UNIVERSIDADE DE SÃO PAULO
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica e de Computação

SEL0337/SEL0630
PROJETOS EM SISTEMAS EMBARCADOS

Capítulo 5

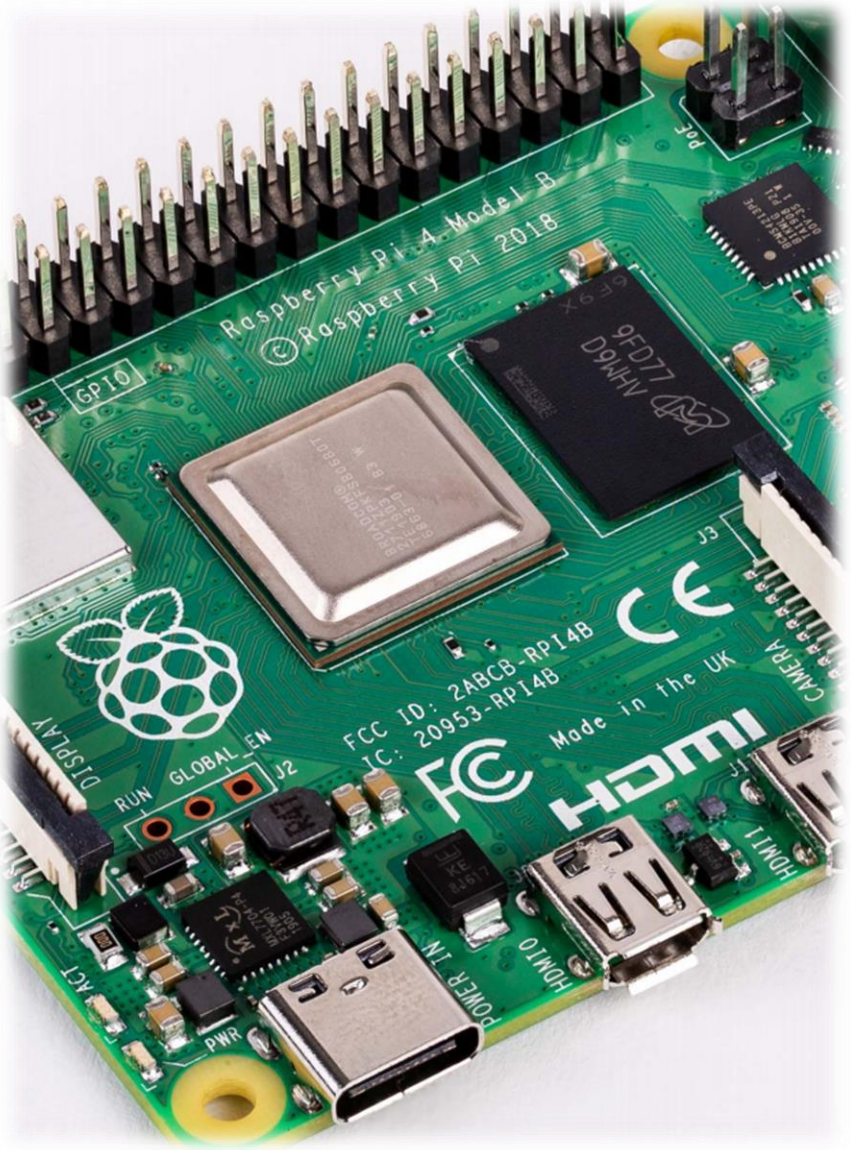
Protocolos da Comunicação Serial em Linux Embarcado

Prof. Pedro de Oliveira C. Junior

pedro.oliveiracjr@usp.br

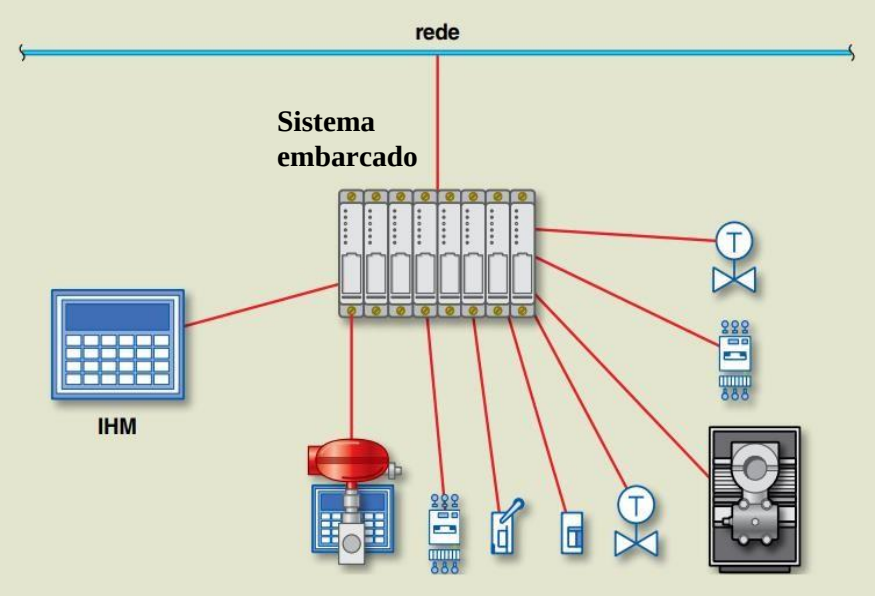
OBJETIVOS

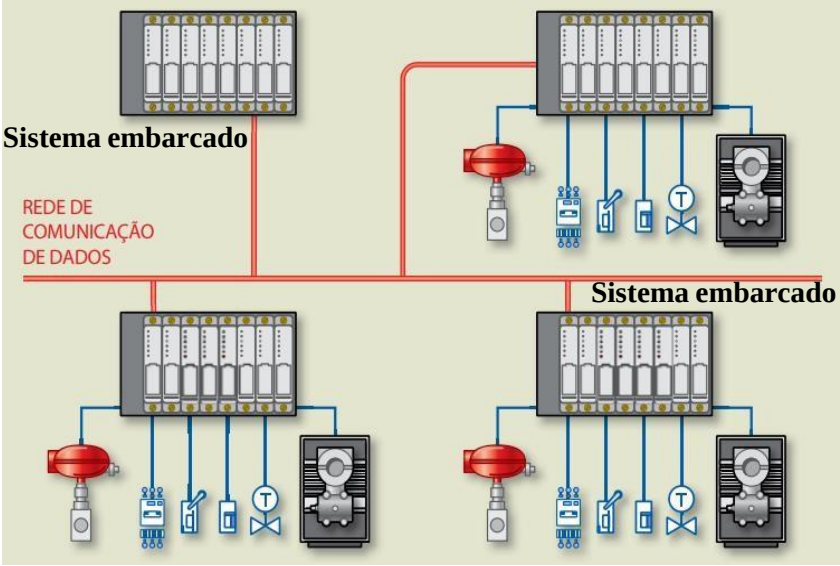
Desenvolver projetos envolvendo interfaces de comunicação serial UART, I2C e SPI em Linux embarcado



Protocolos de Comunicação em Sistemas Embarcados

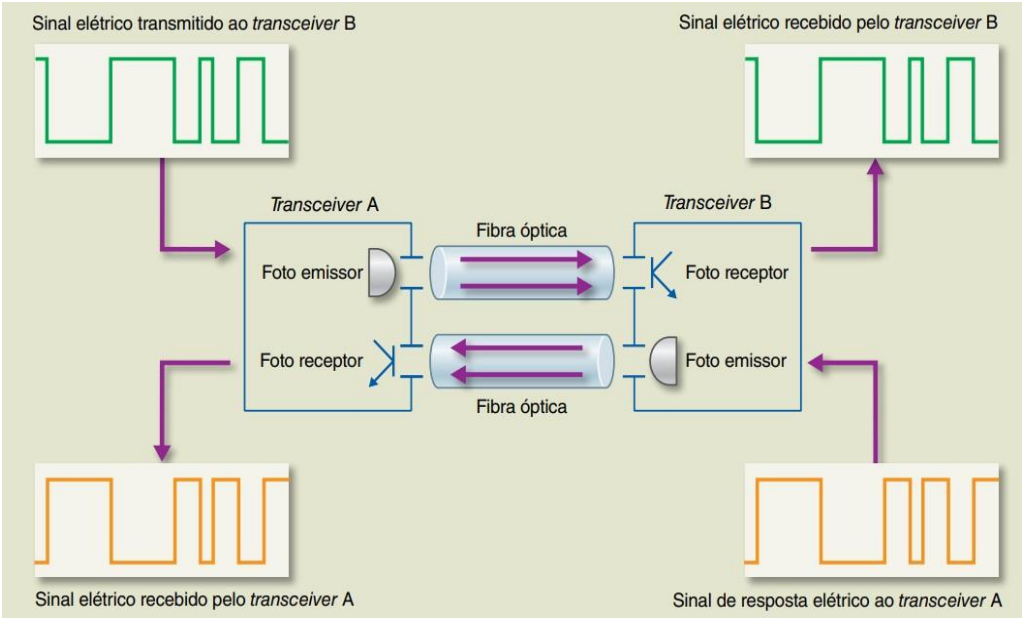
- ✓ **Redes de comunicação** possibilitam a troca de dados entre computadores, compartilhando recursos de hardware e software;
- ✓ Em **sistemas embarcados**, permite interligar dispositivos, módulos processadores ou sensores **por meios de transmissão** (via cabo, wireless etc.) e um conjunto de regras que organizam a comunicação (**protocolos**).

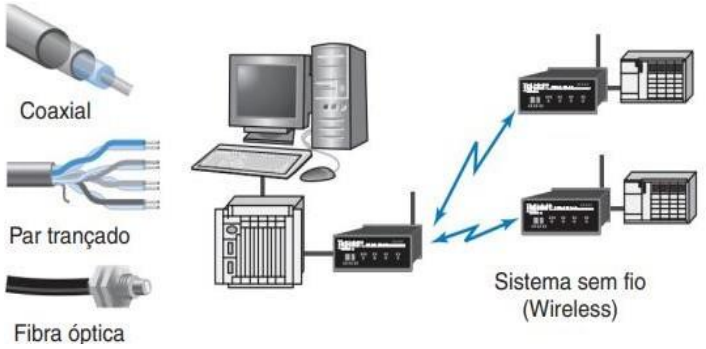


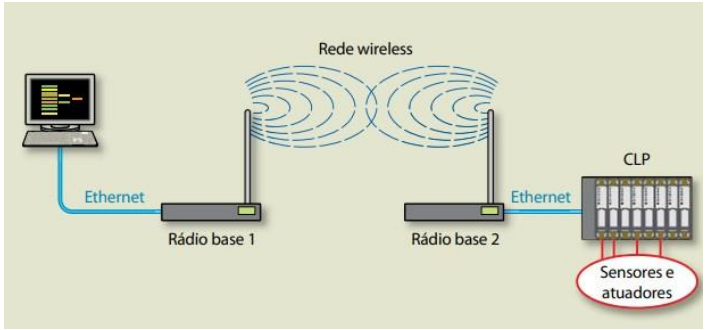


Redes de comunicação

- ✓ A **internet** é exemplo de sistema de comunicação mais amplo existente, o qual interliga muitas redes de computadores
- ✓ Abaixo exemplos de redes de comunicação e meios de transmissão de dados

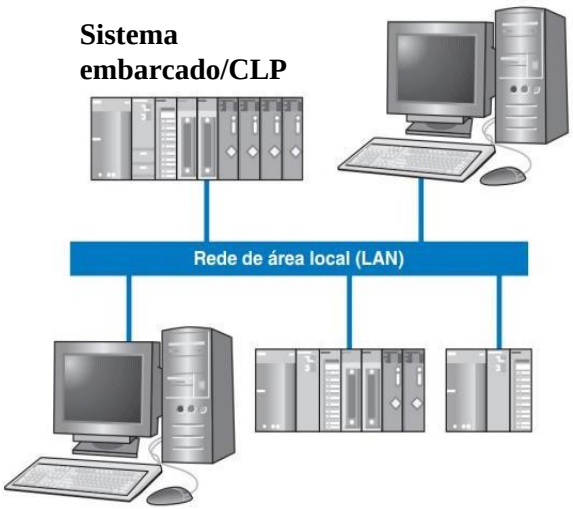


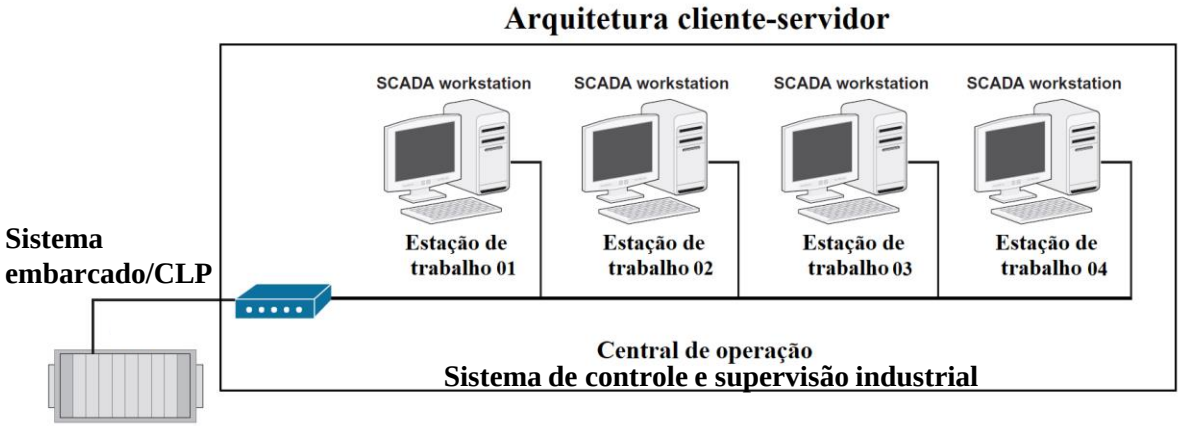




Classificação de redes de computadores

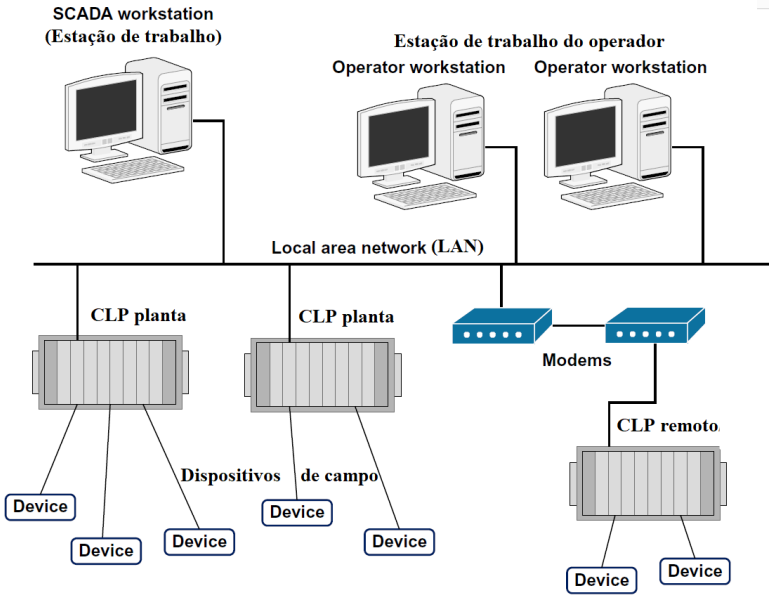
- **LAN (*Local Area Network*):** coleta de dados e processamento para um grupo de controladores, usando um computador host como ponto central. **Exemplo abaixo com arquitetura cliente-servidor**
- **MAN (*Metropolitan Area Network*)** – interconexão de computadores em locais diferentes da mesma cidade. Pode usar a rede telefônica pública ou linha dedicada. Até 50 km.
- **WAN (*Wide Area Network*):** interconexão em qualquer ponto do mundo via satélite/cabos submarinos.

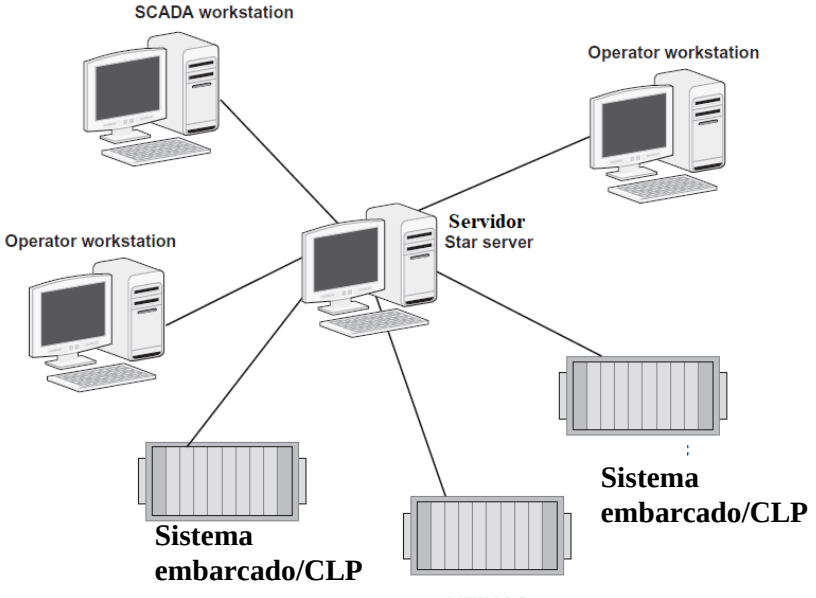




Arquitetura "cliente-servidor"

- ✓ **Servidor/*server*:** um computador/SBC (geralmente com mais recursos e alta capacidade) que fornece serviços para um "cliente".
  - ✓ **Serviços:** controle de usuário, armazenar arquivos e páginas web, compartilhamento de dados, backup; comunicação, banco de dados, histórico etc.
- ✓ **Cliente/*client***: computador/sistema embarcado que recebe/consome os serviços acima – Ex.: por meio do navegador solicita ao servidor, por meio do endereço, acesso à uma página web

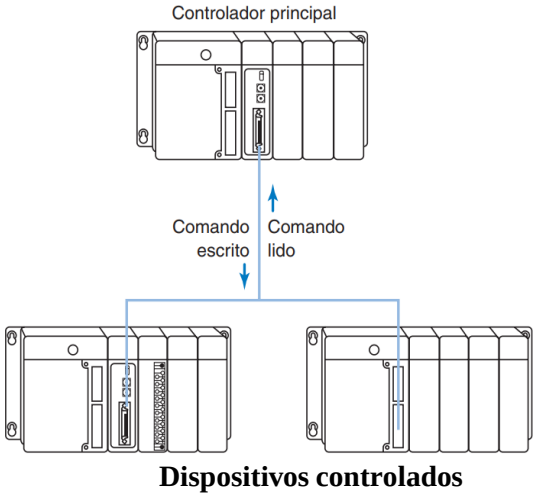


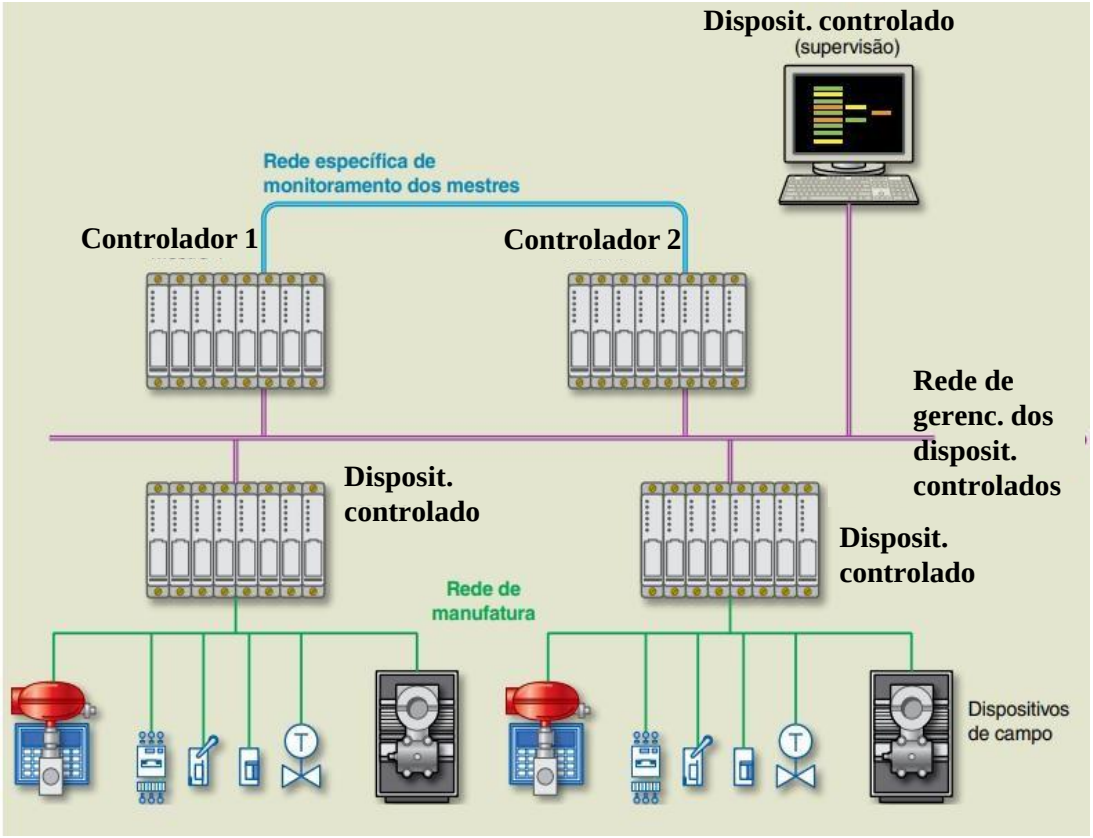


Arquitetura "controlador-dispositivo controlado"

- ✓ Por muito tempo, a nomenclatura "master-slave" foi usada em redes de comunicação.
  - ✓ Entretanto, devido ao termo pejorativo, recentemente existem apelos para adoção de uma nova nomenclatura: https://www.allaboutcircuits.com/news/how-master-slave-terminology-reexamined-in-electrical-engineering/

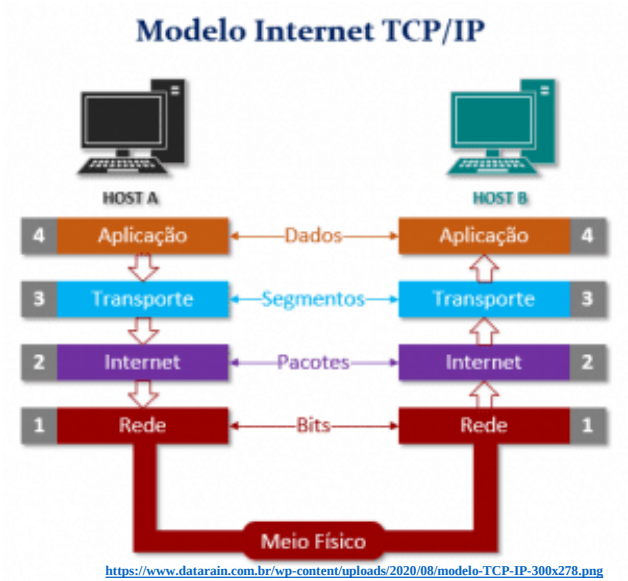
- ✓ O conceito remete à: um **dispositivo controlador** e **um ou mais dispositivos controlados** em uma rede de comunicação.
- ✓ **Controlador:** controla todas as comunicações oriundas de outros dispositivos – ocupa a função de enviar dados para os dispositivos controlados.
- ✓ **Dispositivos controlados:** reponde a solicitação de dados (endereço/leitura/escrita) do controlador



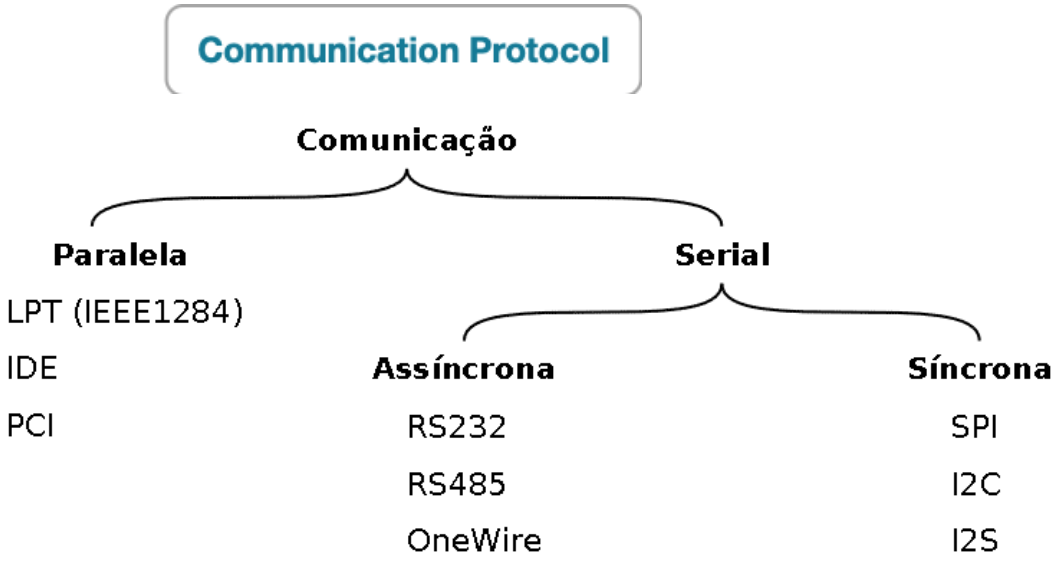


Protocolos de comunicação

- ✓ Protocolos industriais evoluíram da necessidade de enviar e receber informações padronizadas em curtas ou longas distâncias e em tempo determinístico.
- ✓ São instruções e um conjunto de regras ou diretrizes que orientam a comunicação entre computadores, sistemas embarcados e periféricos.
- ✓ Determinam como uma "mensagem" deve circular em uma rede; os mais importantes são **TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*).**
  - ✓ Enquanto **Ethernet** é a arquitetura de interconexão cabeada mais utilizada para redes LAN; o protocolo TCP/IP permite a comunicação/endereçamento entre dois ou mais computadores na rede.



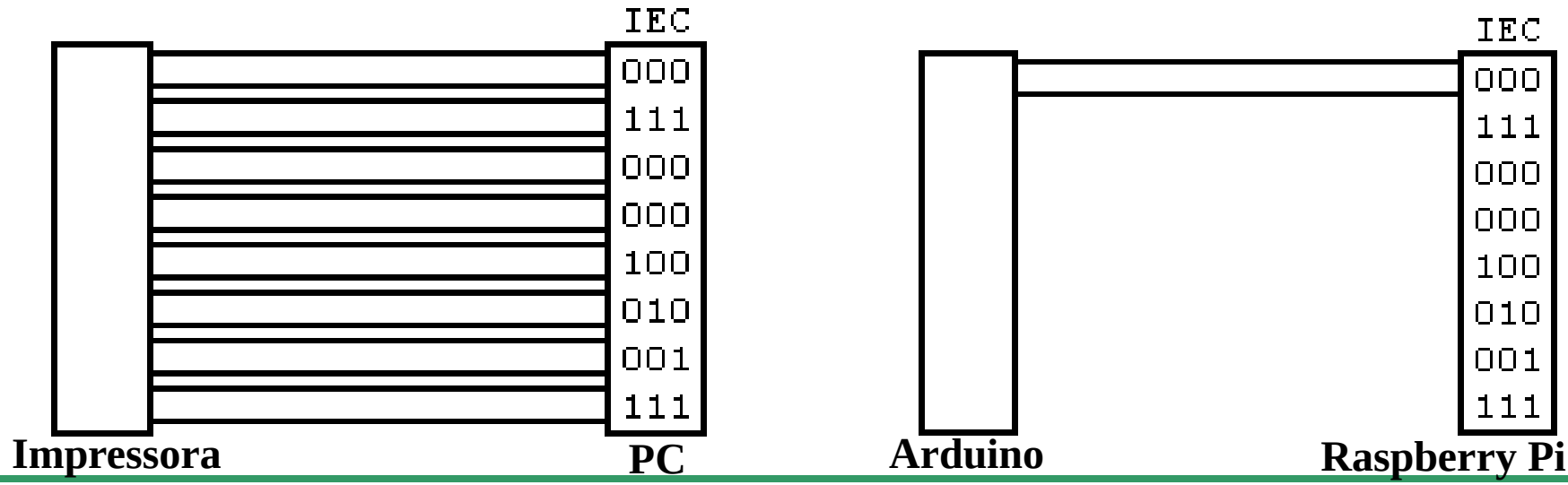
https://www.datarain.com.br/wp-content/uploads/2020/08/modelo-TCP-IP-300x278.png



Comunicação Serial *vs.* Paralela

- ✓ Quando bits são transferidos na forma **serial** de um ponto para outro, eles são enviados um bit de cada vez ao longo de um único canal;
- ✓ Quando bits são transferidos de forma **paralela**, todos os bits de um grupo são enviados em linhas separadas, mas ao mesmo tempo.
- ✓ **Protocolos de com. Serial:** I2C, UART, SPI, RS-232, USB;
- ✓ **Protocolos de com. Paralela:** PCI, IEEE-488

Exemplo de transmissão da mensagem IEC (com codificação ASCII): I= 01001001; E= 01000101; C= 01000011 (colunas)

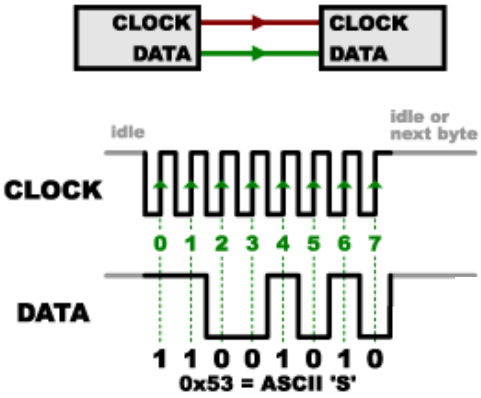


Características da transmissão de dados

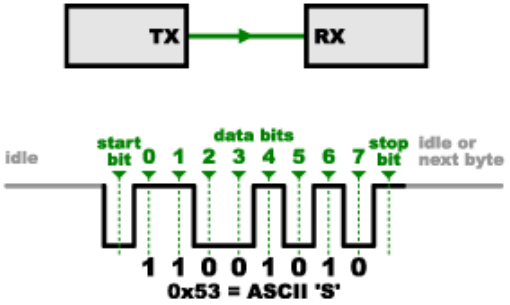
- ✓ **Síncrona:** os dados (bits, bytes) enviados depende um sinal **clock** – transmissão mais rápida seguindo um ordem bem definida, mas depende de um fio extra para o **clock.**
- ✓ **Assíncrona:** dados enviados periodicamente (ou em qualquer ordem), não depende de clock e mais suscetível a erros – os dois dispositivos devem usar mesma taxa;
- ✓ **Taxa de transmissão (bits/segundos)**: velocidade da comunicação – ex. 9600 bits/s

- ✓ **Full-duplex**: indica que o dispositivo pode transmitir e receber dados ao mesmo tempo
- ✓ **Half-duplex**: o dispositivo pode enviar ou receber, mas não de forma simultânea.
- ✓ **Simplex:** comunicação é unidirecional, o dispositivo apenas envia ou recebe
- ✓ **RX** é o termo usado para representar o pino **receptor** de uma comunicação serial
- ✓ **TX** representa o **transmissor**.

Síncrona



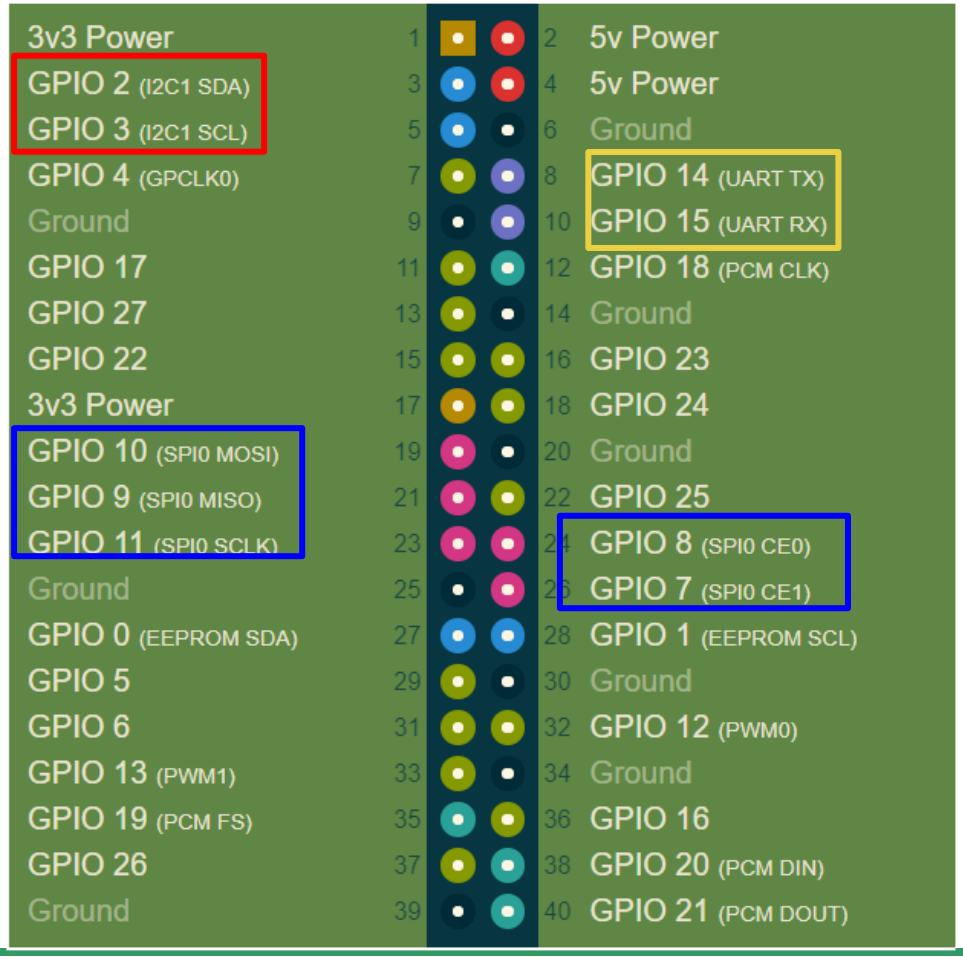
Assíncrona



https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi https://www.robocore.net

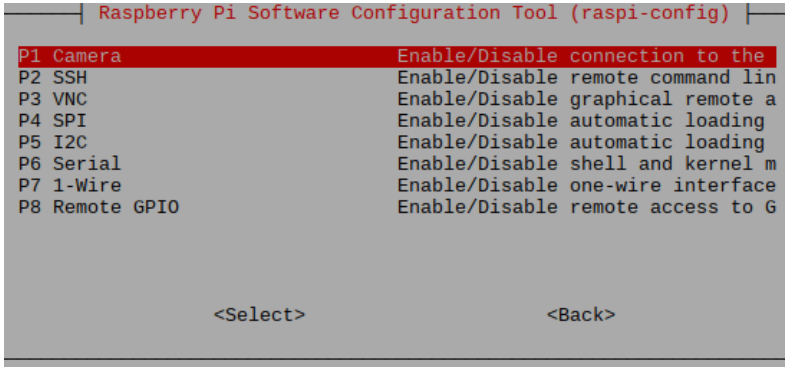
Interfaces de comunicação serial na Raspberry Pi

✓ **GPIO: UART, SPI, I2C, 1-Wire https://pinout.xyz**



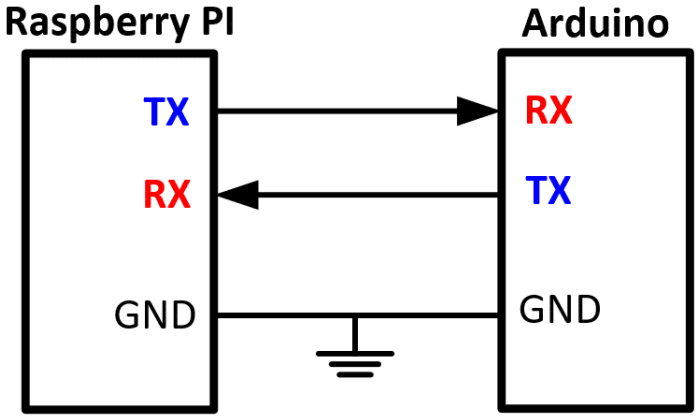
✓ **Para habilitar: *sudo raspi-config***

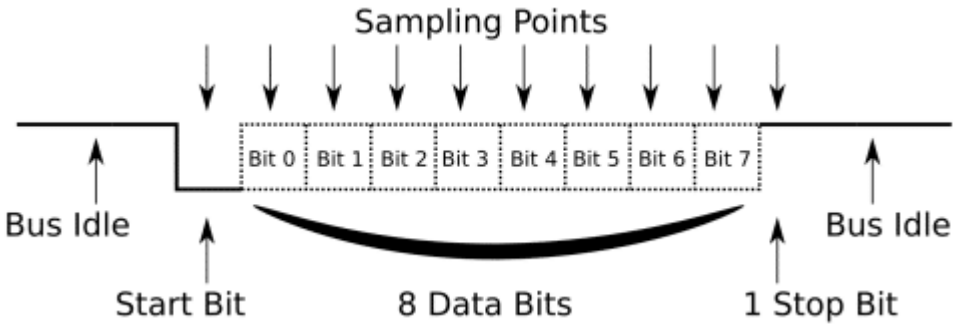
✓ **Acessar "Interfaces"**



Comunicação UART

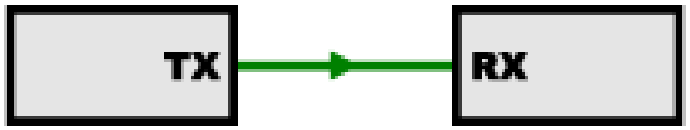
- ✓ ***Universal Asynchronous Receiver/ Transmitter***
- ✓ Protocolo de dois fios que habilita comunicação serial assíncrona entre dispositivos (**formato full-duplex**).
- ✓ Por meio do pino **TX**, um pacote de bits é enviado e interpretado bit a bit pelo pino receptor (**RX**).
- ✓ Cada pacote enviado contém **1 start bit** que indica o início da mensagem, **1 ou 2 stop bits** para indicar o final da mensagem, **5 a 9 bits de informação**, e **1 bit de paridade** para evitar a recepção de erros.



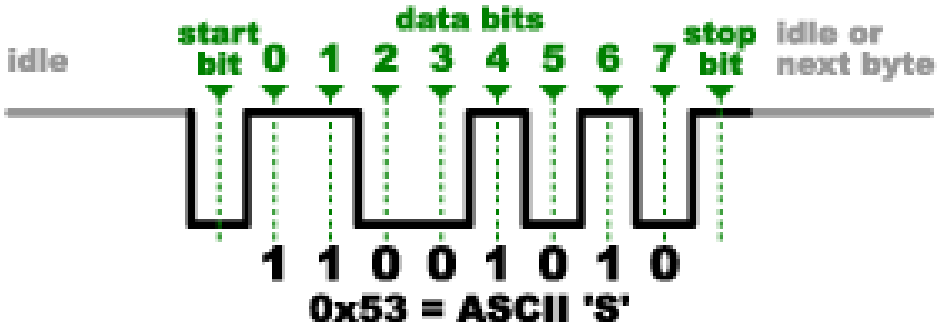


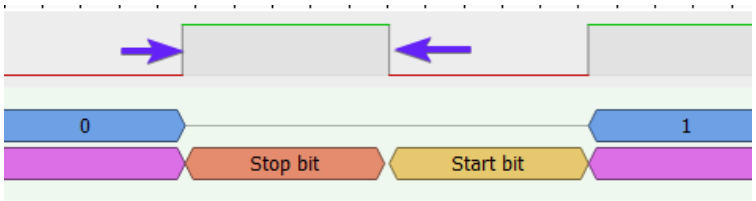
https://embarcados.com.br/raspberry-pi-comunicacao-serial-uart/

Comunicação UART

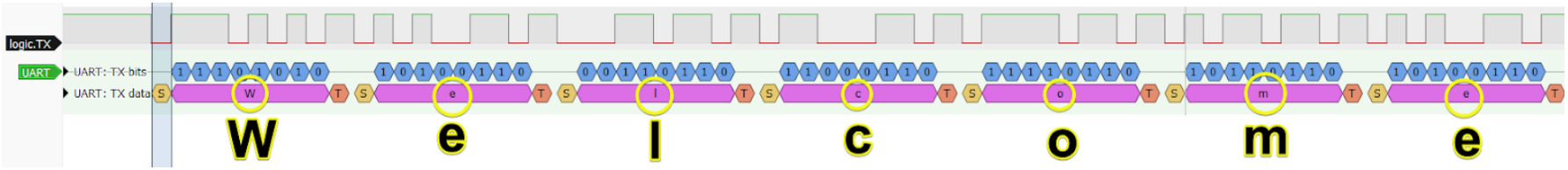


Condições dos bits start (0) e stop(1)



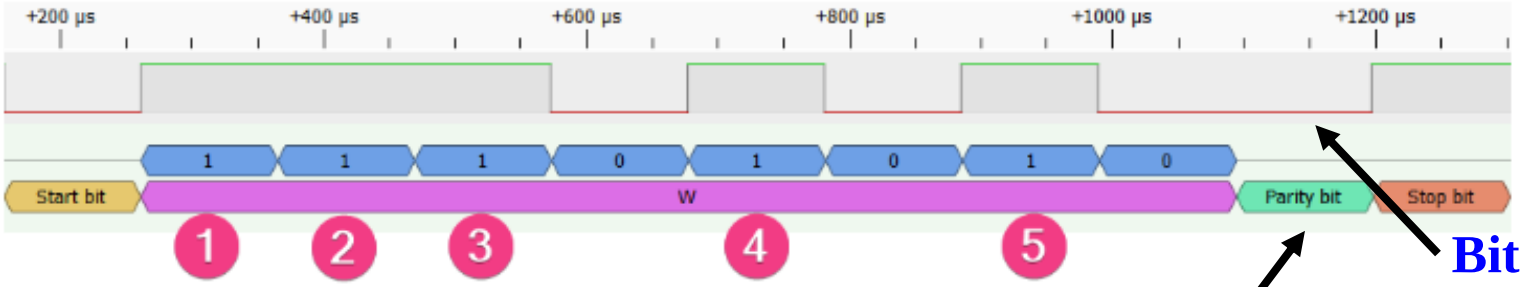


Exemplo de envio de uma mensagem via UART



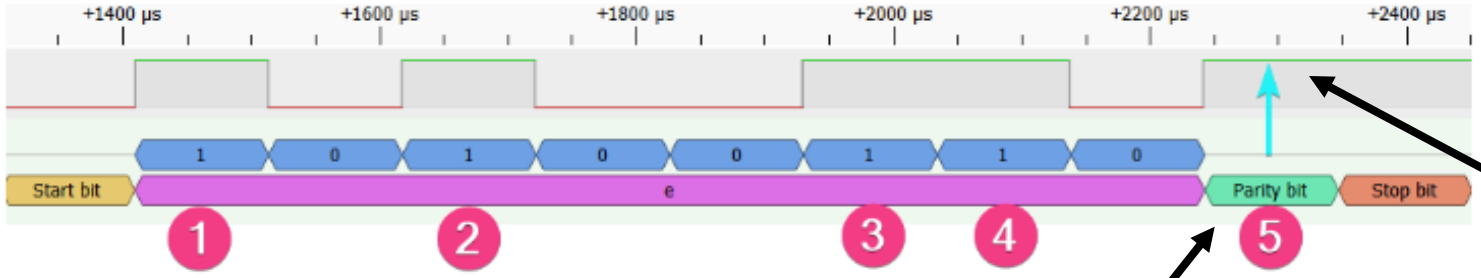
Paridade

- A paridade é usada para verificar se a transmissão ocorre sem erros. Se o número de bits "1" for par, o bit de paridade será "1" (paridade par)



Número de bits "1" = 5, um número ímpar!

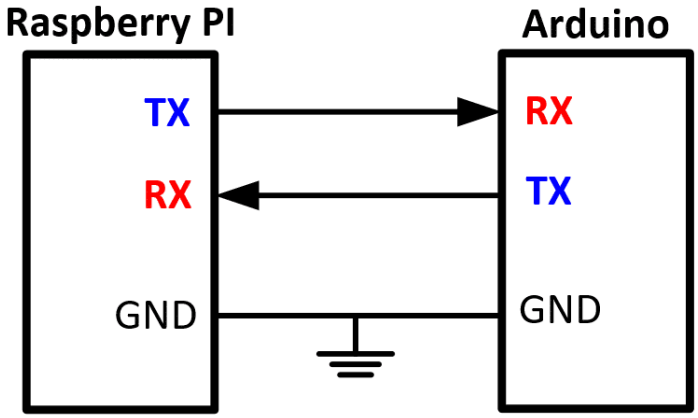
Bit de paridade = 0 (ímpar)



Bit de paridade = 1 (par)

Número de bits "1" = 4, portanto, par!

Comunicação UART



Programa em Python (Raspberry Pi)

```python
import serial
import time

bus = serial.Serial('/dev/ttyS0', 9600, timeout=1)
def enviar_comando(comando):
    ser.write(comando.encode())

try:
    while True:
        comando = input("Digite '1' para ligar o LED ou '0' para desligar: ")
        if comando == '1' or comando == '0':
            enviar_comando(comando)
        else:
            print("Comando inválido. Digite '1' ou '0'.")
except KeyboardInterrupt:
    bus.close()
    print("Comunicação encerrada.")
```

Acender/apagar um LED via UART, no Arduino, enviando 1 ou 0 a partir da Raspberry Pi

```arduino
int ledPin = 13;

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
```

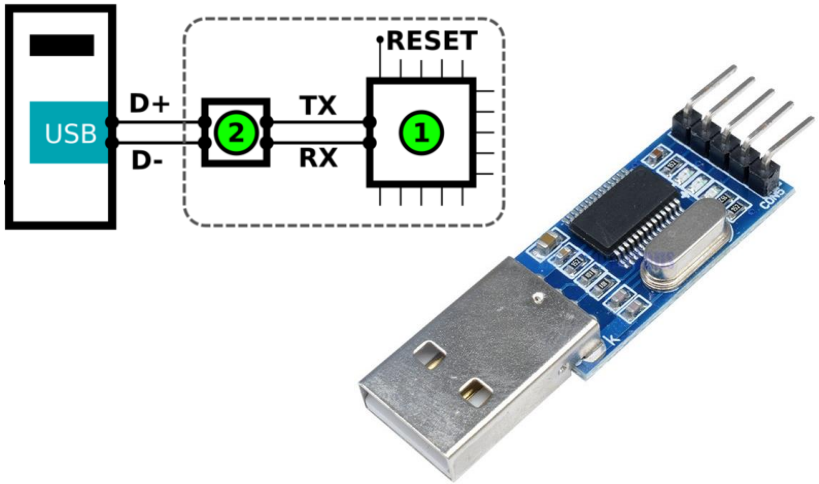
Programa em C (Arduino IDE)

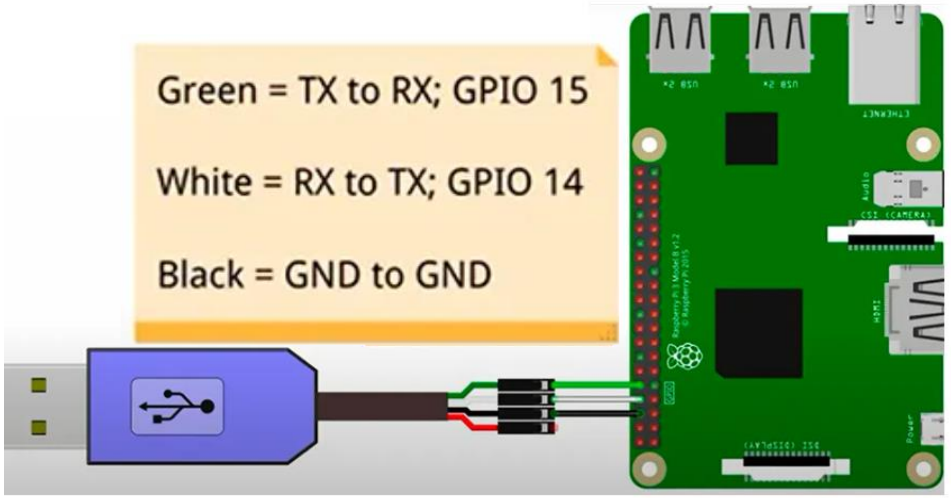
Continuação

```arduino
void loop() {
  if (Serial.available() > 0) {
    char comando = Serial.read();
    if (comando == '1') {
      digitalWrite(ledPin, HIGH);
      Serial.println("LED ligado");
    } else if (comando == '0') {
      digitalWrite(ledPin, LOW);
      Serial.println("LED desligado");
    }
  }
}
```

Comunicação UART

- ✓ **Outro exemplo: Acessando Rasp no PC via console serial (UART)**
  - Conforme slide anterior, é possível a comunicação da Rasp. com Arduino (ou outros sistemas embarcados) via UART, por meio dos pinos **TX** e **RX**.
  - Também é possível a comunicação serial UART com um PC, i.e., acesso a Rasp a partir de um terminal Linux no PC visa console serial.
  - Para tanto, é necessário utilizar um dispositivo: **USB-TTL** (um conversor que permite usar a **porta USB** no PC, com porta **serial RS-232 e em nível TTL**, sendo compatível com sistemas embarcados que operam em **5V ou 3.3V**).
  - Como isso é possível conectar a Rasp via pinos TX e RX.





https://electronicshacks.com/raspberry-pi-serial-uart-tutorial/

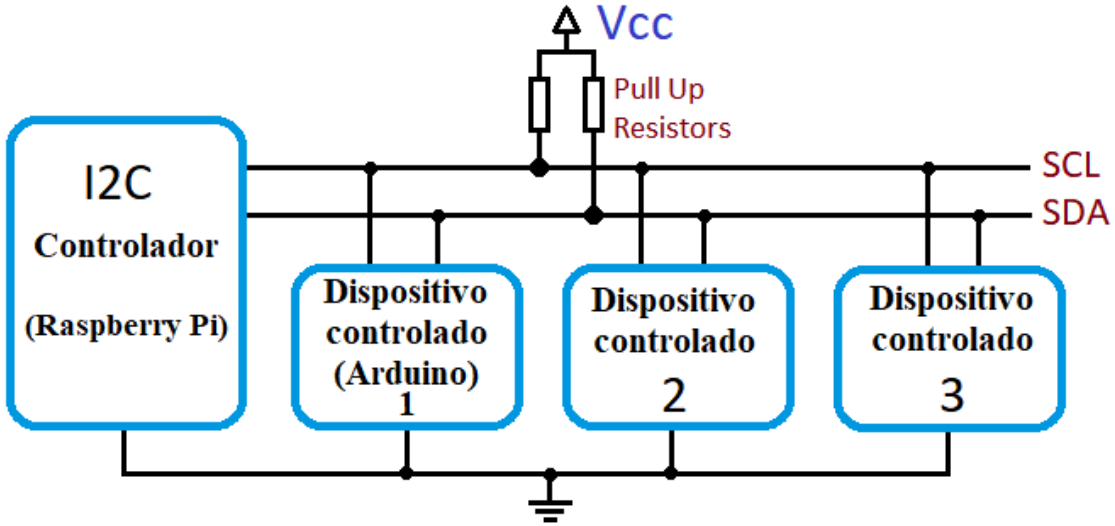
Comunicação UART

- ✓ **Acessando Rasp no PC via console serial (UART)**
  - Vantagens: não requer conexão de mouse, teclado e monitor diretamente na Rasp.
  - Desvantagem: não possui interface gráfica – o acesso é somente via terminal
  - Habilitar em <sudo raspi-config> *interface options* – habilitar a opção *"serial"*

```
#Na rasp:  realizar ligação conforme slide anterior e conectar normalmente a fonte de alimentação da rasp
sudo raspi-config #em interfaces = habilitar serial port

sudo nano /boot/cmdline.txt #excluir linhas após "rotwait"

sudo nano/boot/config.txt # colocar "enable_uart=1"

# No PC (Linux debian):

sudo apt install screen

# lista a conexão USB serial- geralmente USB0

ls /dev/tty* # USB0

sudo screen /dev/ttyUSB0 115200

#login e senha da rasp - o sistema deve inicializar
```

Comunicação I2C

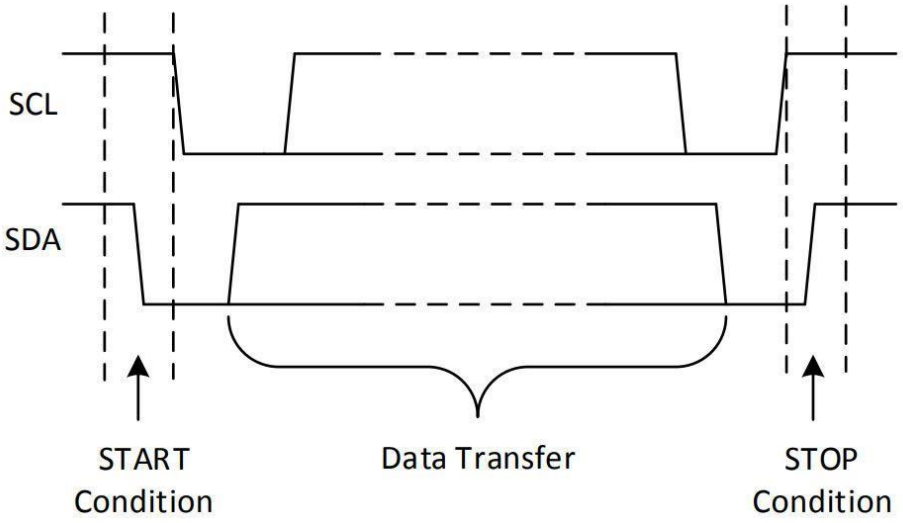
- ✓ ***Inter-Integrated Circuit Bus*** é um protocolo de dois fios que habilita a comunicação serial **sincronizada** entre dois ou mais dispositivos no formato **Half-duplex**.
- ✓ Consiste de um barramento "*controller*" (dispositivo controlador) e um ou mais barramentos "*responder*" (dispositivos controlados).
  - **SDA – *the serial data line*** - conexão de dados bidirecional que habilita a comunicação entre dispositivo controlador e dispositivo(s) controlado(s).
  - **SCL – *the clock line*** - fornece um sinal de **clock** para sincronização

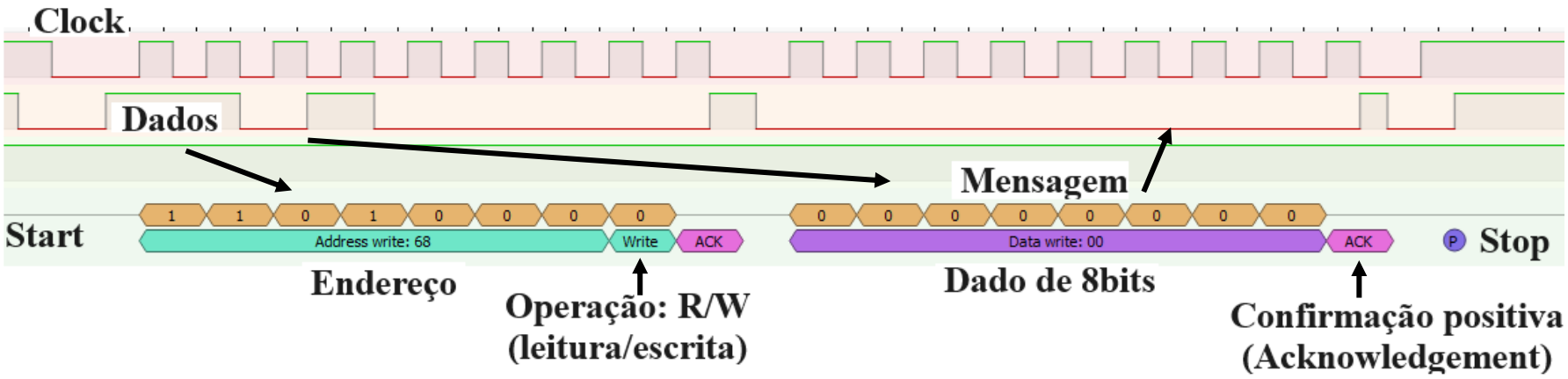


https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi https://www.robocore.net

Barramento I2C

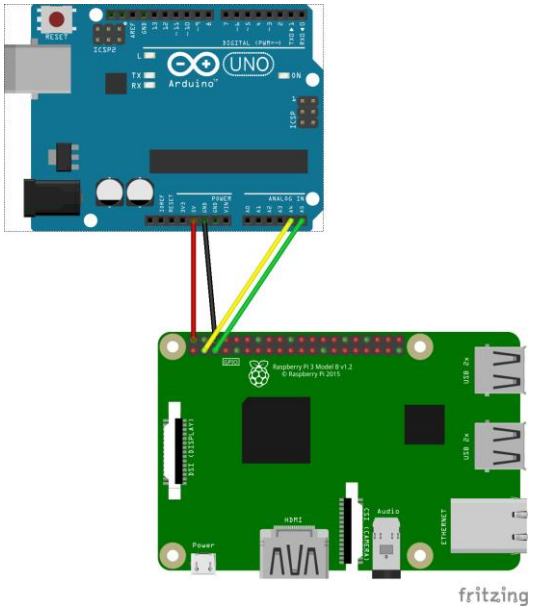
Comandos: START, STOP, R – read; W – write
ACK (confirmação positiva) e NACK (confirmação negativa)





Barramento I2C

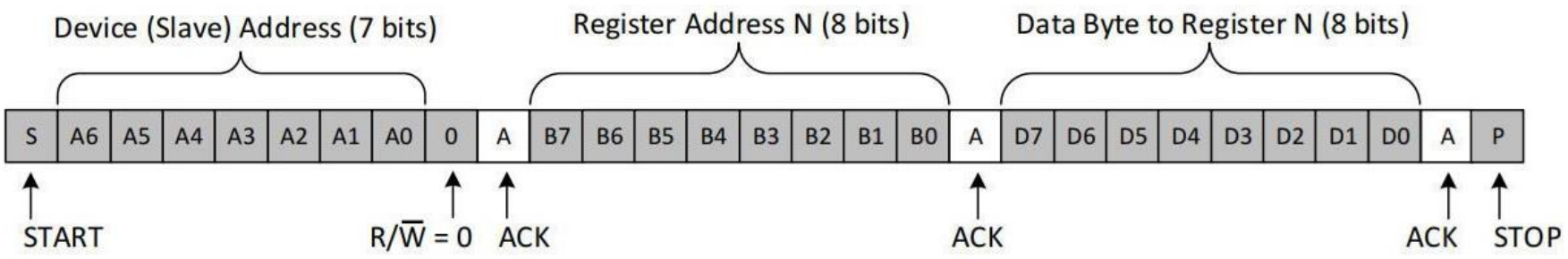
ACK (confirmação positiva – dados recebidos com sucesso!)
NACK (confirmação negativa – dado não recebido, endereço não atribuído, ou não é mais necessário envio de dados/liberar barramento
O byte de registrador é uma convenção usada para indicar qual dado ou configuração específica o controlador deseja acessar dentro do disp. Controlado.



Controlador

Disp. Controlado

Write to One Register in a Device



Comunicação serial entre Raspberry Pi e Arduino via I2C

- ✓ **Raspberry Pi** já é bem conhecida como uma plataforma versátil com diversas funcionalidades de alto nível
- ✓ Entretanto, sendo uma SBC que roda um sistema operacional com kernel Linux, possui suas limitações quando se trata de **temporização precisa**.
- ✓ Ademais, a Raspberry Pi por si só também não é capaz de lidar com **dados analógicos**.
- ✓ Por essa razão, em determinadas aplicações se faz necessário o uso dela em conjunto a um **microcontrolador** (que por sua vez possui **hardware de tempo real e conversor A/D**).

- ✓ A plataforma Arduino se apresenta como uma solução alternativa para atender o propósito acima (poderia ser outro microcontrolador).
- ✓ A Raspberry Pi será a plataforma de alto nível "**controladora**" que irá requisitar informações (no caso, dados analógicos) do Arduino que, na comunicação I2C, será o dispositivo **"controlado"**. E plataforma de mais "baixo nível" e mais adequada no quesito tempo real.
- ✓ Evidentemente, no caso do Arduino, essa função será cumprida parcialmente, tendo vista suas limitações.
- ✓ Existem microcontroladores mais modernos e com arquiteturas mais adequadas para cumprir essa função.

O problema da comunicação I2C entre Raspberry Pi e Arduino

- ✓ A interface entre elas é um desafio em razão de **não operarem no mesmo nível lógico de tensão;**
- ✓ A Raspberry Pi opera com 3.3 V, ao passo que o Arduino Uno (assim como diversos outros modelos) opera com 5 V.

- ✓ Do ponto de vista da lógica de tensão, não existe diferença entre **5V ou 3.3V (ou 2.2 V, 1.8V e 0.8V);**
- ✓ Do ponto de vista da tecnologia, **valores de tensão mais baixos permitem o uso de transistores menores e uma resposta mais rápida.**
- ✓ Tensões mais baixas também sinalizam menores oscilações entre dois níveis lógicos, portanto, garantindo resposta mais rápida, além de dissipar menos energia/menor consumo de energia (permite fontes de menor potência).
- ✓ A desvantagem de valores mais baixos (ex.: 3V) é a **imunidade ao ruído prejudicada** em razão do sinal mais próximo ao nível de ruido e a **necessidade de conversores** em casos de periféricos que operam com valor de tensão maior (ex.: 5V)

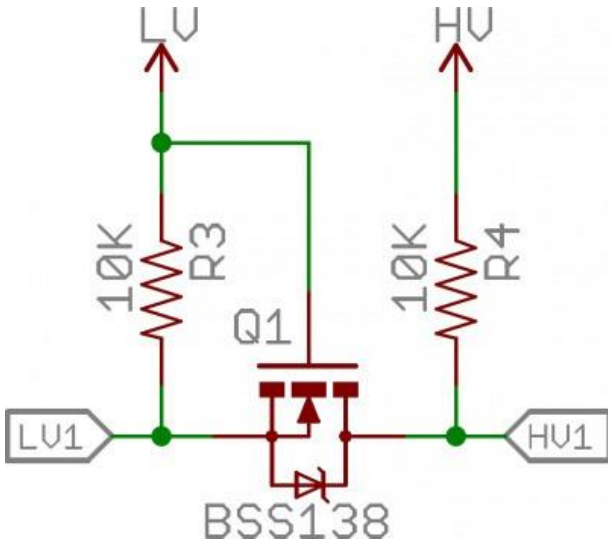
O problema da comunicação I2C entre Raspberry Pi e Arduino

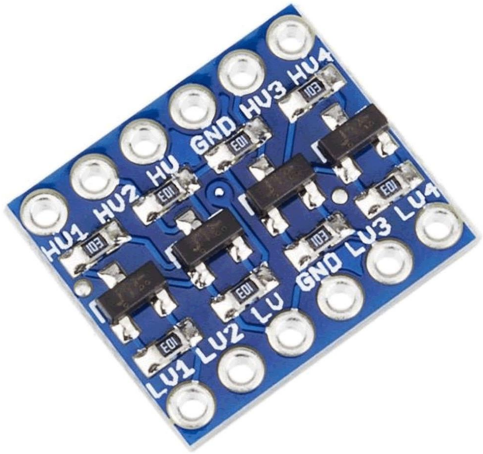
- ✓ Importa destacar que resistores ***pull-up*** adequam os níveis lógicos e de **clock** ao nível de tensão de referência (VCC) .
- ✓ Cumpre também frisar que na comunicação I2C o nível lógico de tensão é determinado pelo dispositivo controlador. Isto posto, o seguinte cuidado deve ser tomado:
  - ✓ (i) **uma saída 3.3 V da Raspberry Pi pode ser conectada à uma entrada 5V do Arduino**;
  - ✓ (ii) **uma saída 5 V do Arduino NÃO deve ser conectada à um entrada 3.3 V da Raspberry Pi**

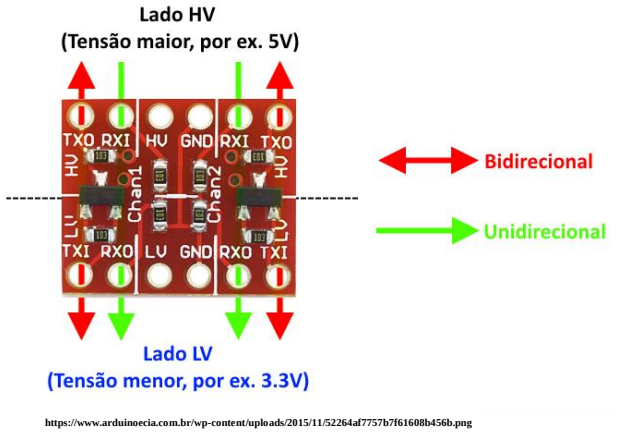
- ✓ A solução mais adequada para este problema é usar um **conversor de nível lógico bidirecional 3.3V - 5V (*Bidirectional Logic Level Converter* ou *"level-shifter"*).**
- ✓ É um circuito que recebe e converte sinais de tensão de nível lógico 5V para 3.3V e vice-versa, fornecendo uma comunicação segura entre os dispositivos.

Level-shifter para conversão 5V-3.3V

- ✓ **Lado "*low voltage*" (Raspberry Pi)** - 4 canais de conversão de níveis lógicos (LV1, LV2, LV3 e LV4), um canal para alimentação, denominado "**LV**" e **GND**.
  - **Os pinos SDA e SCL da Rasp. (GPIO 2 e GPIO3) devem se conectar à LV1 e LV2, ou LV3 e LV4**, respectivamente.
- ✓ **Lado "*high voltage*" (Arduino) -** 4 canais de conversão de níveis lógicos (HV1, HV2, HV3 e HV4), um canal para alimentação, denominado "**HV**" e **GND**.
  - **Os pinos SDA e SCL do Arduino (A4 e A5) devem se conectar à HV1 e HV2, ou HV3 e HV4**, respectivamente.

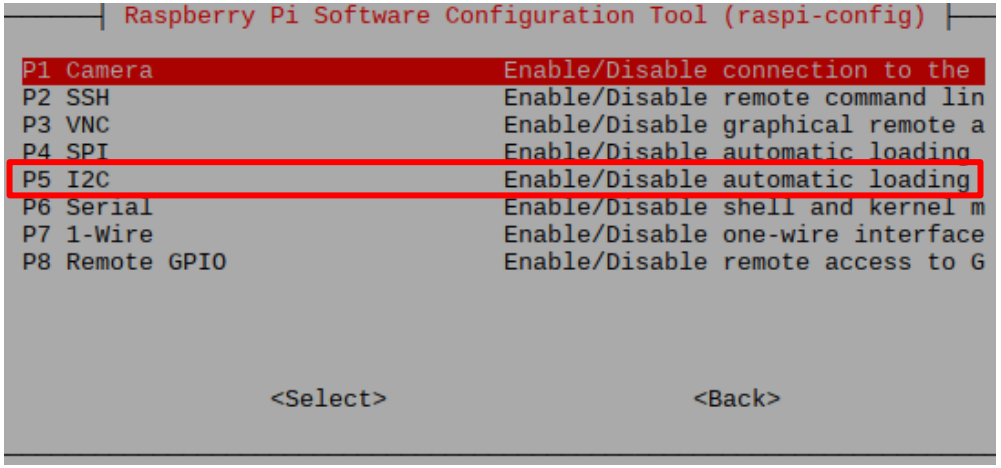


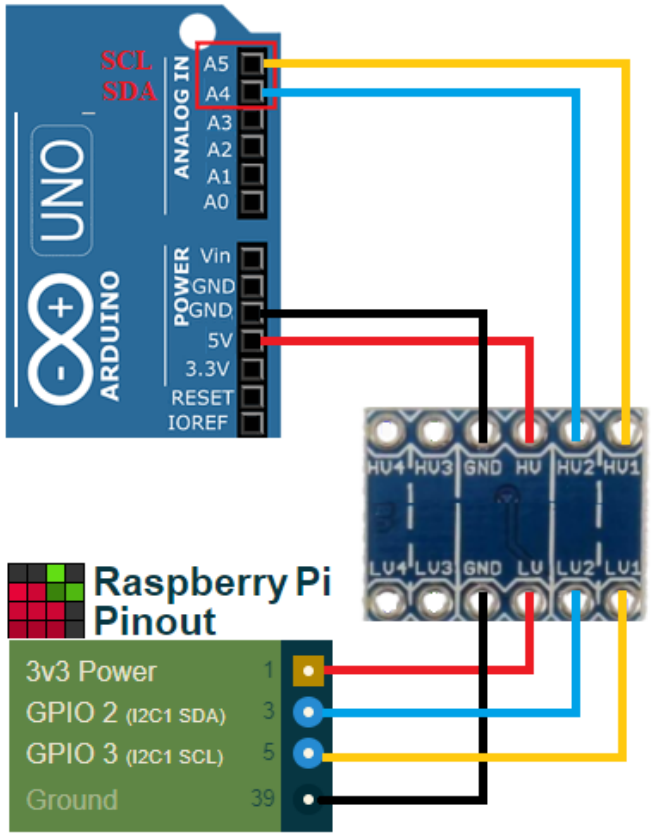




Solução do problema usando o level-shifter

- ✓ **Habilitando a comunicação entre Rasp e Arduino via I2C**
  - É necessário habilitar o I2C a partir de <sudo raspi-config> *interface options*;
  - Deve-se utilizar os **pinos 3 e 5 (GPIO 2 e GPIO 3)** para SDA e SCL;





Raspberry Pi				Arduino		
I2C	**GPIO**	**N°**	***Level- shifter***	**I2C**	**GPIO**	***Level-shifter***
SDA	2	3	LV3	SDA	A4	HV3
SCL	3	5	LV4	SDA	A5	HV4
VCC	3.3 power	1	LV	VCC	5V	HV
GND	Ground	9*	GND	GND	GND	GND

Explorando recursos da comunicação I2C na Rasp (parte Alto Nível)

- ✓ **Habilitando a comunicação entre Rasp e Arduino via I2C**
  - Para verificar o barramento I2C na Rasp, execute: <**sudo i2cdetect -y 1**> (output na imagem abaixo indica o endereço da conexão no barramento I2C)

- ✓ Em Python, a biblioteca responsável por gerenciar a comunicação é o **SMBus.**
- ✓ **Documentação SMBus:** https://buildmedia.readthedocs.org/media/pdf/smbus2/latest/smbus2.pdf

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- 0b -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Biblioteca SMBus (parte Alto Nível em Python na Raspberry Pi)

✓ Conhecendo recursos do módulo **SMBus** em Python para permitir a Rasp. controlar outro dispositivo (Arduino, no caso) via I2C.

```
# Mais detalhes: documentação SMbus
https://buildmedia.readthedocs.org/media/pdf/smbus2/latest/smbus2.pdf


Para acessar o barramento I2C na Rasp usando o módulo SMBus:

import smbus # ou from smbus import SMBus


# Criar objeto de classe SMBus para acessar I2C
    #<Object name> = smbus.SMBus(I2C port no.)
    # ou <Object name> = SMBus(I2C port no.)

     # I2C port no : I2C port no. i.e. 0 or 1
    #Exemplo:
Bus = smbus.SMBus(1) #ou - Bus = SMBus(1)
#============================================================================
# Agora é possível acessar a classe SMBus com objeto "bus"
# <bus.write_byte_data(Device Address, Register Address, Value)>
  # Função usada para escrever dados no registrador solicitado:
  #Device Address : 7-bit or 10-bit device address
  #Register Address : Registrador de endereço necessário para escrita
  #Value : valor necessário para escrita no registrador
  #Exemplo:

Bus.write_byte_data(0x68, 0x01, 0x07)
```

Biblioteca SMBus (parte Alto Nível em Python na Raspberry Pi)

✓ Conhecendo recursos do módulo **SMBus** em Python para permitir a Rasp controlar outro dispositivo (Arduino, no caso) via I2C. Continuação...

```
#============================================================================
# Agora é possível acessar a classe SMBus com objeto "bus"
# <bus.write_byte_data(Device Address, Register Address, Value)>
  # Função usada para escrever dados no registrador solicitado:
  #Device Address : 7-bit or 10-bit device address
  #Register Address : Registrador de endereço necessário para escrita
  #Value : valor necessário para escrita no registrador
  #Exemplo:

Bus.write_byte_data(0x68, 0x01, 0x07)
#============================================================================
#bus.write_i2c_block_data(Device Address, Register Address, [value1,
value2,....])
  # Função usada para escrita de bloco de 32 bytes.
  #Device Address : 7-bit or 10-bit device address
  #Register Address : Registrador de endereço necessário para escrita
  #Value1 Value2... : escrita de blocos de bytes no endereço solicitado
  #Exemplo:

Bus.write_i2c_block_data(0x68, 0x00, [0, 1, 2, 3, 4, 5]) # escrita de 6 bytes no
endereço "0"

#============================================================================
```

Biblioteca SMBus (parte Alto Nível em Python na Raspberry Pi)

✓ Conhecendo recursos do módulo **SMBus** em Python para permitir a Rasp controlar outro dispositivo (Arduino, no caso) via I2C. Continuação...

```
#============================================================================
# bus.read_byte_data(Device Address, Register Address)
  #Função para leitura de bytes do registrador
  #Device Address : 7-bit or 10-bit device address
  #Register Address : endereço do registrado requisitado para leitura    de
dados
  #Exemplo:

Bus.read_byte_data (0x68, 0x01)

#============================================================================
#Bus.read_i2c_block_data(Device Address, Register Address, block of bytes)
  #função para leitura de um bloco de 32 bytes
  # Device Address - 7-bit or 10-bit device address
  # Register Address - " "
  #Block of Bytes  - N° de bytes no endereço requisitado
  #Exemple:

Bus.read_i2c_block_data(0x68, 0x00, 8) #  o valor retornado é uma lista de 6
bytes
```

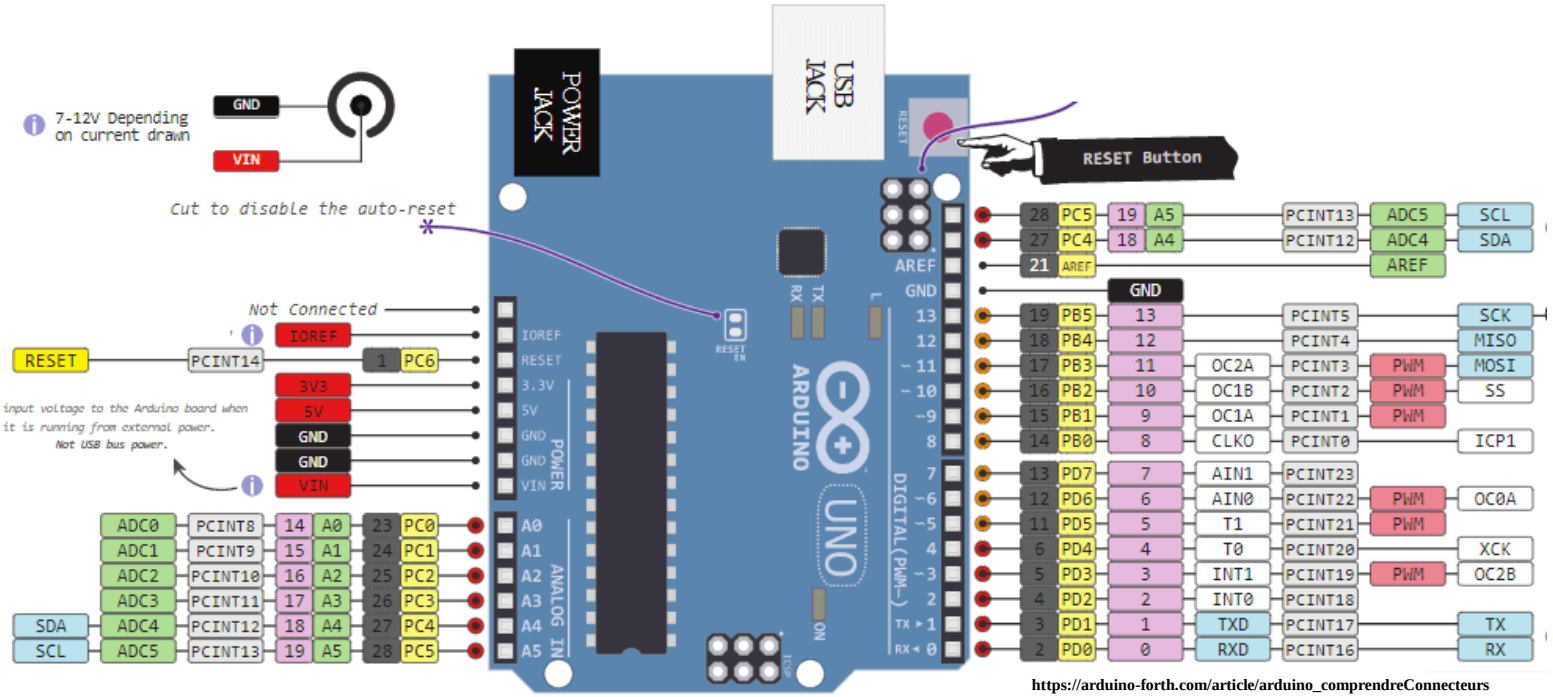
Biblioteca SMBus (parte Alto Nível em Python na Raspberry Pi)

- ✓ **Controlar um LED no Arduino a partir da Rasp. via I2C**
  - Desenvolvendo o Programa em Python para implementar a função de controle e solicitar ao dispositivo controlado (Arduino) que acenda ou apague o LED.

```
#  Tutorial: Raspberry Pi controlando o Arduino
from smbus import SMBus

addr = 0x8 # bus address
bus = SMBus(1) #  /dev/ic2-1

flag = True

print ("Digite 1 para ON ou 0 para OFF")
while flag:

ledstate = input(">>>>   ")

if ledstate == "1":
bus.write_byte(addr, 0x1)
elif ledstate == "0":
bus.write_byte(addr, 0x0)
else:
flag = False


#  Os valores de entrada acima, enviam 1 ou 0 para o barramento I2C - ou 0 em caso de valor de entrada diferente
de 0 ou 1.
```

Explorando a comunicação I2C no Arduino (parte baixo nível)

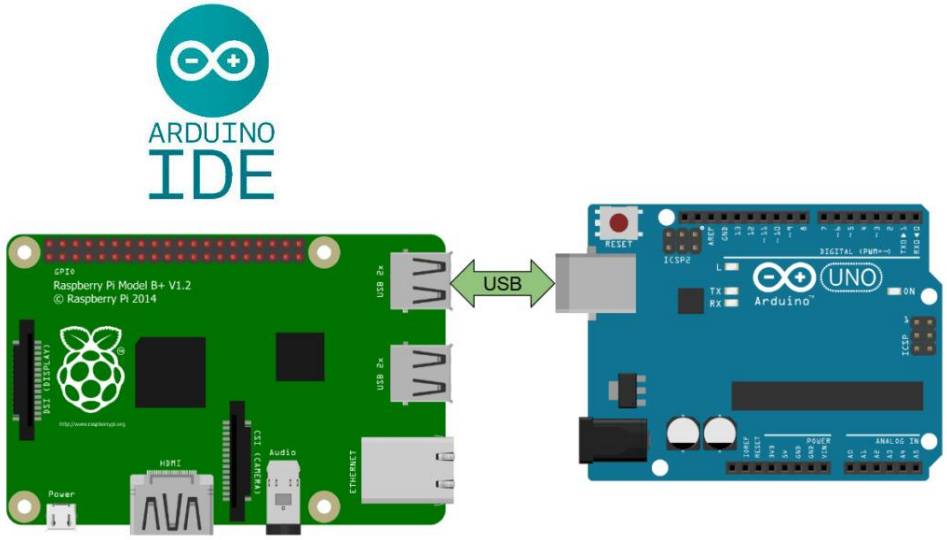
- ✓ **Arduino** é uma plataforma open source de baixo custo para prototipagem eletrônica.
- ✓ **A versão mais popular, modelo UNO, possui um microcontrolador Atmel AVR, ATMega328P, de 8 bits.** Incorpora diversos recursos, conforme segue abaixo.
- ✓ Comunidade: https://www.arduino.cc
- ✓ Book: Get Start with Arduino: https://www.mclibre.org/descargar/docs/revistas/hackspace-books/hackspace-get-started-with-arduino-01-201911.pdf

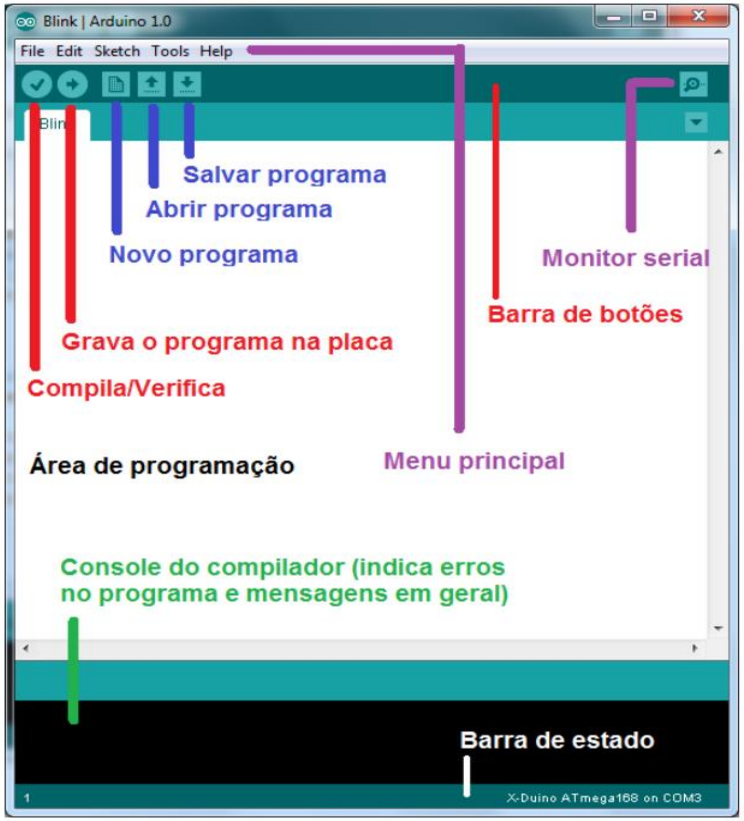


https://arduino-forth.com/article/arduino_comprendreConnecteurs

Explorando a comunicação I2C no Arduino (parte baixo nível)

- ✓ **É possível instalar a IDE do Arduino na Raspberry Pi**
  - Na Rasp, execute: <**sudo apt-get install Arduino -y**> #entretanto, a Rasp. 3B+ é mais limitada e não suporta muito bem uso dessa IDE,podendo travar. Portanto, recomenda-se na aula, usar no PC.



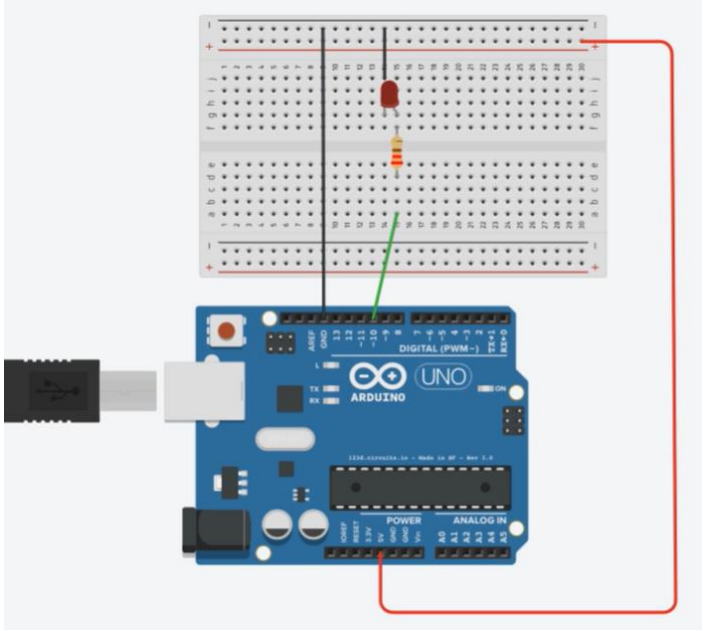


Biblioteca Wire (programa em ling. C no Arduino)

- ✓ **Explorando recursos do Arduino**
  - O **módulo <u>Wire</u>** deve ser utilizado na prática I2C: https://www.arduino.cc/reference/en/language/functions/communication/wire/
  - **Estruturação de programas em C no Arduino: http://www.sel.eesc.usp.br/jcarmo/pdfs/Arduino_SimonMonk_2011.pdf**
  - Ferramentas úteis de prototipagem de circuitos com Arduino: Fritzing e TikerCAD

Blink

```
/*
  Blink

  Pisca um led de dois em dois segundos
 */

//Função de inicialização
void setup() {
  // inicializa o pino digital como saída
  // O pino 13 possui um led integrado na maioria das placas Arduino
  pinMode(13, OUTPUT);
}


//looping principal
void loop() {

  digitalWrite(13, HIGH);   // acende o led conectado no pino 13
  delay(1000);              // espera um segundo (1000 mS)
  digitalWrite(13, LOW);    // apaga o led
  delay(1000);              // espera um segundo (1000 mS)

}
```



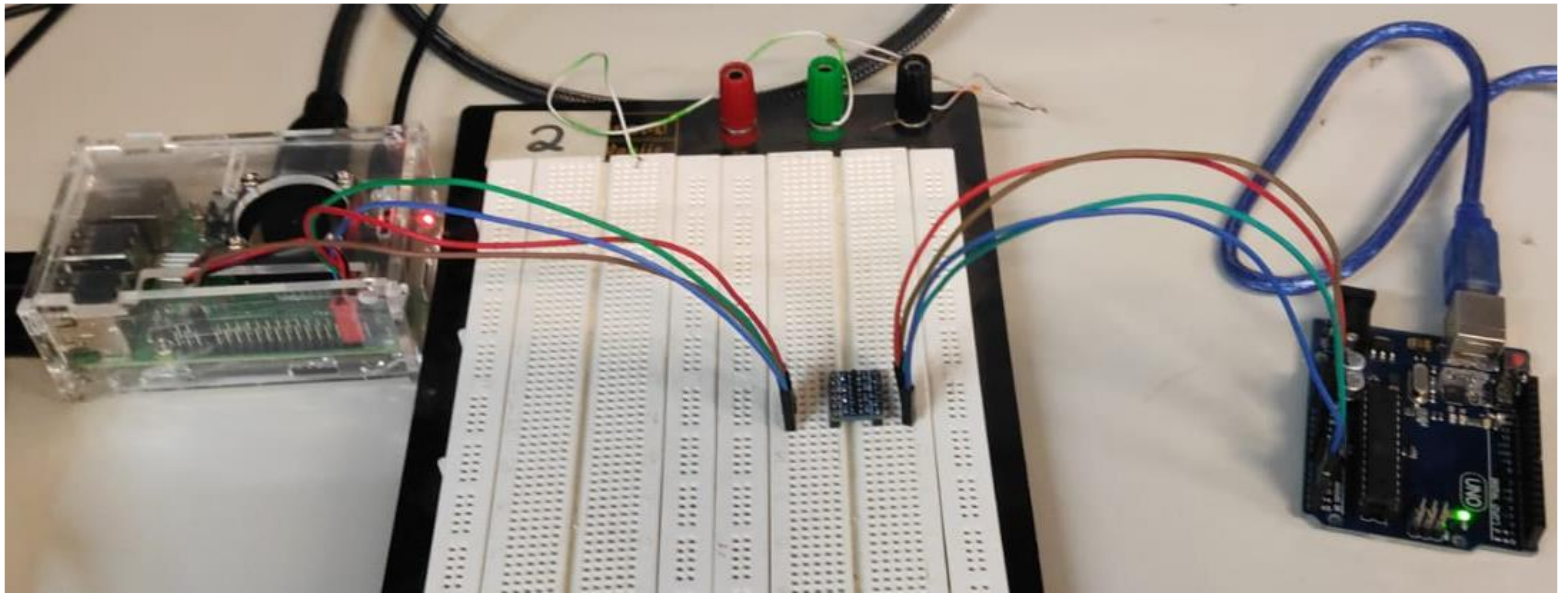
Biblioteca Wire (programa em ling. C no Arduino)

- ✓ **Controlar um LED no Arduino a partir da Rasp. via I2C**
  - Faça com que um LED (o **Arduino já possui um LED em sua placa, pelo nome** LED_BUILTIN) acenda ou apague a partir do valor 0 ou 1 recebido a partir da Rasp.

```
// Tutorial - Arduino como dispositivo controlado na comunicação I2C tendo a Raspberry Pi como controladora
// biblioteca Wire  para  I2C – código em C
#include <Wire.h>

// Controlar o LED da própria placa Arduino
const int ledPin = LED_BUILTIN;

void setup() {
 // adicionando endereço no barramento I2C com dispositivo controlado
  Wire.begin(0x8);

  //Reporar "receiveEvent" quando receber dados
  Wire.onReceive(receiveEvent);

  // Define o pino do LED como saída e o desliga
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
}

// Função executada sempre que dados são recebidos do controlador (Raspberry Pi)
void receiveEvent(int howMany) {
  while (Wire.available()) { // loop
    char c = Wire.read(); // recebe o byte como char
    digitalWrite(ledPin, c);
  }
}
void loop() {
  delay(100);
}
```

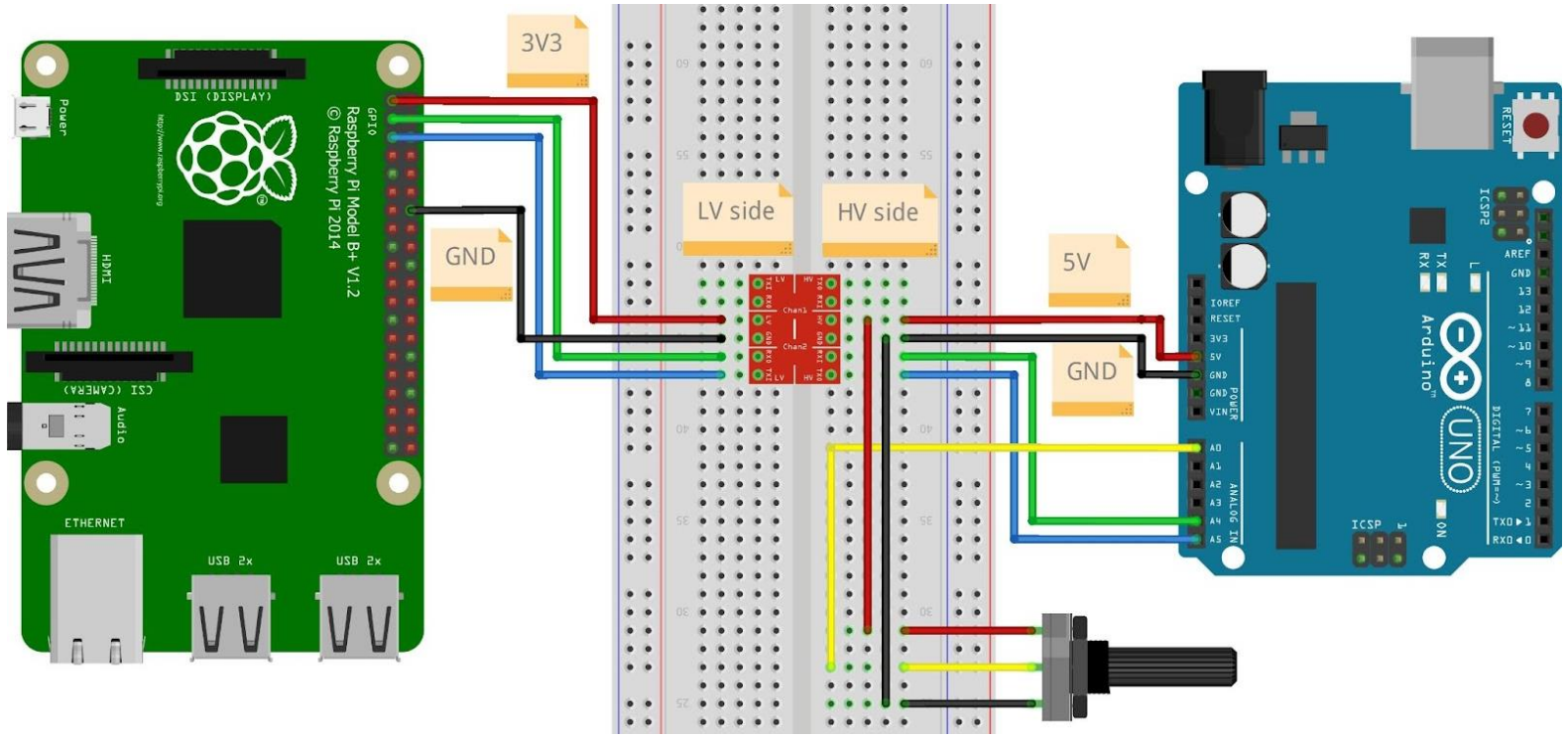
Biblioteca Wire (programa em ling. C no Arduino)

- ✓ **Controlar um LED no Arduino a partir da Rasp. via I2C**
  - – Solução: montagem prática abaixo.



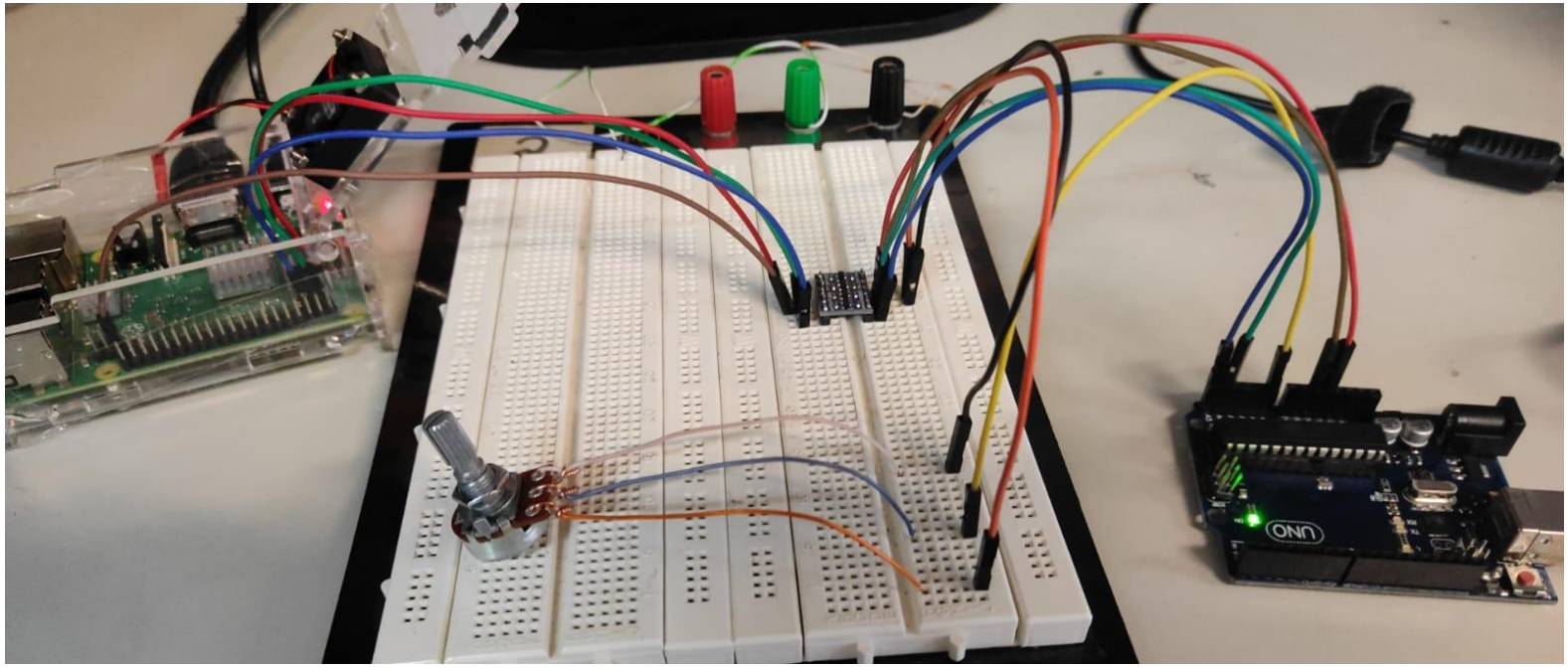
Recebendo dados analógicos lidos pelo Arduino na Raspberry Pi via I2C

- ✓ **Leitura analógica no Arduino usando um potenciômetro**
  - Conectar a um dos **pinos analógicos** do Arduino a um potenciômetro,
  - Realizar leituras analógicas, excursionando o valor recebido de 0 a 5V entre 0 e 1023 (o conversor A/D do Arduino possui resolução de 10 bits)



Recebendo dados analógicos lidos pelo Arduino na Raspberry Pi via I2C

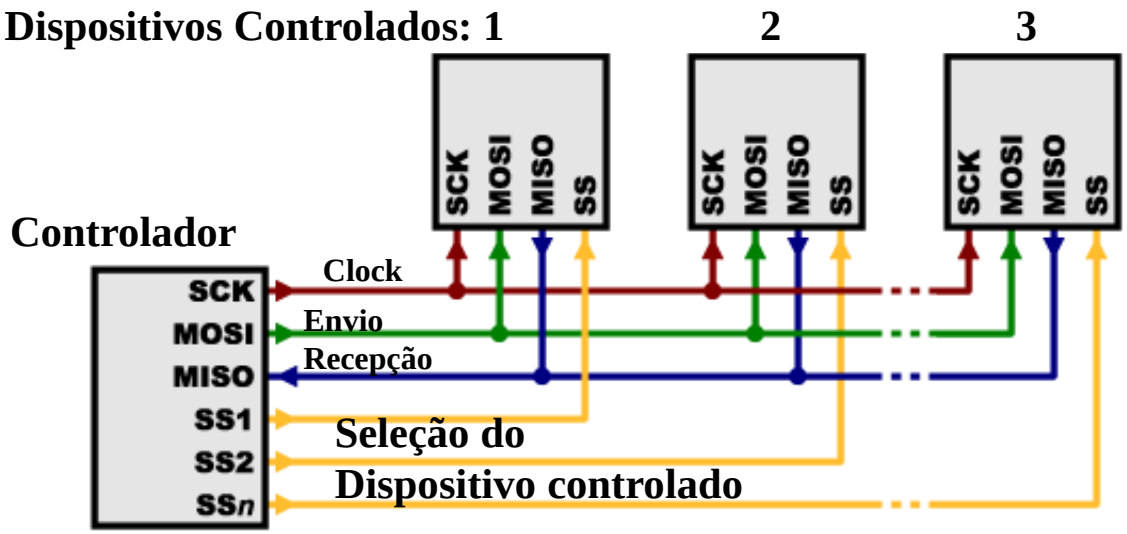
- ✓ **Leitura analógica no Arduino usando um potenciômetro**
  - Para o envio dos **10 bits**, deve-se dividir o inteiro em dois bytes,
  - Arduino possui uma função denominada "**highByte"** e "**lowByte**", utilize-a na chamada da função de envio de bytes pelo I2C.
  - Os valores devem ser lidos na Rasp de 0 a 255.
  - Mais detalhes: **Roteiro da prática no e-Disciplinas.**



Comunicação serial SPI

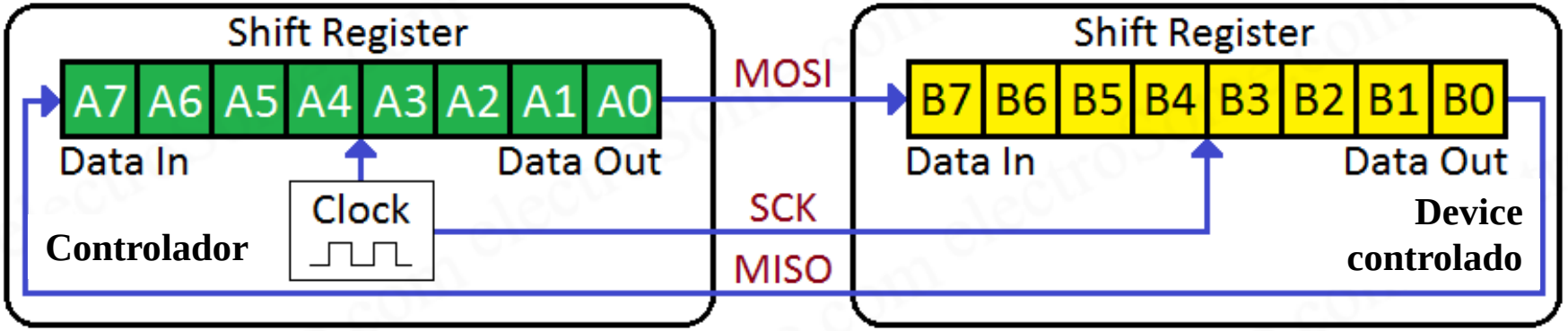
- ✓ ***Serial Peripheral Interface***
- ✓ Protocolo de comunicação serial composto por 4 vias, formato **full duplex** com maior velocidade de comunicação entre dispositivos.
- ✓ O dispositivo controlador (ex. Raspberry Pi), gera um clock e seleciona, por meio do **pino SS**, o dispositivo que será efetuada a comunicação.
- ✓ Em seguida os dados são enviados para o dispositivo de destino pelo **pino MOSI** e então o dispositivo controlado (ex. Arduino), envia uma resposta (se necessário) à Raspberry Pi pelo pino **MISO, de forma simultânea, sendo mais rápido do que o I2C.**

- ✓ **I2C vs. SPI:** **I2C** é uma comunicação de 2 fios **half duplex**, suporta **multi dispositivos controladores** e **multi dispositivos controlados**. **SPI** é full duplex, de 4 fios, suporta único dispositivo controlador, e a velocidade é mais rápida.



Comunicação serial SPI

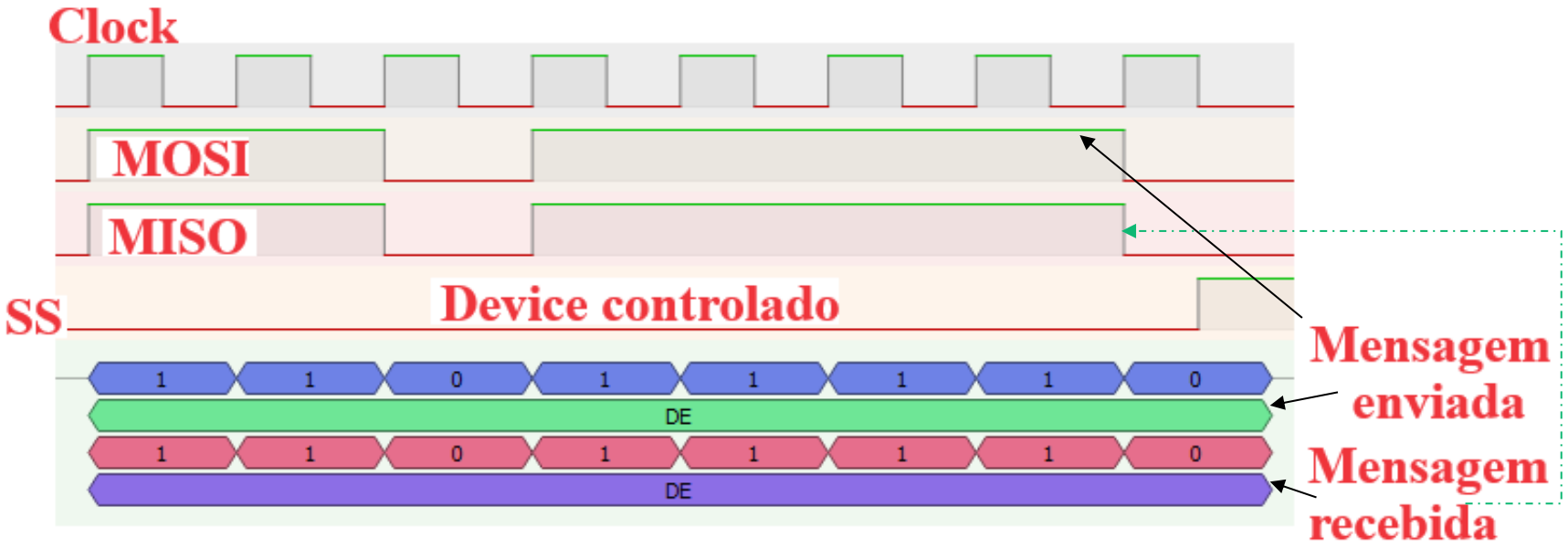
Utiliza o clock gerado pelo controlador



Pino	Nome Padrão	Significado	Do ponto de vista do controlador
Saída do controlador para a entrada do device controlado	MOSI	Master Output Slave Input	Envia dados
Saída do device controlado para a entrada do controlador	MISO	Master Input Slave Output	Recebe dados
Clock	SCLK	Serial Clock	O controlador gera o clock
Seleção do device controlado	SS	Slave Select	Endereço

Comunicação serial SPI

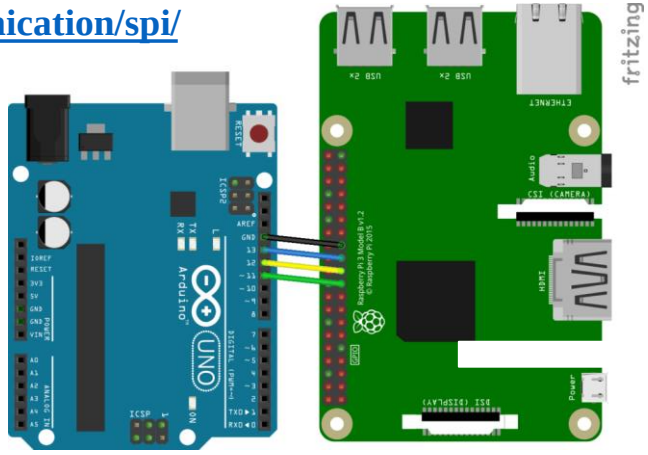
Exemplo de envio e recepção da mesma mensagem via SPI: no caso, o dado "DE" (em hexa = 0xDE) = 11011110



Bibliotecas para comunicação SPI

- **Bibliotecas usadas para a abstração da comunicação SPI entre Raspberry Pi e Arduino, por exemplo, por meio de programas em Python e em C na Arduino IDE**

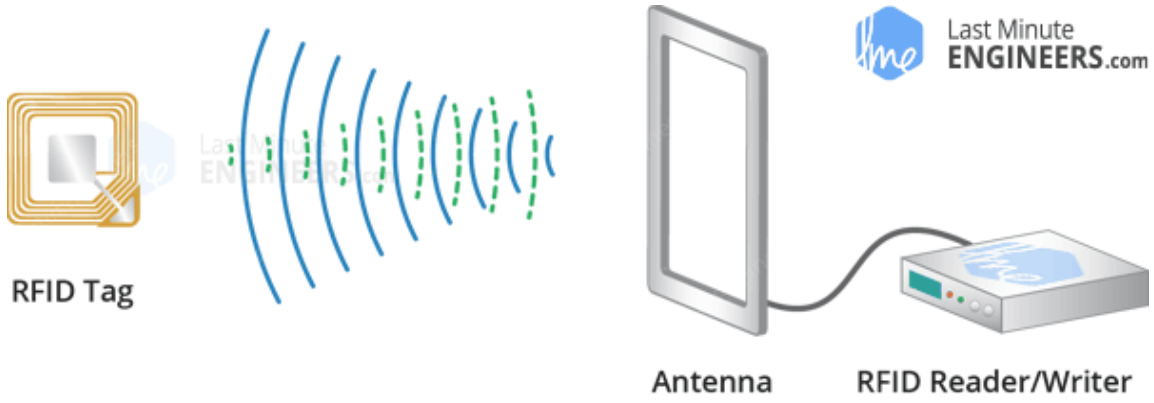
- ✓ **Por exemplo:** Arduino pode ser controlado pela Raspberry via SPI – para também acender/apagar um LED no Arduino digitando 1 ou 0 no terminal Linux da Raspberry Pi, conforme realizado anteriormente via UART ou I2C.
- ✓ **Programa em Python – Raspberry Pi – utiliza a biblioteca "spidev"-** https://pypi.org/project/spidev/
- ✓ **Programa em C – Arduino IDE- utiliza a biblioteca "spi.h"** https://www.arduino.cc/reference/en/language/functions/communication/spi/



Sistema RFID

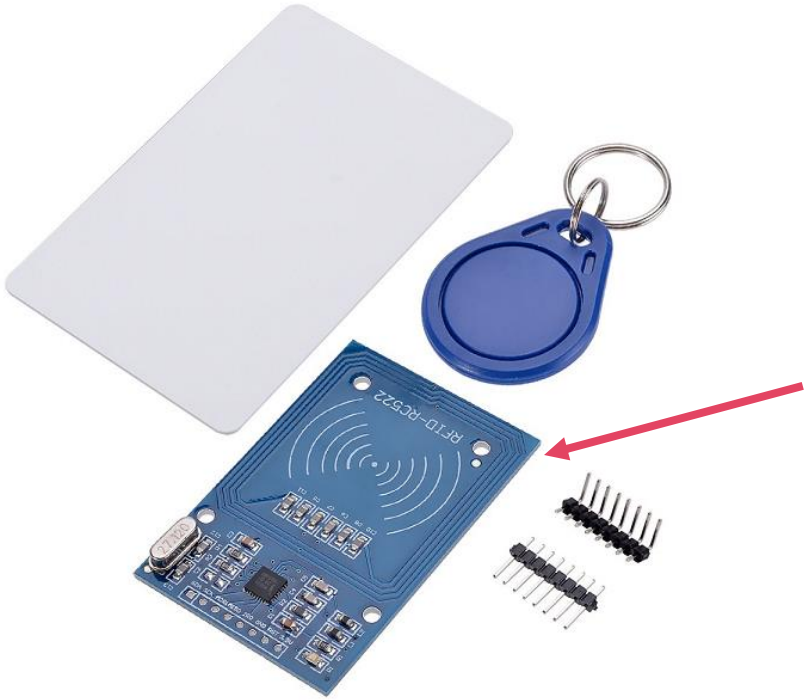
- *Radio frequency identification*
- Um módulo RFID é um CI composto por um leitor e um gravador RF (radio frequency – da ordem de MHz)

O **leitor RFID** gera uma campo eletromagnético por meio de uma antena e fica constantemente esperando que uma **tag** entre neste campo. No momento em que isso ocorre, a **tag** recebe energia gerada pelo leitor e os dois componentes desta comunicação (**leitor e tag**) começam a trocar informações. Quando a comunicação se inicia, o leitor envia requisições de dados para a **tag**, e a **tag** então retorna com todas as informações disponíveis.



Módulo RFID MFRC522

- **Leitor RFID MFRC522/Tag Chaveiro/Tag Cartão – 13,56MHz**
- Opera com frequência de 13,56 MHz - até 5m, e é capaz de gravar informações nas tags (cartões que possuem um CI e memória programável via RF).

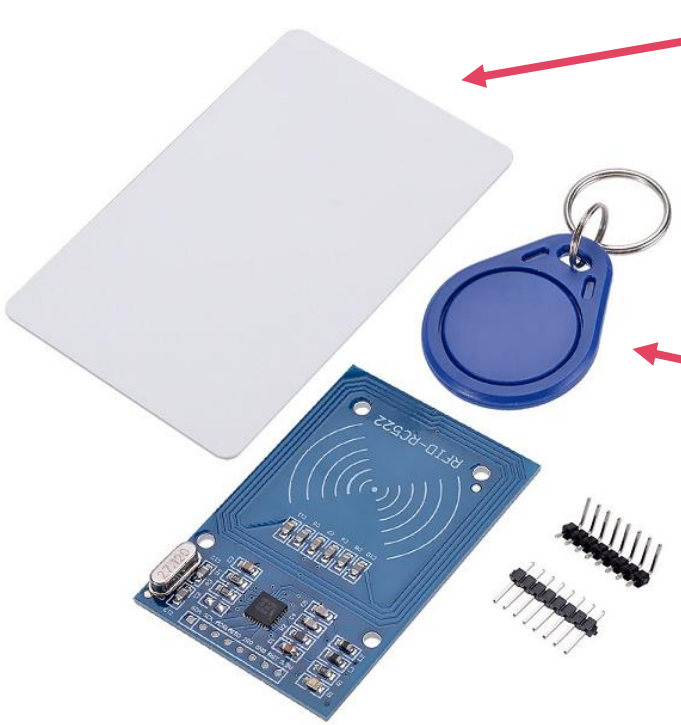


– Especificações e características (Leitor RFID MFRC522):

– Modelo: MFRC522
– Tensão de operação: 3,3VDC
– Corrente de operação: 13mA a 26mA
– Tensão em modo inativo: 3,3VDC
– Corrente em modo inativo: 10mA a 13mA
– Frequência de operação: 13,56MHz
– Interface: SPI
– Taxa de transferência: 10Mbit/s
– Alcance: 0 a 3cm
– Cartões suportados: Mifare1 S50, Mifare1 S70, Mifare Ultralight, Mifare Pro, Mifare Desfire
– Temperatura de operação: -20° a 80° celsius

Módulo RFID MFRC522

- **Leitor RFID MFRC522/Tag Chaveiro/Tag Cartão – 13,56MHz**
- Opera com frequência de 13,56 MHz - até 5m, e é capaz de gravar informações nas tags (cartões que possuem um CI e memória programável via RF).



Especificações e características (Tag Cartão 13,56MHz):

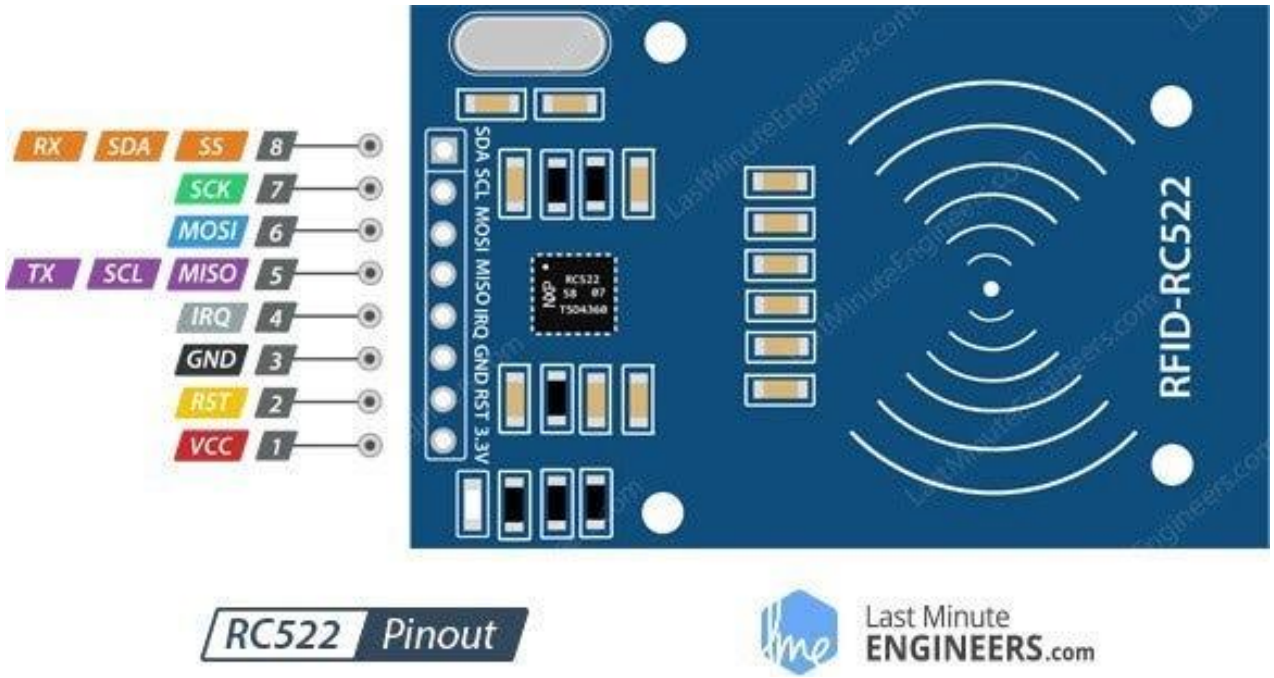
- Frequência de operação: 13,56MHz
- Taxa de transferência: 106Kbaud
- Capacidade: 8Kbit / 16 partições
- Alcance: 2 a 10cm
- Tempo de leitura e escrita: 1 a 2ms
- Temperatura de operação: -20° a 55° celsius
- Material: PVC

Especificações e características (Tag Chaveiro 13,56MHz):

- Frequência de operação: 13,56MHz
- Taxa de transferência: 106Kbaud
- Capacidade: 8Kbit / 16 partições
- Alcance: 2 a 10cm
- Tempo de leitura e escrita: 1 a 2ms
- Temperatura de operação: -20° a 80° celsius
- Material: ABS

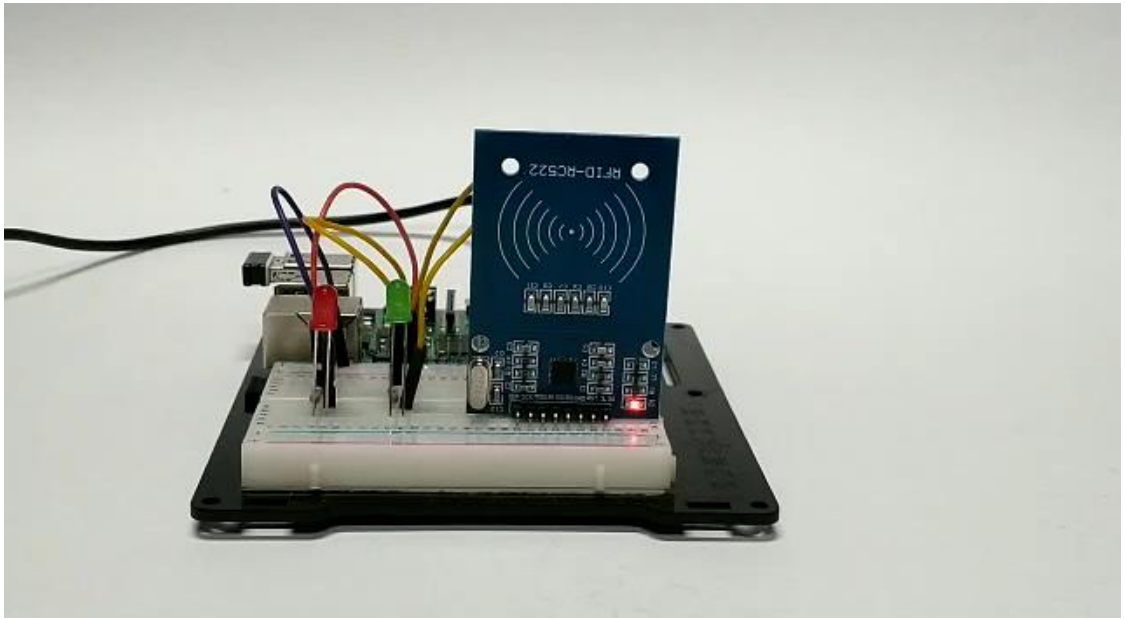
Módulo RFID MFRC522

- **Leitor RFID MFRC522/Tag Chaveiro/Tag Cartão – 13,56MHz**
- Datasheet MFRC522 https://blogmasterwalkershop.com.br/arquivos/datasheet/Datasheet%20MFRC522.pdf



Módulo RFID – controle de acesso

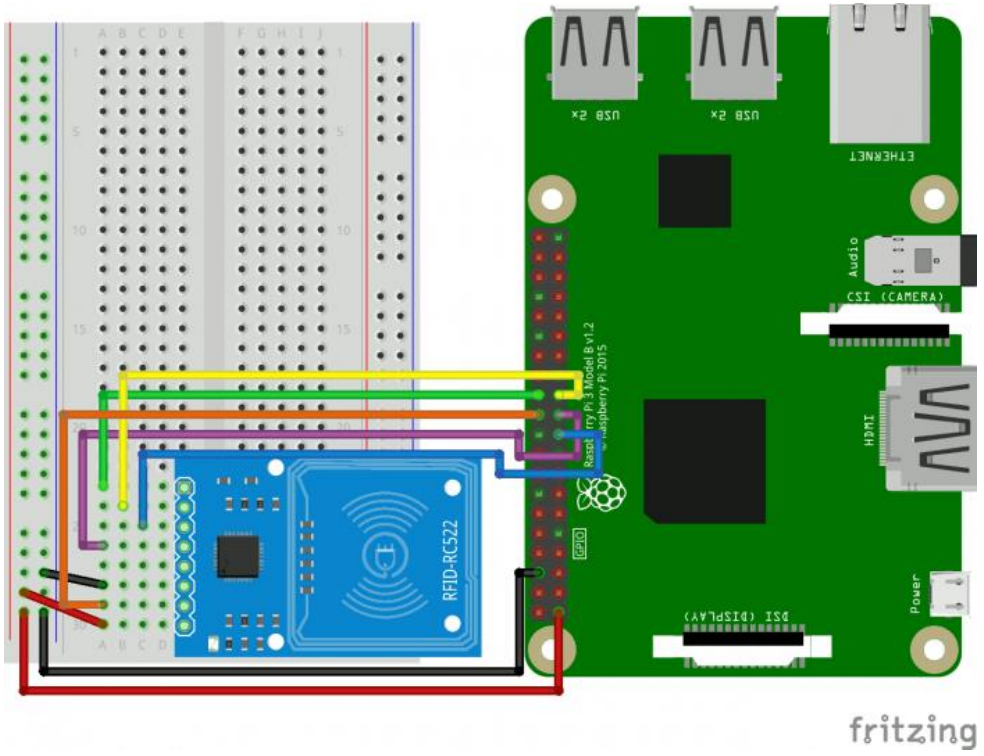
Os sistemas de controle de acesso via tags RFID oferecem gerenciamento de acesso, rastreamento de ativos e automação em fechaduras, sistemas de pagamento como em praças de pedágio, leitura de códigos etc.



Realizando a comunicação com a Raspberry Pi via SPI

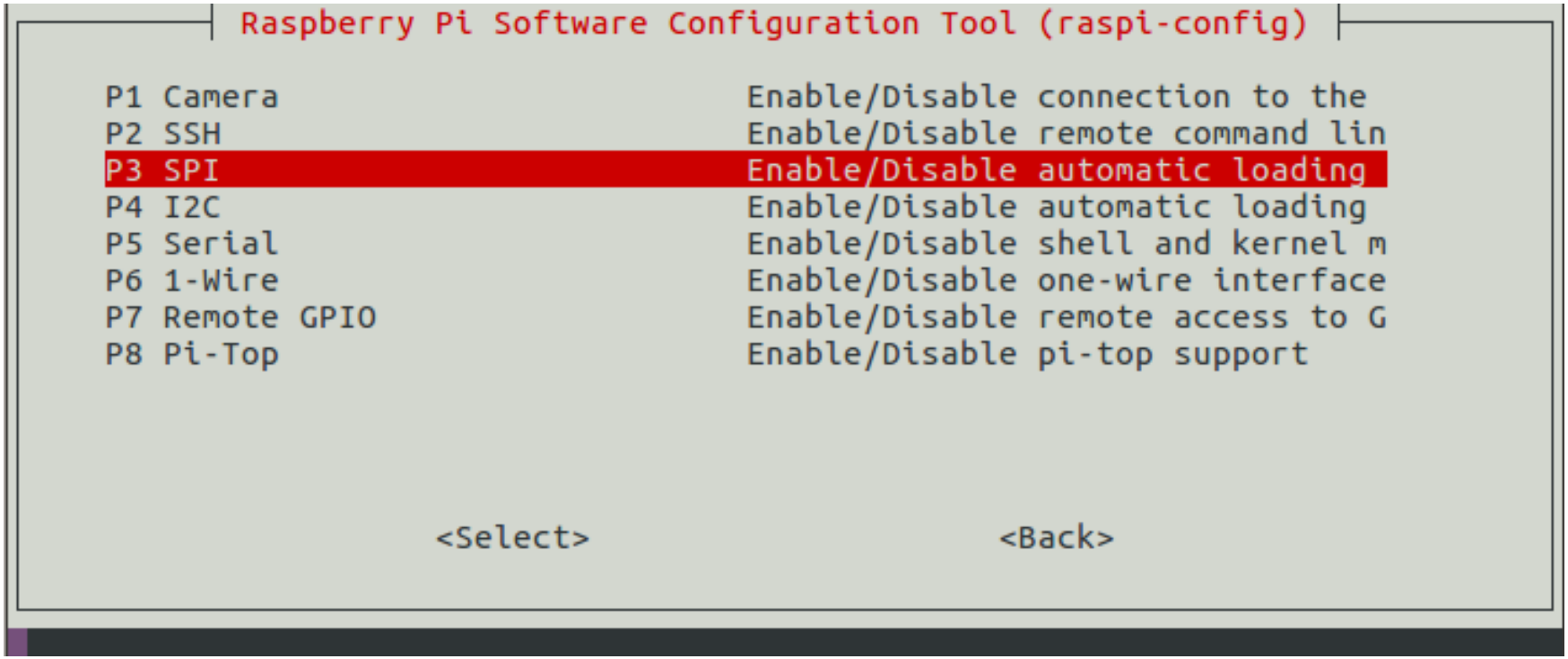
✓ Realizar a comunicação SPI por meio das ligações abaixo.

Módulo RFID MFC522	Pinout da RaspberryPi
SDA	GPIO 8 (SDA)
SCK	GPIO 11 (SCK)
MOSI	GPIO 10 (MOSI)
MISO	GPIO 9 (MISO)
IRQ (interrupt - não usado)	-
GND	GND
RST (reset)	GPIO 25
3.3V	3.3 V



Realizando a comunicação com a Raspberry Pi via SPI

✓ Habilitar a comunicação SPI: *sudo raspi-config* > *3- interfaces*



Tutorial para gravação de texto na Tag

✓ Instalar a biblioteca Python MFRC522: *pip3 install mfrc522*

```
#para gravação de texto na Tag

import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522

#desabilita os avisos
GPIO.setwarnings(False)

#cria o objeto "leitor" para a instância "SimpleMFRC522" da biblioteca
leitor = SimpleMFRC522()

#criacao da variavel que armazena o texto que será gravado na tag
dado= "SEL0630/0337" #altere para o texto que deseja gravar

#escreve a tag assim que ela for aproximada do leitor, e informa a conclusão
print("Aproxime a tag do leitor para gravar.")
leitor.write(dado) #função que realiza a gravação do texto configurado
print("Concluído!") #quando essa mensagem for impressa, significa que a informação já foi
gravada na Tag
```

Tutorial para verificar qual o ID da Tag

✓ Seguir o tutorial abaixo para coletar o n° de identificação da Tag e confirmar se o texto do código do slide anterior foi gravado

- A partir do código abaixo, tendo sido descoberto o ID, é possível criar um projeto de controle de acesso contendo uma base de dados com IDs de tags, para liberar ou bloquear o acesso (caso o ID não constar no cadastro) ao aproximar a tag do leitor.
- MFRC522 – Python (exemplos) : https://pypi.org/project/mfrc522-python/

```
# Para descobrir qual a codificação da Tag com texto gravado anteriormente
from mfrc522 import SimpleMFRC522
from time import sleep
import RPi.GPIO as GPIO

#desabilitar os avisos
GPIO.setwarnings(False)

#cria o objeto "leitor" para a instância "SimpleMFRC522" da biblioteca
leitor = SimpleMFRC522()

print("Aproxime a tag do leitor para leitura.")
while True: #loop
#cria as variáveis "id" e "texto", e as atribui as leituras da id e do texto coletado da tag pelo leitor, respectivamente
    id,texto = leitor.read()
    print("ID: {}\nTexto: {}".format(id, texto)) #exibe as informações coletadas – verifique o texto e ID
    sleep(3) #aguarda 3 segundos para nova leitura
```

Considerações finais sobre UART, I2C e SPI

Tecnologia	N° de Barramento de comunicação	Taxa máxima	Fluxo de dados
UART (RS232)	Sem barramento de controle É um hardware de comunicação ponto a ponto entre 2 dispositivos de forma assíncrona	115.200 bps	Half ou Full Duplex
SPI	3 + nº de devices controlados (comunicação ponto a ponto + sinal de clock)	2 Mbps	Full Duplex
I2C	2 (até 127 dispositivos controlados na rede)	400 Kbps	Half Duplex

SPI geralmente é usado para comunicação de curta distância com taxas de transferência mais altas (consome mais energia). O **I2C** é mais adequado para sistemas que requerem extensibilidade em distâncias maiores devido à sua topologia em barramento (mais lenta e menor consumo). **UART** é uma comunicação ponto a ponto, simples, e direta entre dois dispositivos.

Considerações finais sobre UART, I2C e SPI

Característica	UART	I2C	SPI
Topologia	Point-to-Point	Multi-Ponto (Barramento)	Point-to-MultiPoint ou Daisy Chain
Número de Fios	2 (TX, RX)	2 (SDA, SCL)	4 (MISO, MOSI, SCLK, SS)
Sincronização	Assíncrona	Síncrona	Síncrona
Controle/clock	Não aplicável	Controlador (pode ter vários devices controlados)	Controlador (pode ter vários devices controlados)
Endereçamento	Não aplicável	7 ou 10 bits de endereço	Não aplicável
Uso Comum	Comunicação simples e direta entre dispositivos, periféricos e PC	Sensores, Periféricos	Comunicação rápida com periféricos
Vantagens	Simplicidade, Baixo Custo	Topologia Multi-Ponto, Multi-dispositivos, longas distâncias, baixo consumo	Alta Velocidade, Topologia Flexível
Desvantagens	Limitado por distância, Menor taxa de transferência	Pode ser mais complexo, lentidão	Requer mais fios, maior consumo de energia, limitado a curtas distâncias

Material complementar

- **Protocolo 1-wire – Raspbery Pi:**
  https://core-electronics.com.au/guides/raspberry-pi/temperature-sensing-with-raspberry_pi/
- **I3C:** https://sergioprado.org/i3c-o-futuro-substituto-dos-barramentos-i2c-e-spi/
- **I2C *vs* SPI *vs* UART:** https://www.totalphase.com/blog/2021/12/i2c-vs-spi-vs-uart-introduction-and-comparison-similarities-differences/
- **Redes CAN:**
  https://www.embarcados.com.br/wp-content/uploads/filebase/eventos/tdc_2016_trilha_embarcados/TDC2016-Rede-CAN-Conceitos-e-Aplicacoes.pdf
- **Uma Aplicação Didática do Protocolo I2C em Sistemas de Comunicação:**
  https://ojs.brazilianjournals.com.br/ojs/index.php/BRJD/article/view/36870/pdf
- **Projeto e simulações de comunicação serial no Wokwi:**
  - ✓ https://wokwi.com/projects/366980653780379649
  - ✓ https://apollolabsblog.hashnode.dev/esp32-embedded-rust-at-the-hal-spi-communication
  - ✓ https://wokwi.com/projects/355503418638451713
  - ✓ https://wokwi.com/projects/380652816498266113
  - ✓ https://blog.wokwi.com/wokwi-logic-analyzer-uart-part-1/

Referências e créditos

- Arduino - https://www.arduino.cc
- Floyd. T. Sistemas Digitais. Bookman 8ª ed. 2007
- GitHub, Inc. Disponível em: https://github.com
- Get Started with Arduino – Raspberry Pi Press. https://www.mclibre.org/descargar/docs/revistas/hackspace-books/hackspace-get-started-with-arduino-01-201911.pdf
- Python MFRC522 - https://pypi.org/project/mfrc522-python/
- Portal Embarcados. https://embarcados.com.br
- Python –Disponível em: https://www.python.org.
- Raspberry Pi Foundation https://www.raspberrypi.org - https://www.raspberrypi.com

- Obrigado pela atenção!

FIM

Coffee Break