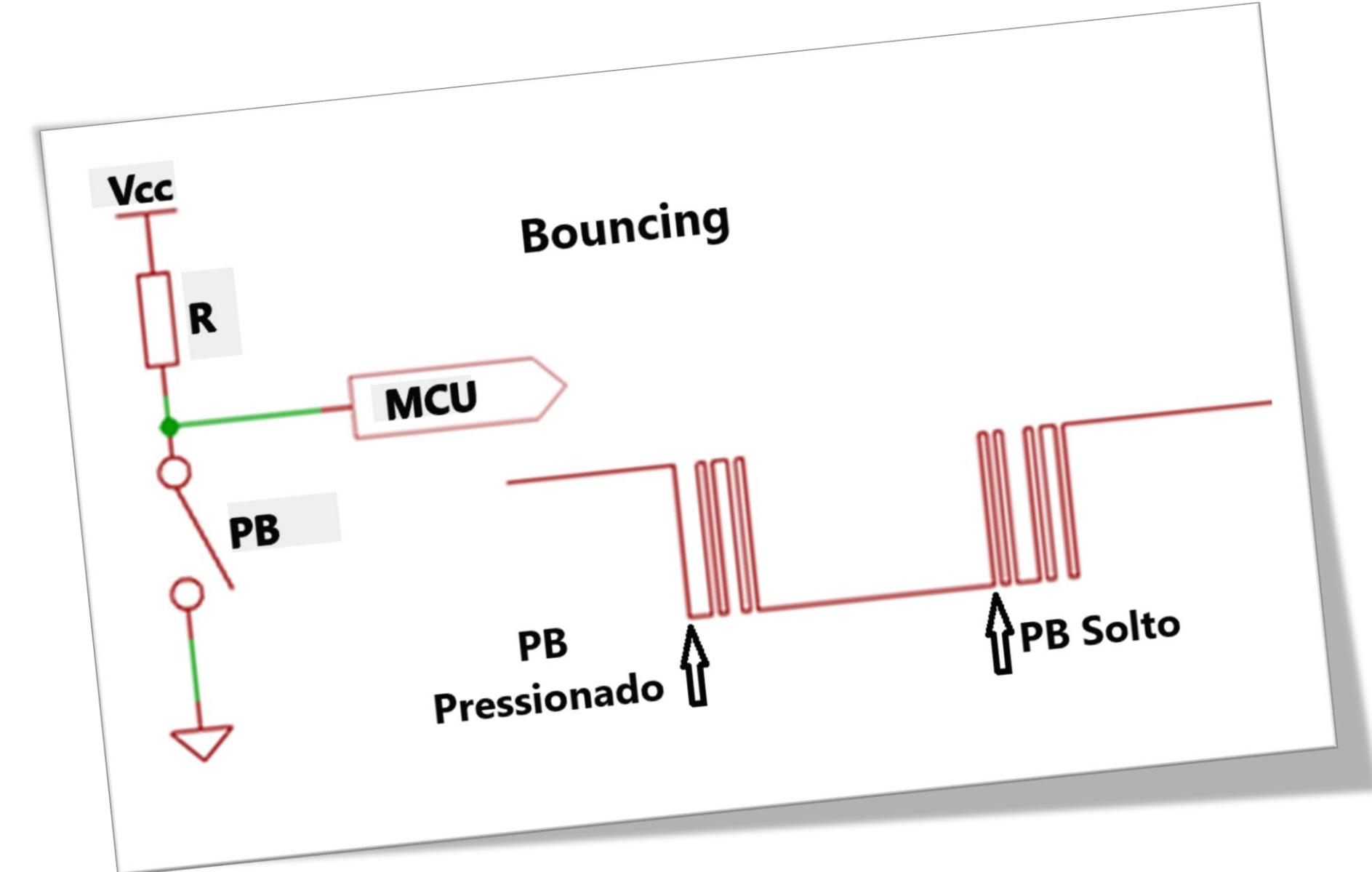
```
C Interrupt_Blink.c x
C Interrupt_Blink.c > gpio_irq_handler(uint, uint32_t)
1 #include "pico/stdlib.h"
2 const uint ledA_pin = 12;      // Blue => GPIO12
3 const uint ledB_pin = 11;      // Green => GPIO11
4 const uint button_0 = 5;       // Botão A = 5, Botão B = 6 , BotãoJoy = 22
5 #define tempo 2500
6 static void gpio_irq_handler(uint gpio, uint32_t events);
7 int main()
8 {
9     gpio_init(ledA_pin);          // Inicializa o pino do LED
10    gpio_set_dir(ledA_pin, GPIO_OUT); // Configura o pino como saída
11    gpio_init(ledB_pin);          // Inicializa o pino do LED
12    gpio_set_dir(ledB_pin, GPIO_OUT); // Configura o pino como saída
13    gpio_init(button_0);          // Inicializa o botão
14    gpio_set_dir(button_0, GPIO_IN); // Configura o pino como entrada
15    gpio_pull_up(button_0);        // Habilita o pull-up interno
16    gpio_set_irq_enabled_with_callback(button_0, GPIO_IRQ_EDGE_FALL, true, &gp
17
18    while (true)
19    {
20        gpio_put(ledB_pin, true);
21        sleep_ms(tempo);
22        gpio_put(ledB_pin, false);
23        sleep_ms(tempo);
24    }
25 }
26 // Função de interrupção "É possível observar o bounce do botão"
27 void gpio_irq_handler(uint gpio, uint32_t events)
28 {
29     bool estado_atual = gpio_get(ledA_pin); // Obtém o estado atual
30     gpio_put(ledA_pin, !estado_atual);      // Alterna o estado
31 }
```

Ln 31, Col 2 Spaces: 4 UTF-8 CRLF {} C Compile Run Pico SDK: 2.1.0 Board: pico

Bouncing

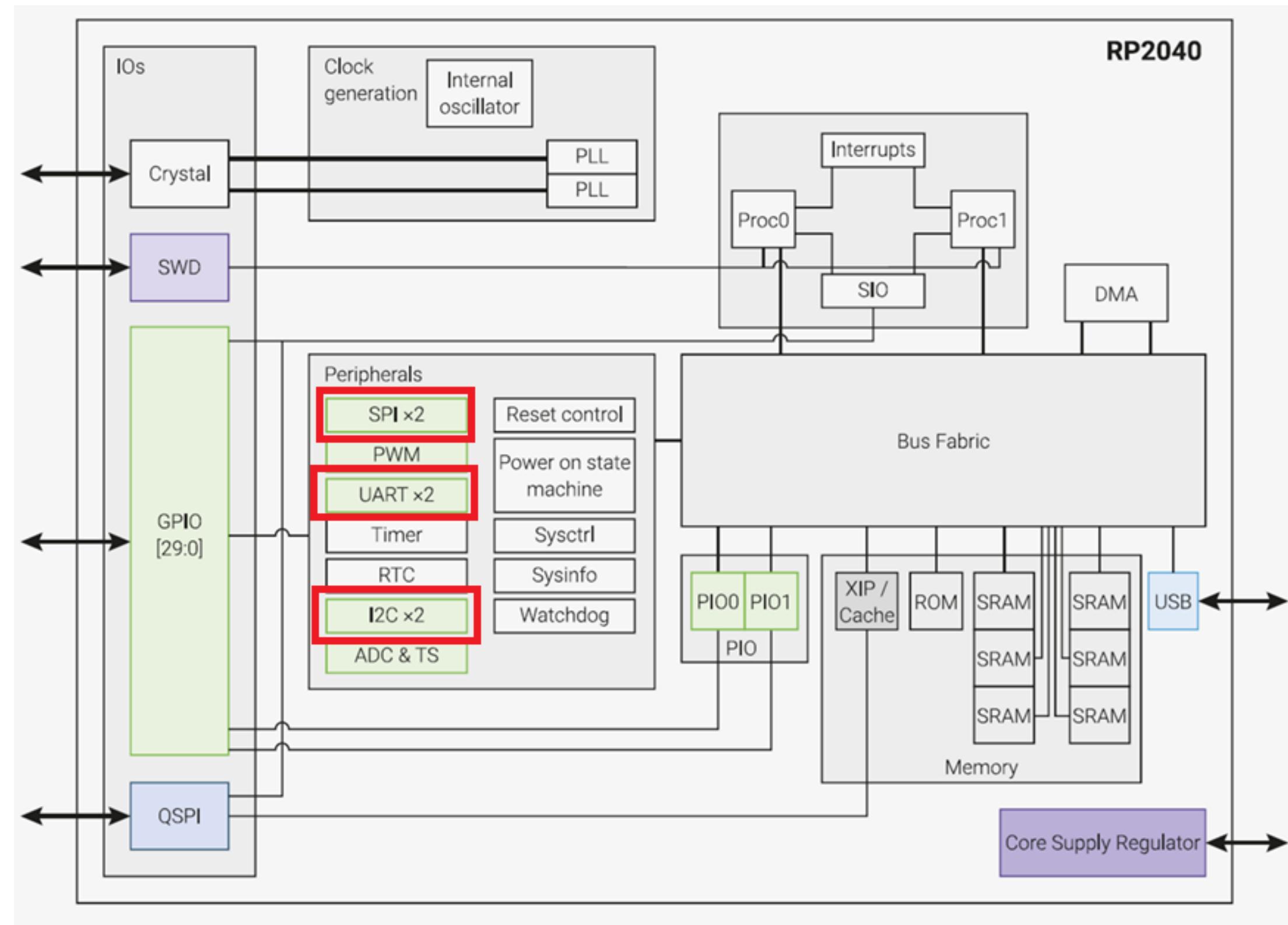
O que é Efeito Bouncing?

O Efeito Bouncing nada mais é que um fenômeno de **trepidação**. Ele é comum em Chaves, push buttons, reed-switch, entre outros. Ocorre quando os contatos internos não se conectam ou desconectam instantaneamente ao serem acionados. Isso resulta em múltiplos sinais de liga/desliga (picos de tensão) em um curto intervalo de tempo, em vez de uma transição limpa entre os estados.



Interfaces de Comunicação Serial

UART, SPI e I2C no RP2040.



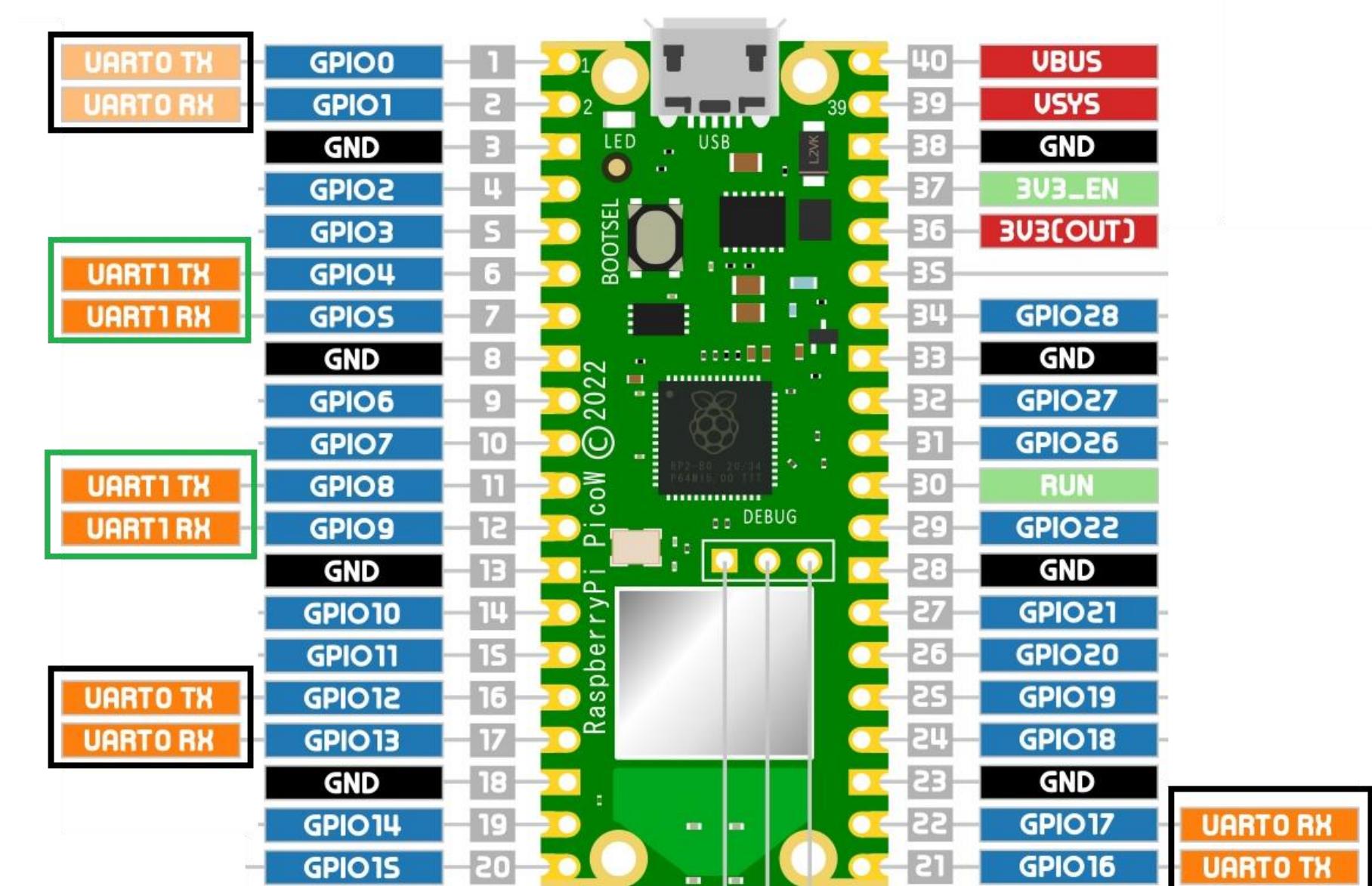
Comunicação Serial UART

UART - Universal Asynchronous Receiver/Transmitter

A UART é uma interface de comunicação assíncrona que permite a troca de dados entre dois dispositivos. Ela é frequentemente utilizada em módulos Bluetooth, GPS e para depuração de código.

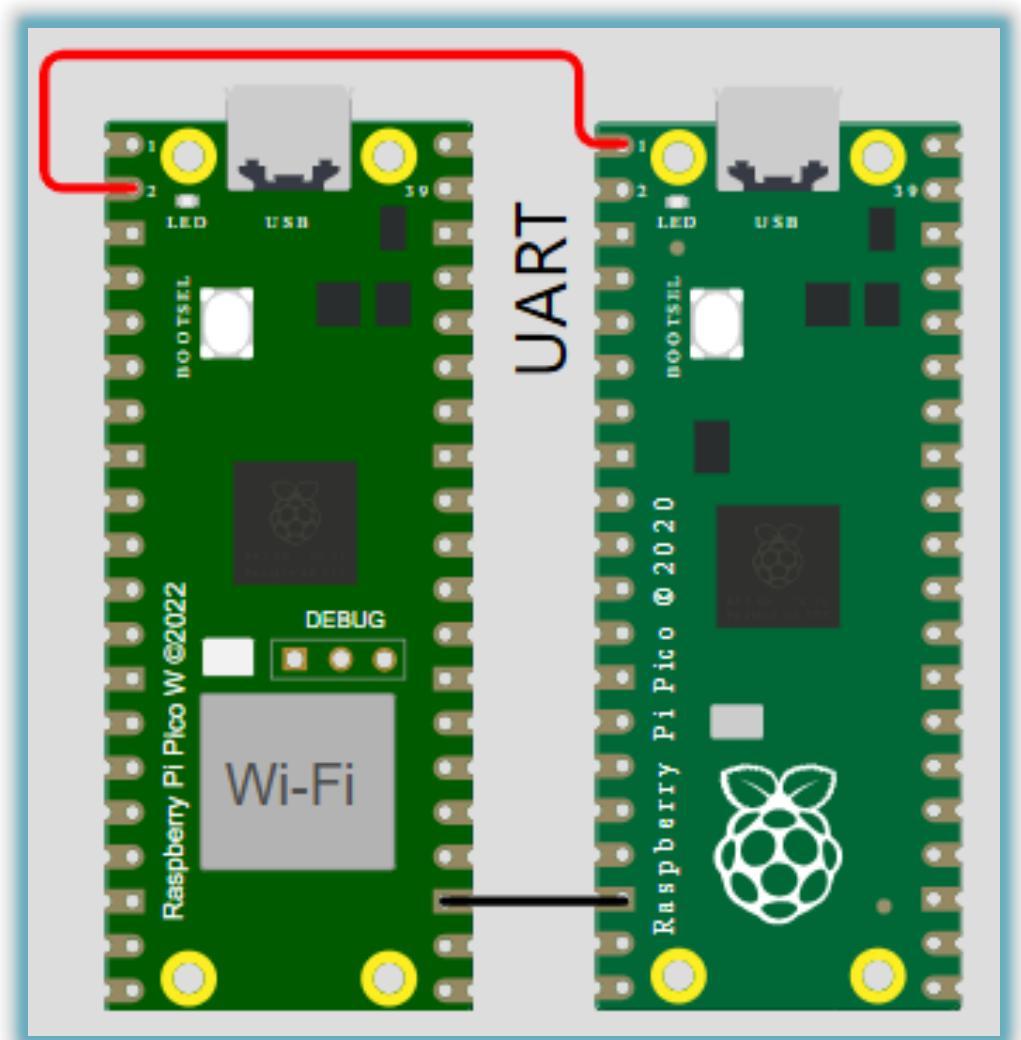
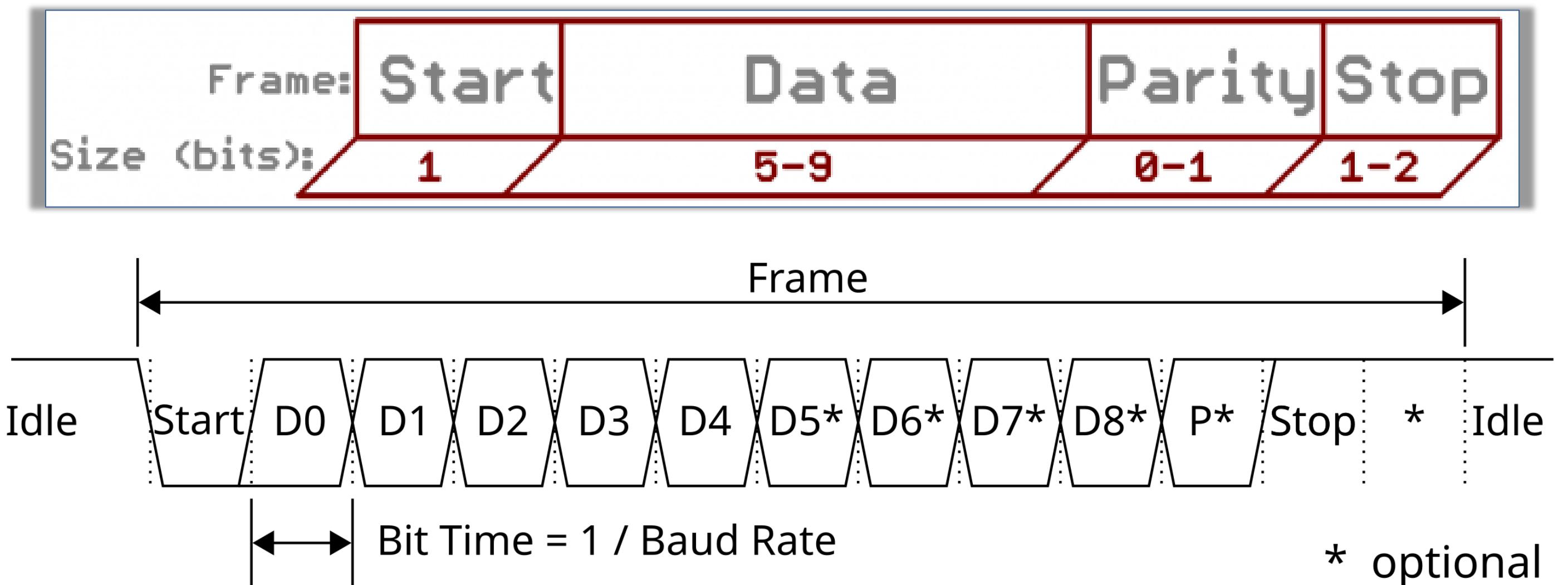
Funcionamento da UART

UART utiliza apenas dois fios principais: **TX** (transmissor) e **RX** (receptor), além do fio de **GND** (terra). A **comunicação é assíncrona**, o que significa que não utiliza um sinal de clock compartilhado. Isso simplifica a implementação, mas requer que os dispositivos estejam configurados com a mesma taxa de transmissão, conhecida como **baud rate**.



Formatação do Frame UART

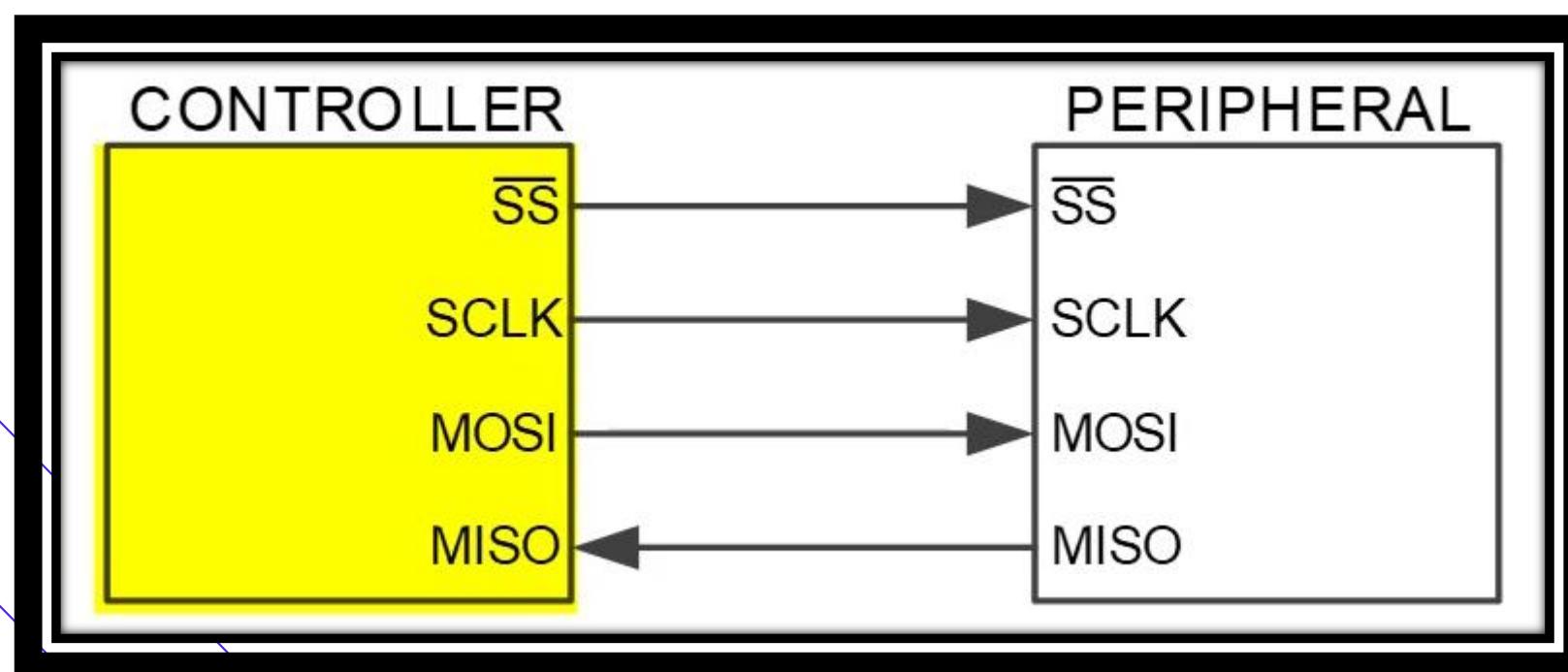
Cada bloco (geralmente um byte) de dados transmitido é, na verdade, enviado em um pacote ou quadro de bits. Os quadros são criados adicionando **bits de sincronização** e **bits de paridade** aos dados. A quantidade de **dados** em cada pacote pode variar **entre 5 a 9 bits**. O bit de início (**start bit**) é sempre indicado por uma linha de dados em repouso (**IDLE**) que muda **de 1 para 0**, enquanto o bit(s) de parada (stop bit(s)) retornará ao estado de repouso mantendo a linha em **1**. A paridade (**parity**) é uma forma muito simples e de baixo nível de verificação de erros. Ela pode ser de dois tipos: **ímpar** (odd) ou **par** (even). Finalmente, basta escolher a ordem de envio: (**MSB**, most-significant bit) é enviado primeiro, ou vice-versa.



Comunicação serial SPI

SPI (Serial Peripheral Interface)

O SPI é um protocolo de comunicação serial amplamente utilizado para conectar dispositivos eletrônicos, como microcontroladores, sensores, displays, memórias, conversores analógicos/digitais e outros periféricos. Ele é rápido, simples e eficiente, sendo ideal para trocas de dados de alta velocidade em curtas distâncias.



Características em destaque do SPI:

Comunicação mestre-escravo:

- Um dispositivo atua como mestre (geralmente um microcontrolador) e controla a comunicação com um ou mais escravos.

Síncrono:

- A comunicação é sincronizada por um relógio compartilhado (SCK).

Full-duplex:

- Permite enviar e receber dados simultaneamente.

Linhas de comunicação:

O SPI usa um conjunto mínimo de quatro fios:

- SCK** (Serial Clock): Gerado pelo mestre, sincroniza a transferência de dados.
- MOSI** (Master Out, Slave In): Linha de dados do mestre para o escravo.
- MISO** (Master In, Slave Out): Linha de dados do escravo para o mestre.
- SS** (Slave Select): **CS** (Chip Select) **CE** (Chip Enable)

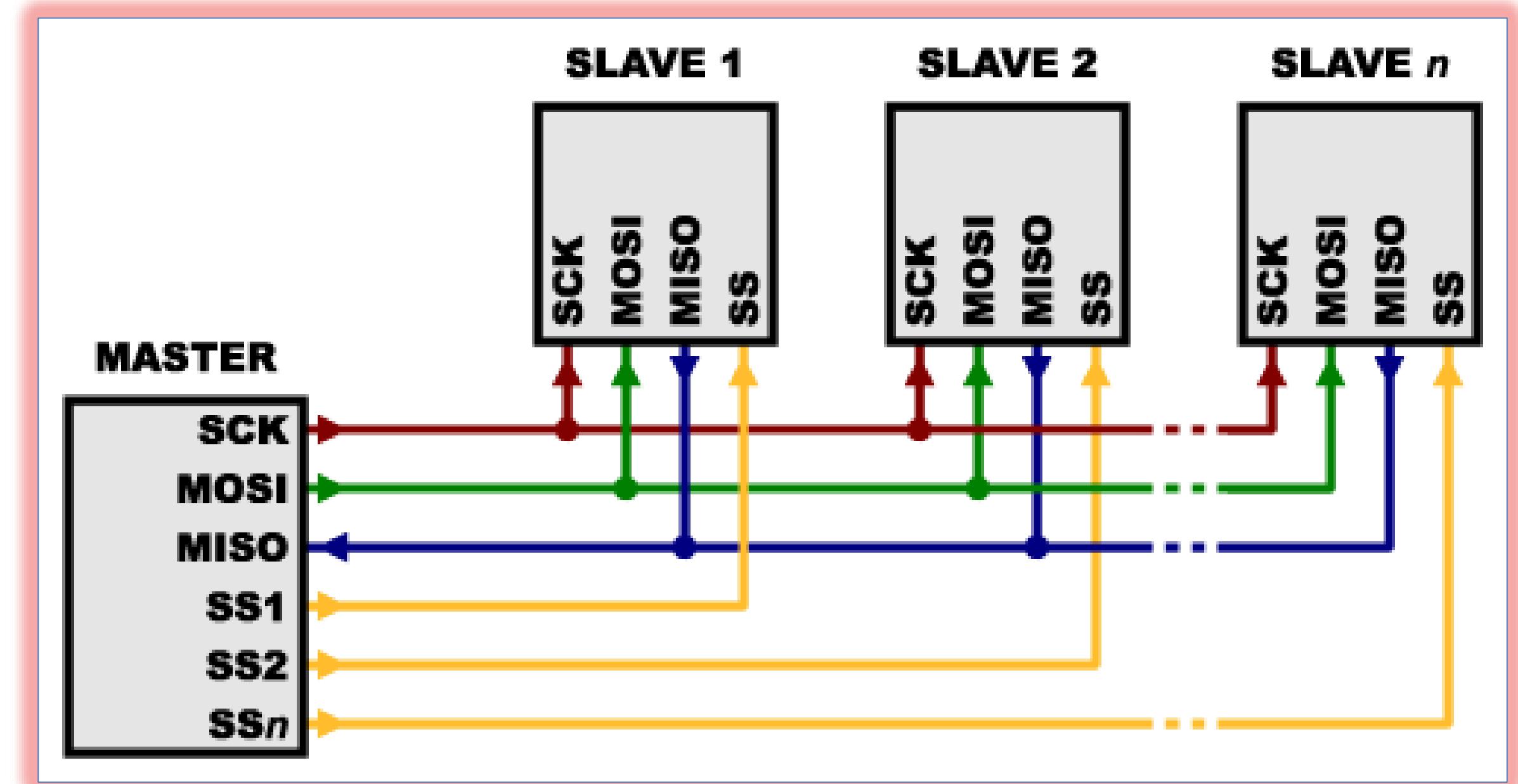
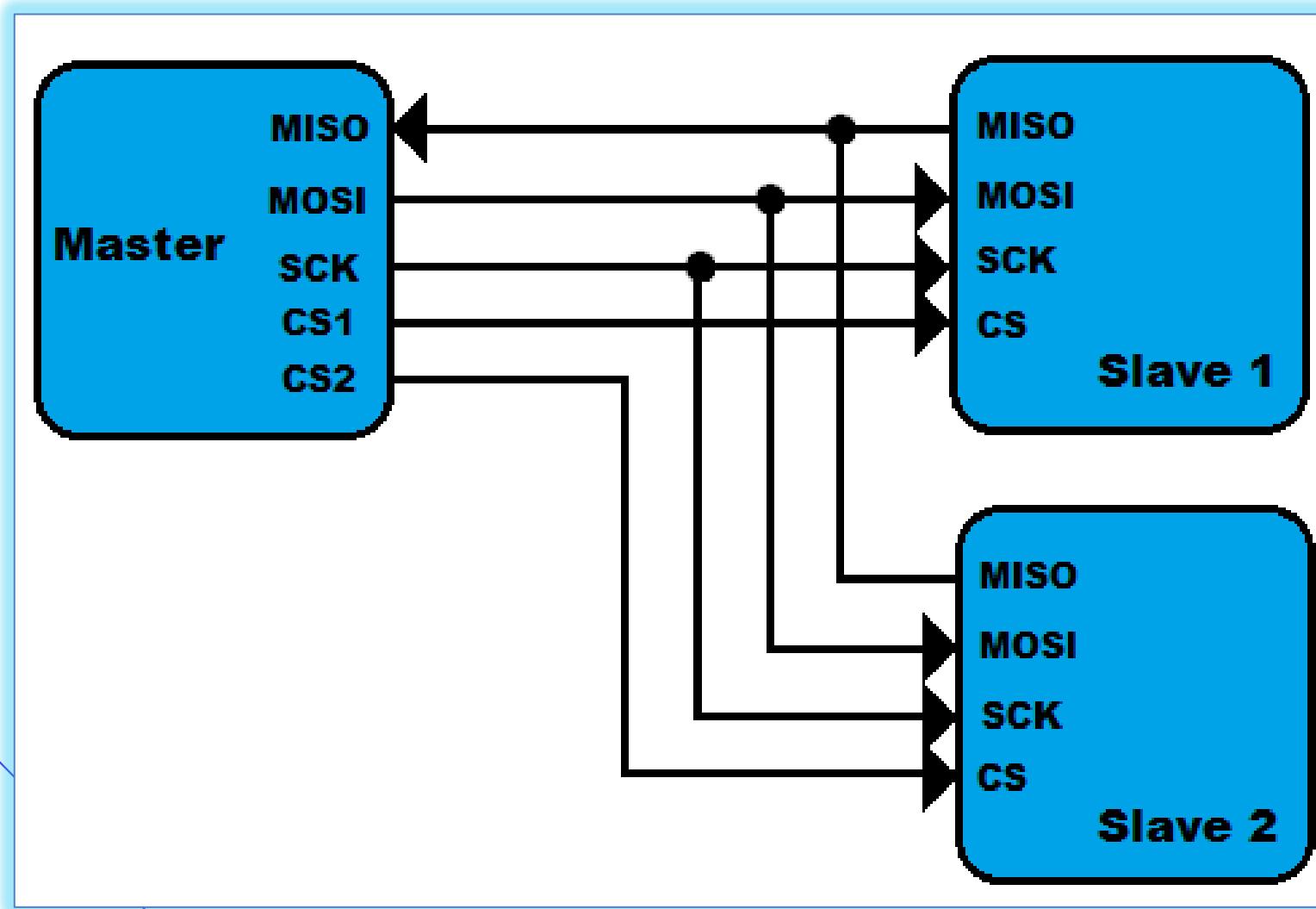
Usada para selecionar o escravo ativo. Em sistemas com múltiplos escravos, cada um tem seu próprio pino SS.

Velocidade configurável:

- A taxa de transferência de dados é determinada pelo mestre, com frequências que podem atingir dezenas de megahertz, dependendo do hardware.

Comunicação serial SPI

Representação
esquemática de
comunicação SPI com 2 ou 3
dispositivos periféricos.



Comunicação serial SPI

No SPI, apenas um lado gera o sinal de clock (geralmente chamado de CLK ou **SCK** para Serial Clock).

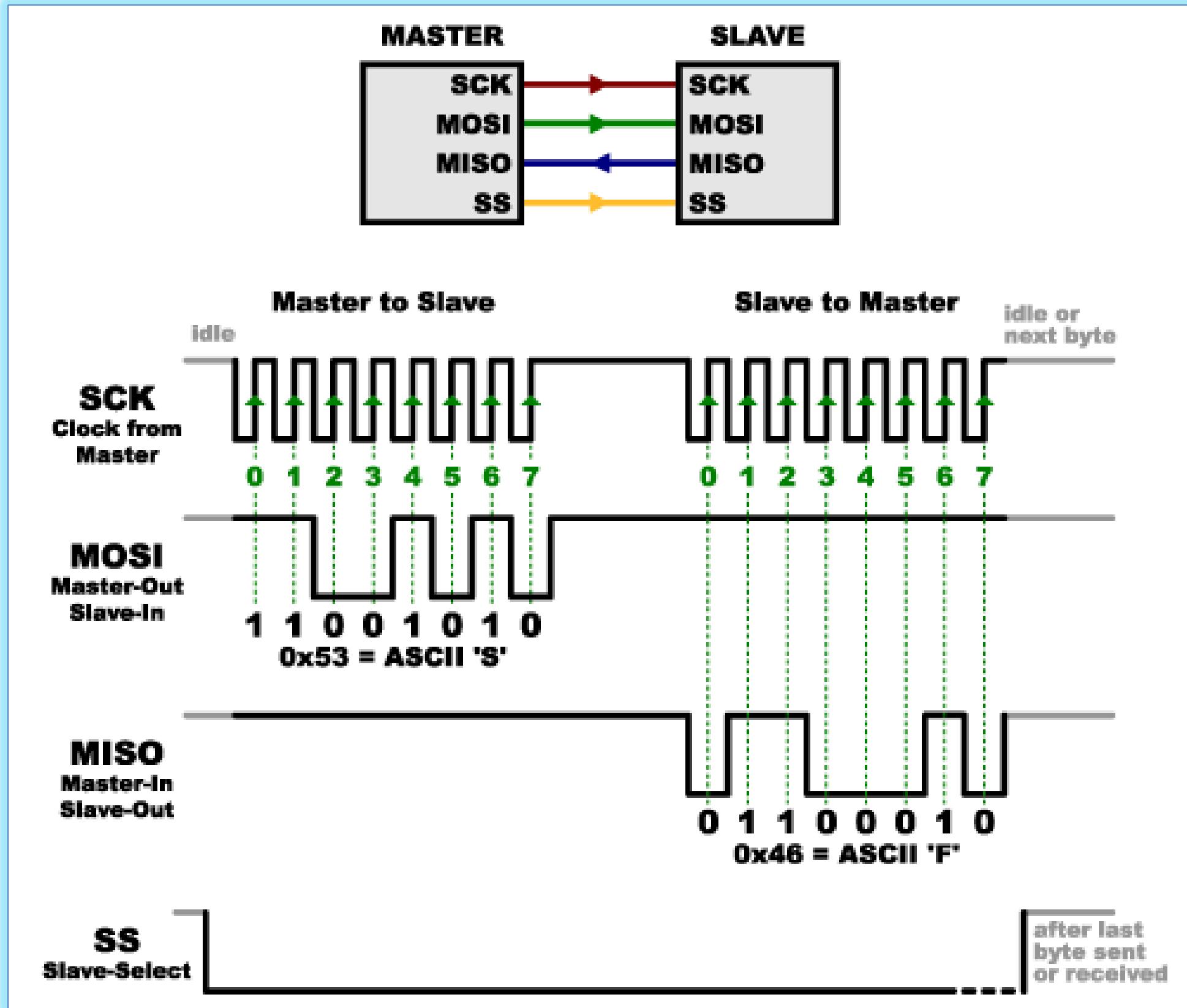
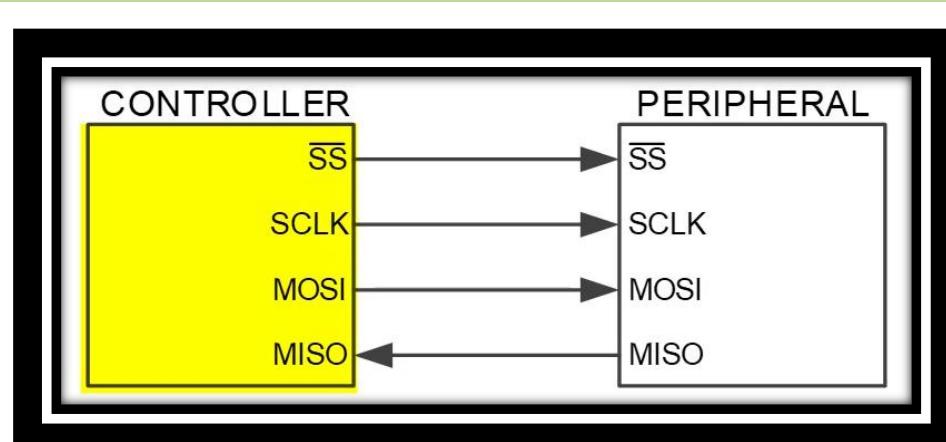
O lado que gera o clock é chamado de **controlador** (anteriormente "master"), e o outro lado é chamado de **periférico** (anteriormente "slave").

Há sempre apenas um controlador (geralmente o microcontrolador), mas podem existir **múltiplos periféricos**.

Quando dados são enviados do **controlador** para um **periférico**, eles são transmitidos por uma linha de dados chamada **MOSI** (Controller Out / Peripheral In ou também **Master Out e Slave In**).

Se o **periférico** precisar enviar uma resposta ao **controlador**, este continuará a gerar o número pré-determinado de ciclos de clock, e o **periférico** enviará os dados por meio de uma terceira linha de dados chamada **MISO** (Peripheral Out / Controller In ou **Master In / Slave Out**).

A linha **SS** (**Seleção**) é normalmente mantida em estado alto, o que desconecta o dispositivo periférico do barramento SPI.



Comunicação serial I²C

I²C (Inter-Integrated Circuit)

O I²C é um **protocolo serial síncrono** usado para a troca de dados entre MCUs e periféricos, tais como: sensores e displays.

Criado pela Philips Semiconductors em 1982. O protocolo suporta **múltiplos dispositivos** alvo em um barramento de comunicação e também pode suportar **múltiplos controladores** que enviam e recebem comandos e dados. A comunicação é **enviada em pacotes** de bytes com um **endereço exclusivo** para cada dispositivo alvo.

Características em destaque do I²C :

Barramento de dois fios:

- **SDA** (Serial Data Line): linha de dados bidirecional.
- **SCL** (Serial Clock Line): linha de clock gerada pelo dispositivo mestre.

Topologia mestre-escravo:

- O dispositivo mestre controla o clock e inicia as comunicações.
- Escravos respondem a comandos do mestre.

Endereçamento:

- Cada dispositivo escravo tem um endereço único de 7 bits.

Velocidade:

Standard Mode: até 100 kbps.

Fast Mode Plus: até 1 Mbps.

Ultra-Fast Mode: até 5 Mbps.

Fast Mode: até 400 kbps.

High-Speed Mode: até 3.4 Mbps.

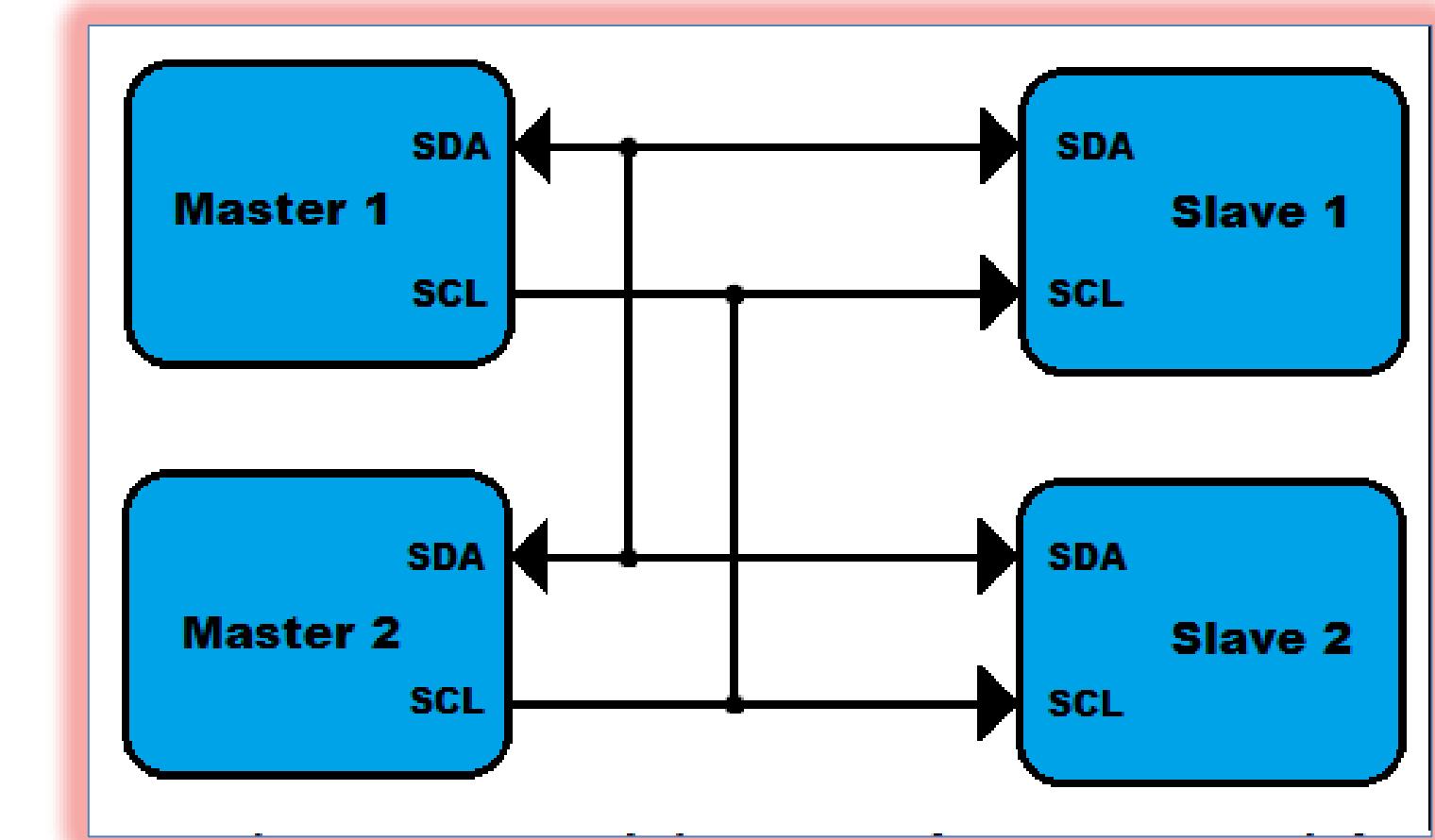
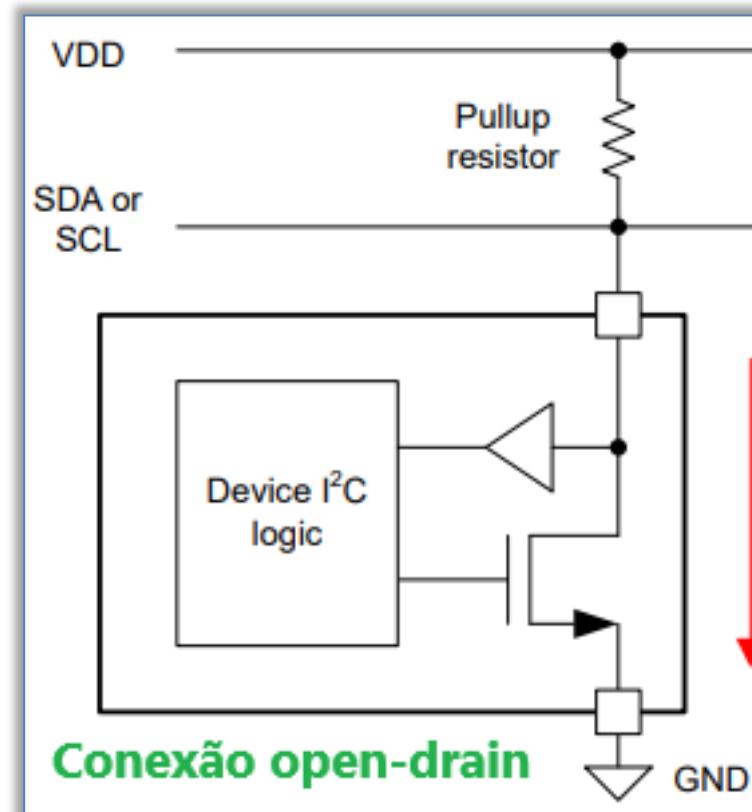
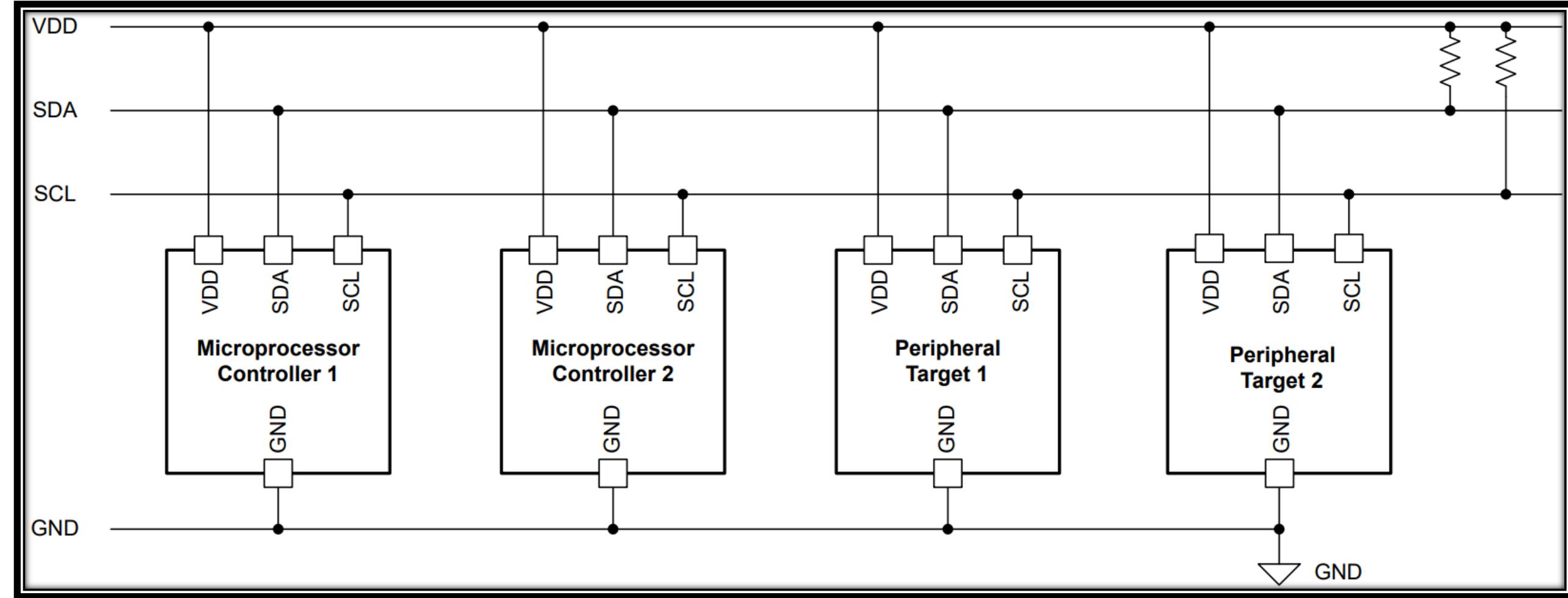
Pull-up resistors:

As linhas SDA e SCL são conectadas a resistores pull-up para manter o estado alto quando não estão em uso.

Comunicação serial I²C

Linhas de comunicação RP2

- **SDA** (Serial Data)
- **SCL** (Serial Clock)



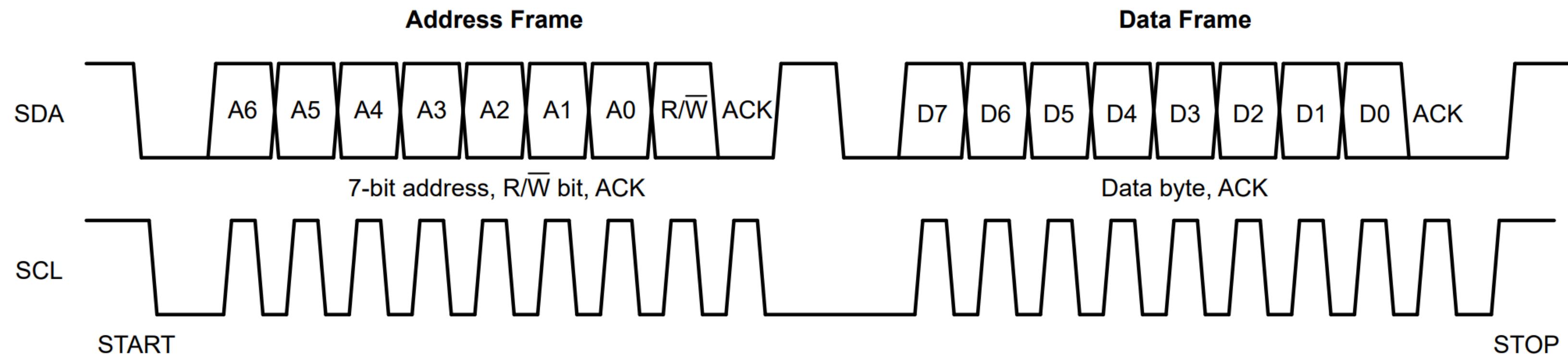
Comunicação serial I²C

Quadro de endereços e Dados do protocolo I²C

As mensagens são divididas em dois tipos de quadro (frame):

- Um **quadro de endereço**, onde o mestre indica o escravo para o qual a mensagem está sendo enviada. (7 bits o que resulta em no máximo **127 dispositivos** endereçáveis).
- Um ou mais **quadros de dados**, que são mensagens de dados de 8 bits passadas de mestre para escravo ou vice-versa .

Os dados são colocados na linha SDA depois que o SCL fica baixo e são amostrados depois que a linha SCL fica alta. O tempo entre a transição do clock e a leitura/gravação dos dados é definido pelos dispositivos no barramento e varia de chip para chip.



Sinais Digitais

Um sinal digital é uma representação discreta de uma grandeza física, composta por valores específicos em intervalos de tempo definidos.

Características:

- Discreto no tempo e na amplitude.
- Assume um número **finito de valores**.
(geralmente representados em binário: 0 e 1).

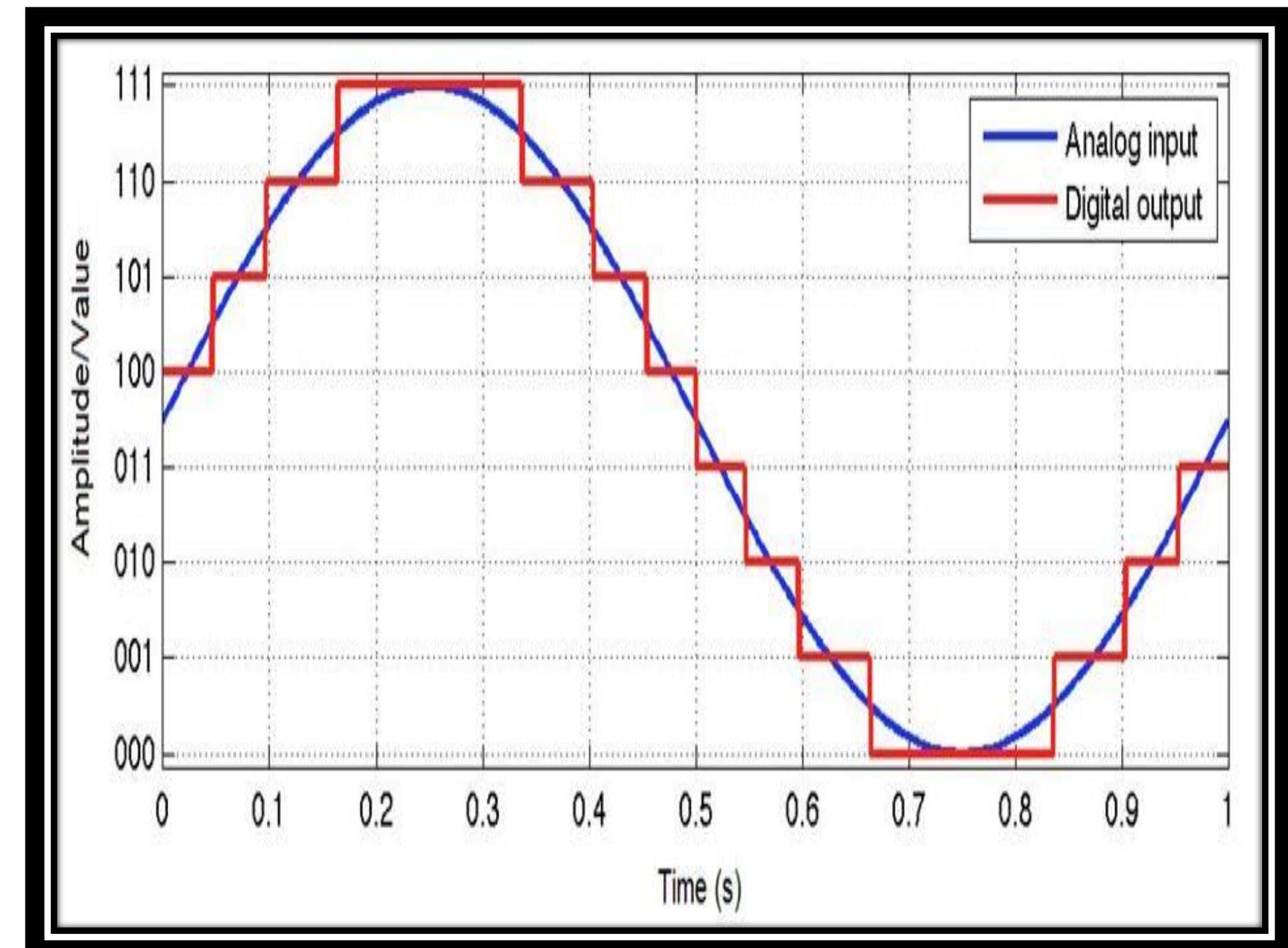
Exemplos:

Dados em um computador, mensagens de texto, imagens digitais.

Representação Gráfica:

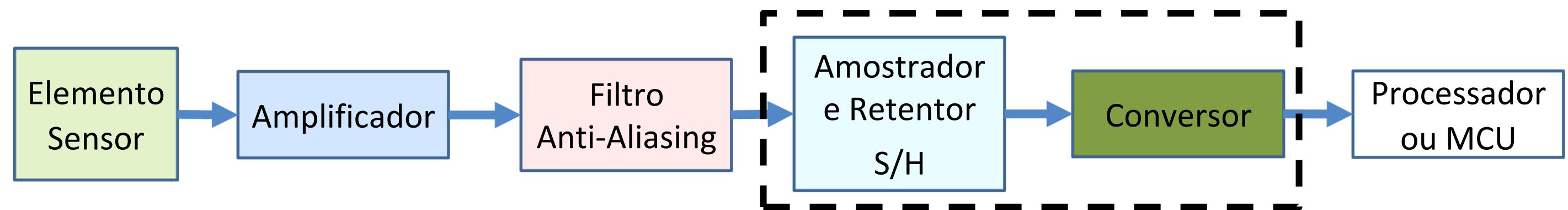
Uma onda senoidal discretizada.

Veja o seu gráfico Eixo X (tempo) vs. Eixo Y (amplitude).



Visão Geral do Sistema de Conversão

Um sistema de conversão analógico-digital (ADC) é composto por vários estágios, cada um com uma função específica. Esses estágios garantem que o sinal analógico seja convertido em digital de forma precisa e eficiente.



Resolução do ADC

A resolução de um ADC refere-se ao número de bits usados para representar o sinal digital. Quanto maior a resolução, mais preciso é o sinal digital.

Fórmula da Resolução:

$$\text{Resolução} = \frac{V_{\text{ref}}}{2^n}$$

- V_{ref} : Tensão de referência do ADC. Ex. RP2: 3,3V; Arduino Uno: 5,0V
- n : Número de bits do ADC. Ex. RP2 12bits; Arduino Uno 10bits.

Ex. $\text{Resolução} = \frac{V_{\text{ref}}}{2^n} = \frac{3,3V}{2^{12}} = \frac{3,3V}{4096} = 805,6 \times 10^{-6} V/bit$

Tipos de Conversores Analógico-Digital

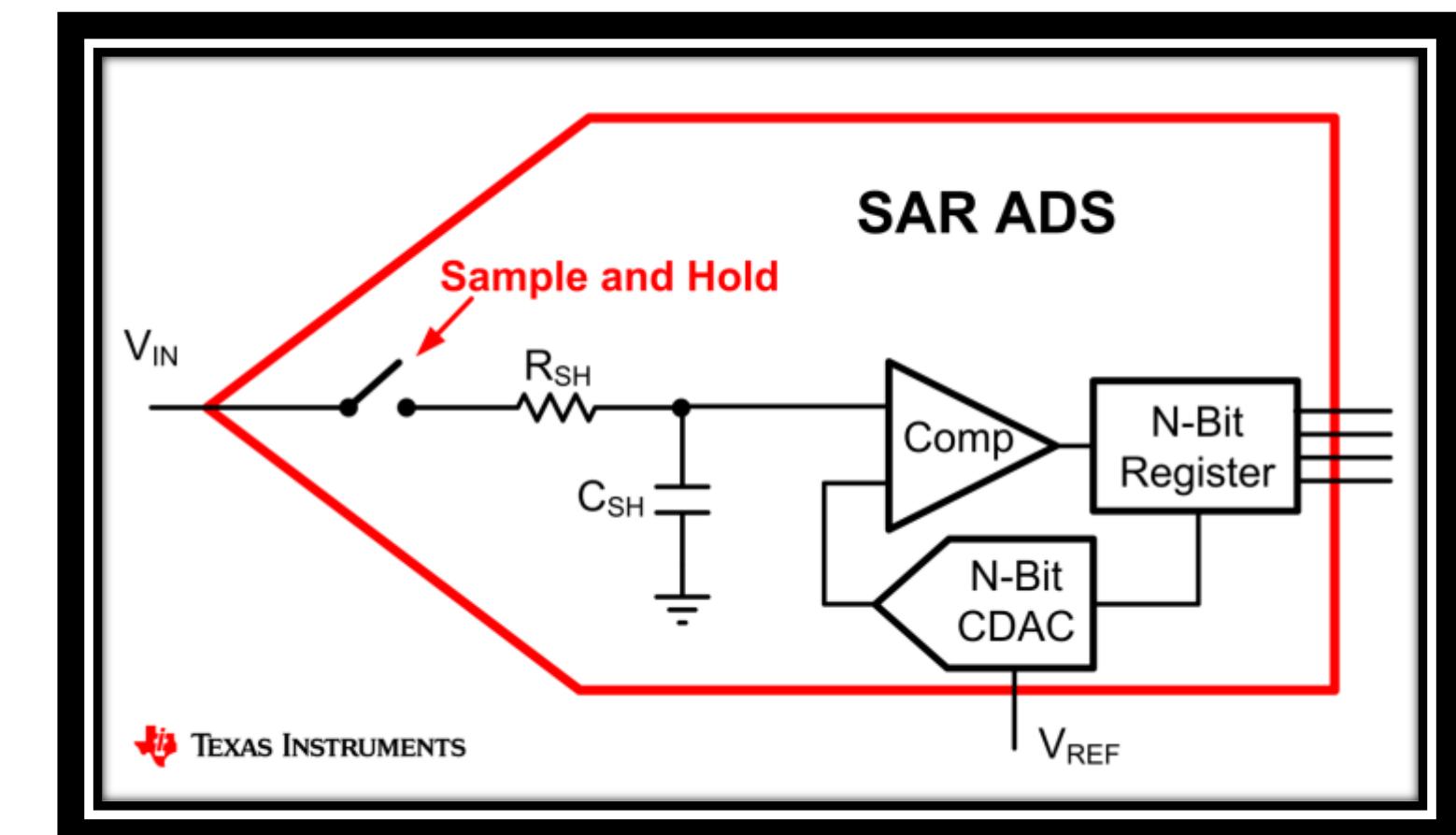
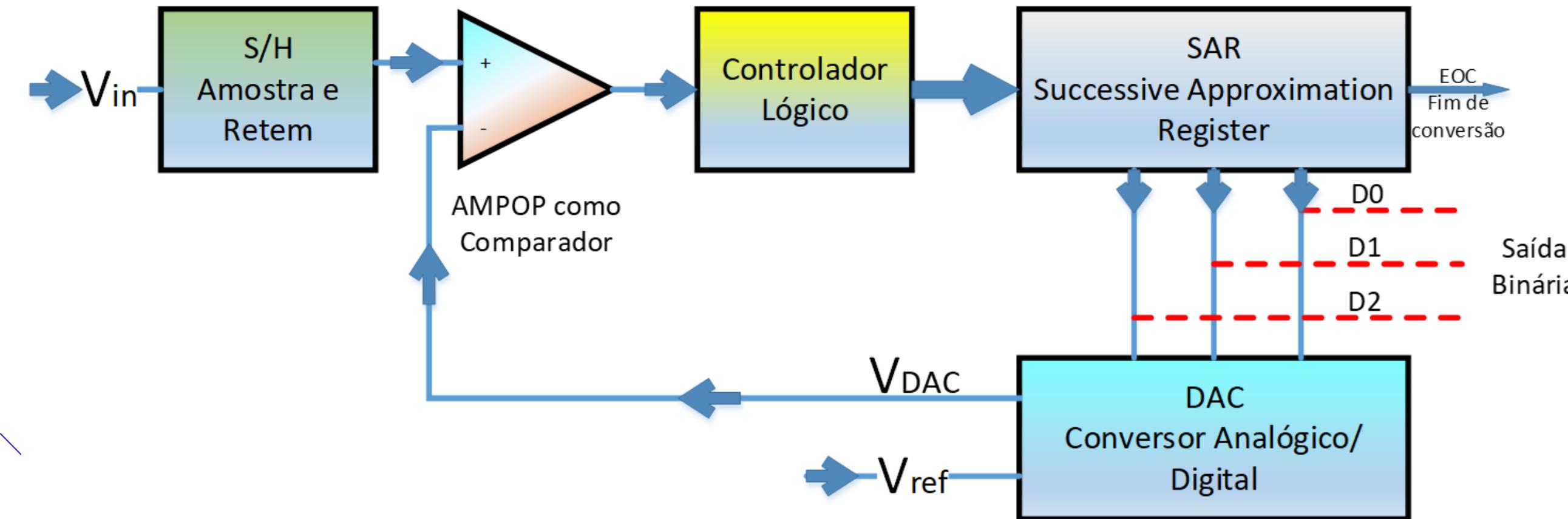
Existem várias arquiteturas de ADCs, cada uma com características específicas que as tornam adequadas para determinadas aplicações. Podemos destacar:

- **ADC Flash.**
- **ADC de Aproximação Sucessiva (SAR).**
- **ADC Sigma-Delta ($\Delta\Sigma$).**
- **ADC de Integração (Dual-Slope).**
- **ADC Pipeline.**

ADC de Aproximação Sucessiva - SAR

O **SAR** (Successive Approximation Register) é um tipo de conversor analógico-digital (ADC) amplamente utilizado devido ao seu equilíbrio entre velocidade, resolução e consumo de energia.

O **SAR ADC** opera usando um método de aproximação sucessiva para determinar o valor digital correspondente ao sinal analógico de entrada.



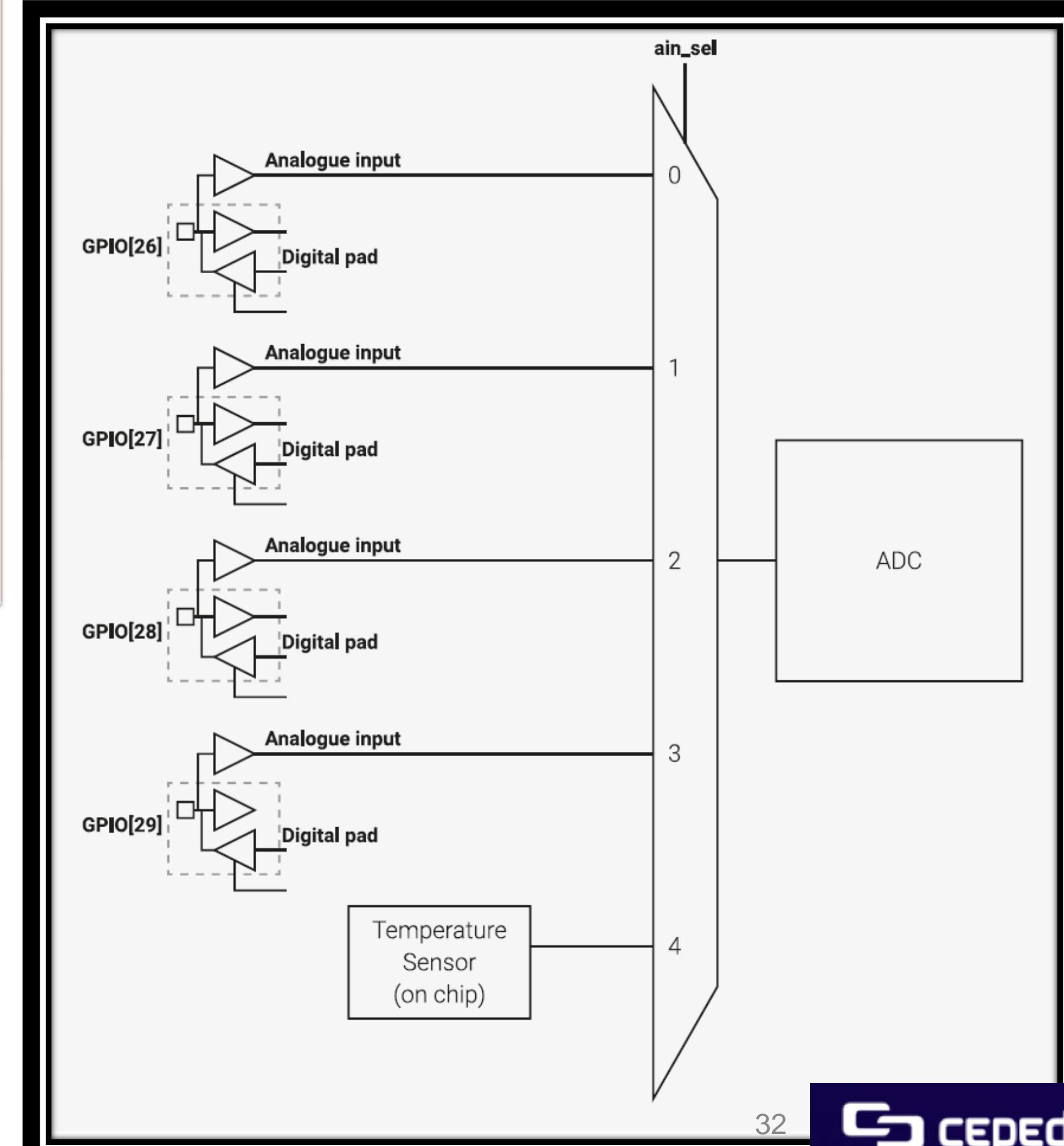
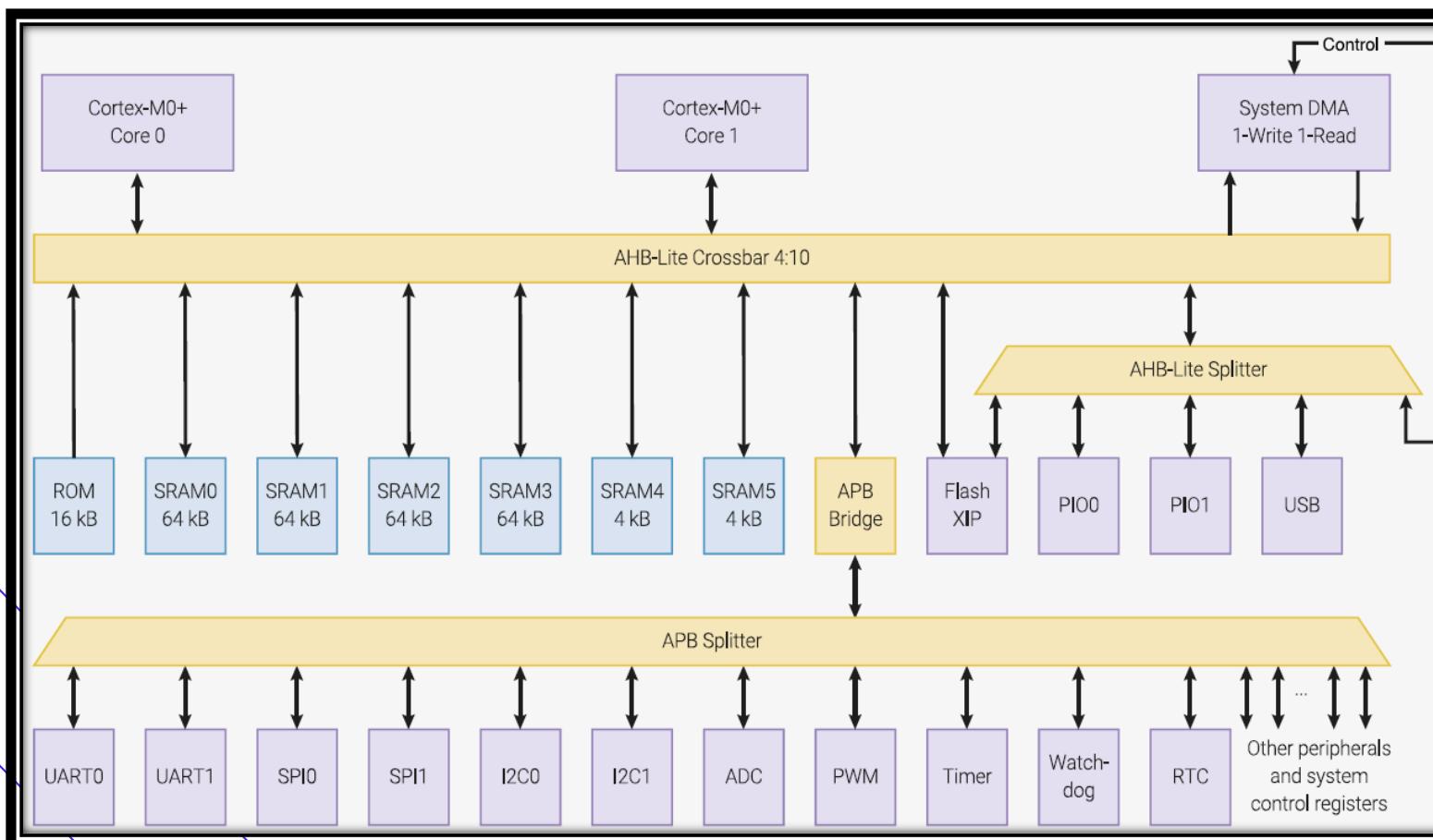
ADC no RP2040

4.9. ADC and Temperature Sensor

(informação completa no datasheet)

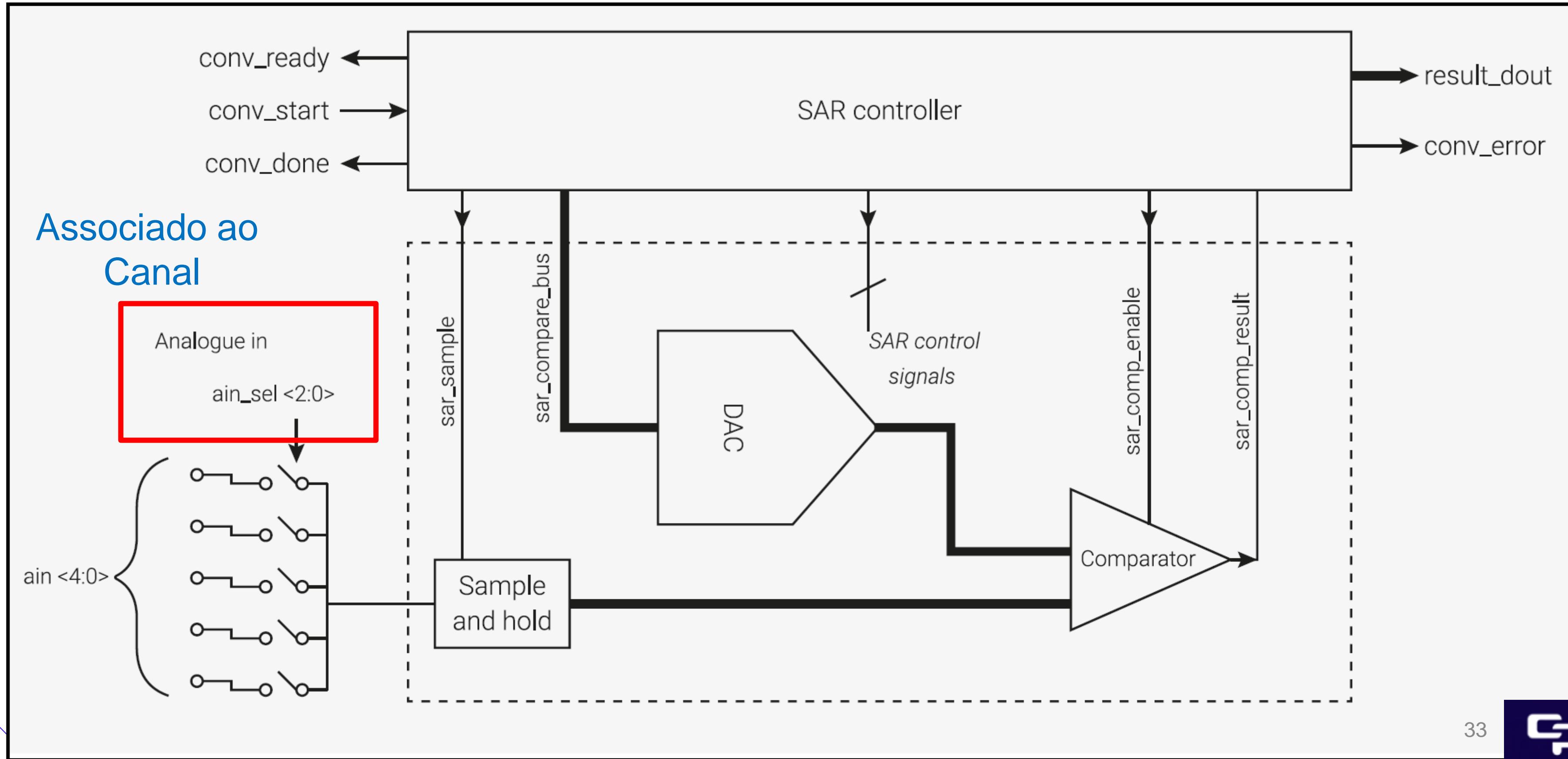
O RP2040 conta com um conversor analogico-digital internamente, com as seguintes características:

- SAR ADC (Seção 4.9.2 do datasheet)
- 500ksps (usando um clock independente de 48MHz)
- 12-bit com uma ENOB de 8.7 bits (Seção 4.9.3)*
- Conta com um multiplexador de cinco entradas, onde:
 - Quatro entradas estão disponíveis nos pinos do chip GPIO[26:29]
 - Uma entrada é dedicada para um sensor interno de temperatura (seção 4.9.5)
- Geração de interrupção
- Interface DMA (Seção 4.9.2.5)
- A referência de tensão do ADC é a alimentação de 3,3V do chip (VREF = VDD)



ADC no RP2040

The ADC requires a 48MHz clock (`clk_adc`), which could come from the USB PLL. Capturing a sample takes 96 clock cycles ($96 \times 1/48\text{MHz} = 2\mu\text{s}$ per sample (**500ksps**)). The clock must be set up correctly before enabling the ADC.



ADC no RP2040

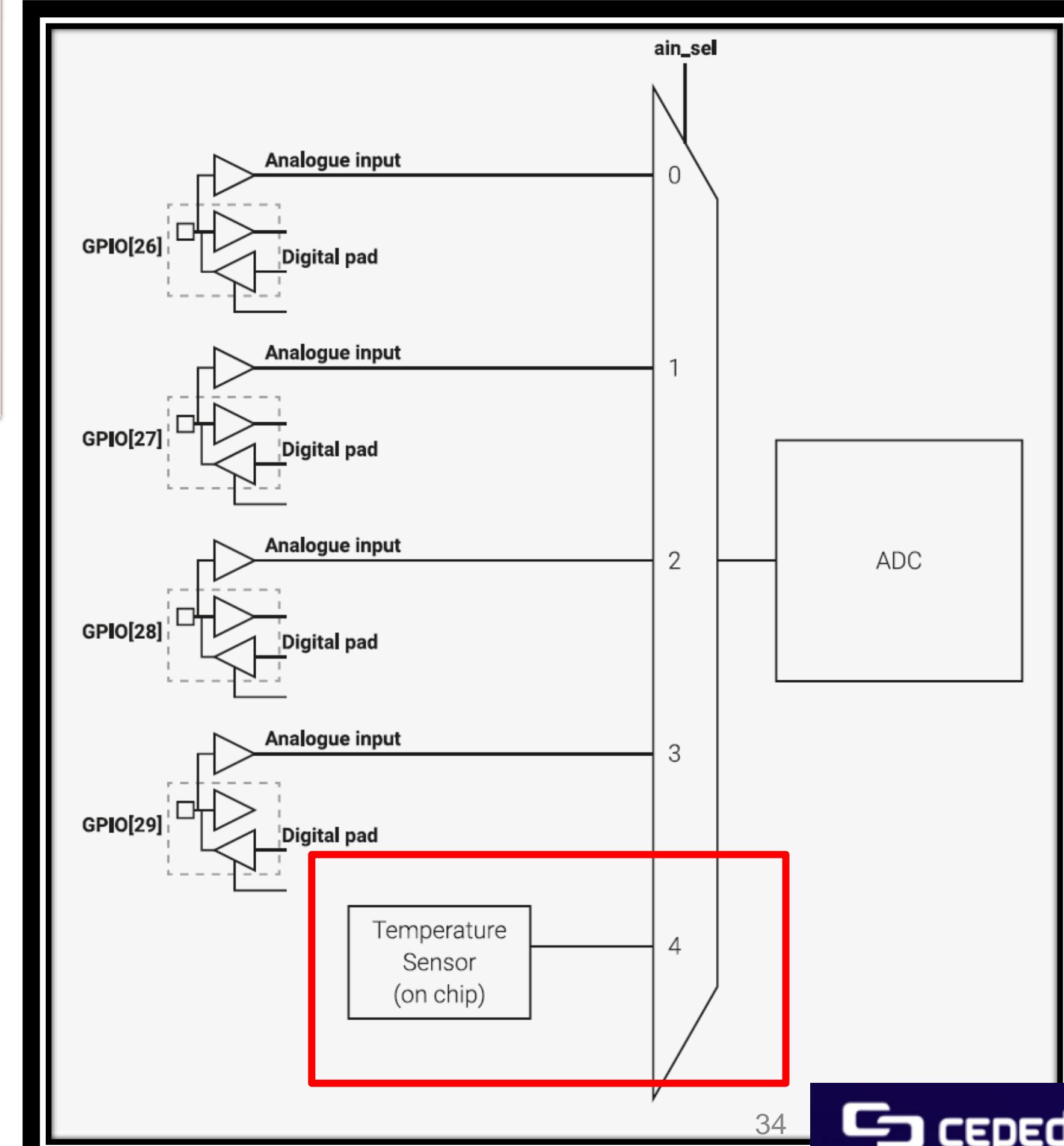
Sensor de Temperatura

- Ele é um canal separado dentro do ADC do RP2040, acessado diretamente por software (**CANAL 4**).
- Não possui um pino externo e sua leitura ocorre via o próprio periférico ADC.
- Ele só pode ser usado para medir a temperatura interna do chip.
- Sua leitura retorna um valor proporcional à temperatura, mas precisa ser convertida para graus Celsius usando a seguinte equação:

$$T = 27 - \frac{(V_{sense} - 0.706)}{0.001721}$$

Onde:

- T é a temperatura em graus Celsius.
- V_{sense} é a tensão lida no ADC3.
- 0.706V é a tensão típica do sensor a 27°C.
- 0.001721 V/°C é o coeficiente de variação da tensão com a temperatura.



Kit de ferramentas

- Multímetro
- Protoboard
- Jumpers
- Resistores, LEDs, LDR, Potenciômetro, etc.
- Ferramentas: Alicate, pinças, ferro de solda, solda, etc.

Tarefa da revisão

Com o objetivo de revisar e consolidar os conhecimentos adquiridos sobre o microcontrolador RP2040 e os principais recursos disponíveis na placa de desenvolvimento BitDogLab, propõe-se a realização de um projeto **prático individual**, cuja concepção será de responsabilidade do aluno.

Observação: Este trabalho é somente para os alunos que estão na trilha de **Sistemas Embarcados**.

- **Leitura analógica** por meio do **potenciômetro do joystick**, utilizando o conversor **ADC** do RP2040;
- **Leitura de botões físicos (push-buttons)** com tratamento de **debounce**, essencial para garantir a confiabilidade das entradas digitais;
- Utilização da **matriz de LEDs**, do **LED RGB** e do **buzzer** como saídas para **feedback visual e sonoro**;
- Exibição de informações em tempo real no **display gráfico 128x64 (SSD1306)**, que se comunica com o RP2040 via **interface I2C**;
- Transmissão de dados e mensagens de depuração através da **interface UART**, permitindo a visualização em um terminal serial no computador;
- Emprego de **interrupções** para o tratamento eficiente de eventos gerados pelos botões;
- Estruturação do projeto no ambiente **VS Code**, previamente configurado para o desenvolvimento com o RP2040.



Obrigado!

Executores:



Coordenação:



Iniciativa:

