



🎓 Desenvolvimento de sensores e atuadores IoT – Parte 3

Aula síncrona (22/07/2025)

Prof. Wilton Lacerda Silva

Executores:



Coordenação:



Iniciativa:



Sumário

- Objetivos
- Entender o sensor inercial IMU
- Armazenamento de dados em memória FLASH
(Cartão SD)
- Exemplos de aplicação

Objetivos

- Conceituar algumas grandezas físicas.
- Projetar e integrar sensores e atuadores específicos.
- Desenvolver alguns exemplos para fixação dos conceitos e realizar atividades práticas com sensores e atuadores.



Conceitos: Aceleração



Conceitos: Aceleração

A aceleração é uma das grandezas mais fundamentais para descrever como um objeto se move. Em termos simples, é a **taxa** na qual a velocidade de um objeto muda ao longo do tempo.

Aceleração Média: É a mudança total na velocidade dividida pelo tempo que levou para essa mudança acontecer.

$$a_{média} = \frac{\Delta v}{\Delta t}$$

Aceleração Instantânea: É a taxa de alteração na velocidade de um objeto em um momento específico no tempo.

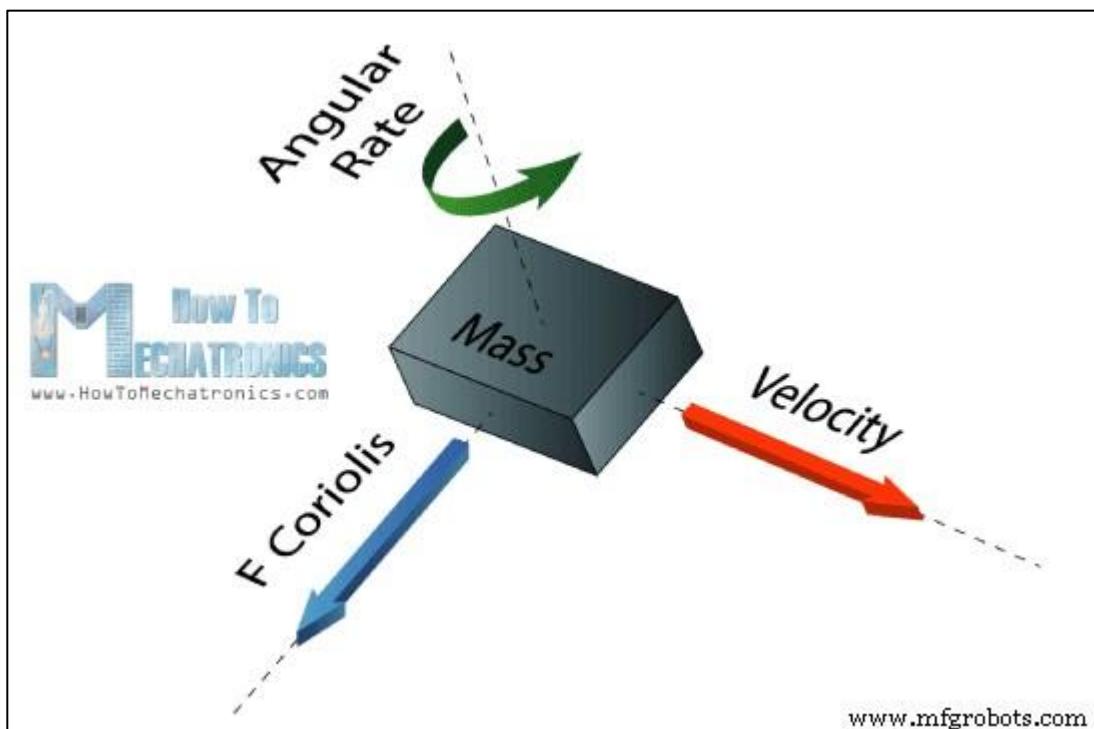
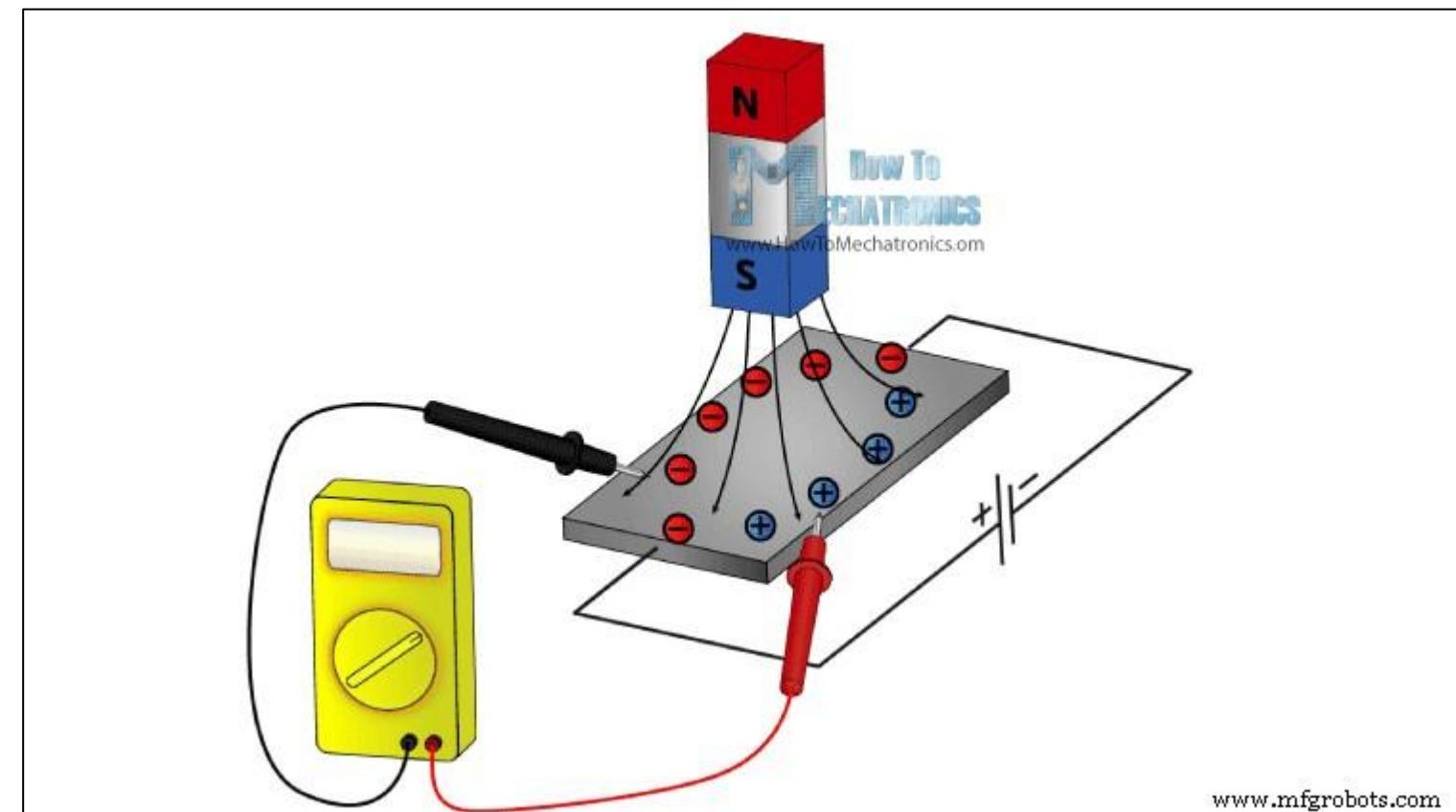
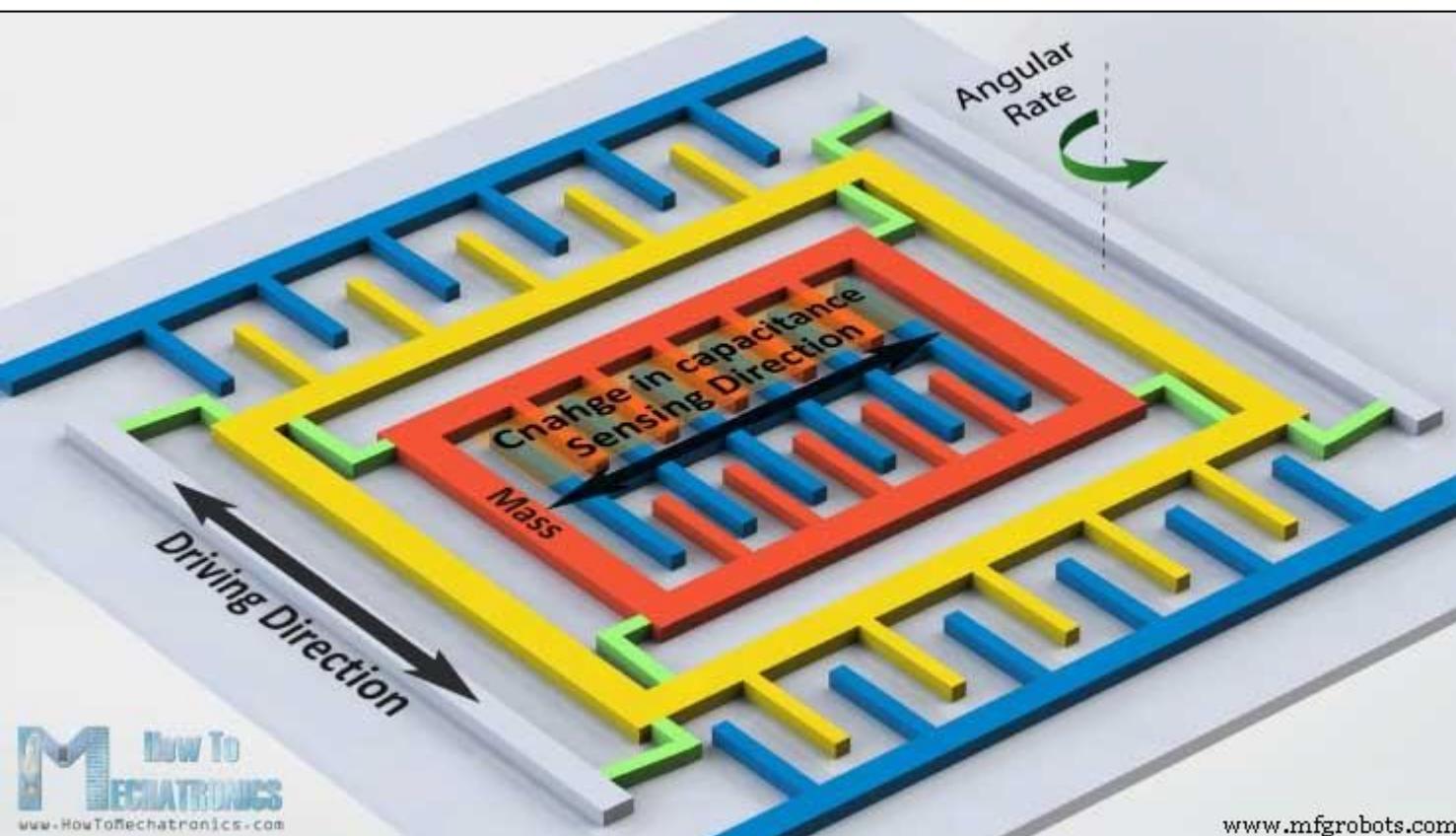
$$a = \lim_{t \rightarrow 0} \left(\frac{\Delta v}{\Delta t} \right) = \frac{dv}{dt}$$

Aceleração da Gravidade: Na superfície da Terra, todos os objetos caem (em um vácuo) com uma aceleração constante devido à gravidade, denotada por 'g'. O valor aproximado é de 9.8m/s²



Conceitos: Sensor inercial.

Os **sensores iniciais**, conhecidos como **Unidade de Medição Inercial (IMU)**, são dispositivos eletrônicos essenciais para medir e relatar o movimento, a orientação e a posição de um objeto. Eles fazem isso combinando dados de diferentes tipos de sensores, principalmente **acelerômetros** e **giroscópios**, e em alguns casos, **magnetômetros**.



Conceitos: Sensor inercial.

Um **IMU** é um componente-chave em sistemas de navegação e rastreamento de movimento. A sigla IMU vem do inglês "Inertial Measurement Unit" (**Unidade de Medição Inercial**).

Basicamente, uma IMU mede:

- **Força específica:** A aceleração linear que um corpo experimenta.
- **Taxa angular:** A velocidade de rotação de um corpo em torno de seus eixos.
- **Campo magnético (em algumas IMUs):** A presença e a direção do campo magnético circundante, o que ajuda na determinação da orientação em relação ao norte magnético.

Essas medições são fundamentais para entender como um objeto está se movendo no espaço tridimensional.

O que mede?

Aceleração linear (movimento de translação) ao longo de três eixos (X, Y, Z). Ele detecta a força específica que age sobre o sensor, incluindo a gravidade.

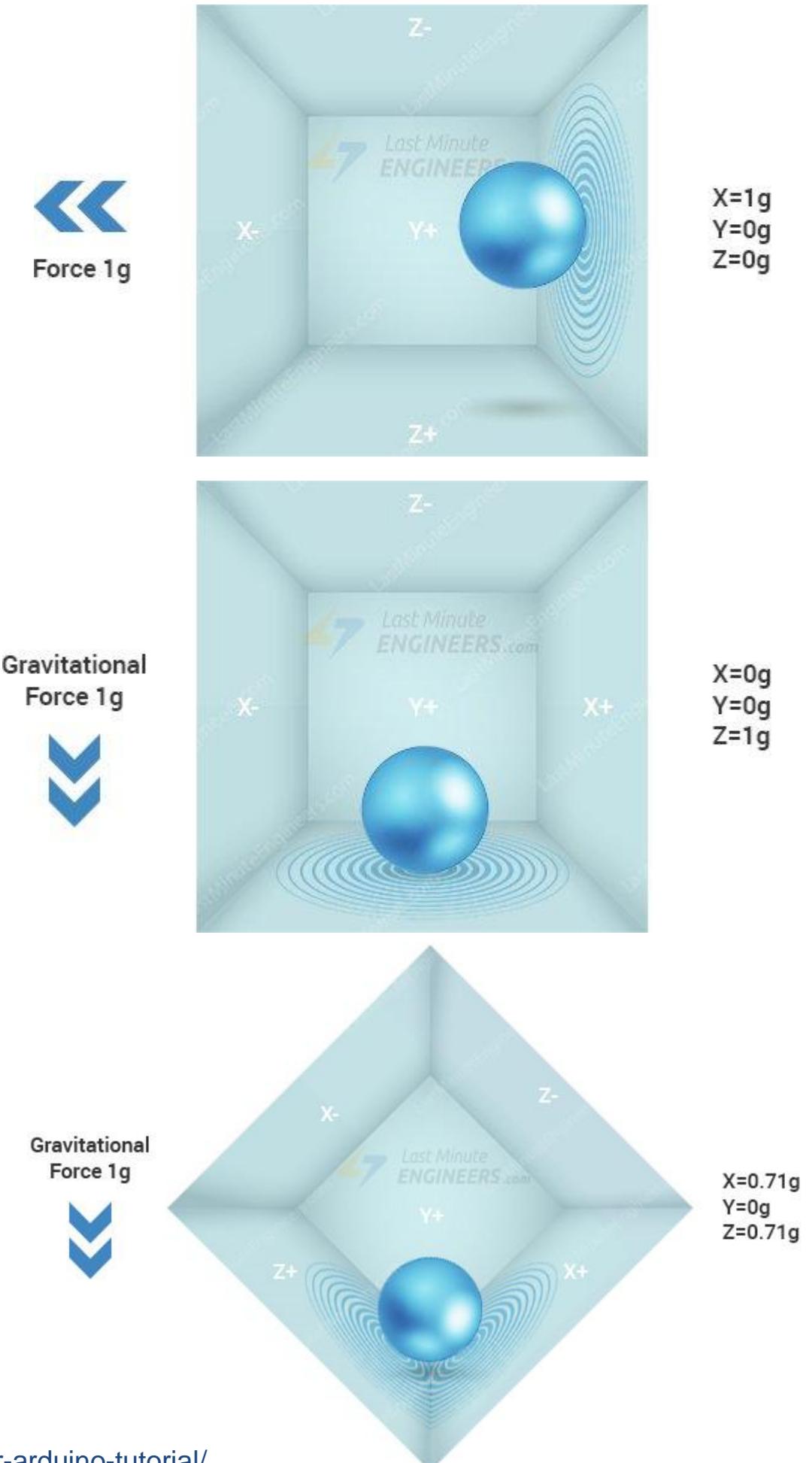
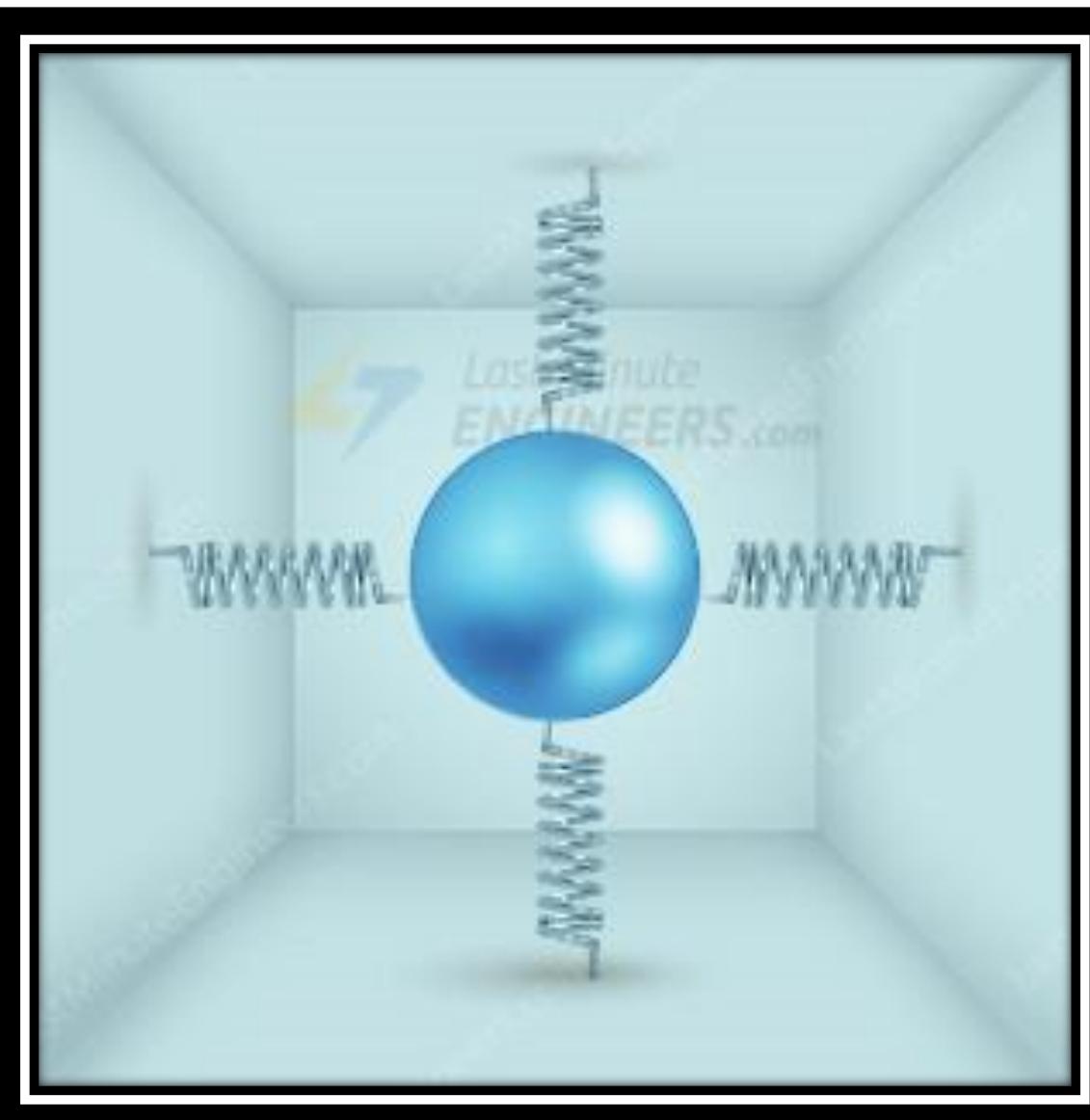
Como funciona?

Geralmente baseado em tecnologia **MEMS** (Sistemas Micro eletromecânicos), ele contém microestruturas que se deslocam quando submetidas a uma força (aceleração). A mudança na capacidade devido a esse deslocamento é convertida em um sinal elétrico.

Qual utilidade?

Pode ser usado para determinar a inclinação estática de um objeto (Pitch e Roll) em relação à gravidade e para detectar vibrações ou movimentos bruscos. No entanto, sua precisão é afetada por acelerações dinâmicas.

Acelerômetro: Princípio de funcionamento



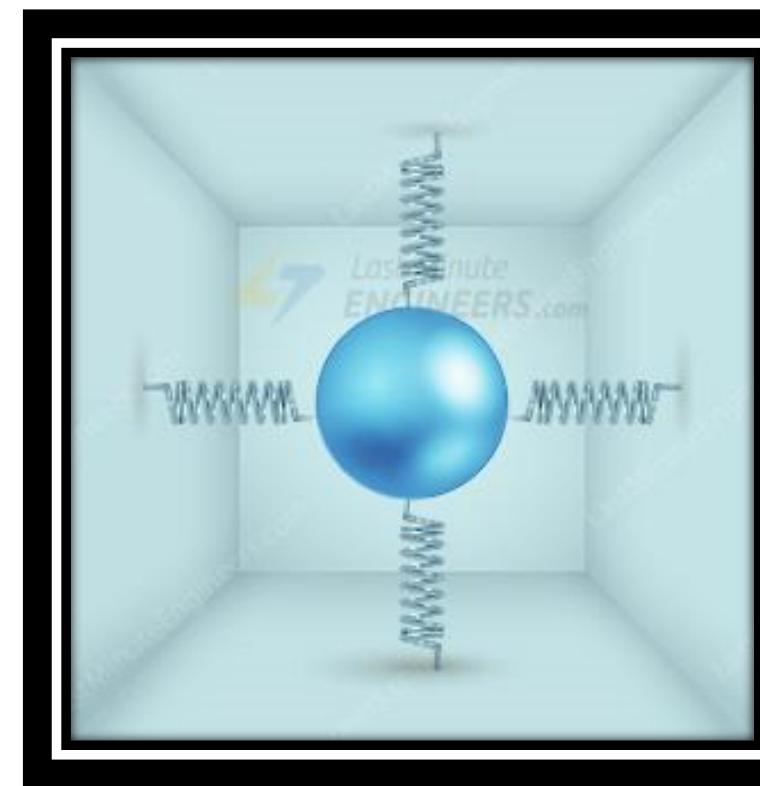
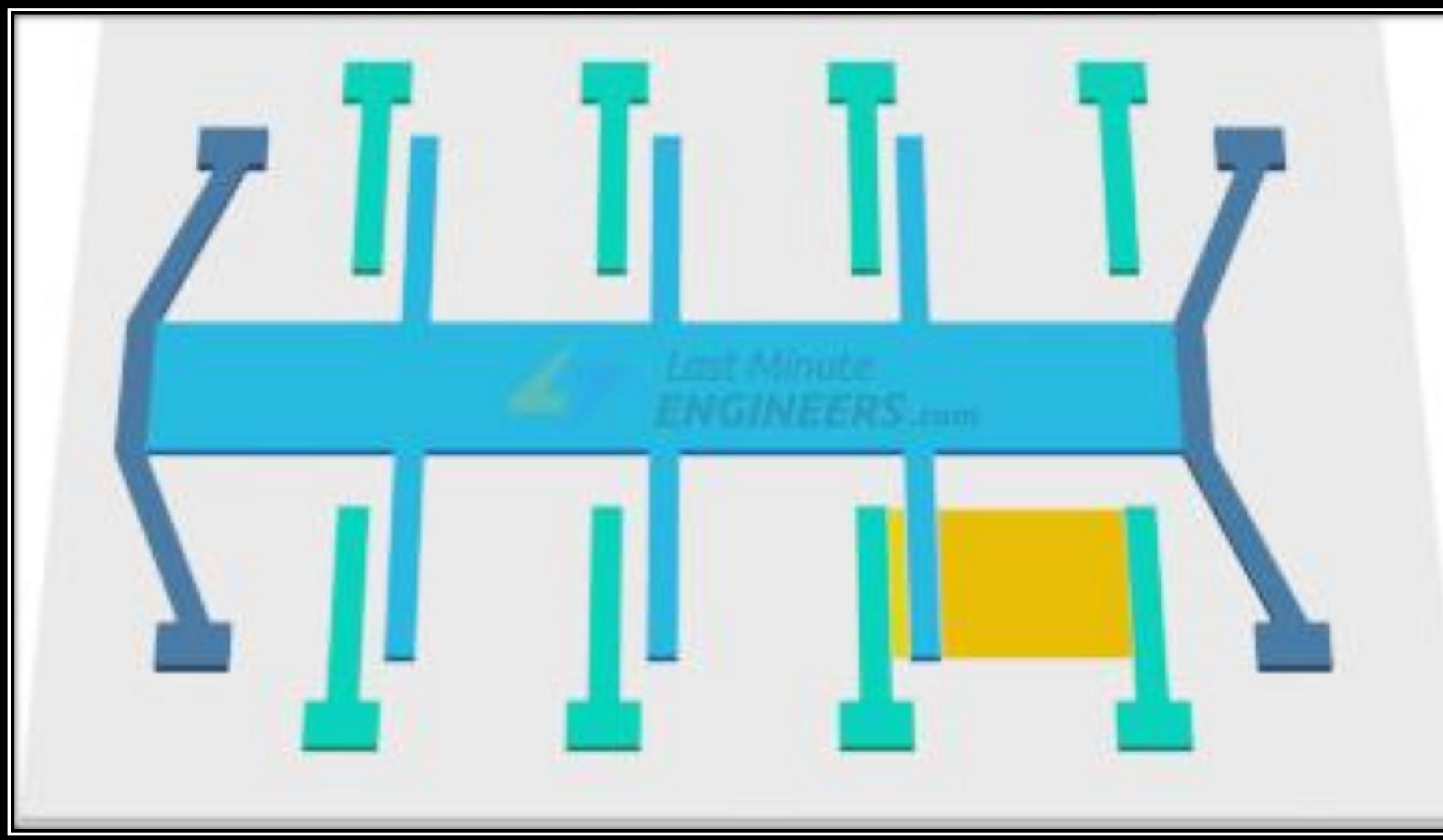
Funcionamento de **acelerômetros** usando a analogia de uma **bola com molas** para ilustrar como as forças de aceleração e gravidade são medidas.

Inércia: Quando o cubo se move (acelera) em uma direção, a bola, por inércia, tende a ficar para trás, esticando as molas no sentido oposto ao movimento. Ao medir a tensão nessas molas, é possível determinar a direção e a intensidade da aceleração.

Parado em uma mesa: O sensor registrará 1g no eixo Z (para baixo) devido à atração da gravidade, e 0g nos eixos X e Y.

Ao inclinar o dispositivo a 45 graus, a força se divide entre os eixos Z e X, resultando em uma leitura de aproximadamente 0.71g em cada um.

Acelerômetro: Tecnologia MEMS



Um acelerômetro **MEMS** (Sistema Microeletromecânico) é uma estrutura mecânica microscópica construída diretamente em um chip de silício.

No seu centro, há uma pequena "massa de prova" suspensa por molas flexíveis. Essa massa, junto com placas fixas ao redor, forma um **capacitor**.

Quando o dispositivo acelera, a massa de prova se move por **inércia**, alterando a **distância entre ela e as placas fixas**.

Como a capacitância depende da distância entre as placas, seu valor muda. Um circuito no chip detecta essa mudança na capacitância, converte-a em um sinal elétrico e **calcula o valor exato da aceleração**.

Giroscópio

O que mede?

Taxa angular (velocidade de rotação ou "giro") ao redor de três eixos (Pitch, Roll, Yaw).

Como funciona?

Também tipicamente MEMS, usa um ressonador vibratório. Quando o sensor gira, o **efeito Coriolis** causa uma deflexão na vibração, que é medida e convertida em uma taxa angular (geralmente em graus por segundo - dps).

Qual a utilidade?

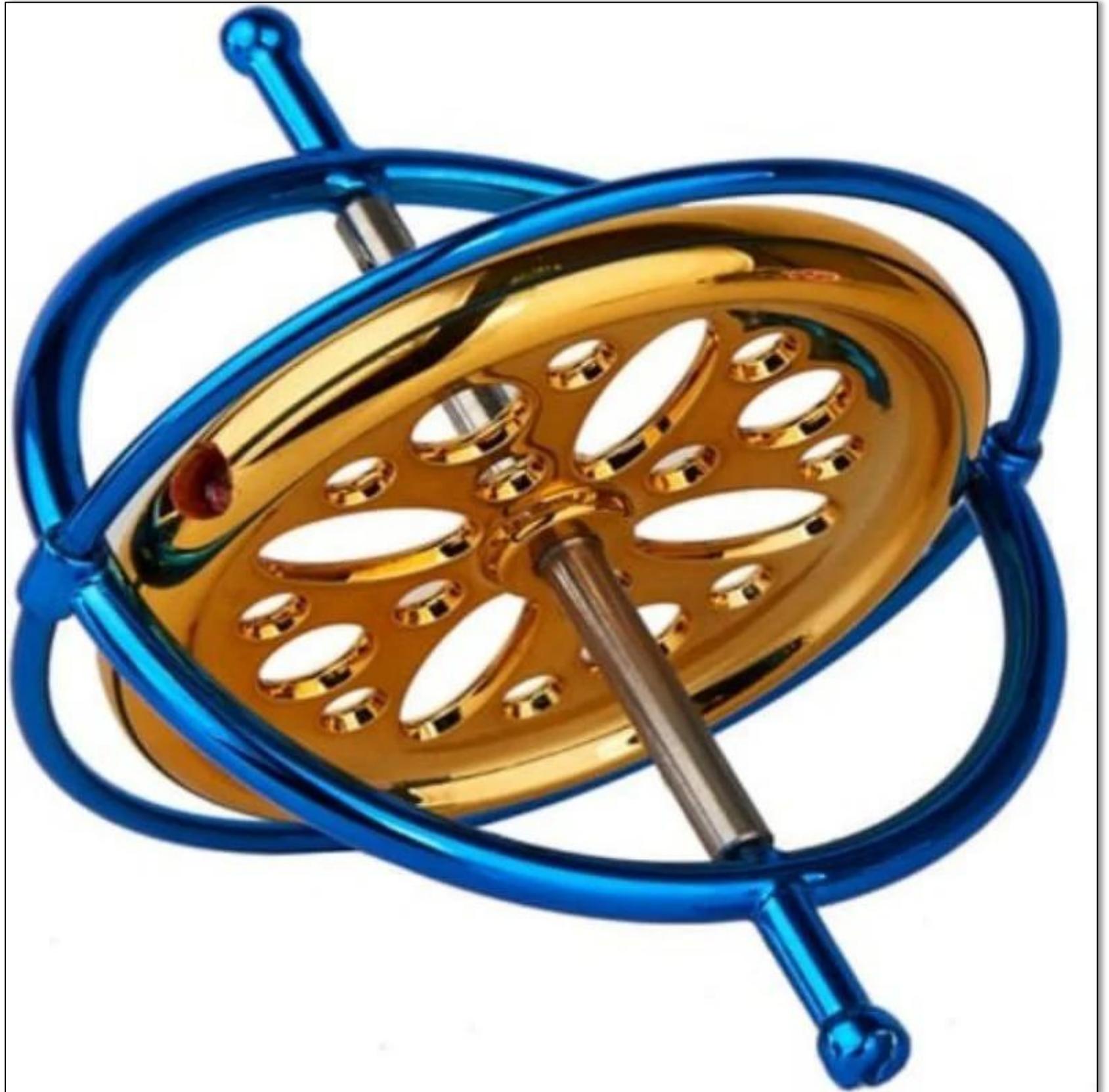
Fundamental para rastrear a orientação em movimentos dinâmicos, complementando o acelerômetro. Ele pode medir a rotação independentemente da gravidade.



Giroscópio: Medição

Assim como no acelerômetro, existem placas de capacitor posicionadas para detectar esse movimento lateral induzido pelo **Efeito de Coriolis**.

O circuito mede a mudança na capacitância causada por esse movimento e a converte em um sinal elétrico que representa a velocidade angular (ex: 150 graus por segundo).



Fusão dos sensores.

A combinação dos dados desses sensores é onde a **IMU** realmente brilha. Sozinho, um **acelerômetro** pode ser enganado pela aceleração do movimento (não apenas pela gravidade), e um **giroscópio** acumula erros ao longo do tempo. A **fusão de sensores**, geralmente através de algoritmos como Filtro de Kalman ou Filtro Complementar, combina os pontos fortes de cada sensor para fornecer uma estimativa mais precisa e estável da orientação e movimento.

Portanto, para ter uma noção completa do movimento de um objeto (como seu smartphone ou um drone), precisamos de ambos os sensores trabalhando juntos em um **IMU**.

- O **Acelerômetro** é o especialista em "**movimento em linha reta**". Ele sente empurrões, puxões e a força constante da gravidade.
- O **Giroscópio** é o especialista em "**giros e rotações**". Ele sente o quanto rápido o objeto está virando, independentemente de estar se movendo em linha reta ou não.

Aplicações dos Sensores IMU

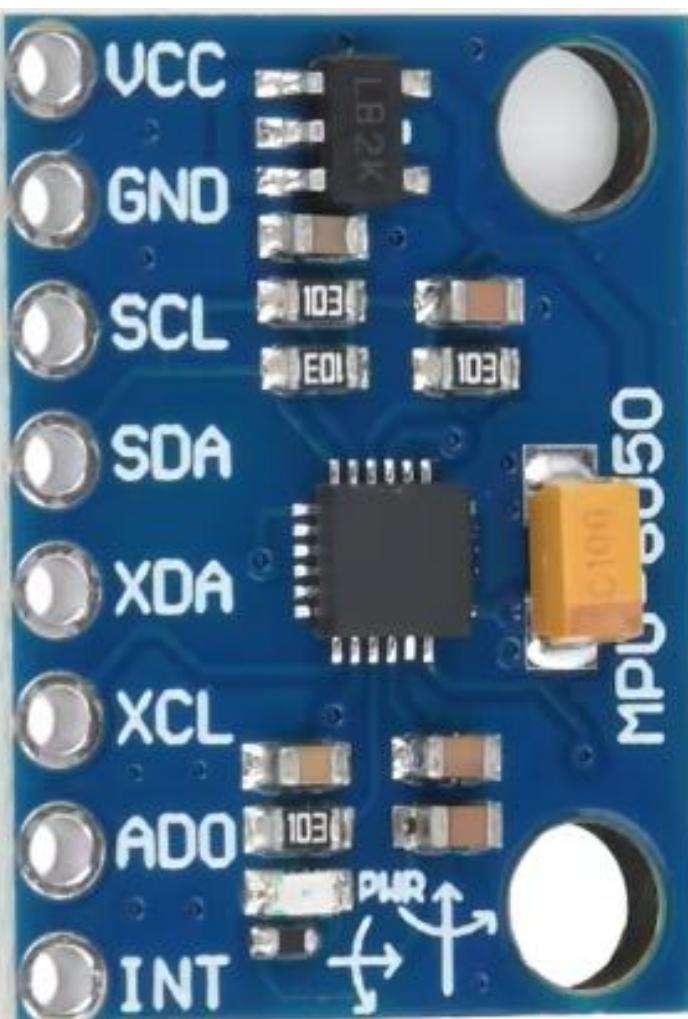
- **Eletrônicos de Consumo:** Smartphones, tablets, smartwatches e controladores de jogos usam IMUs para detecção de orientação da tela, navegação por gestos, rastreamento de fitness e jogos de realidade virtual/aumentada.
- **Robótica:** Essenciais para a navegação autônoma de robôs, drones (UAVs), veículos autônomos e estabilização de plataformas.
- **Sistemas de Navegação:** Usadas em sistemas de navegação inercial para aeronaves, mísseis, navios e veículos terrestres, muitas vezes em conjunto com GPS para maior precisão.
- **Esportes e Fitness:** Para análise de movimento em esportes, rastreadores de atividade e reabilitação.
- **Realidade Virtual e Aumentada:** Para rastrear os movimentos da cabeça e do corpo do usuário, proporcionando imersão.
- **Estabilização:** Em câmeras (gimbals), drones e outros equipamentos que precisam manter uma orientação estável.
- **Medição Industrial:** Para monitoramento de inclinação e aceleração em máquinas e equipamentos pesados, compensando movimentos dinâmicos.

Aplicações dos Sensores IMU – MPU6050

O **MPU6050** é um dos módulos IMU mais populares e amplamente utilizados, especialmente em projetos de hobby e prototipagem. Ele é um sensor de **6 Graus de Liberdade (6DOF)**, o que significa que ele integra:

- **Acelerômetro de 3 eixos:** Mede aceleração linear nos eixos X, Y e Z.
- **Giroscópio de 3 eixos:** Mede a taxa angular (velocidade de rotação) nos eixos X, Y e Z.
- **Sensor de Temperatura:** Embora não seja um sensor primário de movimento, o sensor de temperatura é importante para a compensação de temperatura das leituras do acelerômetro e giroscópio, pois as características desses sensores podem variar com a temperatura.

O MPU6050 se comunica via protocolo I²C, tornando-o fácil de integrar com microcontroladores como o Raspberry Pi Pico. Sua combinação de recursos e custo-benefício o torna uma escolha excelente para uma vasta gama de projetos que exigem detecção de movimento e orientação.

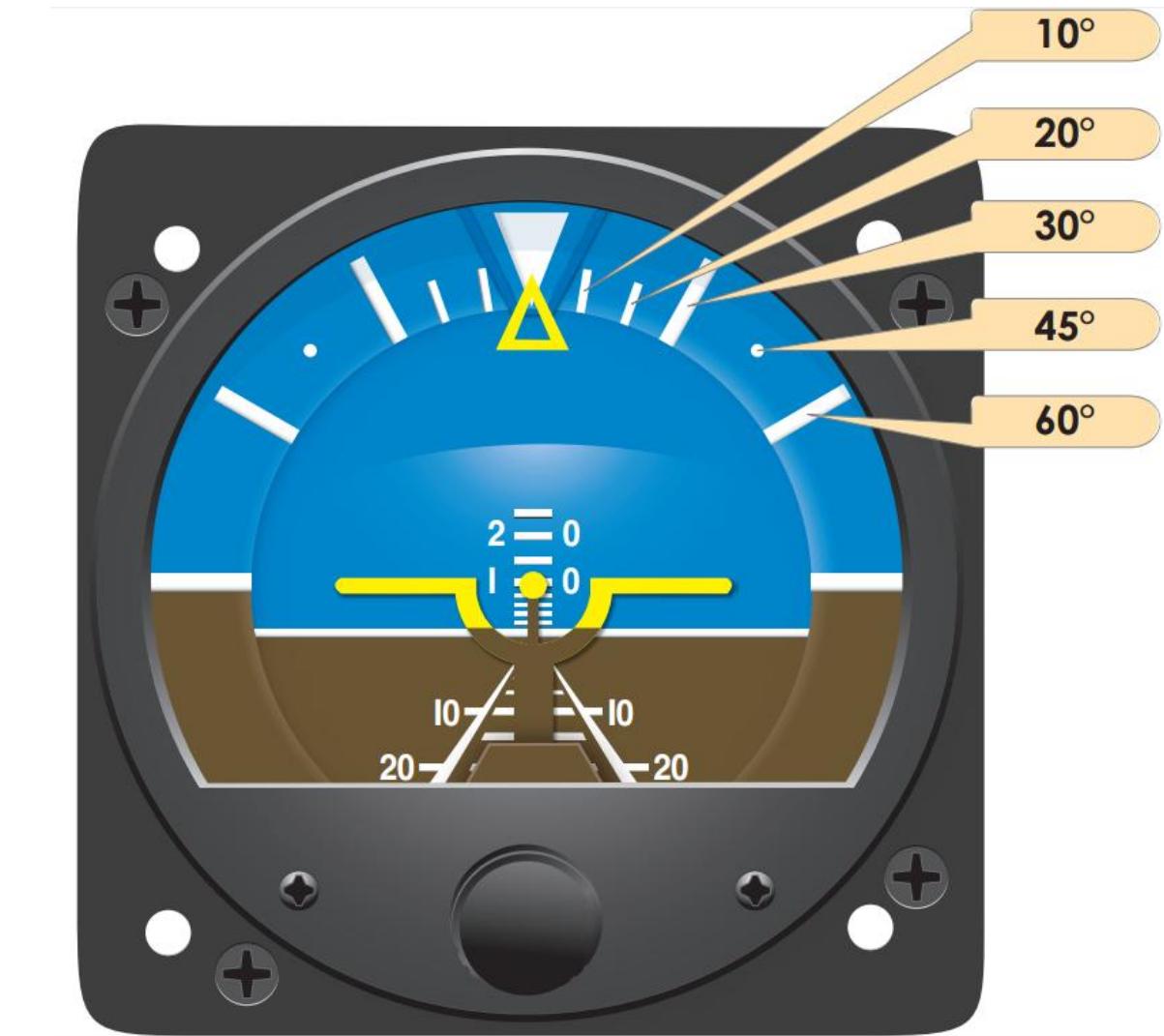
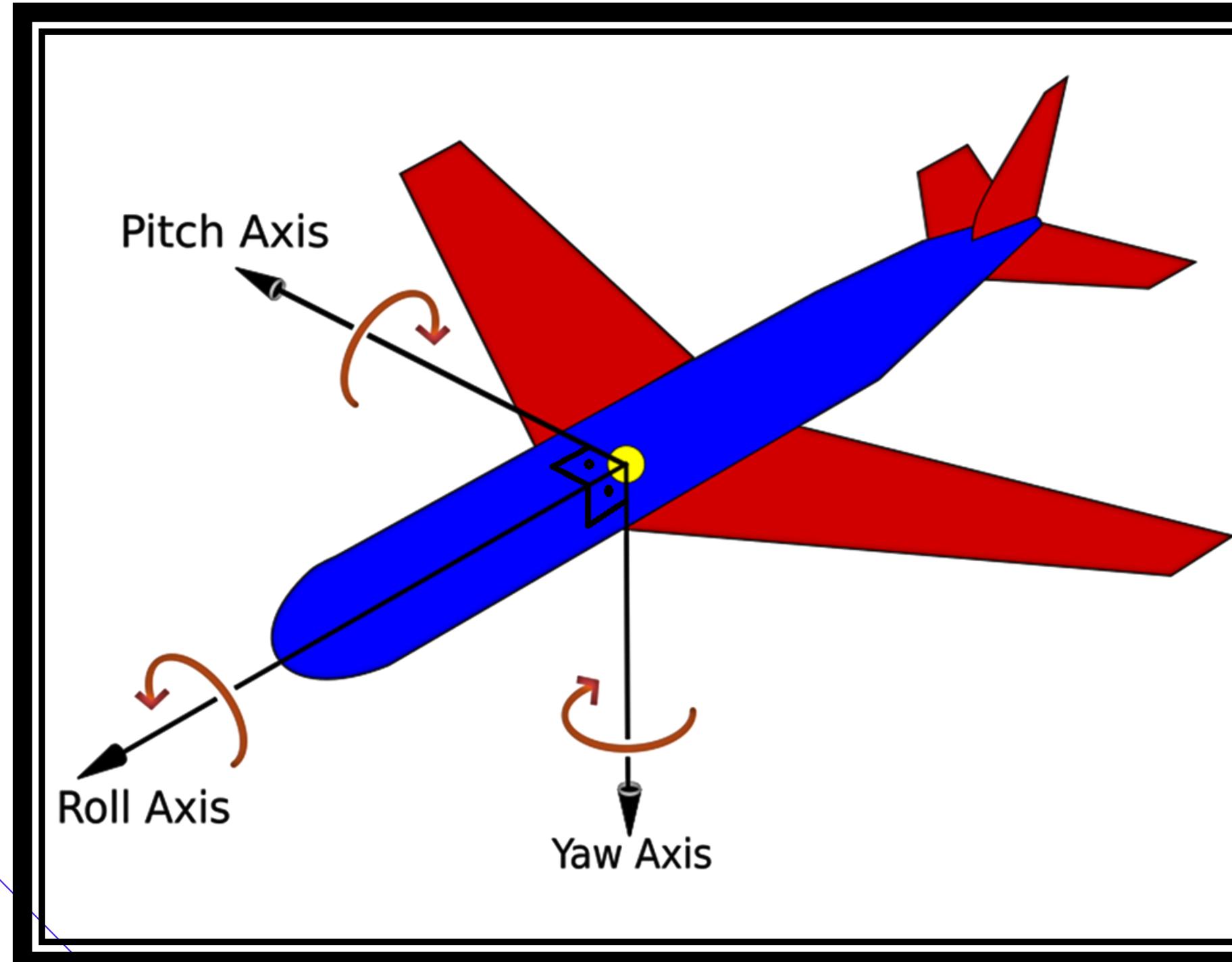


Aplicações dos Sensores IMU – MPU6050

Pitch: Eixo Lateral (de uma ponta de asa à outra). É o movimento do nariz do avião para cima ou para baixo.

Roll: Eixo Longitudinal (do nariz até a cauda). É a inclinação lateral do avião, onde uma asa sobe e a outra desce. Este é o principal movimento para iniciar uma curva.

Yaw: Eixo Vertical (atravessa o centro do avião de cima para baixo). É o movimento do nariz do avião para a esquerda ou para a direita, sem inclinar as asas. É usado para ajustar a direção e para coordenar as curvas, evitando derrapagens.



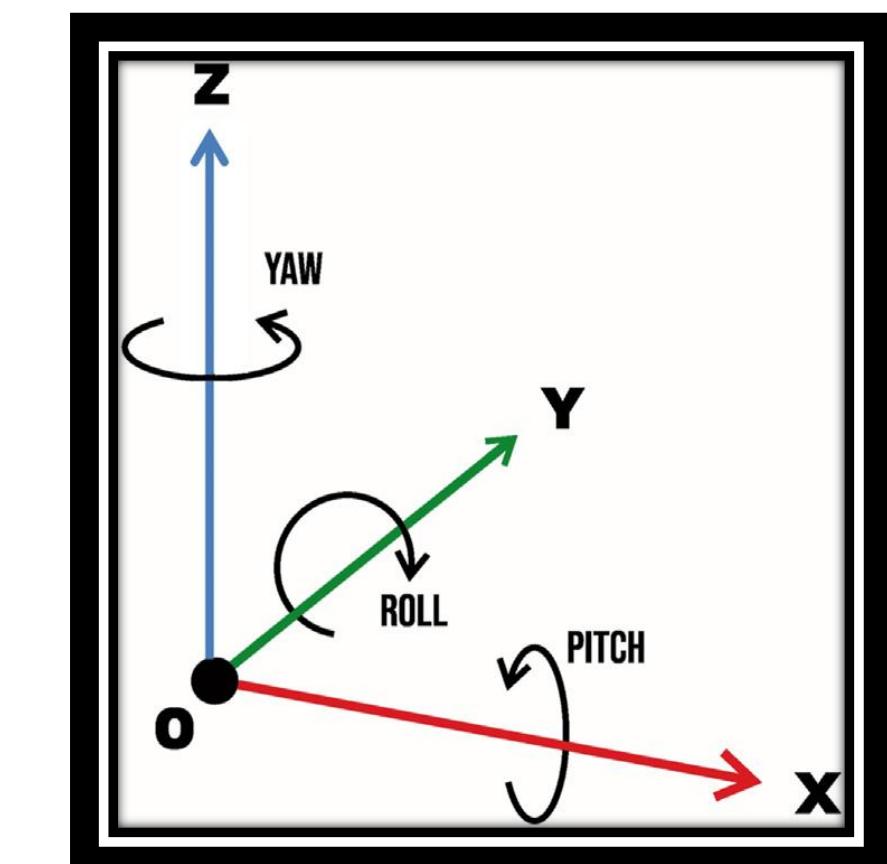
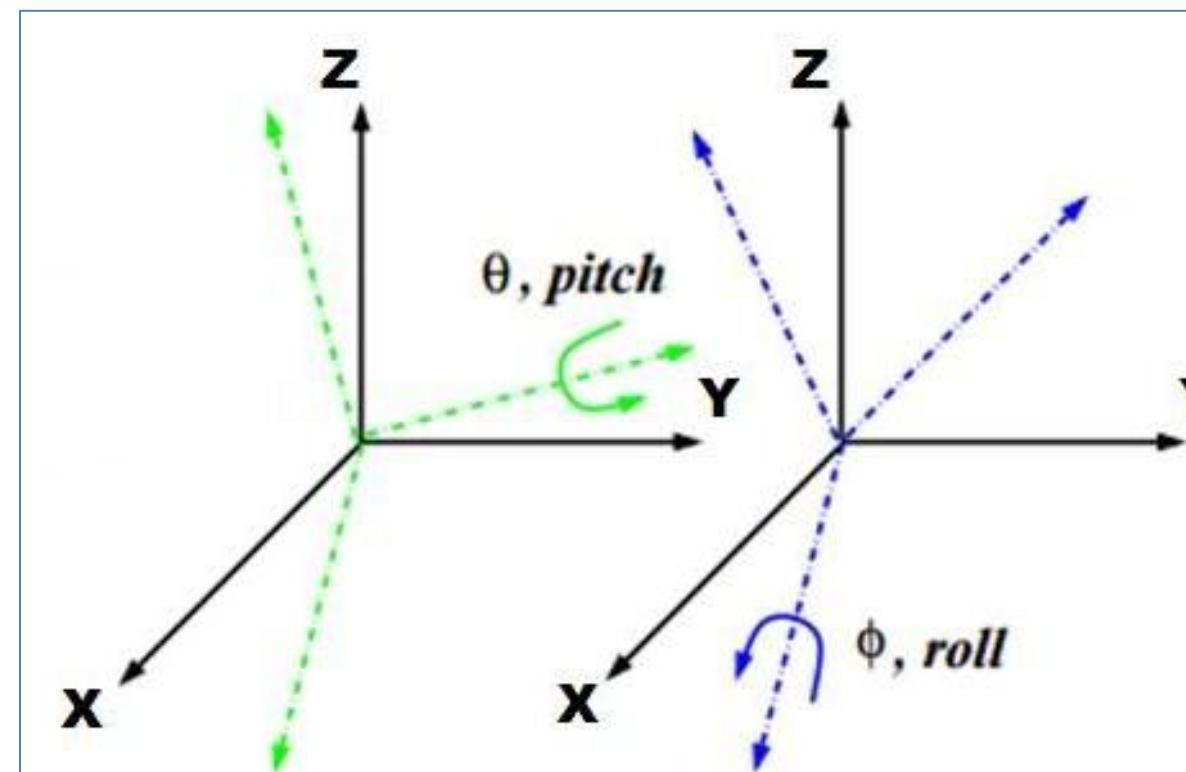
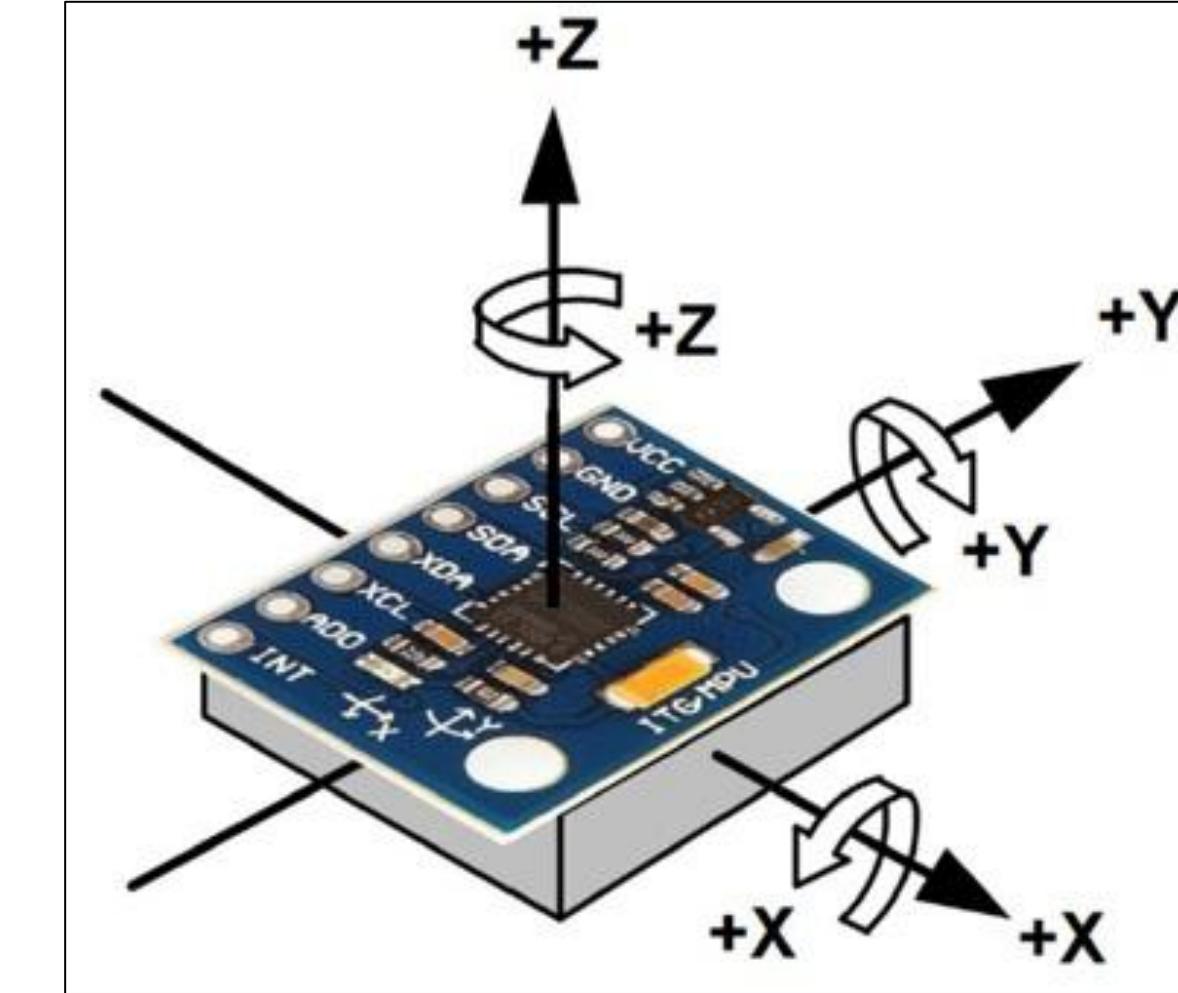
Aplicações dos Sensores IMU – MPU6050

Cálculo do ângulo do **Pitch**:

$$\theta = \arctan\left(\frac{A_y}{A_z}\right)$$

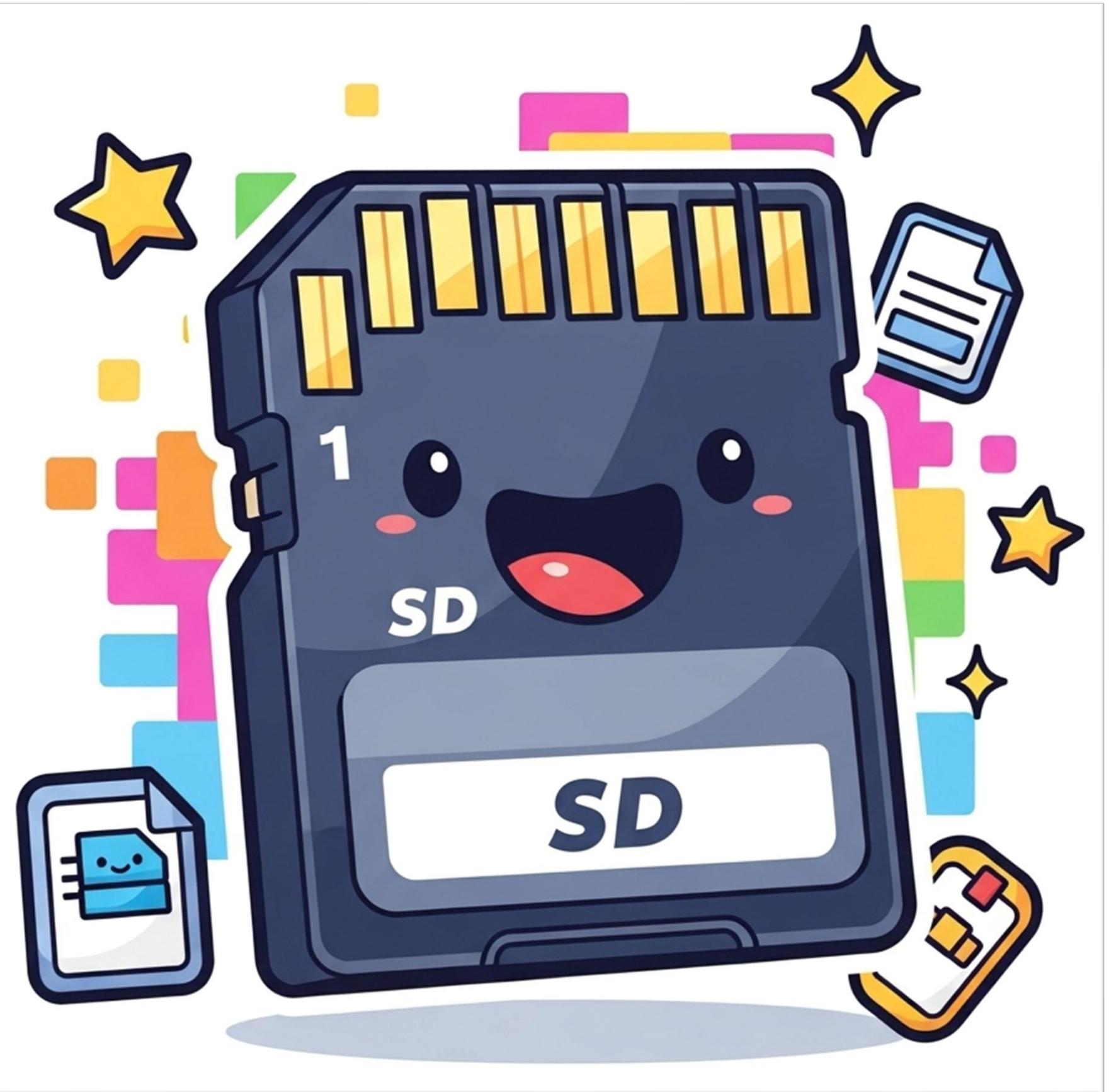
Cálculo do ângulo do **Roll**:

$$\phi = \arctan\left(\frac{A_x}{\sqrt{A_y^2 + A_z^2}}\right)$$



Baseado na **tecnologia de memória flash**, o **Cartão SD** (Secure Digital) é um formato de cartão de memória removível padronizado pela SD Association.

Ele foi projetado para fornecer uma solução de armazenamento de dados em um formato compacto e portátil, tornando-o ideal para uma vasta gama de dispositivos eletrônicos.



Comunicação com Cartões SD em Baixo Nível via Modo SPI

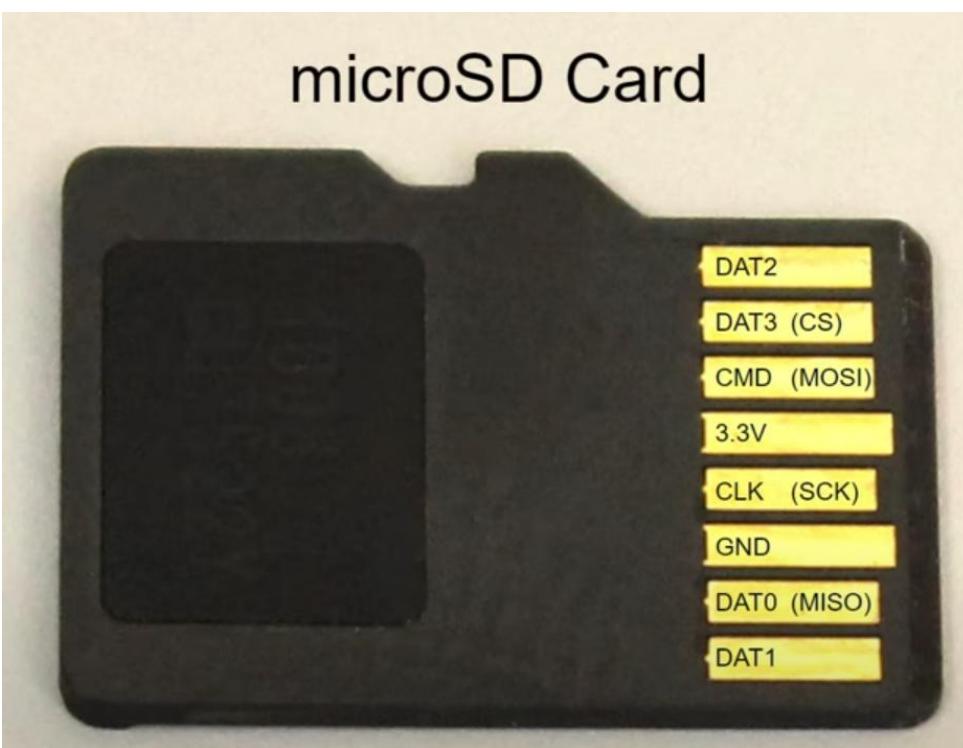
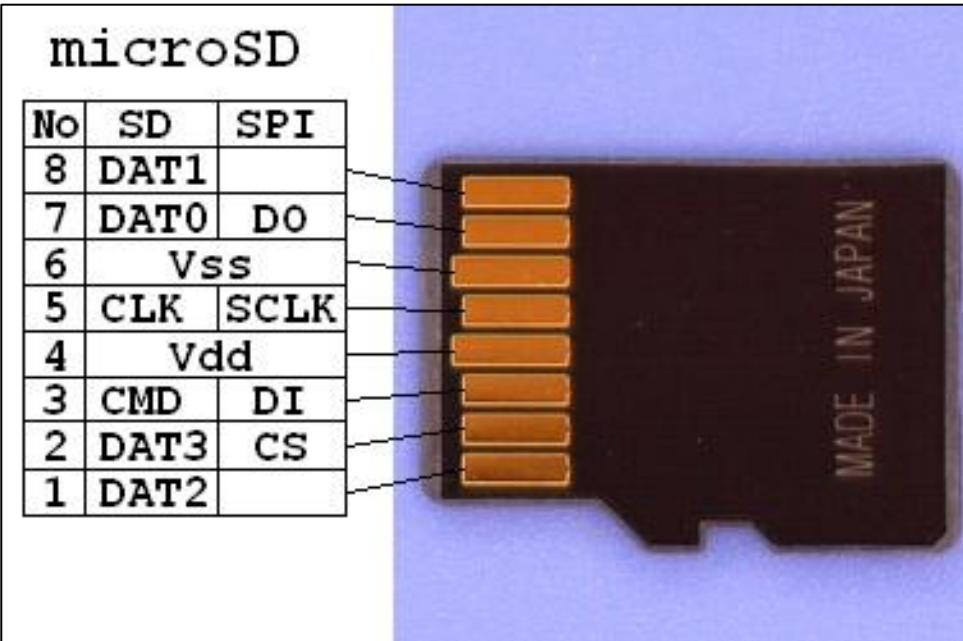


A página ([elm-chan.org](https://elm-chan.org/docs/mmc/mmc_e.html)) é uma referência para entender o controle de baixo nível de cartões SD/MMC, especialmente para MCUs.

Objetivo: Entender o protocolo de comunicação para ler e escrever blocos de dados brutos em um cartão SD.

Protocolo: A comunicação nativa do SD é um barramento de 4 bits, mas eles oferecem um modo alternativo mais simples e universal: **SPI (Serial Peripheral Interface)**, que é ideal para microcontroladores.

Modelo de Comunicação: O microcontrolador (MCU) atua como Mestre (Master) e o cartão SD como Escravo (Slave).

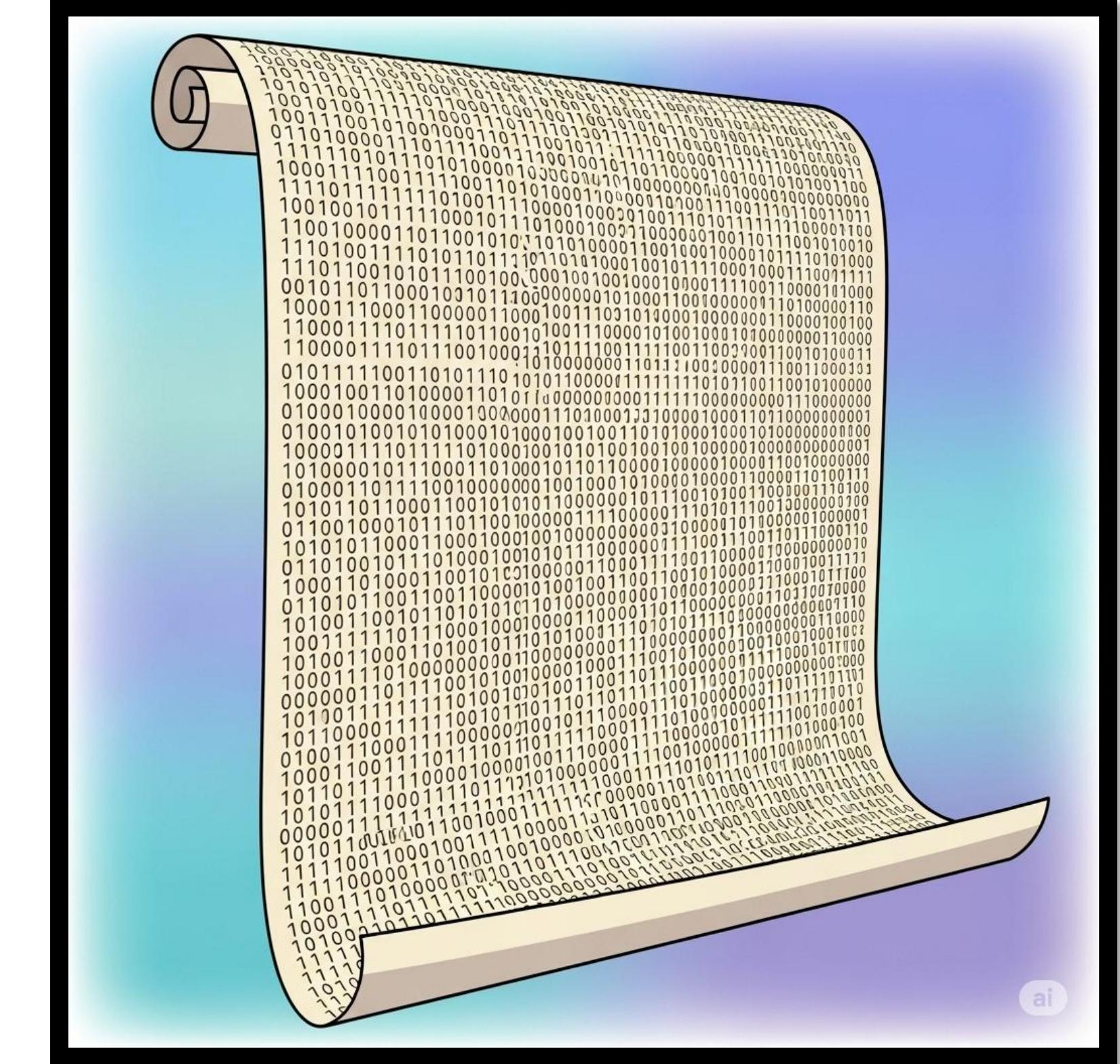


Sistemas de arquivos (Armazenamento de dados)

Um **sistema de arquivos** é um conjunto de regras e estruturas de dados que, por exemplo, um sistema operacional usa para controlar como os dados são salvos e recuperados. Ele traduz a nossa noção de "**arquivos**" e "**pastas**" em operações de baixo nível, dizendo exatamente em que setor do disco físico cada pedaço de um arquivo está.

Quando uma unidade (como um **cartão SD**) é formatada com FAT, ela é dividida em **três áreas principais e lógicas**:

- 1. Setor de Boot (Boot Sector)**
- 2. Tabela de Alocação de Arquivos**
- 3. Área de Dados**



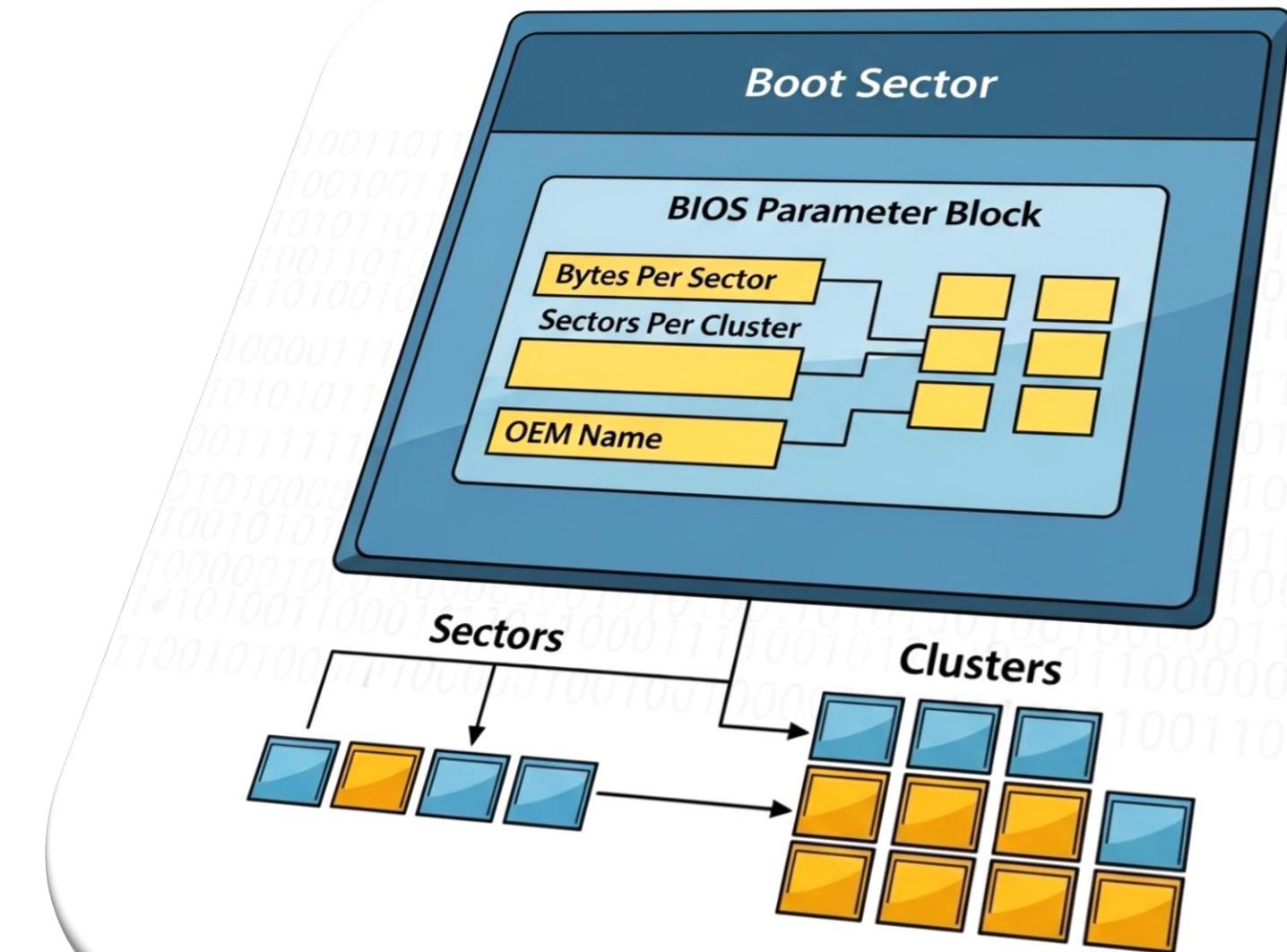
1. Setor de Boot (Boot Sector)

É o primeiro setor da partição. Pense nele como o "mapa do mapa". Ele contém informações vitais sobre a estrutura do volume, conhecidas como **BPB (BIOS Parameter Block)**.

Informações no BPB:

- Tamanho do setor (geralmente 512 bytes).
- Número de setores por **cluster**.
- Número de cópias da tabela FAT (geralmente 2, por segurança).
- Tamanho total do volume.
- Versão do FAT (FAT12, FAT16 ou FAT32).

Função: Sem o Setor de Boot, o sistema operacional não saberia como ler o restante do disco. Ele não saberia o tamanho de um cluster, onde a Tabela FAT começa ou quantos setores existem.



Sistemas de arquivos (Armazenamento de dados)

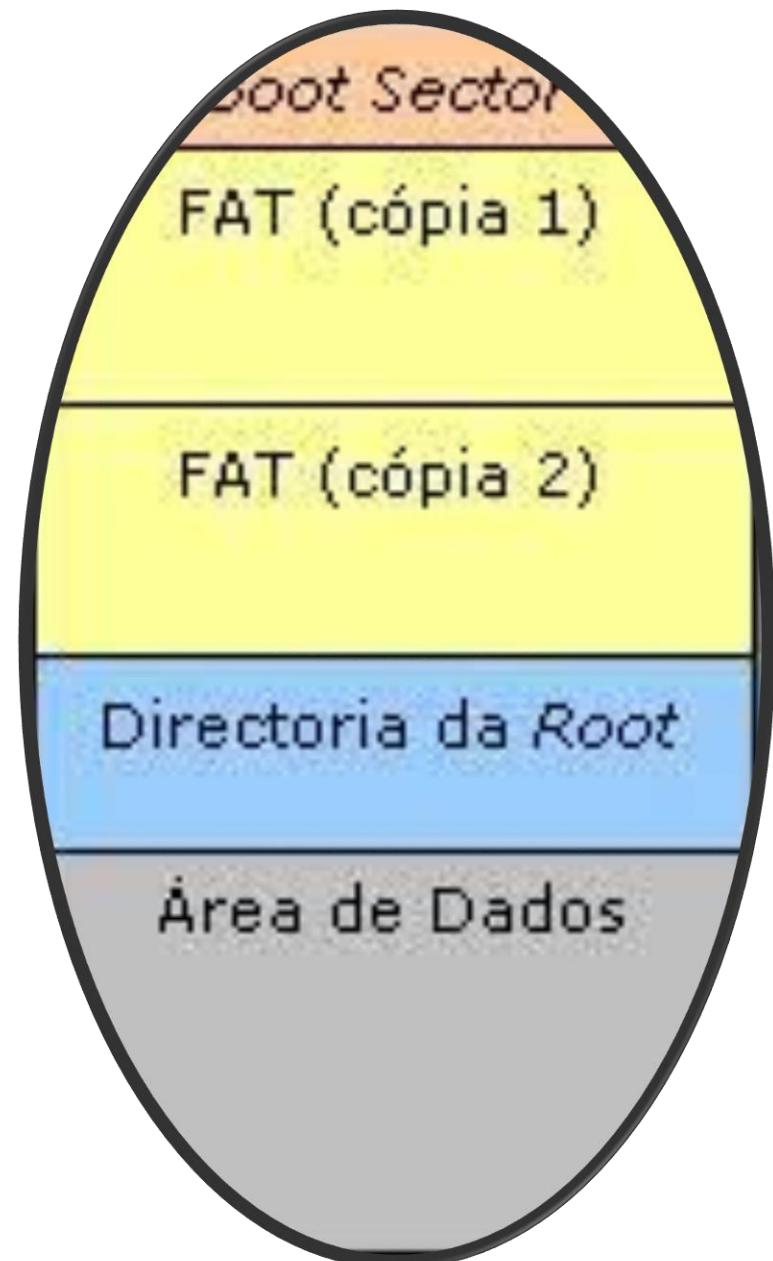
2. Tabela de Alocação de Arquivos (A Própria FAT)

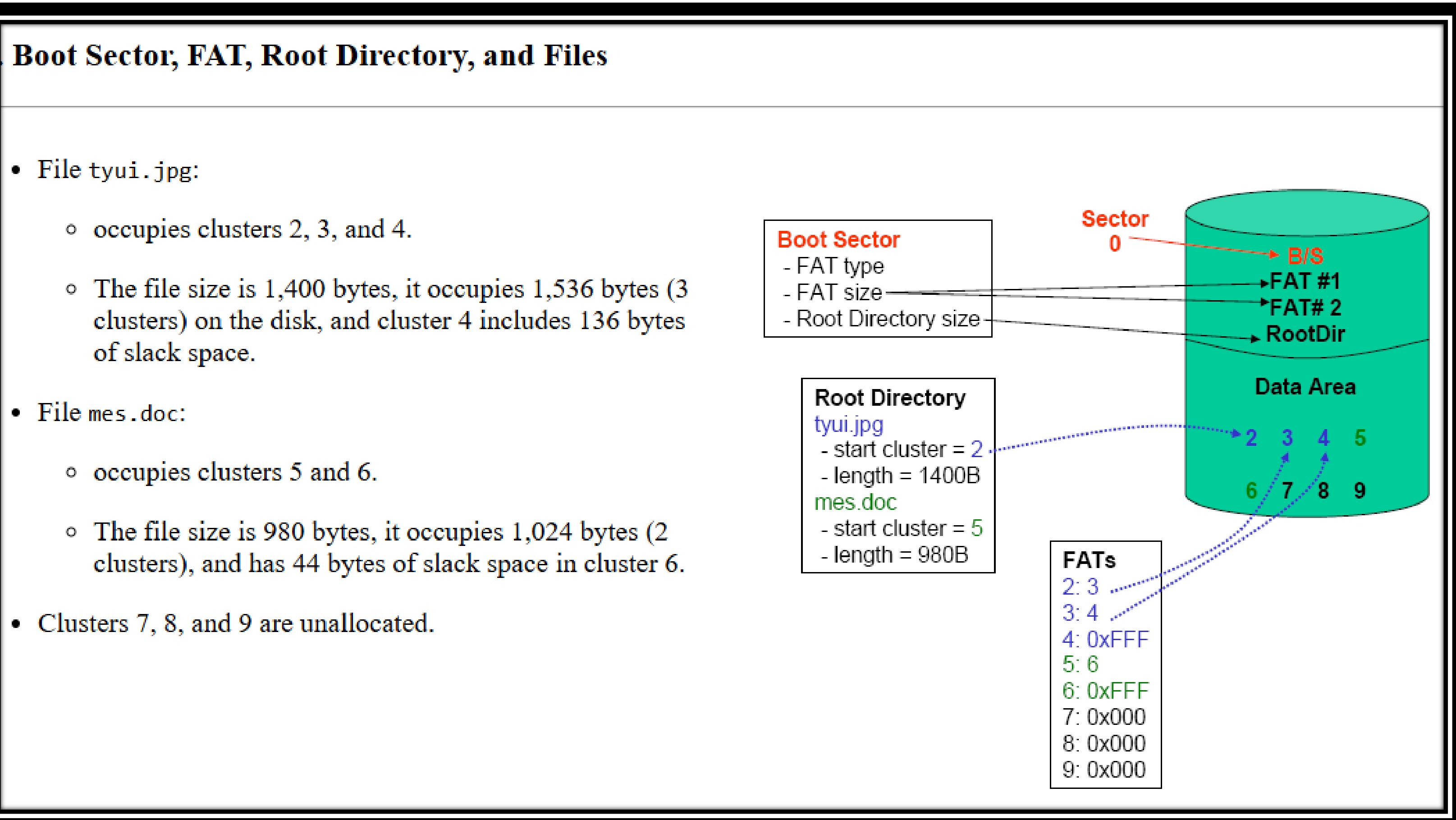
Esta é a parte mais importante e que dá nome ao sistema. A FAT é um grande "índice" ou "livro de registros" que rastreia o estado de cada parte do disco.

• **Clusters:** A área de dados do disco é dividida em blocos de tamanho fixo chamados **clusters**. Um cluster é a menor unidade de espaço que pode ser alocada a um arquivo. Ele pode ter 4KB, 8KB, 16KB, etc., dependendo do tamanho da partição.

• Como a FAT funciona:

1. A tabela FAT possui uma entrada para cada cluster no disco.
2. Quando um arquivo é criado, o sistema de arquivos encontra o primeiro cluster livre e grava os dados lá.
3. A entrada do arquivo no diretório aponta para o número deste primeiro cluster.
4. Se o arquivo for maior que um cluster, o sistema encontra o próximo cluster livre. Na entrada da tabela FAT correspondente ao *primeiro* cluster, ele escreve o número do *segundo* cluster. Na entrada do *segundo*, ele escreve o número do *terceiro*, e assim por diante.
5. Isso cria uma **lista encadeada**. Para ler um arquivo, o sistema começa no primeiro cluster e segue a "trilha de migalhas de pão" através da tabela FAT até encontrar o fim.





Sistemas de arquivos: Comparação entre FATs

Característica	FAT12	FAT16	FAT32
Ano de Lançamento	1980	1984	1996
Bits de Endereçamento	12-bit	16-bit	32-bit (28-bit efetivos)
Tamanho Máx. da Partição	~32 MB	2 GB (típico)	2 TB (teórico), 32 GB (limite no Windows)
Tamanho Máx. do Arquivo	~32 MB	2 GB	4 GB
Uso Principal	Disquetes	Primeiros discos rígidos, MS-DOS	Pendrives, cartões de memória, discos externos
Compatibilidade	Universal (obsoleto)	Alta	Universal (padrão de fato para mídias removíveis)

Acessar Cartão SD no SDK

O SDK do RP2040 **não tem suporte nativo para SD**, mas podemos usar a FatFs, uma biblioteca para manipulação de sistemas de arquivos FAT em cartões SD para dispositivos embarcados.

Para isso, precisamos:

- Configurar a SPI no RP2040
- Usar a FatFs para leitura e escrita no SD

Podemos adicionar suporte ao sistema de arquivos FAT usando a biblioteca **FatFs de ChaN**. O fluxo geral do código será:

1. Inicializar a SPI e configurar o pino CS.
2. Implementar a interface de baixo nível para comunicação SPI com o cartão SD.
3. Integrar a **FatFs** para leitura e escrita de arquivos.

A biblioteca FatFs pode ser baixada de:

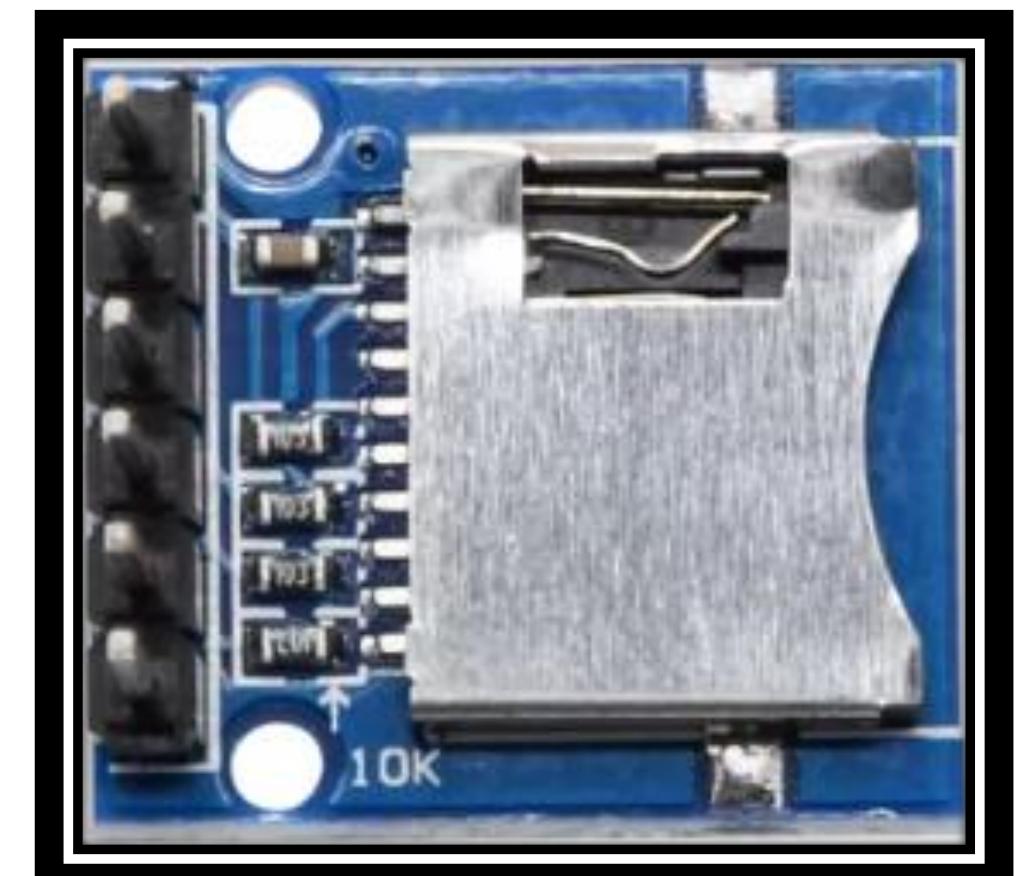
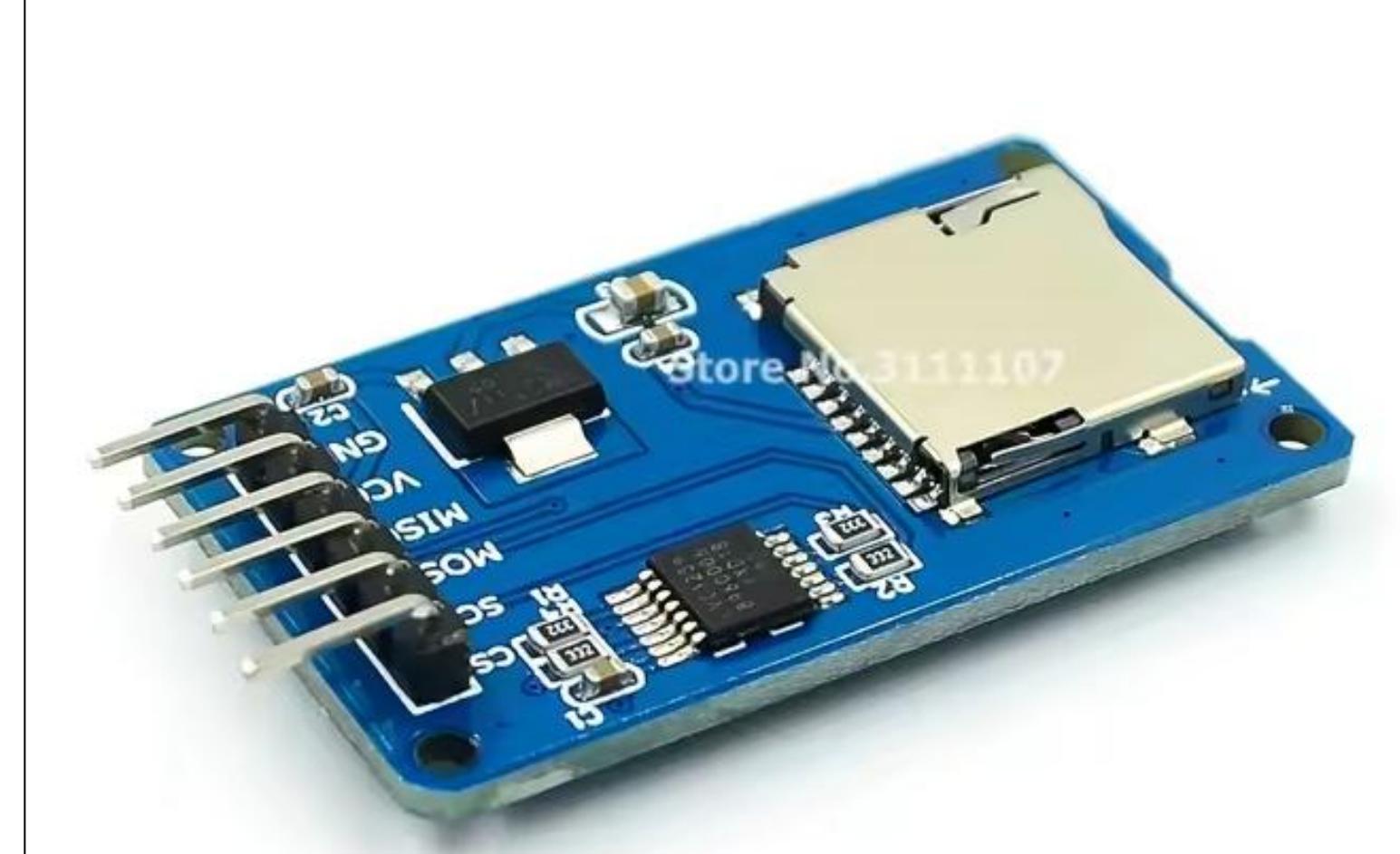
http://elm-chan.org/fsw/ff/00index_e.html

Adaptador para cartões SD

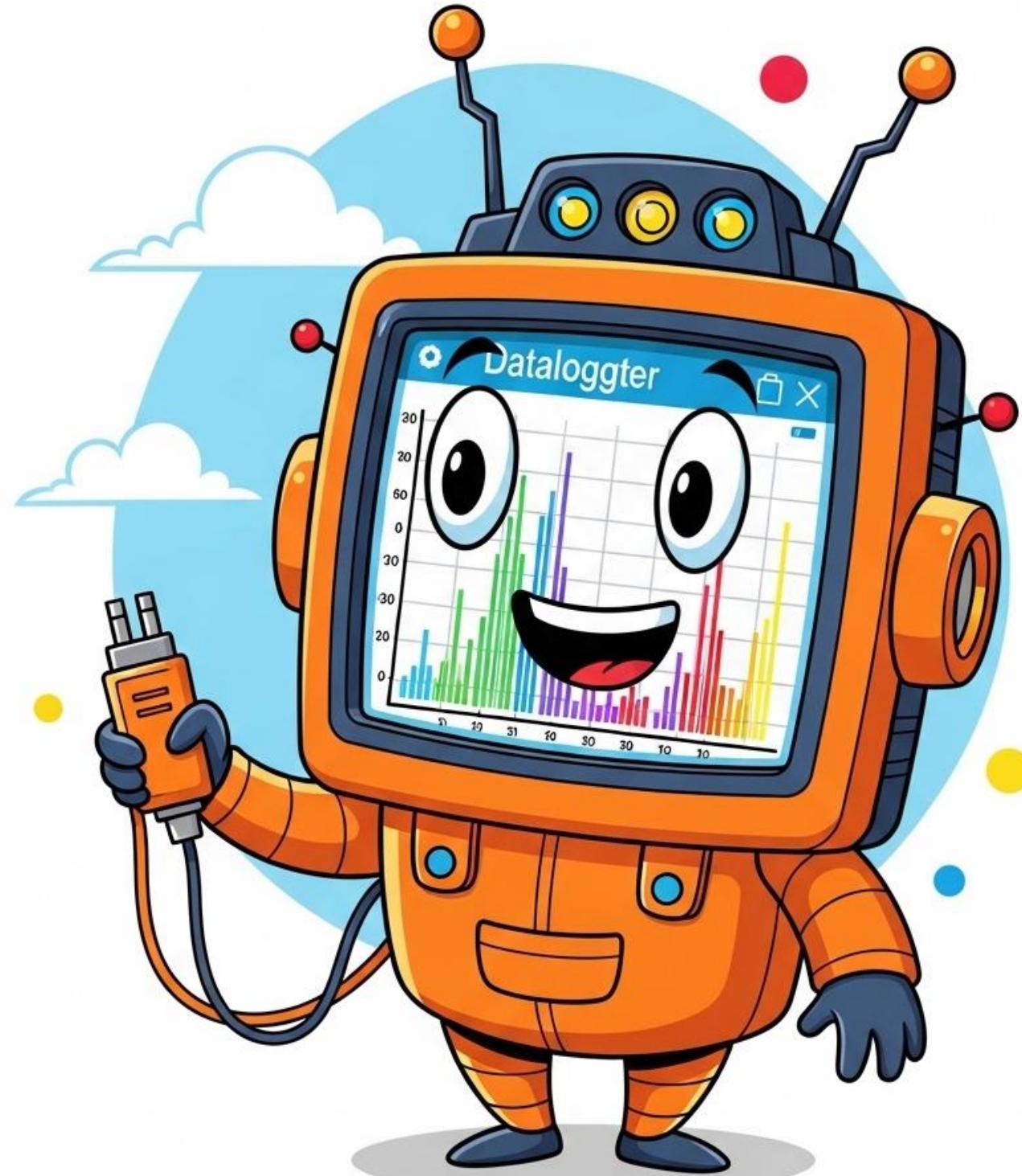
Um **módulo de cartão micro SD** é uma placa de circuito impresso (PCB) de interface que garante a compatibilidade elétrica e funcional entre um sistema microcontrolado e um cartão de memória flash (micro SD).

Observação:

Cartões SD operam em 3.3V. Assim, é necessário um adaptador de nível lógico se o MCU operar em 5V (por exemplo o Arduino UNO ou Nano).



Datalogger (Registrador de Dados)

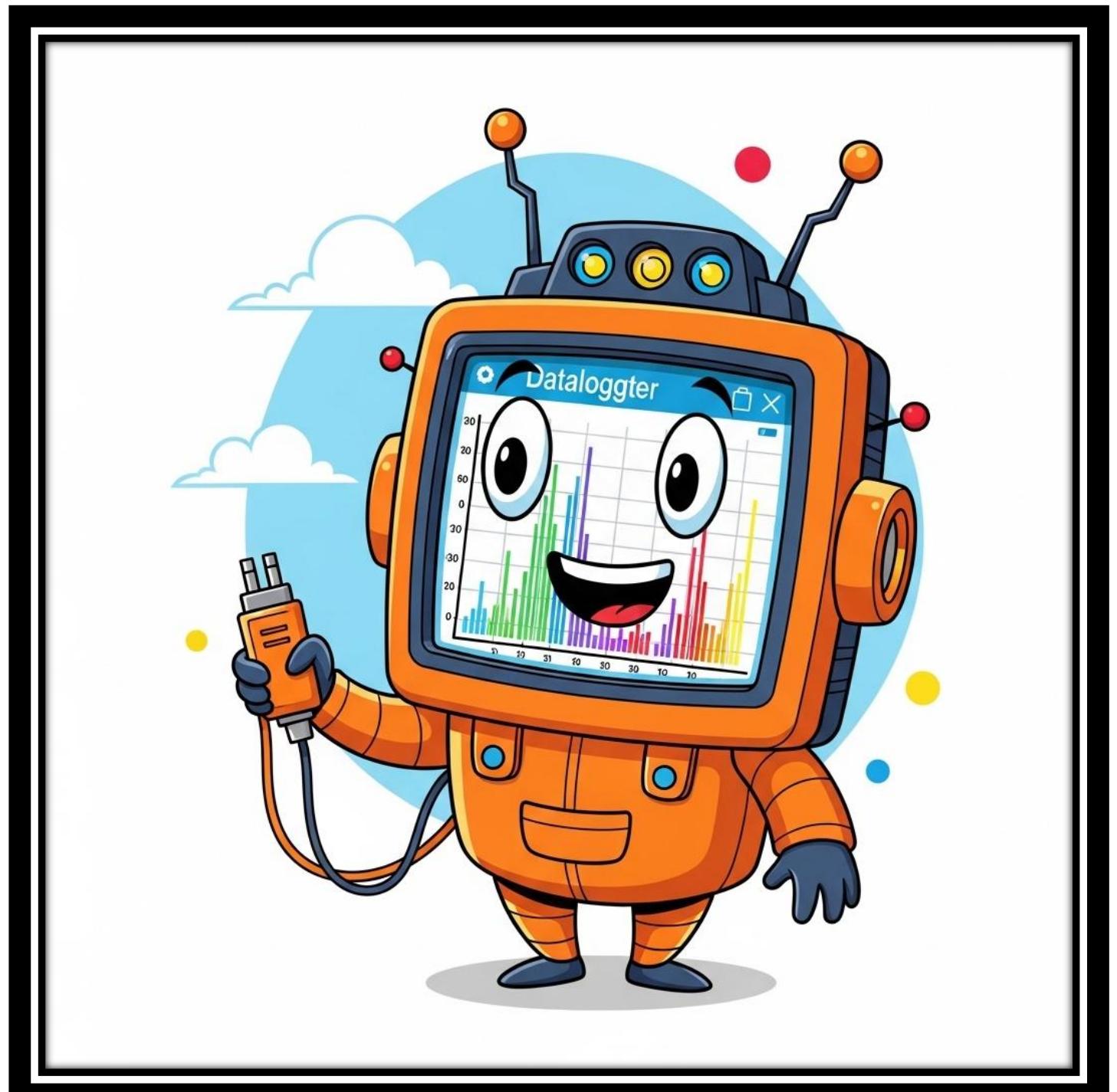


Simplificadamente um **datalogger** (ou registrador de dados) é um dispositivo eletrônico autônomo projetado para coletar e armazenar dados ao longo do tempo.

Um **datalogger** é um dispositivo que automatiza a tarefa de coletar e registrar informações ao longo do tempo, permitindo uma análise detalhada e confiável de qualquer processo ou ambiente.

Datalogger (Registrador de Dados) – Exemplos:

- **Logística e Transporte:** Dataloggers de temperatura e umidade são colocados em caminhões que transportam vacinas, medicamentos e alimentos perecíveis. Ao final da viagem, os dados são analisados para garantir que a "cadeia de frio" não foi quebrada e que os produtos estão em perfeitas condições.
- **Indústria:** Monitoram a temperatura de fornos, a pressão em tubulações ou a vibração de motores para prever falhas, garantir a qualidade do produto e otimizar processos.
- **Meio Ambiente e Agricultura:** Estações meteorológicas autônomas usam dataloggers para registrar dados de chuva, velocidade do vento, radiação solar e umidade do solo por meses, ajudando em estudos climáticos e no manejo de plantações.
- **Pesquisa Científica:** São usados em laboratórios e em campo para registrar dados de experimentos que duram longos períodos, desde o monitoramento de vulcões até o estudo do comportamento de animais.
- **Engenharia Civil:** Podem monitorar a "saúde" de estruturas como pontes e edifícios, registrando deformações e vibrações ao longo do tempo.



Arquivos salvos no formato CSV

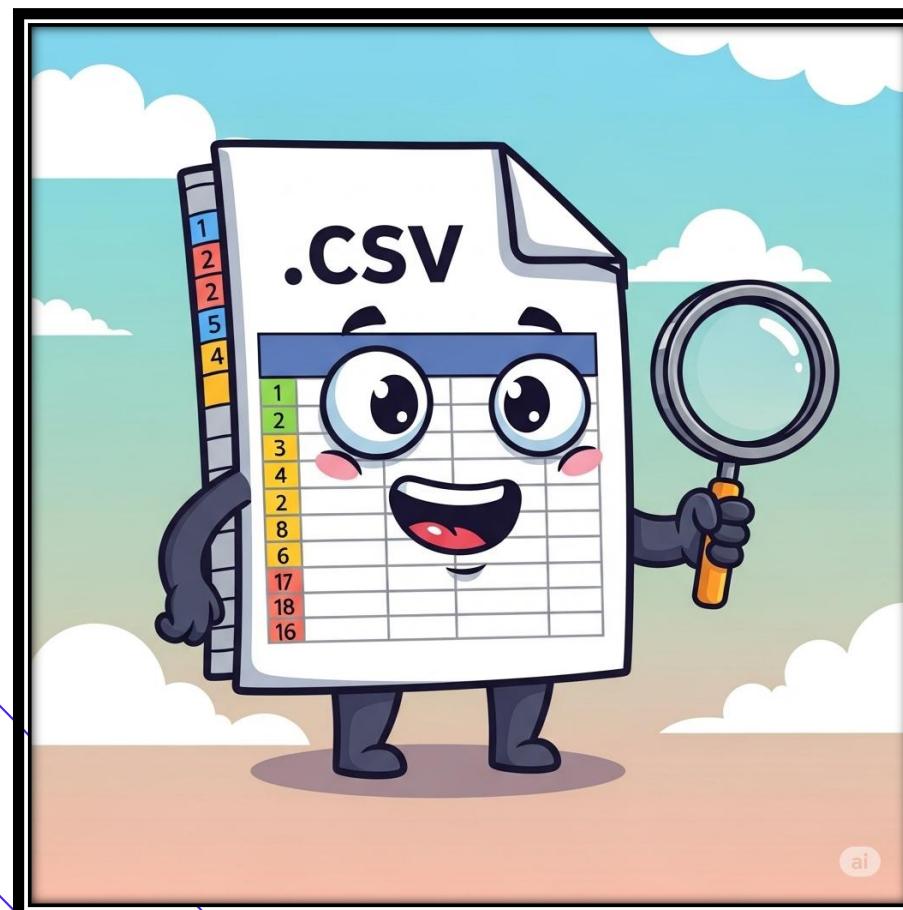
CSV é a sigla para **Comma-Separated Values**, ou "Valores Separados por Vírgula".

O CSV é um **arquivo de texto** que não tem formatação, cores, ou fórmulas como um arquivo do Excel (.xlsx).

Cada linha é uma linha da tabela de dados.

A **vírgula é o separador** que diz "aqui termina uma coluna e começa a próxima".

A primeira linha é (geralmente) o cabeçalho: Por convenção, a primeira linha do arquivo .csv contém os nomes das colunas (o "cabeçalho").

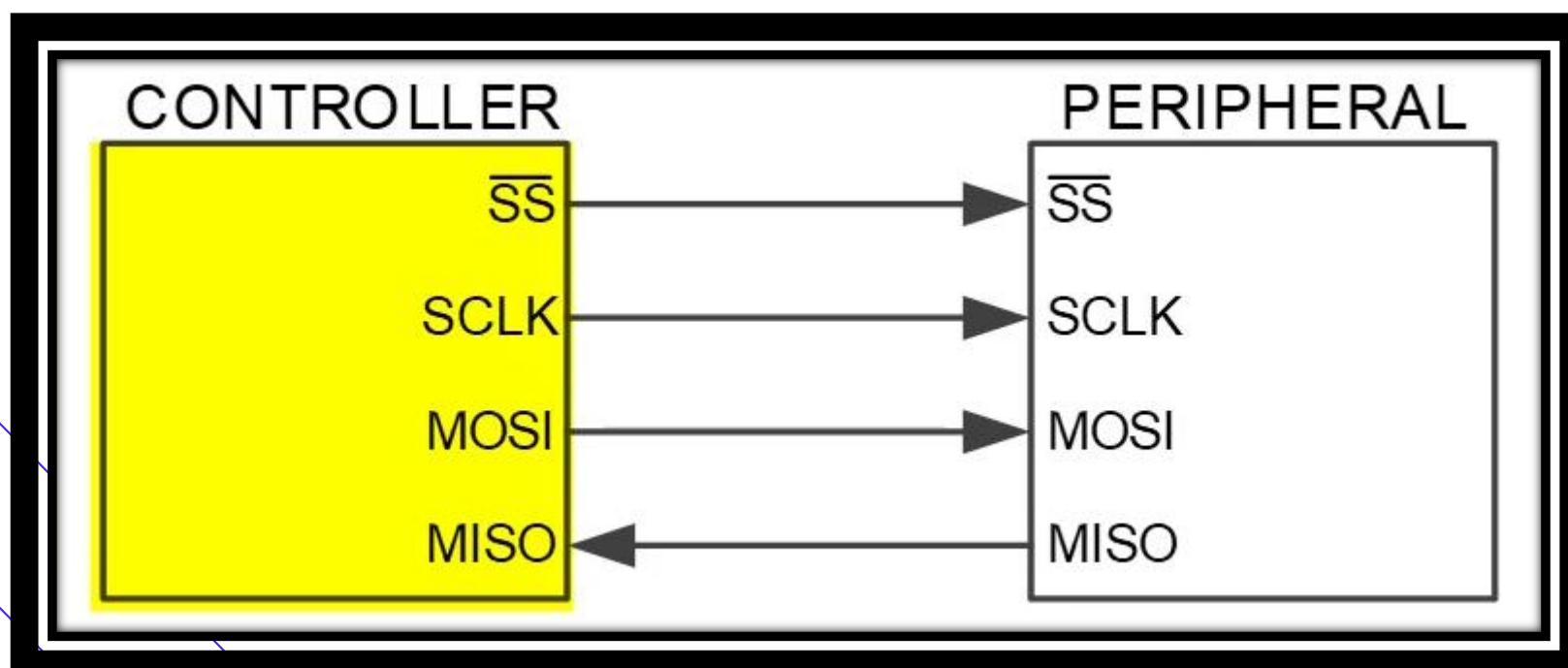


```
tempo_ms,accel_x,accel_y,accel_z,giro_x,giro_y,giro_z
150,0.02,0.01,0.98,-1.2,0.5,0.1 250,0.03,0.01,0.99,-1.3,0.5,0.1
350,0.02,0.02,0.98,-1.2,0.6,0.2 450,0.02,0.01,0.98,-1.2,0.5,0.1
```

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.

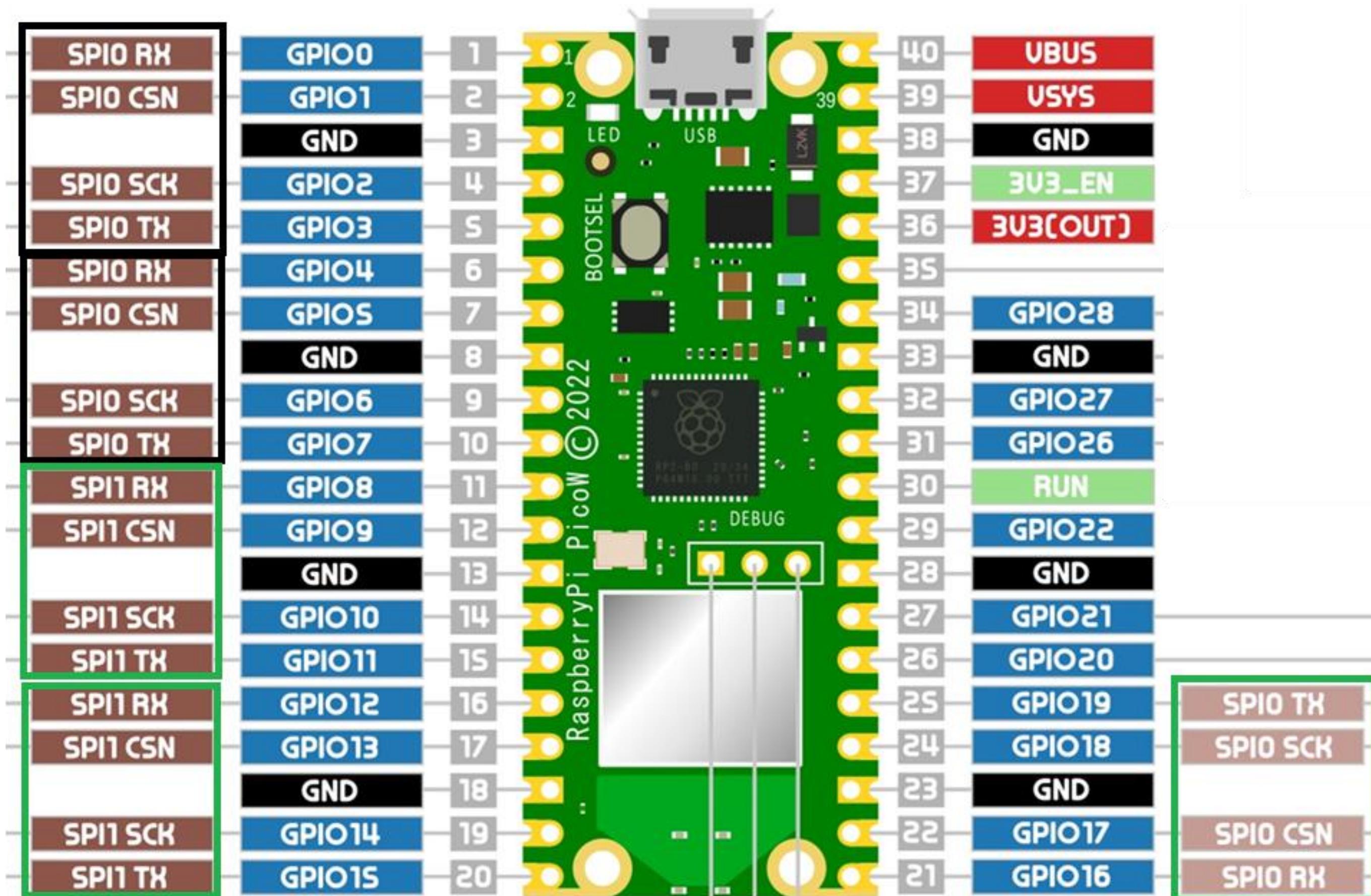
Velocidade configurável:

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

Comunicação serial SPI

Linhas de SPI no RP2

- **SCK** (Serial Clock) = **SCK**
- **MOSI** (Master Out, Slave In) = **TX**
- **MISO** (Master In, Slave Out) = **RX**
- **SS** (Slave Select): **CSN**



Comunicação serial SPI

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

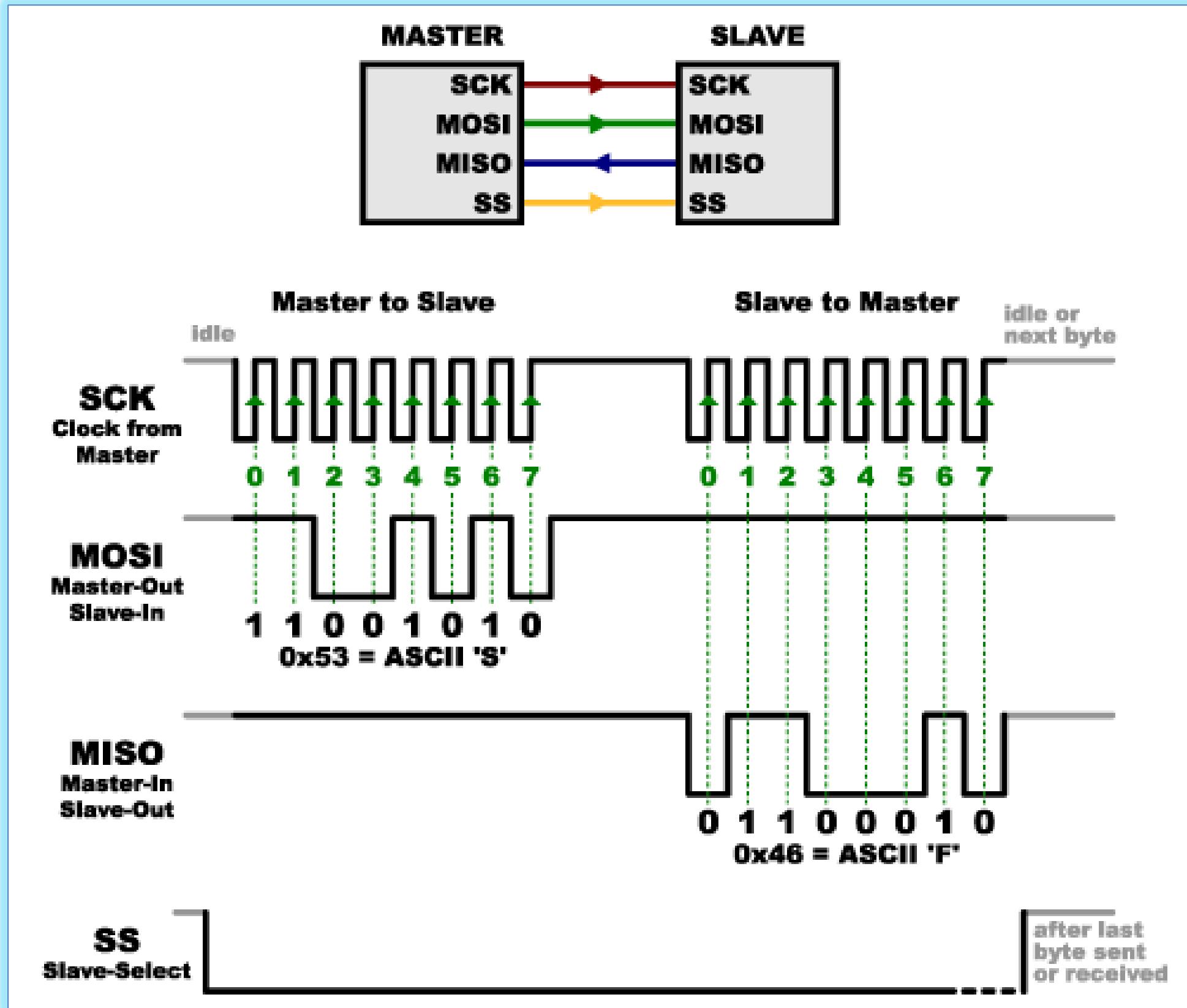
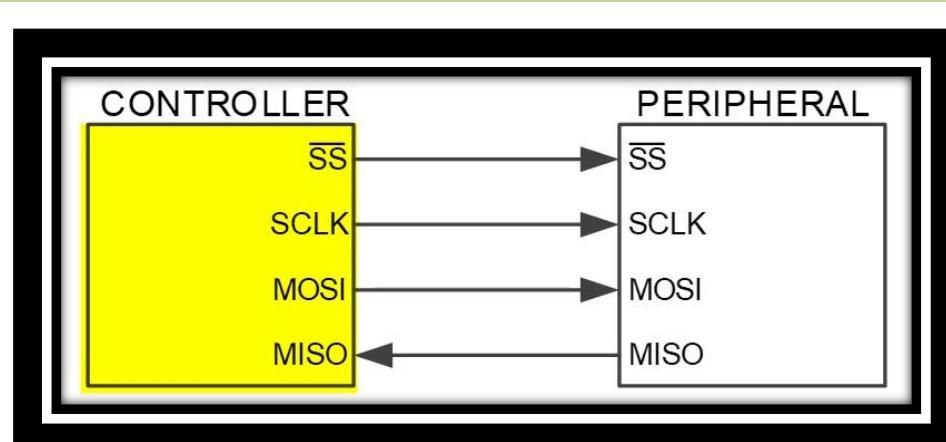
O lado que gera o clock é chamado de **controlador** (anteriormente "master"), e o outro lado é chamado de **periférico** (anteriormente "slave").

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**).

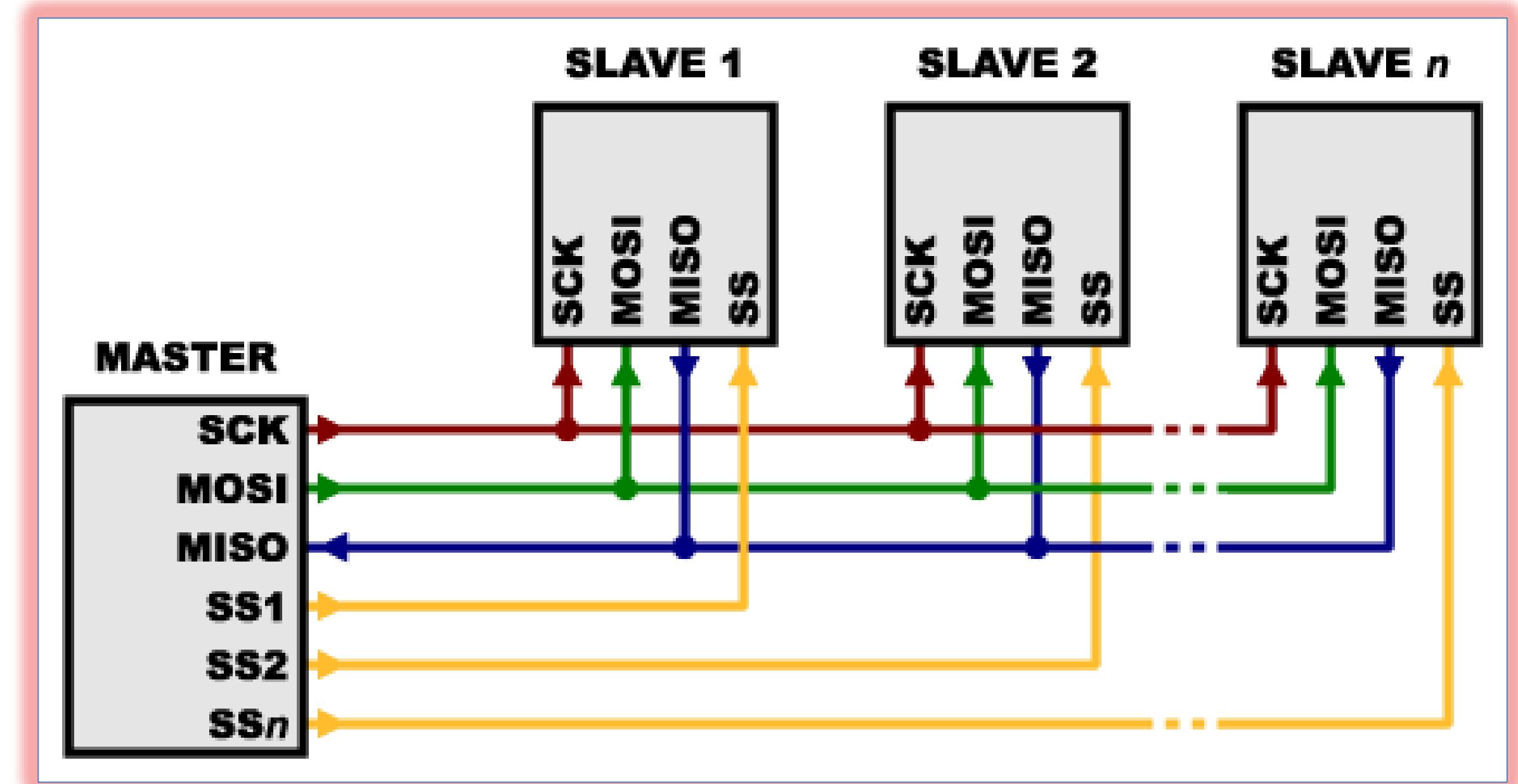
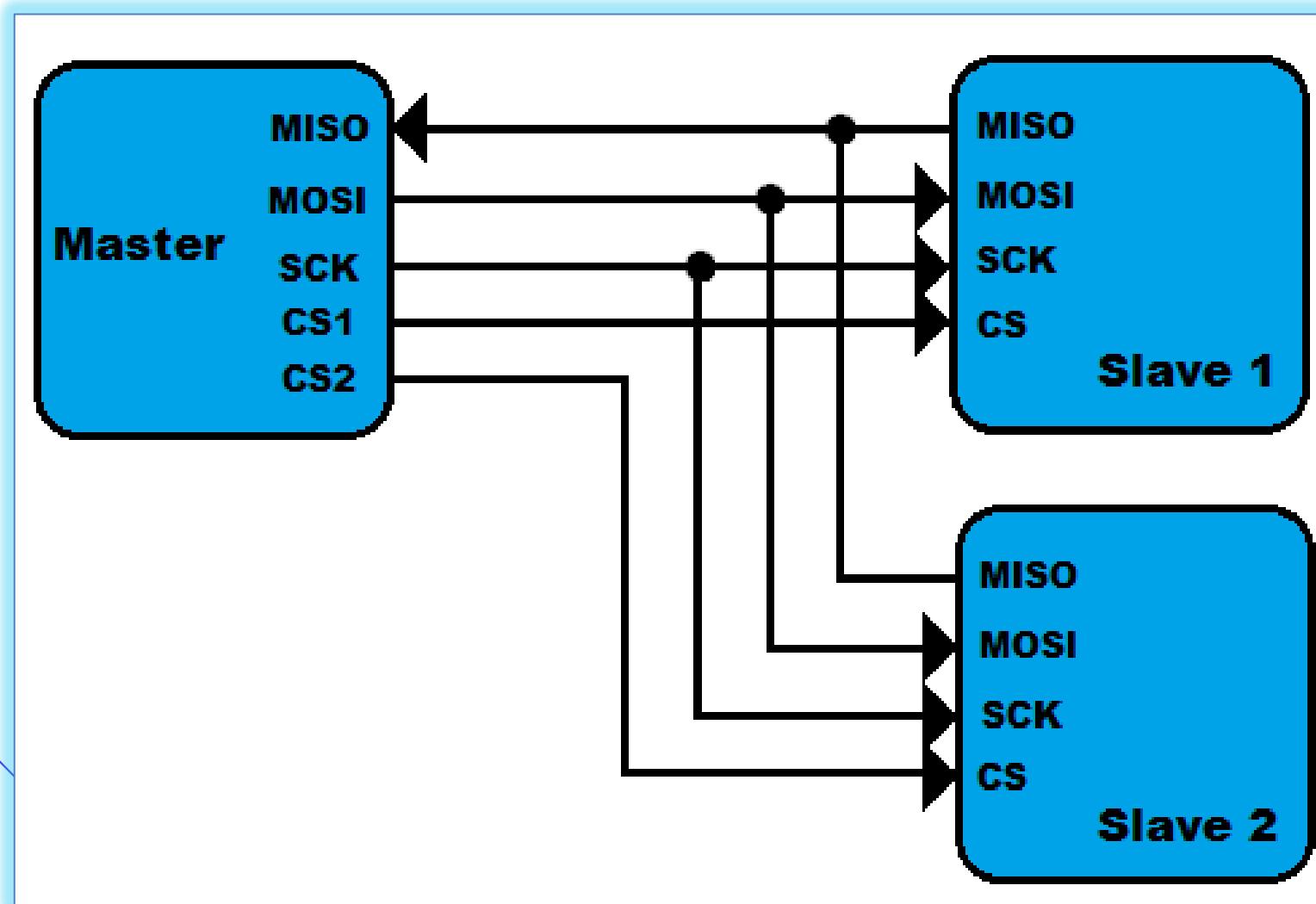
Se o **periférico** precisar enviar uma resposta ao **controlador**, este continuará a gerar o número pré-determinado de ciclos de clock, e o **periférico** enviará os dados por meio de uma terceira linha de dados chamada **MISO** (Peripheral Out / Controller In ou **Master In / Slave Out**).

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



Comunicação serial SPI

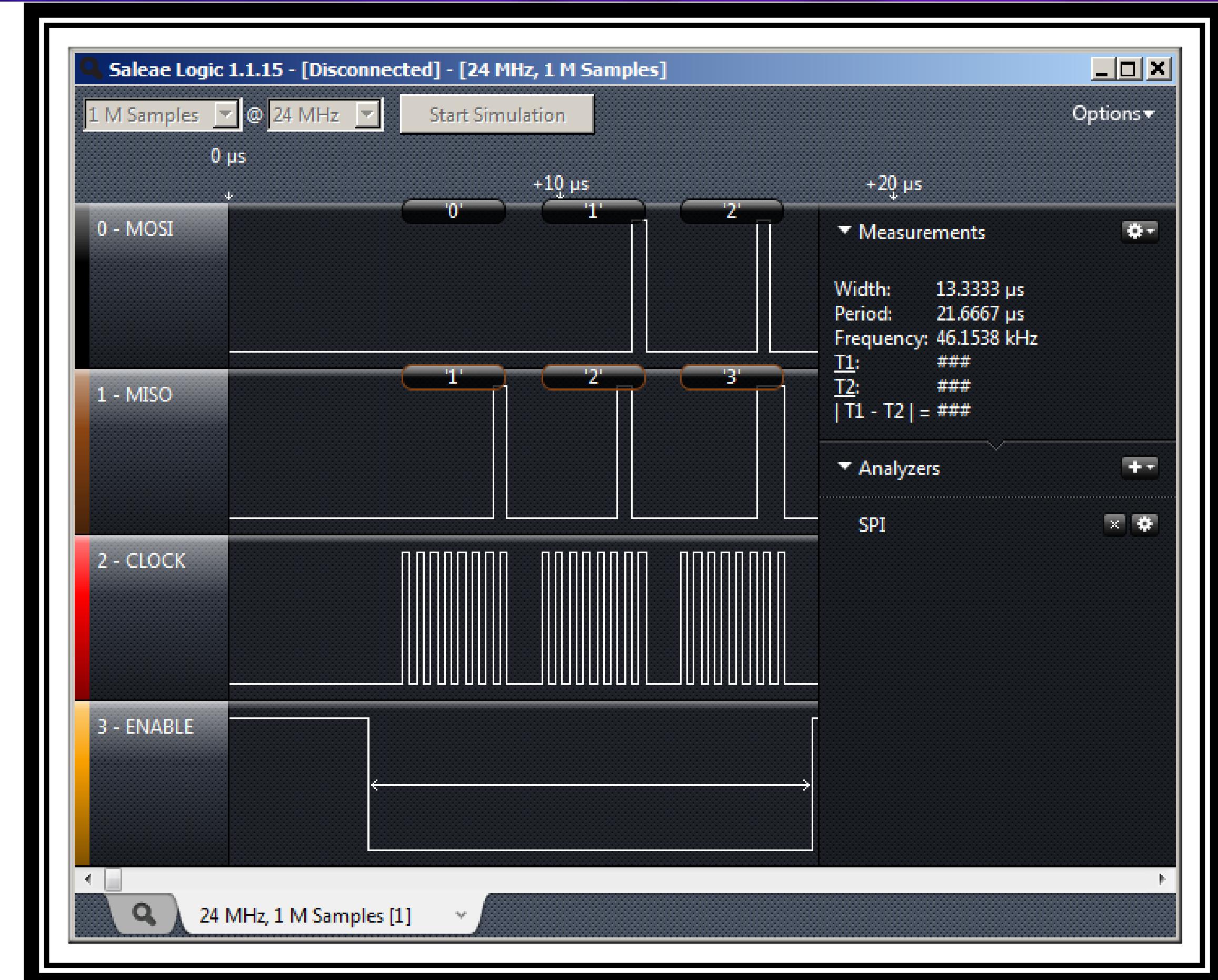
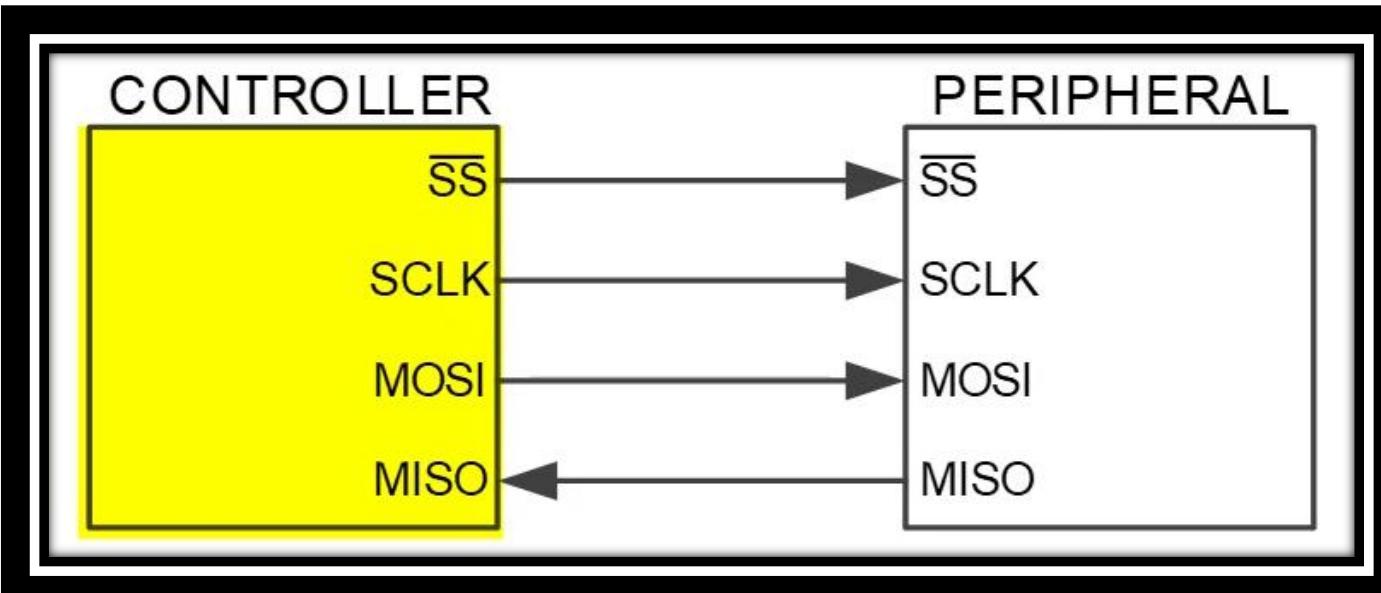
Representação
esquemática de
comunicação SPI com 2 ou 3
dispositivos periféricos.



Comunicação serial SPI

Linhas de comunicação RP2

- **SCK** (Serial Clock) = **SCK**
- **MOSI** (Master Out, Slave In) = **TX**
- **MISO** (Master In, Slave Out) = **RX**
- **SS** (Slave Select): **CSN**



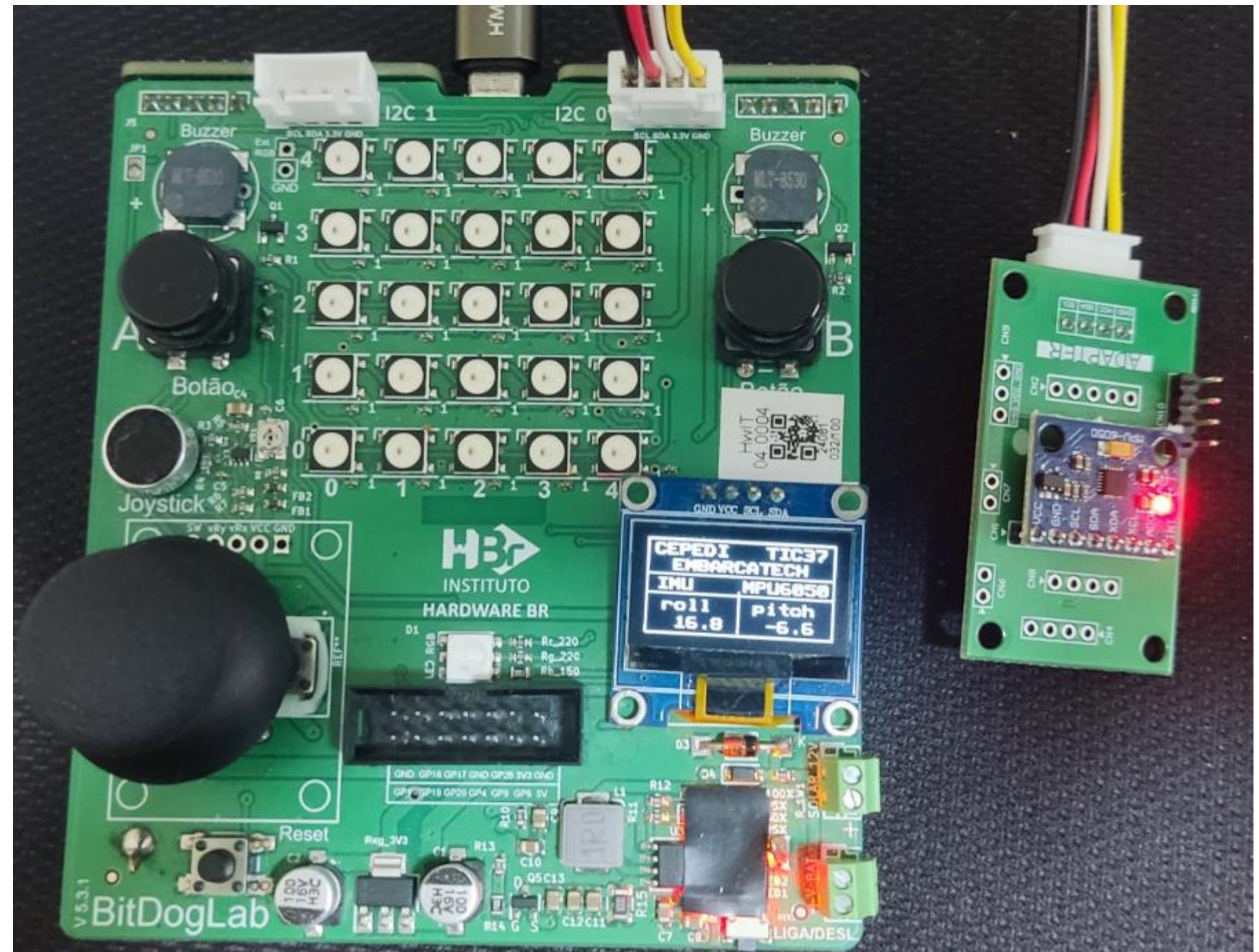
Exemplos de programas desenvolvidos para verificação de funcionamento do IMU.



Sensor: IMU-MPU6050

Exemplo 1:

Este projeto utiliza a Bitdoglab ligada ao sensor MPU 6050 para medir a aceleração dos eixos X, Y e Z. Com estes valores são calculados os ângulos de **Roll** e **Pitch** relativos a posição espacial da placa extensora. Os resultados são apresentados no display OLED.



Sensor: IMU-MPU6050

Exemplo 1:

```
c RPY_mpu6050.c > ...
1 #include <stdio.h>
2 #include <string.h>
3 #include <math.h>
4 #include "pico/stdc.h"
5 #include "pico/binary_info.h"
6 #include "hardware/i2c.h"
7 #include "ssd1306.h"
8 #include "font.h"
9

10 // Definição dos pinos I2C
11 #define I2C_PORT i2c0
12 #define I2C_SDA 0
13 #define I2C_SCL 1
14
15 // Definição dos pinos I2C
16 #define I2C_PORT_DISP i2c1
17 #define I2C_SDA_DISP 14
18 #define I2C_SCL_DISP 15
19 #define ENDERECO_DISP 0x3C
```

```
21 // Endereço padrão do MPU6050
22 static int addr = 0x68;
23
24 // Função para resetar e inicializar o MPU6050
25 static void mpu6050_reset()
26 {
27     // Dois bytes para reset: primeiro o registrador, se
28     uint8_t buf[] = {0x6B, 0x80};
29     i2c_write_blocking(I2C_PORT, addr, buf, 2, false);
30     sleep_ms(100); // Aguarda reset e estabilização
31
32     // Sai do modo sleep (registrador 0x6B, valor 0x00)
33     buf[1] = 0x00;
34     i2c_write_blocking(I2C_PORT, addr, buf, 2, false);
35     sleep_ms(10); // Aguarda estabilização após acordar
36 }
```

Sensor: IMU-MPU6050

Exemplo 1:

```
38 // Função para ler dados crus do acelerômetro, giroscópio e temperatura
39 static void mpu6050_read_raw(int16_t accel[3], int16_t gyro[3], int16_t *temp)
40 {
41     uint8_t buffer[6];
42     // Lê aceleração a partir do registrador 0x3B (6 bytes)
43     uint8_t val = 0x3B;
44     i2c_write_blocking(I2C_PORT, addr, &val, 1, true);
45     i2c_read_blocking(I2C_PORT, addr, buffer, 6, false);
46     for (int i = 0; i < 3; i++)
47     {
48         accel[i] = (buffer[i * 2] << 8) | buffer[(i * 2) + 1];
49     }
50     // Lê giroscópio a partir do registrador 0x43 (6 bytes)
51     val = 0x43;
52     i2c_write_blocking(I2C_PORT, addr, &val, 1, true);
53     i2c_read_blocking(I2C_PORT, addr, buffer, 6, false);
54     for (int i = 0; i < 3; i++)
55     {
56         gyro[i] = (buffer[i * 2] << 8) | buffer[(i * 2) + 1];
57     }
58     // Lê temperatura a partir do registrador 0x41 (2 bytes)
59     val = 0x41;
60     i2c_write_blocking(I2C_PORT, addr, &val, 1, true);
61     i2c_read_blocking(I2C_PORT, addr, buffer, 2, false);
62     *temp = (buffer[0] << 8) | buffer[1];
63 }
64 }
```

Esta função “**mpu6050_read_raw()**”, juntamente com a função “**mpu6050_reset()**” funcionam basicamente como a **biblioteca** do sensor IMU.

Sensor: IMU-MPU6050

Exemplo 1:

```
85 // Inicializa a I2C do Display OLED em 400kHz
86 i2c_init(I2C_PORT_DISP, 400 * 1000);
87 gpio_set_function(I2C_SDA_DISP, GPIO_FUNC_I2C);
88 gpio_set_function(I2C_SCL_DISP, GPIO_FUNC_I2C);
89 gpio_pull_up(I2C_SDA_DISP);
90 gpio_pull_up(I2C_SCL_DISP);

91 ssd1306_t ssd;
92 ssd1306_init(&ssd, WIDTH, HEIGHT, false, ENDERECO_DISP, I2C_PORT_DISP);
93 ssd1306_config(&ssd);
94 ssd1306_send_data(&ssd);

95
96 // Limpa o display
97 ssd1306_fill(&ssd, false);
98 ssd1306_send_data(&ssd);

99
100 // Inicialização da I2C do MPU6050
101 i2c_init(I2C_PORT, 400 * 1000);
102 gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
103 gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
104 gpio_pull_up(I2C_SDA);
105 gpio_pull_up(I2C_SCL);

106
107 // Declara os pinos como I2C na Binary Info
108 bi_decl(bi_2pins_with_func(I2C_SDA, I2C_SCL, GPIO_FUNC_I2C));
109 mpu6050_reset();
```

```
65 // Trecho para modo BOOTSEL usando o botão B
66 #include "pico/bootrom.h"
67 #define botaoB 6
68 void gpio_irq_handler(uint gpio, uint32_t events)
69 {
70     reset_usb_boot(0, 0);
71 }
72 int main()
73 {
74     // Inicialização do modo BOOTSEL com botão B
75     gpio_init(botaoB);
76     gpio_set_dir(botaoB, GPIO_IN);
77     gpio_pull_up(botaoB);
78     gpio_set_irq_enabled_with_callback(botaoB, GPIO_IRQ_EDGE_FALL, true, &gpio_irq_handler);
79 // Fim do trecho para modo BOOTSEL
80 }
```

Botão B como reset do MCU.

Inicialização das portas I2C, tanto do Display quanto do MPU 6050.

Inicialização da biblioteca SSD1306.
Inicialização da “biblioteca” do MPU 6050.

Sensor: IMU-MPU6050

Exemplo 1:

```
112     int16_t aceleracao[3], gyro[3], temp;
113     bool cor = true;
114     while (1)
115     {
116         // Leitura dos dados de aceleração, giroscópio e temperatura
117         mpu6050_read_raw(aceleracao, gyro, &temp);
118
119         // Conversão para unidade de 'g'
120         float ax = aceleracao[0] / 16384.0f;
121         float ay = aceleracao[1] / 16384.0f;
122         float az = aceleracao[2] / 16384.0f;
123
124         // Cálculo dos ângulos em graus (Roll e Pitch)
125         float roll = atan2(ay, az) * 180.0f / M_PI;
126         float pitch = atan2(-ax, sqrt(ay*ay + az*az)) * 180.0f / M_PI;
127
128         // Montagem das strings para o display
129         char str_roll[20];
130         char str_pitch[20];
```

Criação de variáveis

Início do LOOP infinito do MCU.

Leitura dos valores de aceleração nos eixos X,Y e Z.

Conversão dos valores lidos para aceleração em “g”.

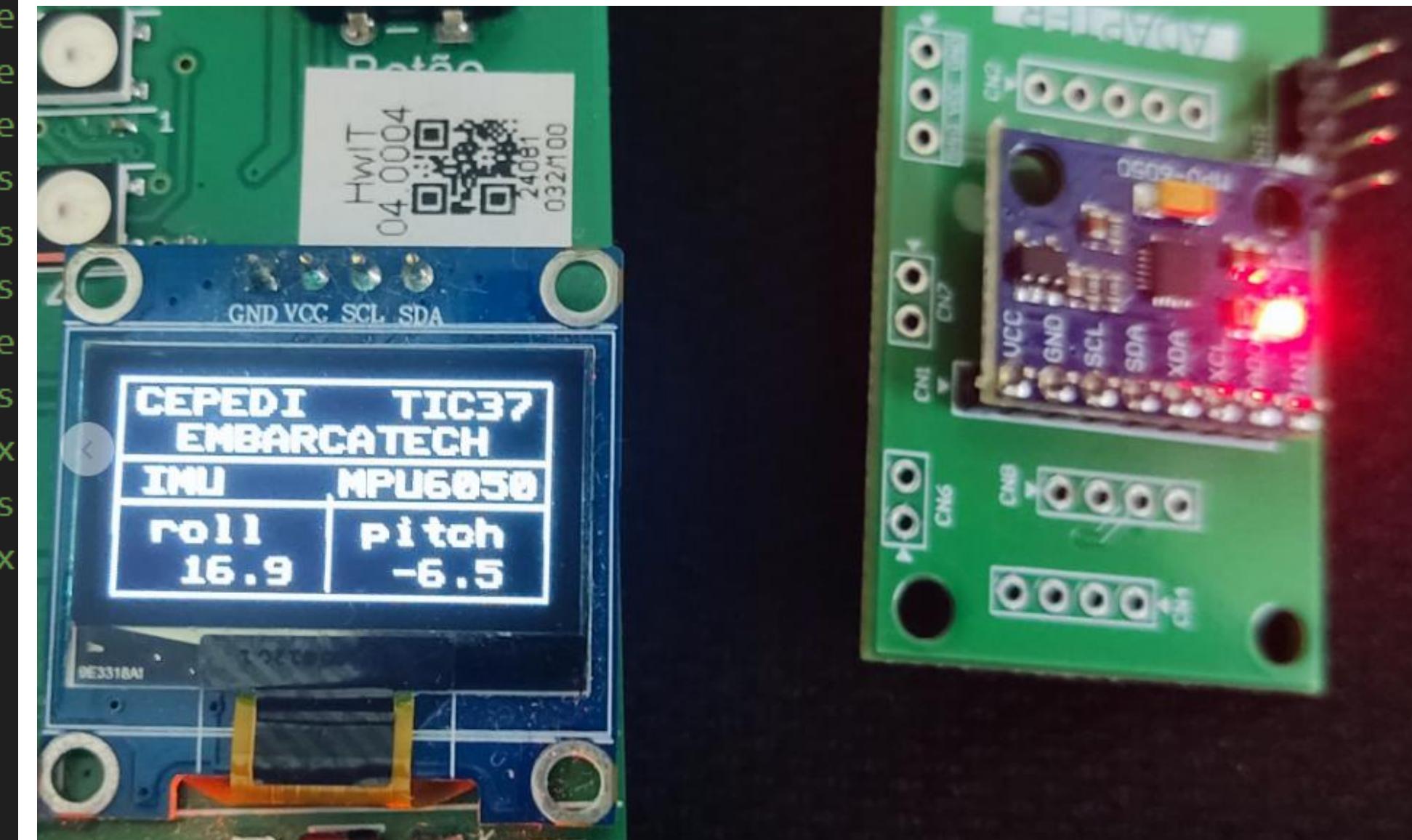
Cálculo dos ângulos para o Roll e o Pitch.

Sensor: IMU-MPU6050

Exemplo 1:

```
132  
133     sprintf(str_roll, sizeof(str_roll), "%5.1f", roll);  
134     sprintf(str_pitch, sizeof(str_pitch), "%5.1f", pitch);  
135  
136  
137     // Exibição no display  
138     ssd1306_fill(&ssd, !cor);                                // Li  
139     ssd1306_rect(&ssd, 3, 3, 122, 60, cor, !cor);          // De  
140     ssd1306_line(&ssd, 3, 25, 123, 25, cor);              // De  
141     ssd1306_line(&ssd, 3, 37, 123, 37, cor);              // De  
142     ssd1306_draw_string(&ssd, "CEPEDI TIC37", 8, 6);    // Es  
143     ssd1306_draw_string(&ssd, "EMBARCATECH", 20, 16);   // Es  
144     ssd1306_draw_string(&ssd, "IMU MPU6050", 10, 28); // Es  
145     ssd1306_line(&ssd, 63, 35, 63, 60, cor);           // De  
146     ssd1306_draw_string(&ssd, "roll", 14, 41);         // Es  
147     ssd1306_draw_string(&ssd, str_roll, 14, 52);        // Ex  
148     ssd1306_draw_string(&ssd, "pitch", 73, 41);        // Es  
149     ssd1306_draw_string(&ssd, str_pitch, 73, 52);       // Ex  
150     ssd1306_send_data(&ssd);  
151     sleep_ms(500);  
152 }  
153 }  
154 }
```

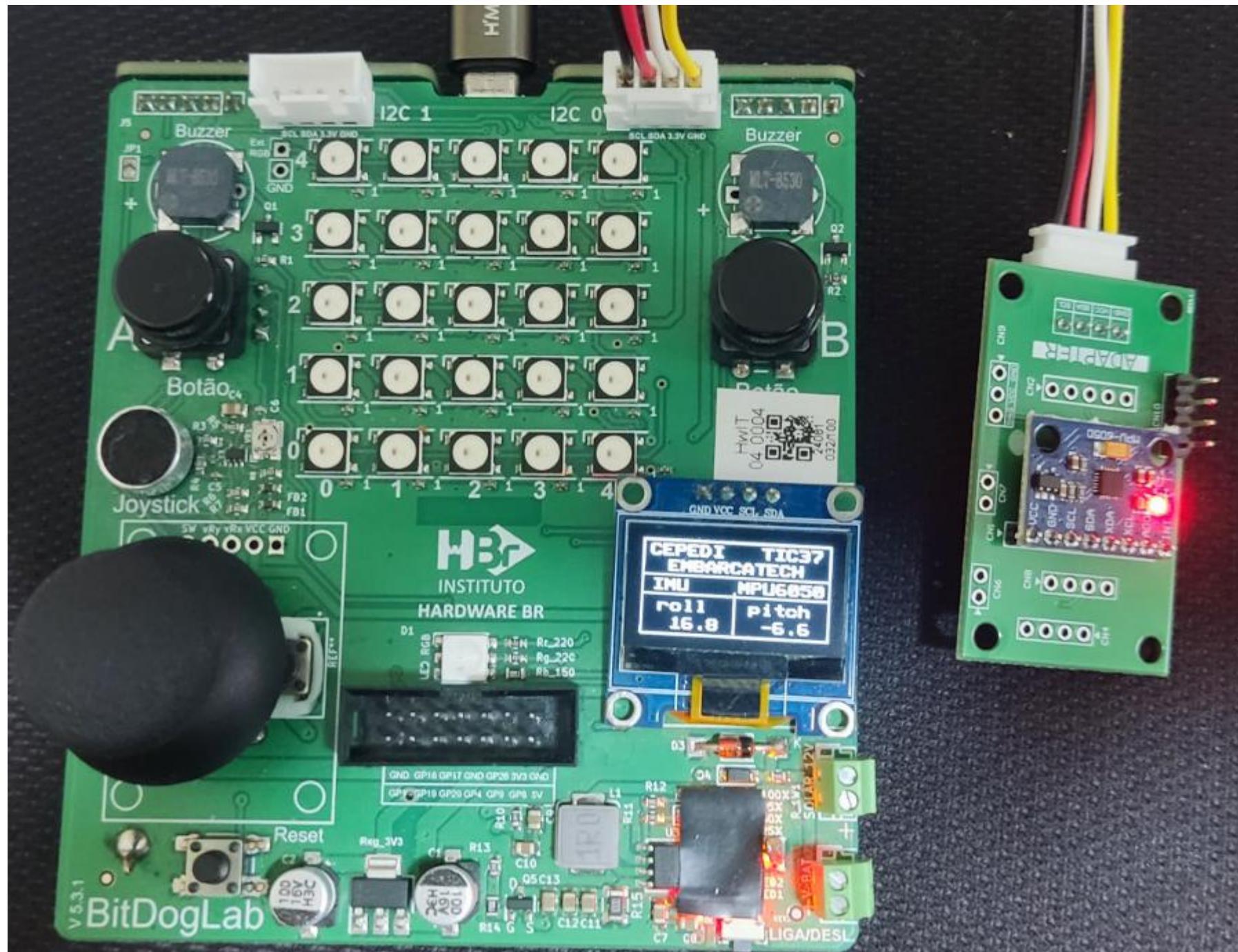
Escrita dos valores dos ângulos do Roll
e Pitch no Display monocromático LCD
128 x 64 da BitDogLab.



Cartão SD + ADC

Exemplo 2:

Este projeto usa um cartão SD com sistema de arquivos FAT (FatFS) realizando operações de leitura, escrita e listagem de arquivos via comandos no terminal.
O código também inclui uma rotina de aquisição de dados do ADC (canal 0, GPIO 26) e registro dos dados em arquivo.



<https://github.com/wiltonlacerda/EmbarcaTechResU1Ex05.git>

C Cartao_FatFS_SPI.c > ...

```
1 #include <ctype.h>
2 #include <stdbool.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <time.h>
7
8 #include "hardware/adc.h"
9 #include "hardware/rtc.h"
10 #include "pico/stdlib.h"
11
12 #include "ff.h"
13 #include "diskio.h"
14 #include "f_util.h"
15 #include "hw_config.h"
16 #include "my_debug.h"
17 #include "rtc.h"
18 #include "sd_card.h"
19
20 #define ADC_PIN 26 // GPIO 26
```

Funcionalidades

- Formatação, montagem e desmontagem do cartão SD (com FatFS).
- Listagem de arquivos e diretórios (comando `ls`).
- Leitura do conteúdo de arquivos (comando `cat`).
- Aquisição de dados do ADC e salvamento automático em arquivo ("adc_data2.txt").
- Exibição de espaço livre no cartão SD.
- Configuração de data/hora do RTC integrado.
- Atalhos de teclado para comandos rápidos no terminal.

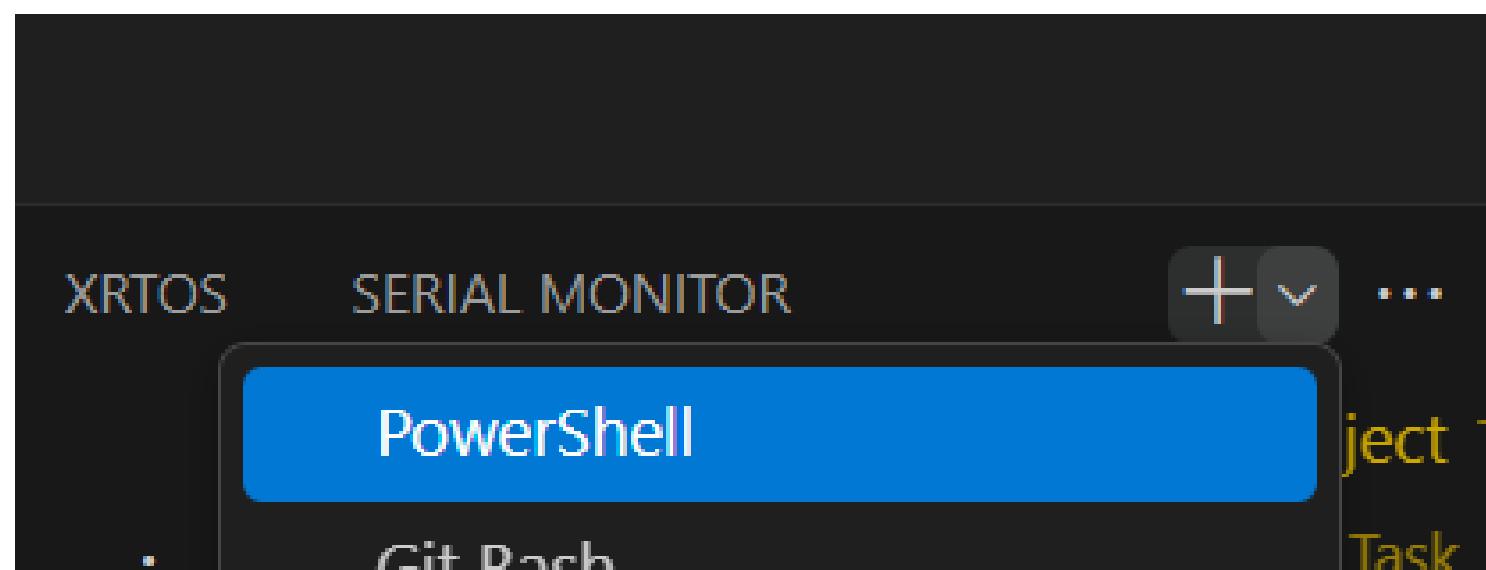
```
26 static char filename[20] = "adc_data1.txt";
```

Atalhos de teclado no terminal (pressione apenas a tecla):

Tecla	Função
~a~	Monta o cartão SD (~mount~)
~b~	Desmonta o cartão SD (~unmount~)
~c~	Lista os arquivos do cartão SD (~ls~)
~d~	Lê e exibe o conteúdo do arquivo ~adc_data2.txt~
~e~	Mostra o espaço livre no cartão SD (~getfree~)
~f~	Captura 128 amostras do ADC e salva no arquivo ~adc_data2.txt~
~h~	Exibe os comandos disponíveis (~help~)

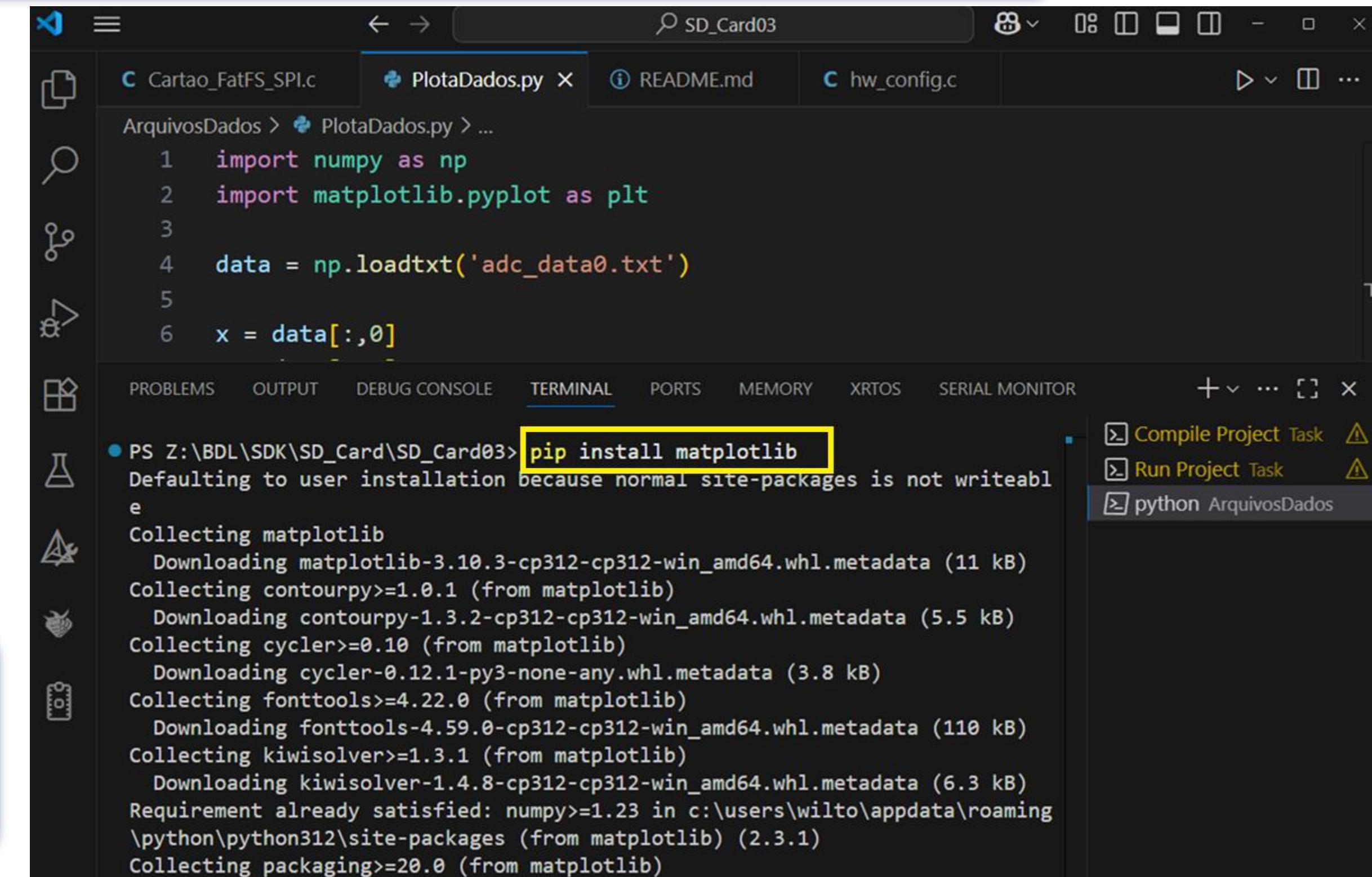
Traçar gráficos utilizando o Python

Instalar a biblioteca “matplotlib” no Terminal do VSCode: `pip install matplotlib`



Caso ainda não tenha a biblioteca `numpy` instalada:
`pip install numpy`

No meu caso já estava instalada.



```
PS Z:\BDL\SDK\SD_Card\SD_Card03> pip install matplotlib
Defaulting to user installation because normal site-packages is not writeable
Collecting matplotlib
  Downloading matplotlib-3.1.0.3-cp312-cp312-win_amd64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.2-cp312-cp312-win_amd64.whl.metadata (5.5 kB)
Collecting cycler>=0.10 (from matplotlib)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.59.0-cp312-cp312-win_amd64.whl.metadata (110 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.8-cp312-cp312-win_amd64.whl.metadata (6.3 kB)
Requirement already satisfied: numpy>=1.23 in c:\users\wilto\appdata\roaming\python\python312\site-packages (from matplotlib) (2.3.1)
Collecting packaging>=20.0 (from matplotlib)
```

Traçar gráficos utilizando o Python

Para rodar o código, vá para dentro da pasta que esta o arquivo Python “PlotaDados.py” e execute o comando no Terminal:

“ python .\PlotaDados.py ”

The screenshot shows a terminal window with the following content:

```
Cartao_FatFS_SPI.c          PlotaDados.py X READING
ArquivosDados > PlotaDados.py > ...
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 data = np.loadtxt('adc_data0.txt')
5
6 x = data[:,0]

PROBLEMS      OUTPUT      DEBUG CONSOLE      TERMINAL      PO
clear, contourpy, matplotlib
[...]
ripts fonttools.exe, pyftmerge.exe, pyftsubset.exe
in 'C:\Users\wilto\AppData\Roaming\Python\Python PATH.
Consider adding this directory to PATH or,
warning, use --no-warn-script-location.
Successfully installed contourpy-1.3.2 cyclenumber-1.4.8
matplotlib-3.10.3 packaging-25.0 pillow-11.5.0 pyparsing-3.2.3
python-dateutil-2.9.0.post0
PS Z:\BDL\SDK\SD_Card\SD_Card03> cd ..\ArquivosDados\
```

The terminal command `cd ..\ArquivosDados\` is highlighted with a green border, and the command `python .\PlotaDados.py` is highlighted with a yellow border.

To the right of the terminal, a plot window titled "Figure 1" displays a scatter plot titled "Gráfico dos dados coletados". The x-axis is labeled "tempo" and ranges from 0 to 120. The y-axis is labeled "Amplitude" and ranges from 1000 to 1800. The plot shows a series of blue dots representing collected data points.

Traçar gráficos utilizando o Python

The screenshot shows a Google Colab notebook titled "PlotaGraficoRP2.ipynb". The code cell contains the following Python script:

```
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt('adc_data1.txt')

x = data[:,0]
y = data[:,1]
plt.plot(x,y,'bo-')
plt.title("Gráfico dos dados coletados")
plt.xlabel("tempo")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```

The resulting plot is a scatter graph titled "Gráfico dos dados coletados". The x-axis is labeled "tempo" (time) and ranges from 0 to 120. The y-axis is labeled "Amplitude" and ranges from 1240.0 to 1257.5. The plot shows a series of blue data points connected by lines, representing collected data over time.

Para demonstrar a utilização aqui é apresentado também podemos utilizar o ambiente do Google Colab, onde as bibliotecas “numpy” e “matplotlib” já estão instaladas por padrão.

<https://colab.research.google.com>

Tarefa



Tarefa:

Desenvolvimento de um Datalogger de Movimento com IMU

Enunciado

Criar um dispositivo portátil (datalogger) capaz de capturar dados de movimento (aceleração e giroscópio) de um sensor IMU MPU6050. Os dados coletados serão armazenados em formato de texto (.csv) num cartão MicroSD. Posteriormente, um programa em Python será usado num computador para ler esses dados e gerar gráficos para análise.

Descrição do Projeto

Cada residente desenvolverá **individualmente** um sistema embarcado que funciona como um registrador de dados de movimento. O Raspberry Pi Pico W, realiza a leitura do sensor, o armazenamento de dados e a interação com o usuário.

O sistema deverá:

- Capturar continuamente os dados de aceleração (eixos X, Y, Z) e giroscópio (eixos X, Y, Z) do sensor MPU6050.
- Armazenar os dados de forma estruturada em um arquivo .csv no cartão SD.
- Utilizar os recursos da plataforma BitDogLab para fornecer feedback em tempo real ao usuário, melhorando a usabilidade do dispositivo.





Obrigado!

Executores:



Coordenação:



Iniciativa:

