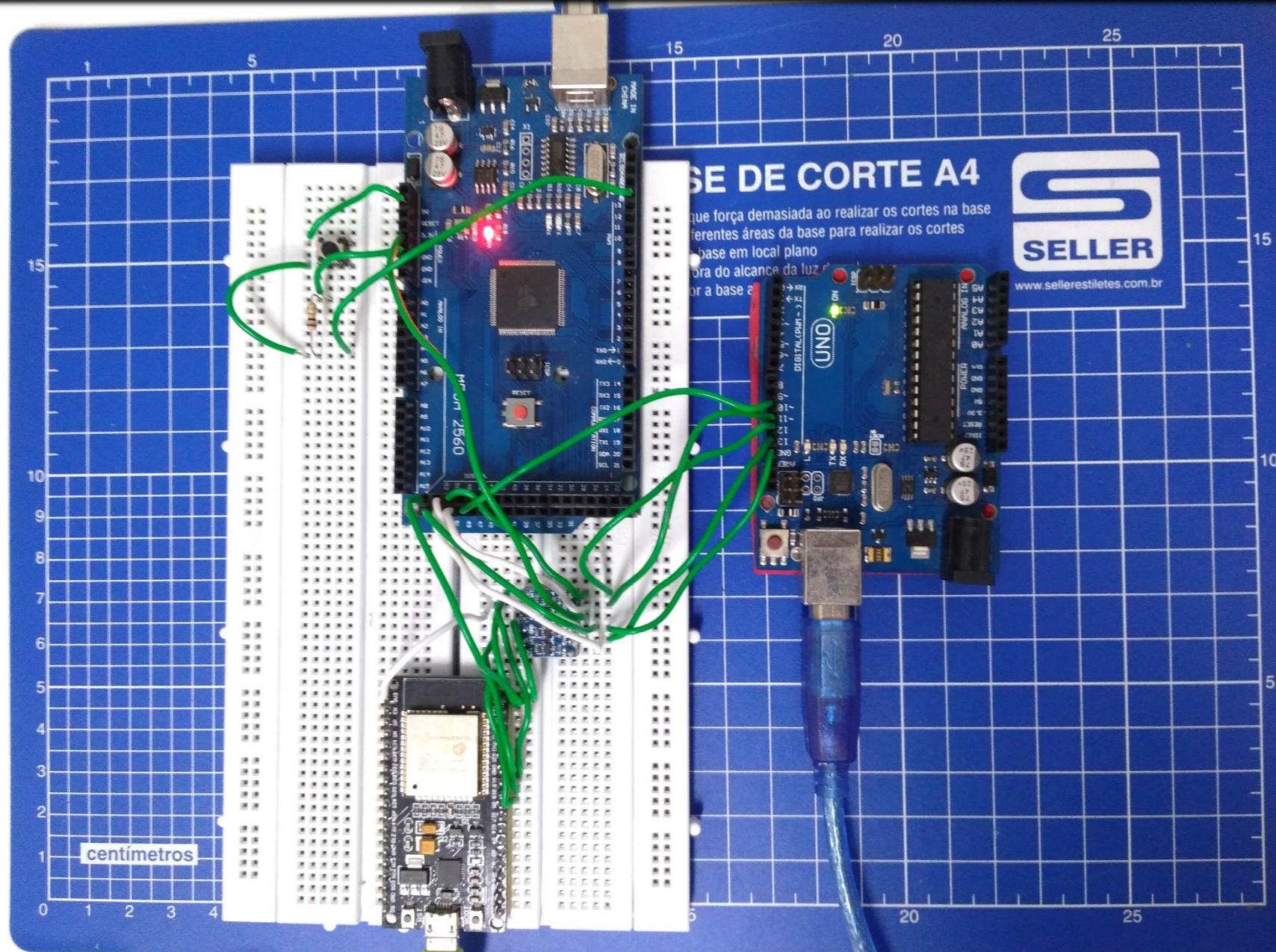


SPI - ESP32, Arduino Mega e Arduino UNO



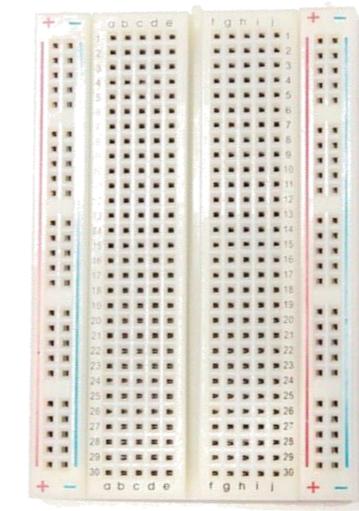
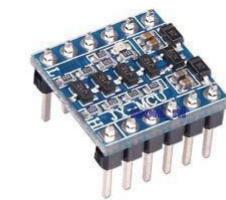
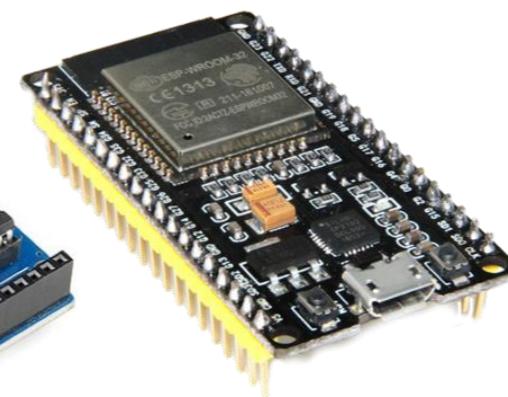
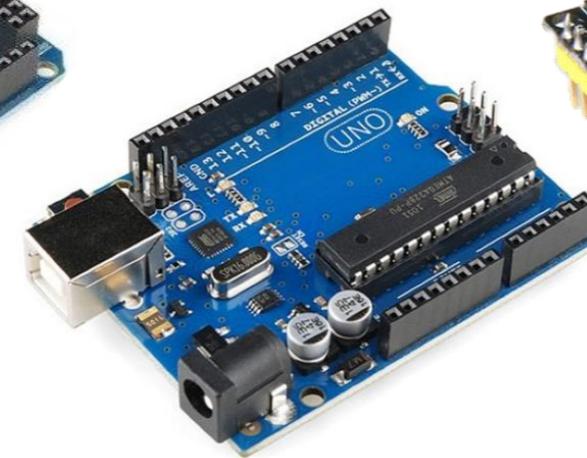
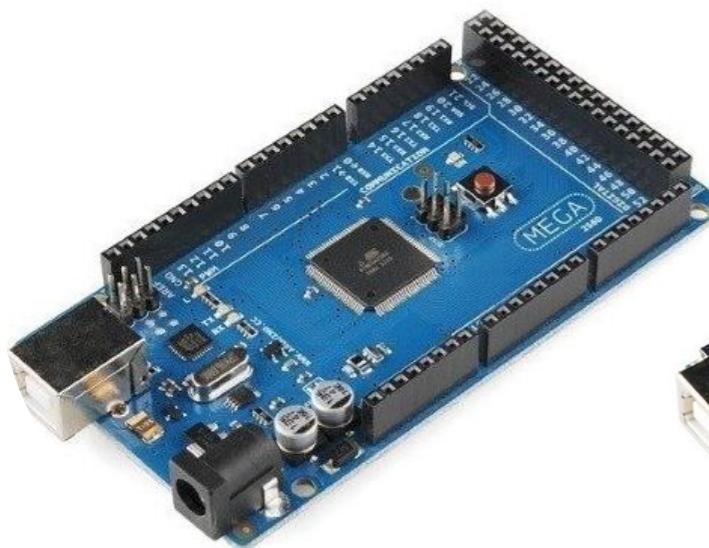
Por Fernando Koyanagi

Intenção dessa aula

- 1. Utilizar comunicação SPI entre ESP32, Arduino Mega e Arduino UNO**
- 2. Explicação do envio do estado dos pinos Digitais e Analógicos (AD) do Arduino Mega para o ESP32 e do Arduino Mega para o Arduino UNO**

Recursos usados

- **ESP-WROOM32 (38 pinos)**
- **Arduino Mega 2560**
- **Arduino UNO**
- **Protoboard**
- **Conversor de nível lógico 3V3-5V**
- **Jumpers**
- **Push button**
- **Resistor 10K ohm**





Tutoriais
Tecnologia
Tendências

FÓRUM

SOBRE

ARDUINO

ESP8266

ESP32

LORAWAN

STM32

MOTOR

DISPLAY

MATERIAL PARA DOWNLOAD



Robô de desenho XY

by Fernando K - 19 fevereiro

Temos aqui hoje um projeto de mecatrônica que é uma derivação de um vídeo que eu já lancei aqui: ROUTER E PLOTTER WIFI COM WEB SERVER ...

[Leia mais](#)



E se o seu link cair?

by Fernando K - 12 fevereiro

Neste vídeo vamos criar um sensor de queda de link com um ESP32 e um SIM800. Isso significa que, com este projeto, poderemos verificar ...

[Leia mais](#)



Tragédia de Brumadinho: sugestão para um sistema de alerta!

by Fernando K - 08 fevereiro

Como a Internet das Coisas pode auxiliar em tragédias como a de Mariana e Brumadinho? Hoje quero trazer uma sugestão de um sistema ...

[Leia mais](#)



Router e Plotter WiFi com Webserver em ESP32

by Fernando K - 05 fevereiro

Já gravei vídeos sobre plotter com o Raspberry Pi e com Laser, mas, hoje, quero falar de uma versão com GRBL e ESP32. Como acredito e...

[Leia mais](#)

CLIQUE AQUI E ACESSE
ONDE EU COMPREI

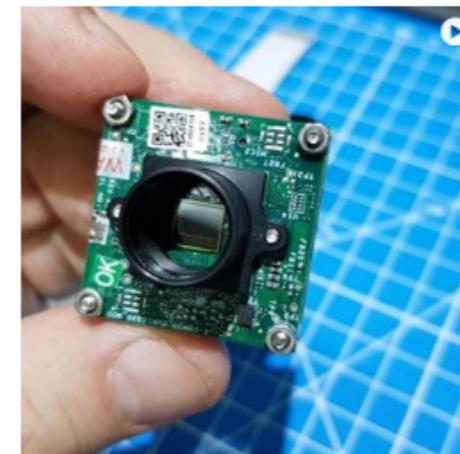
Links onde comprei os componentes

Em www.fernandok.com

Supergrupo de colaboração entre meus seguidores.

Conteúdo exclusivo, que não tem no Youtube!

INSTAGRAM @FERNANDOK_OFICIAL



FACEBOOK



INSCREVA-SE NO YOUTUBE

forum.fernandok.com

Fórum Fernando K Tecnologia
Fórum sobre dúvidas com relação ao conteúdo disponibilizado pelo Fernando Koyanagi

Nosquisar...

www.fernandok.com /fernandokoyanagi /fernandokoyanagi /fernandok_oficial /fernandok_oficial

[Links rápidos](#) [L...](#) [fernandokoyanagi](#)

Bem-vindo: 05/Oct/2018, 11:16 A sua última visita foi em 10/Set/2018, 15:47

Assinalar todos os fóruns como lidos

SUporte Fórum Fernandok

	TÓPICOS	MENSAGENS	ÚLTIMA MENSAGEM
Feedback Dúvidas, críticas ou sugestões sobre o Fórum FernandoK. Para demais questões utilize o fórum correto.	6	11	Re: O russo voltou por Ipmehi 01/Oct/2018, 08:25

FERNANDO K

	TÓPICOS	MENSAGENS	ÚLTIMA MENSAGEM
Arduino Projetos de arduino	31	79	skardy bogii por Sorororcem 05/Oct/2018, 10:55
ESP32 Projetos de ESP32	29	62	Dúvidas sobre como instalar a... por Marcos Sarge 04/Oct/2018, 15:52
ESP8266 O ESP8266 é um microcontrolador do fabricante chinês Espressif que inclui capacidade de comunicação por Wi-Fi.	24	51	Re: NodeMCU não conecta em qu... por ivanribeira 04/Oct/2018, 14:39
LoRa Projetos com LoRa	11	31	Projeto de irrigação de jardim por marlendo 04/Oct/2018, 21:30
STM32 Projetos com STM32	3	8	Re: Imprecisão de tempo de de... por biazoto 12/Sep/2018, 09:15
Motor Projetos com motor	5	11	Re: impressora 3d com motor dc por Magneton 24/Sep/2018, 19:05
Display Projetos com Display	4	11	Re: Alguém conhece o VIRTUINO... por Jod Luz 21/Sep/2018, 11:39

QUEM ESTÁ ONLINE
No total, há 4 usuários online :: 2 usuários registrados, 0 invitado e 2 visitantes (baseado em usuários ativos nos últimos 5 minutos)
O recorde de usuários online foi de 19 em 11/Sep/2018, 05:37

Usuários registrados: alberto, fernandokoyanagi
Legenda: Administradores, Moderadores globais

ANIVERSÁRIOS
Não há aniversários hoje

ESTATÍSTICAS
Total de mensagens 703 • Total de tópicos 114 • Total de membros 469 • Novo usuário: Sorororcem

[L...](#) [☰](#)

Powered by phpBB® Forum Software © phpBB Limited
Traduzido por: Suporte phpBB
Painel de Controle da Administração



Instagram
fernandok_oficial

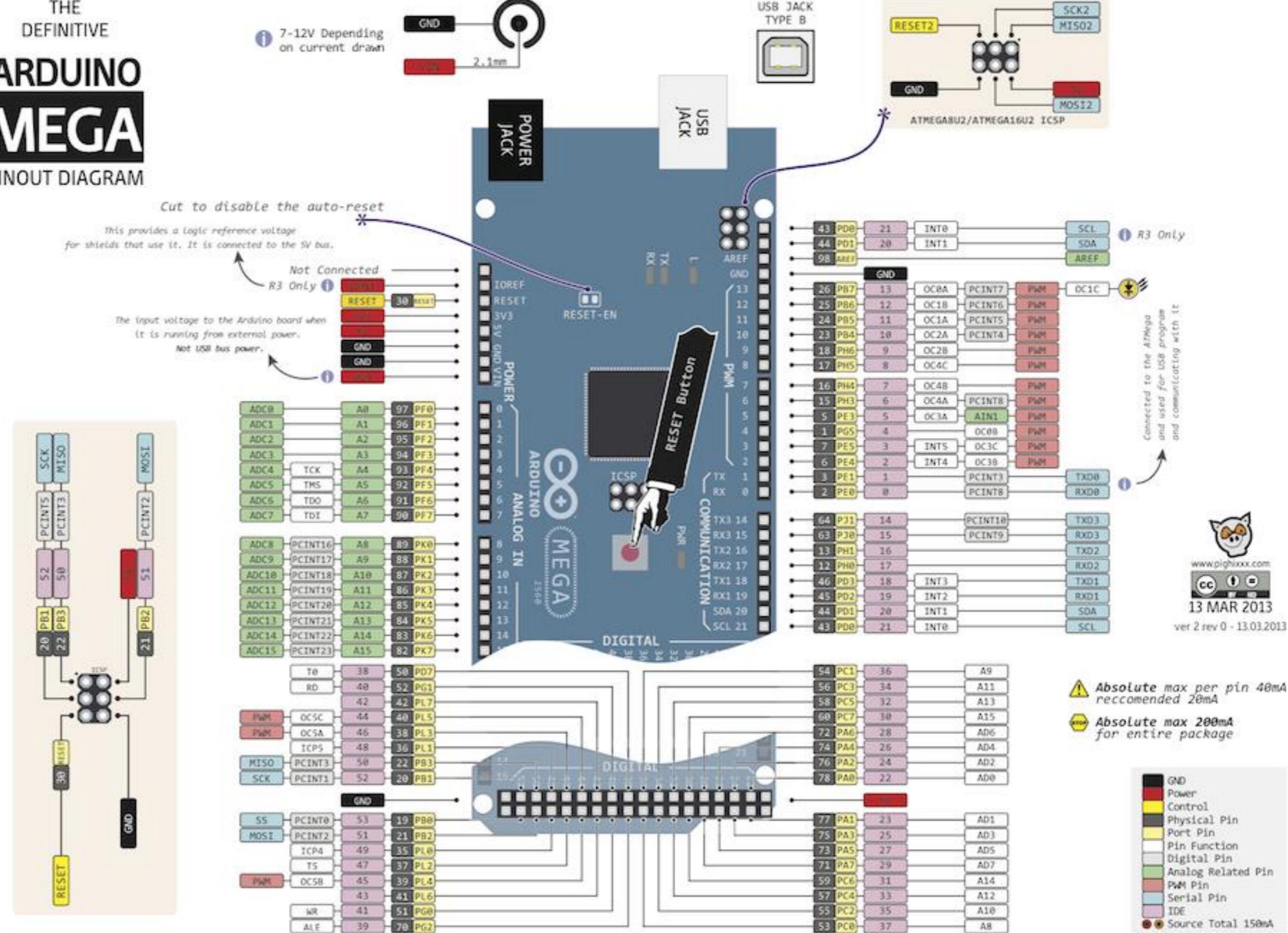


Telegram
fernandok_oficial



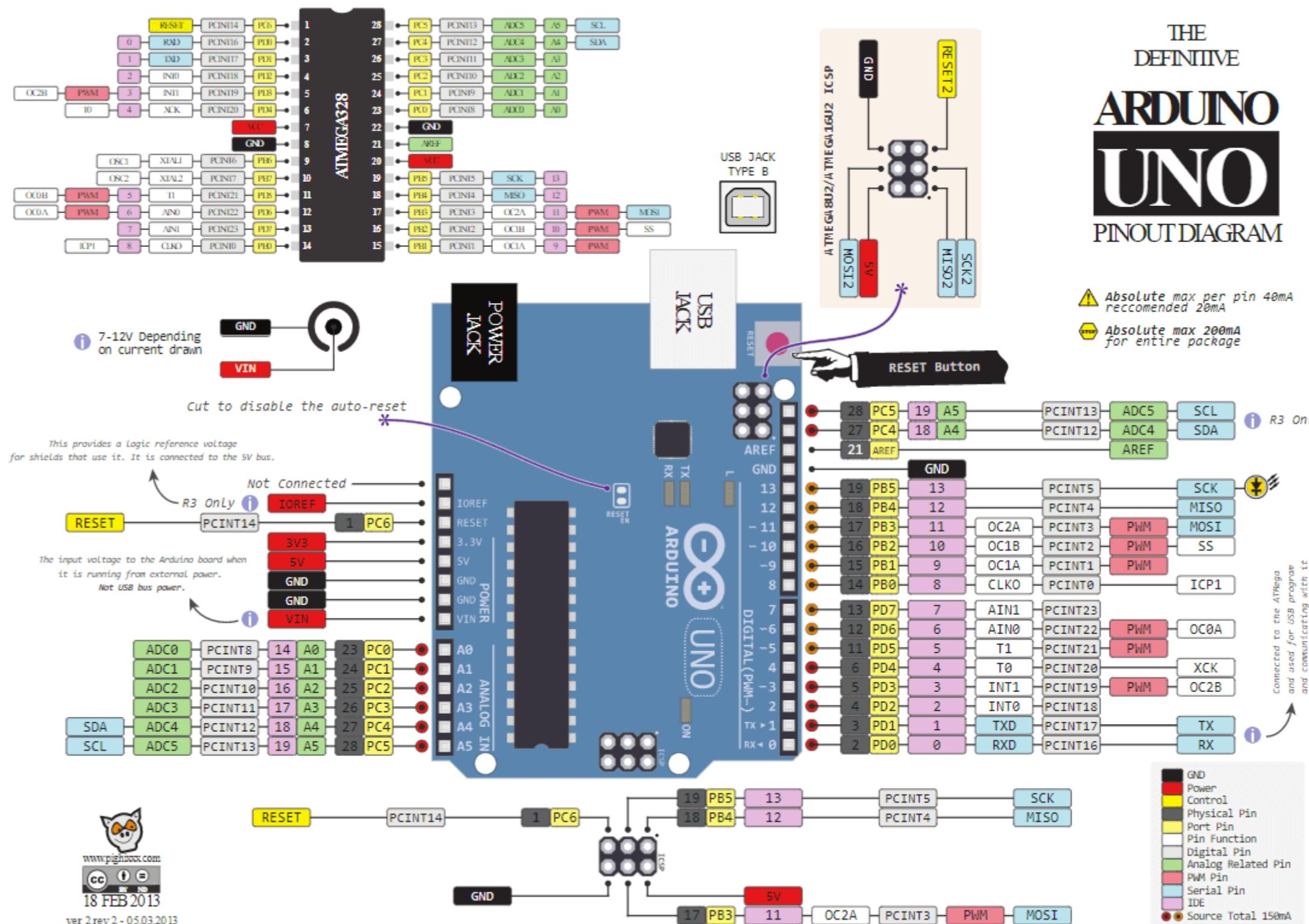
Pinout Arduino Mega

THE DEFINITIVE ARDUINO **MEGA** PINOUT DIAGRAM



Pinout Arduino UNO

THE
DEFINITIVE
ARDUINO
UNO
PINOUT DIAGRAM



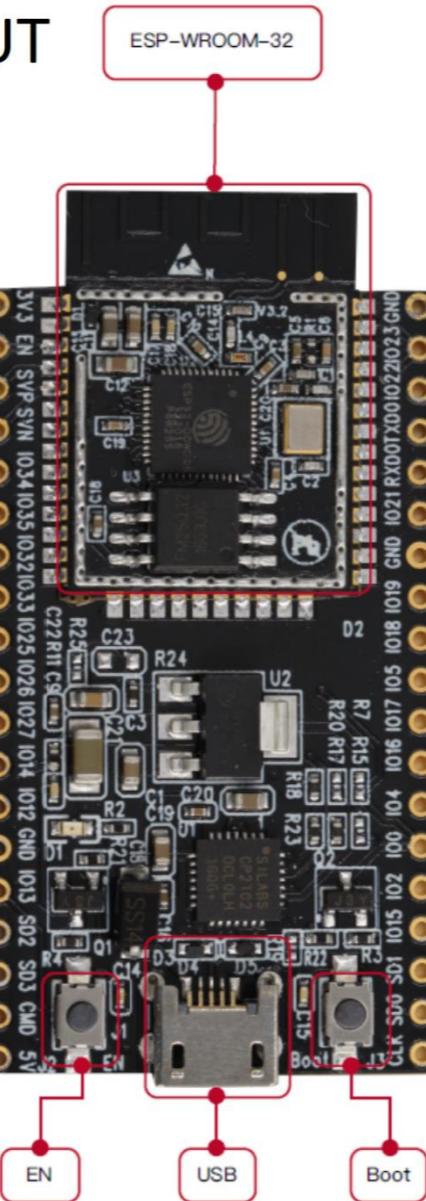
Pinout ESP32



JTAG

	Power
	GND
	Serial Pin
	Analog Pin
	Control
	Physical Pin
	Port Pin
	Touch Pin
	DAC Pin
	PWM Pin

ESP32-WROOM-32 PINOUT (aka ESP32-DevKitC)



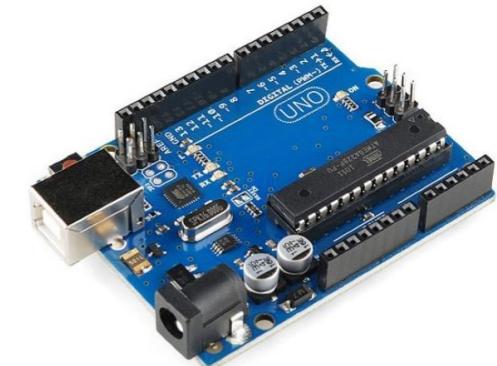
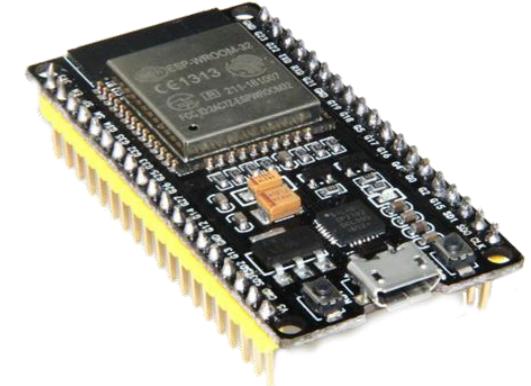
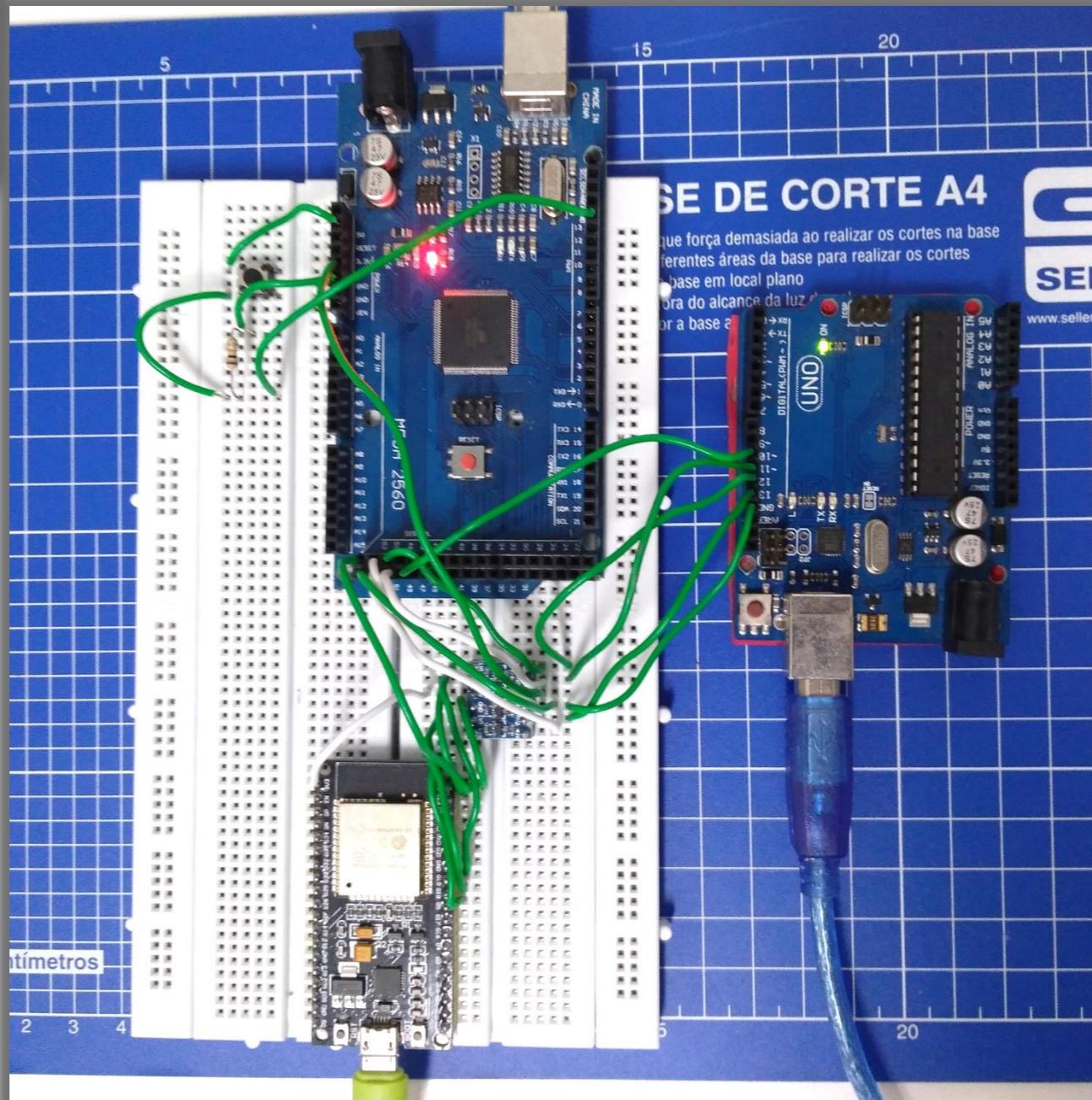
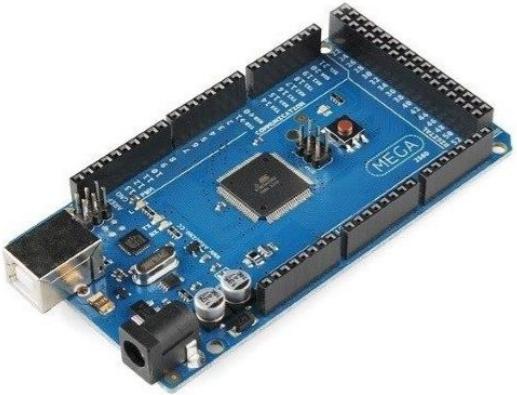
JTAG 20-pin	
Red	3V3 VTref
Gray	RSTnTRST
Orange	IO12 TDI
Yellow	IO14 TMS
Blue	IO13 TCK
(to GND) RTCK	
Purple	IO15 TDO
	RESET
	DBGREQ
	5V-Supply
1 • • 2	---
3 • • 4	GND GND
5 • • 6	GND Black
7 • • 8	GND
9 • • 10	GND Green
11 • • 12	GND White
13 • • 14	GND
15 • • 16	GND
17 • • 18	GND
19 • • 20	GND
Pins 14, 16, 18, 20: On some models like the high-end model J-Link PRO, these pins may not be connected to GND but are reserved for future use/extension. In case of doubt, leave open on target hardware.	
GND	---
IO23	GND
IO22	---
TXD0	---
RXD0	---
IO21	---
GND	---
IO19	---
IO18	---
IO17	---
IO16	---
IO4	---
IO0	---
IO2	---
IO15	---
SD1	---
SD0	---
CLK	---
PROGRAM Port	
36	GPIO23 VSPI ID HS1 STROBE SPI MOSI
39	GPIO22 ENACTD0 U0 RTS VSPI WP Wire SCL
41	GPIO01 EMAC RXD2 U0 TXD CLK OUT3
40	GPIO03 EMAC RXD1 U0 RXD CLK OUT2
42	GPIO21 EMAC TXEN VSPI HD Wire SDA
NC	---
38	GPIO19 EMAC TXD0 U0 CTS VSPI Q SPI MISO
35	GPIO18 VSPI CLK HS1 DATA7 SPI SCK
34	GPIO05 EMAC RXCLK VSPI CS0 HS1 DATA6 SPI SS
27	GPIO17 EMAC CO180 U2 TXD HS1 DATA5
25	GPIO16 EMAC CLK OUT U2 RXD HS1 DATA4
24	GPIO04 EMAC TXER ADC2 0 RTC10 Touch0 HSPI HD SD DATA1 HS2 DATA1
23	GPIO00 EMAC TXCLK ADC2 1 RTC11 Touch1 CLK OUT1
22	GPIO2 HSPI WP ADC2 2 RTC10 Touch2
21	GPIO15 EMAC RXD3 ADC2 3 RTC13 Touch3 MT D0 HSPI CS0 SD CMD HS2 CMD TDO
33	GPIO8 SPI D U2 CTS HS1 DATA1 SD DATA2 FLASH D1
32	GPIO7 SPI Q U2 RTS HS1 DATA2 SD DATA3 FLASH D0
31	GPIO6 SPI CLK U1 CTS HS1 CLK SD CLK FLASH SCK

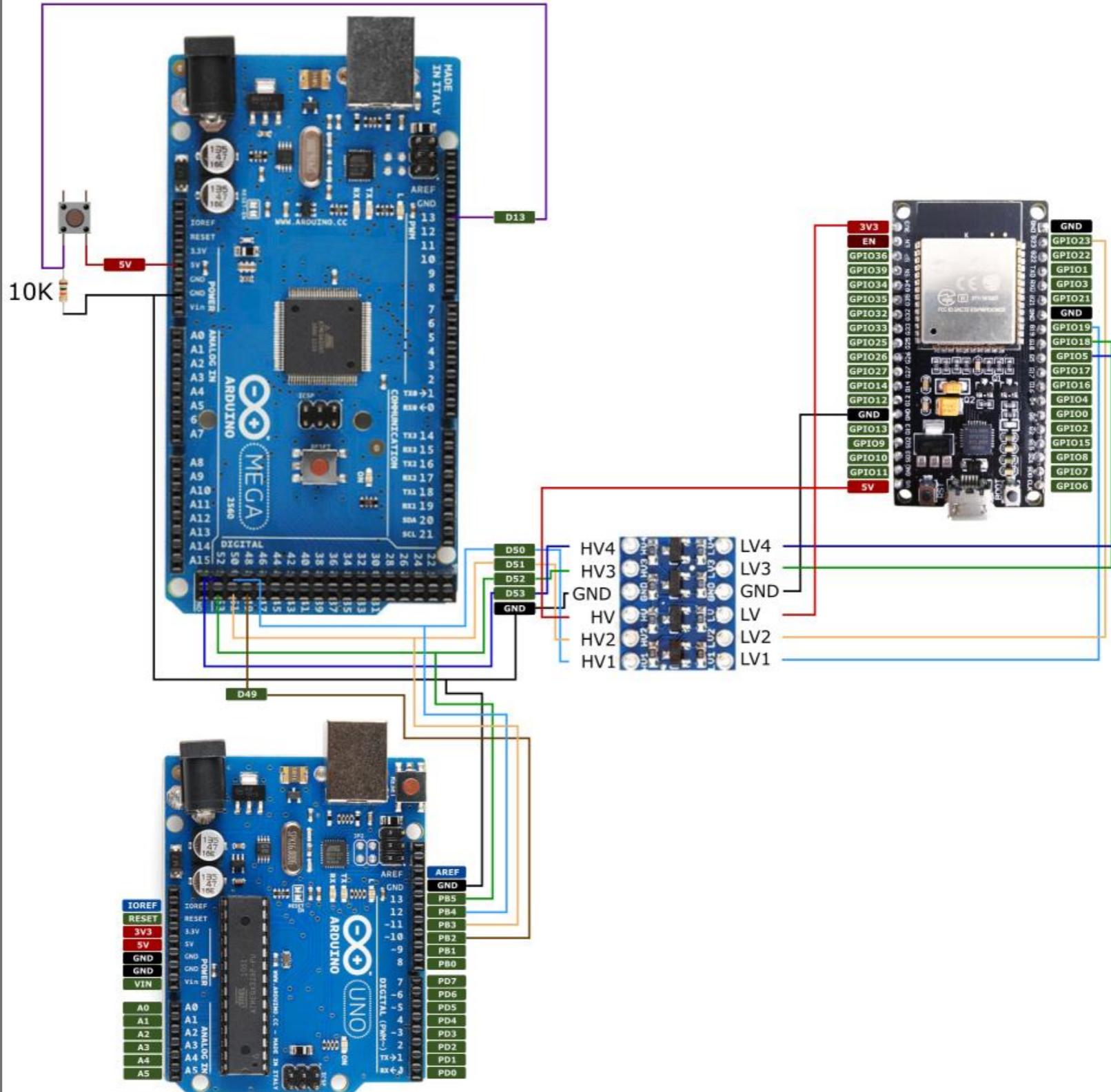
Adapted from data at:

http://www.espressif.com/sites/default/files/documentation/esp_wroom_32_datasheet_en.pdf

https://cdn-shop.adafruit.com/product-files/3269/pinout_wroom_pinout.png

Montagem





Montagem

Arduino Mega	ESP32	Descrição
D50	GPIO19	MISO
D51	GPIO23	MOSI
D52	GPIO18	SCK
D53	GPIO5	SS
GND	GND	GND

Arduino Mega	UNO	Descrição
D50	GPIO12	MISO
D51	GPIO11	MOSI
D52	GPIO13	SCK
D49	GPIO10	SS
GND	GND	GND

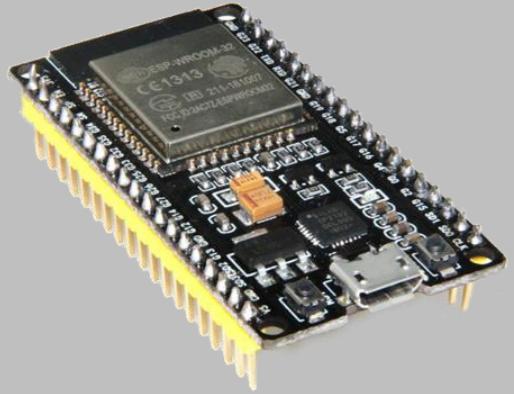
*Conekte utilizando **conversor de nível**:

D50 → H1 L1 ← GPIO19

D51 → H2 L2 ← GPIO23

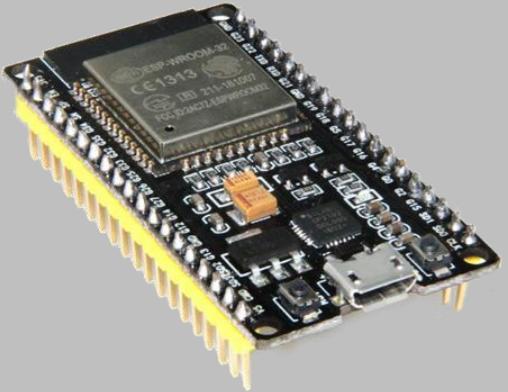
D52 → H3 L3 ← GPIO18

D53 → H4 L4 ← GPIO5



Código

Configurações

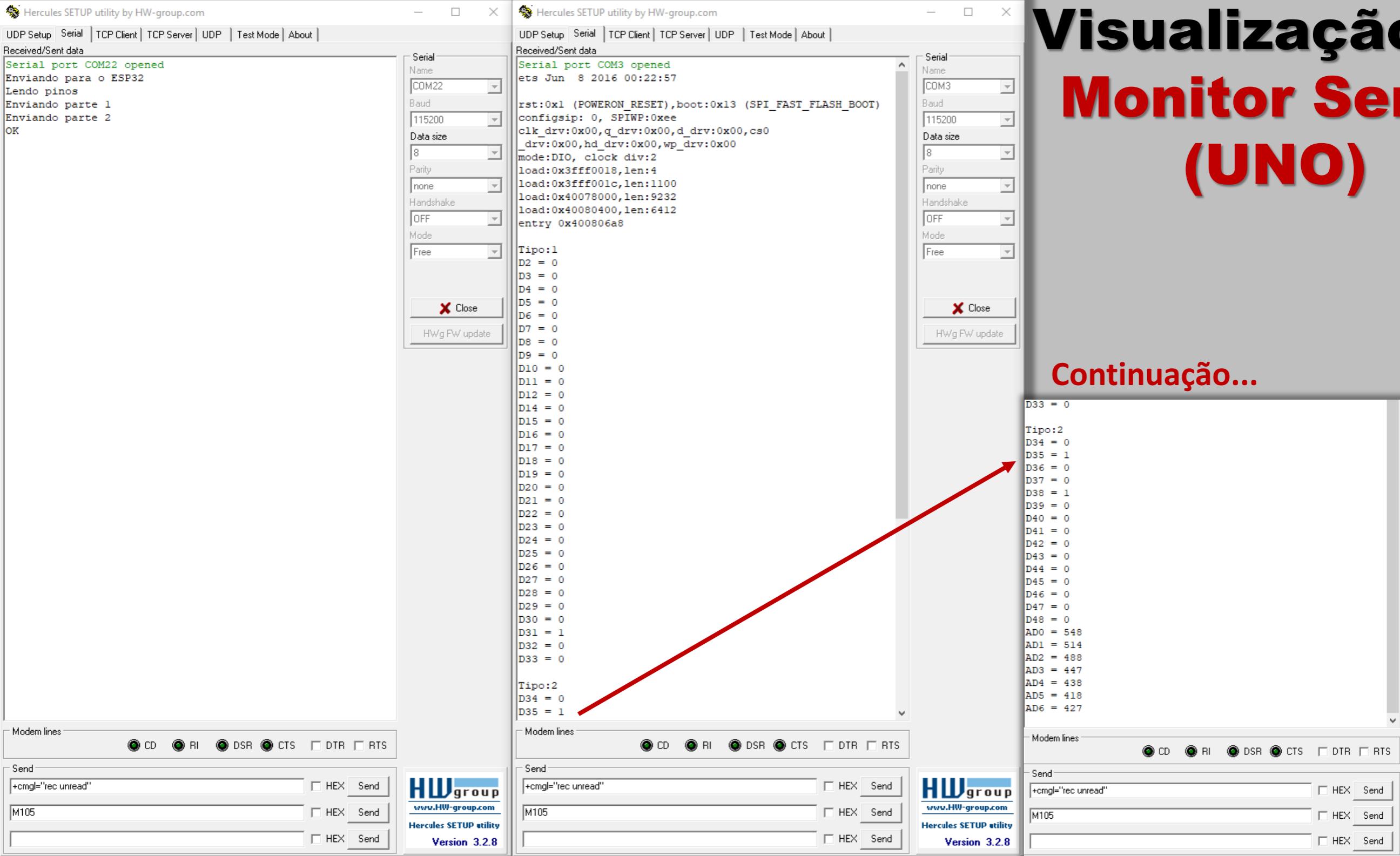


ESP32 - Bibliotecas necessárias

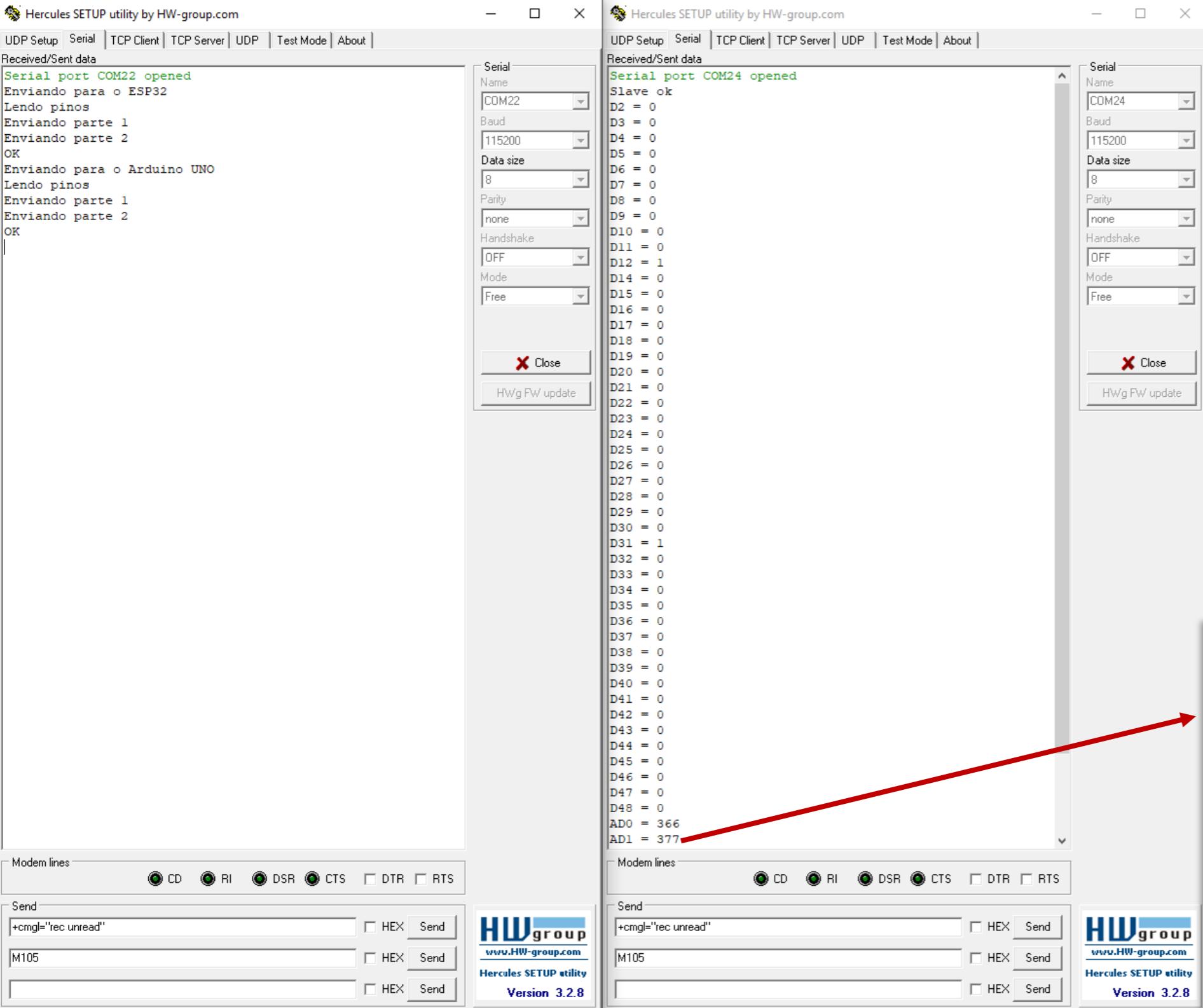
- Biblioteca Slave SPI ESP32
<https://github.com/iPAS/esp32-slave-spi>
- Biblioteca Simple Array
<https://github.com/iPAS/simple-array/>

Visualização no Monitor Serial (UNO)

Continuação...

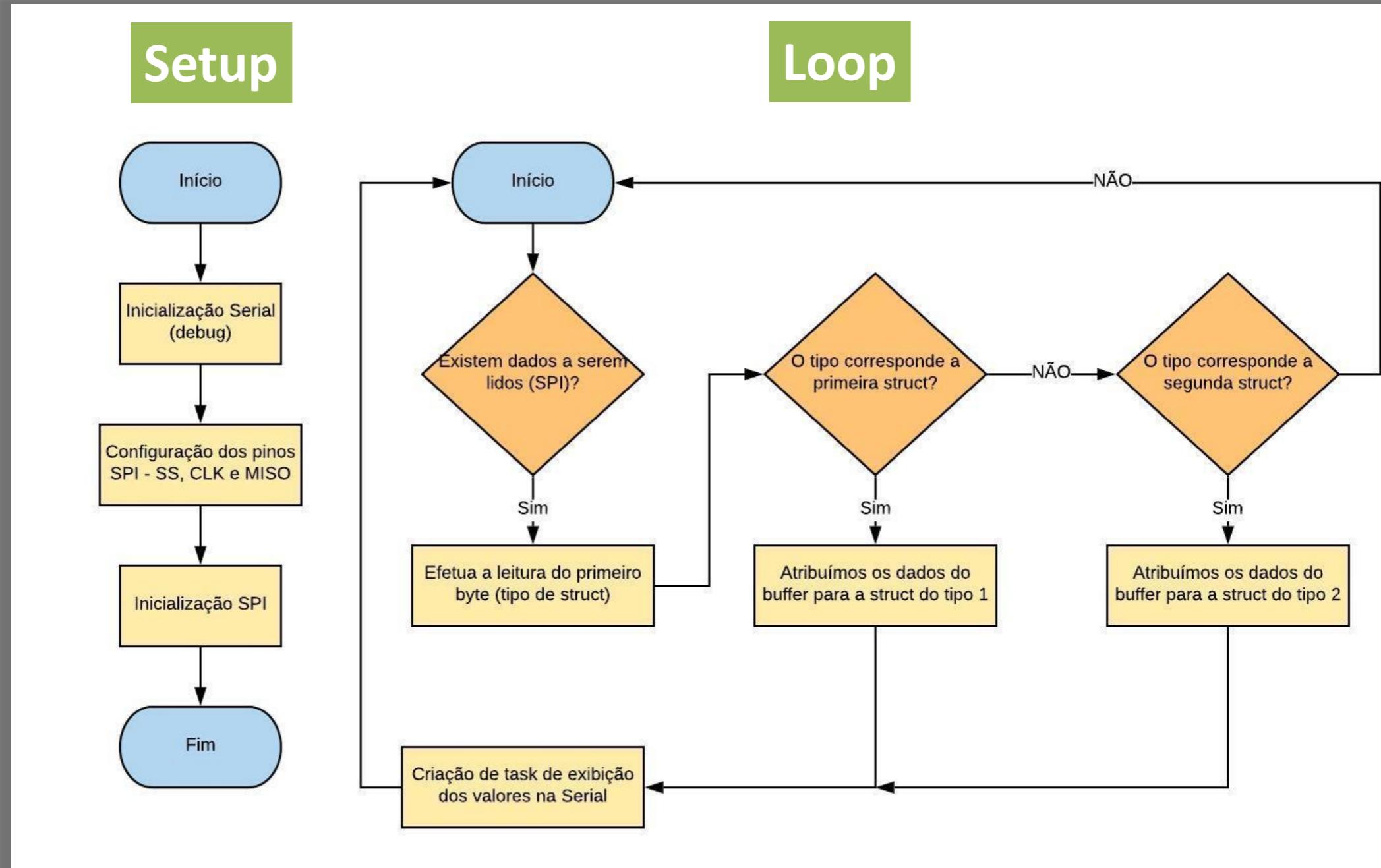


Visualização no Monitor Serial (ESP)



Código ESP32

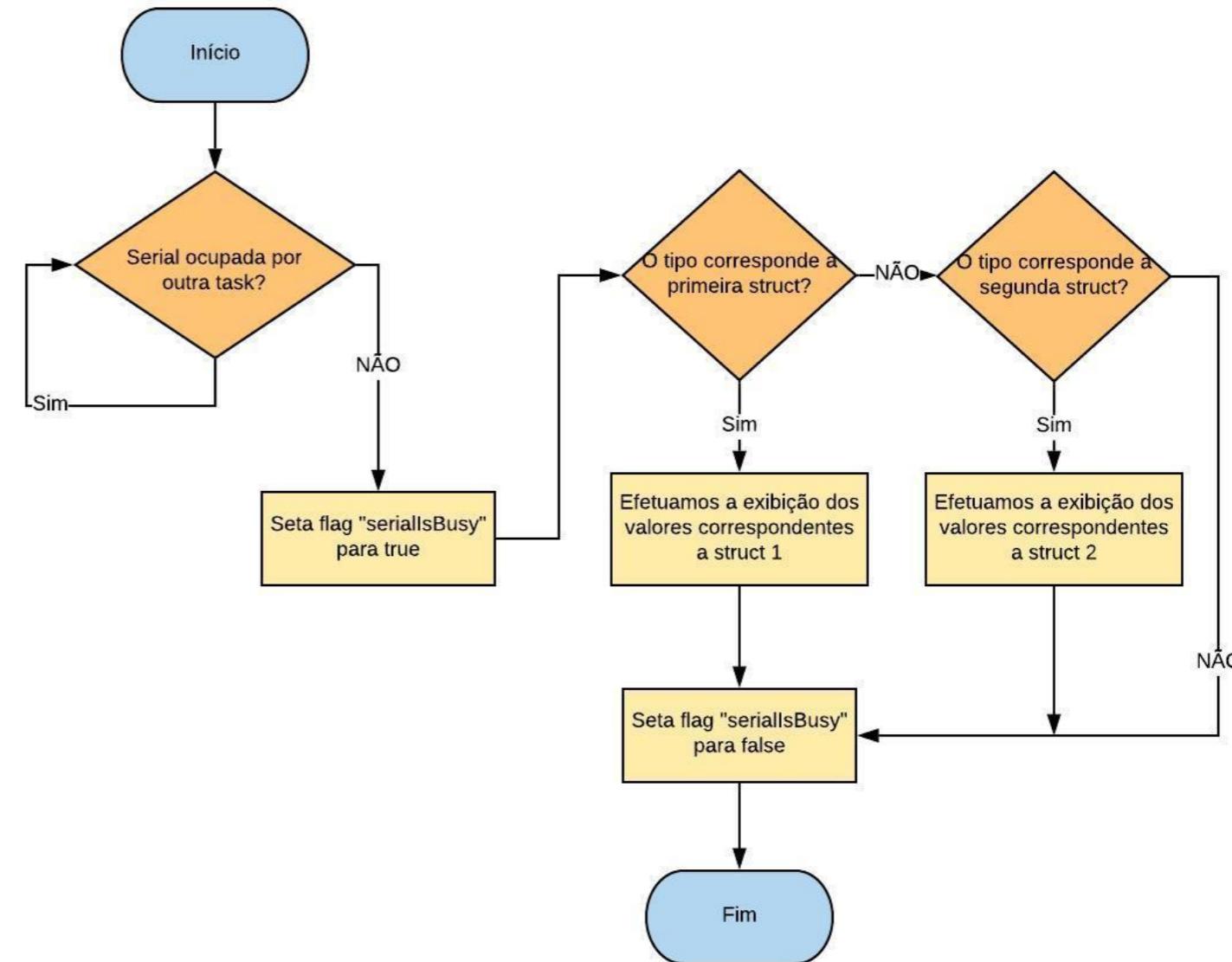
Fluxograma



Código ESP32

Fluxograma

taskPrintValues



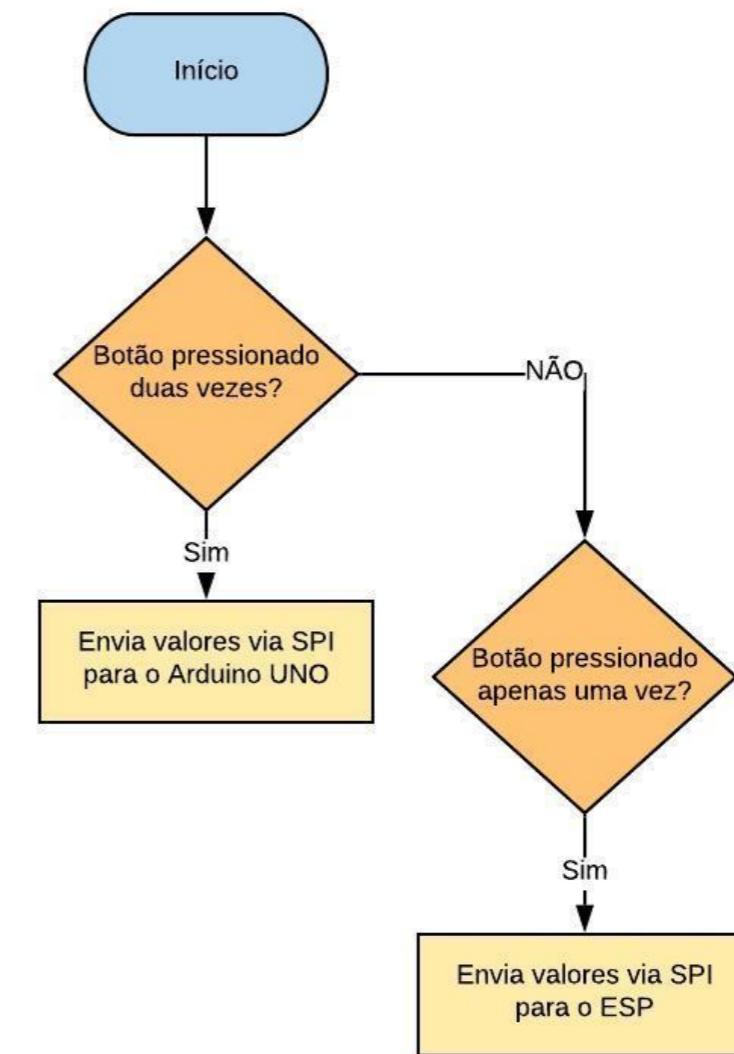
Código Mega

Fluxograma

Setup



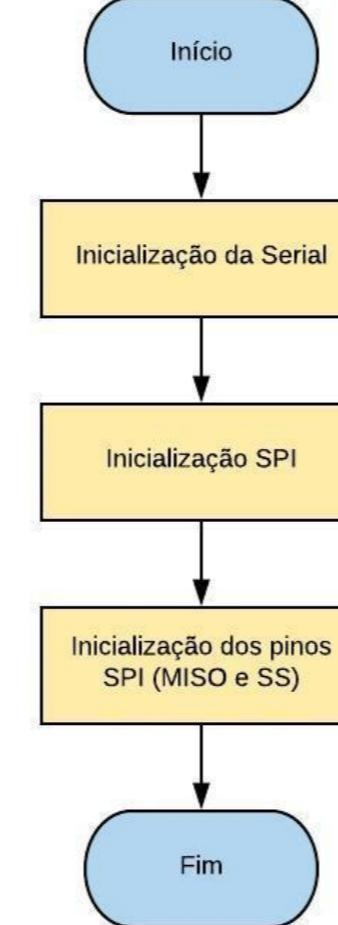
Loop



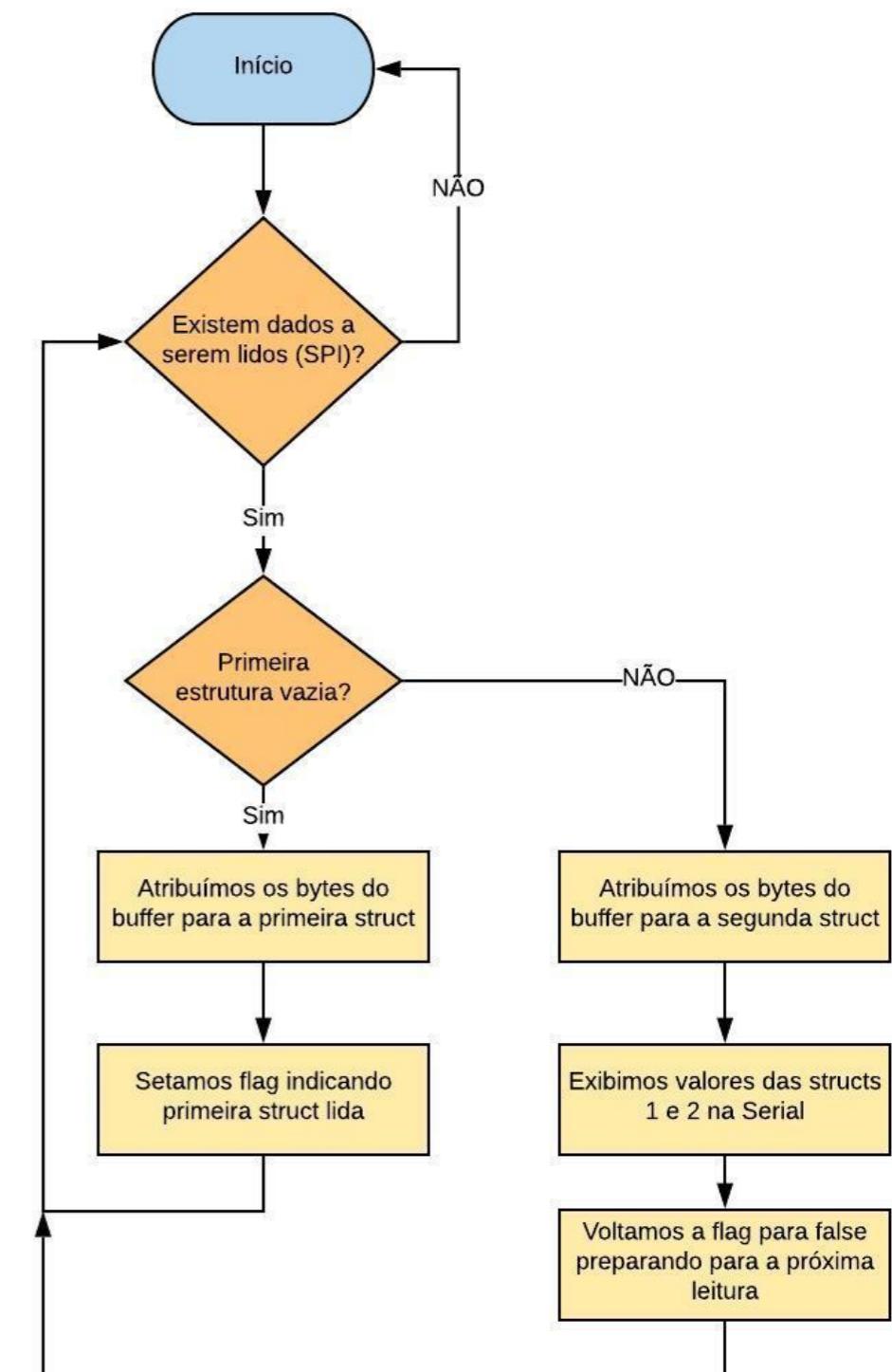
Código UNO

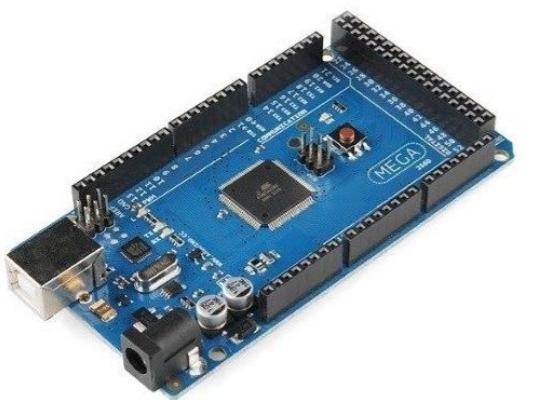
Fluxograma

Setup



Loop





Código

Código Mega Declarações e Variáveis

```
// Biblioteca SPI
#include <SPI.h>
// Pinos SPI default utilizados
#define GPIO_MISO 50
#define GPIO_MOSI 51
#define GPIO_SCK 52
#define GPIO_SS_ESP 53
#define GPIO_SS_UNO 49
// Botão que servirá como disparo de envio SPI
int eventPin = 13;

// Estruturas que guardarão os valores dos pinos Digitais e AD
// As estruturas devem possuir no máximo 32 bytes (tamanho máximo de transações SPI). Um valor digital ocupa 1 byte de espaço, enquanto um valor AD ocupa 2 bytes
// A variável type indica qual o tipo de estrutura foi enviado, assim, do lado do slave, podemos identificar qual é a estrutura que foi recebida

// A struct "digitalPinsUntil33" guarda os valores dos pinos D2 a D33, exceto o D13 (pino usado para o botão). Possui um tamanho de 32 bytes
struct digitalPinsUntil33
{
    uint8_t type = 1;
    uint8_t digitalPins[31];
};
```

Código Mega Declarações e Variáveis

```
// A struct "digitalAndADPins" guarda os valores dos pinos D32 a D49 e também  
os A0 a A6. Possui um tamanho de 30 bytes  
struct digitalAndADPins  
{  
    uint8_t type = 2;  
    uint8_t digitalPins[15];  
    uint16_t ADPins[7];  
};  
  
// Declaração das variáveis especiais (structs) globais  
struct digitalPinsUntil33 DPins1;  
struct digitalAndADPins Dpins2_ADPins1;
```

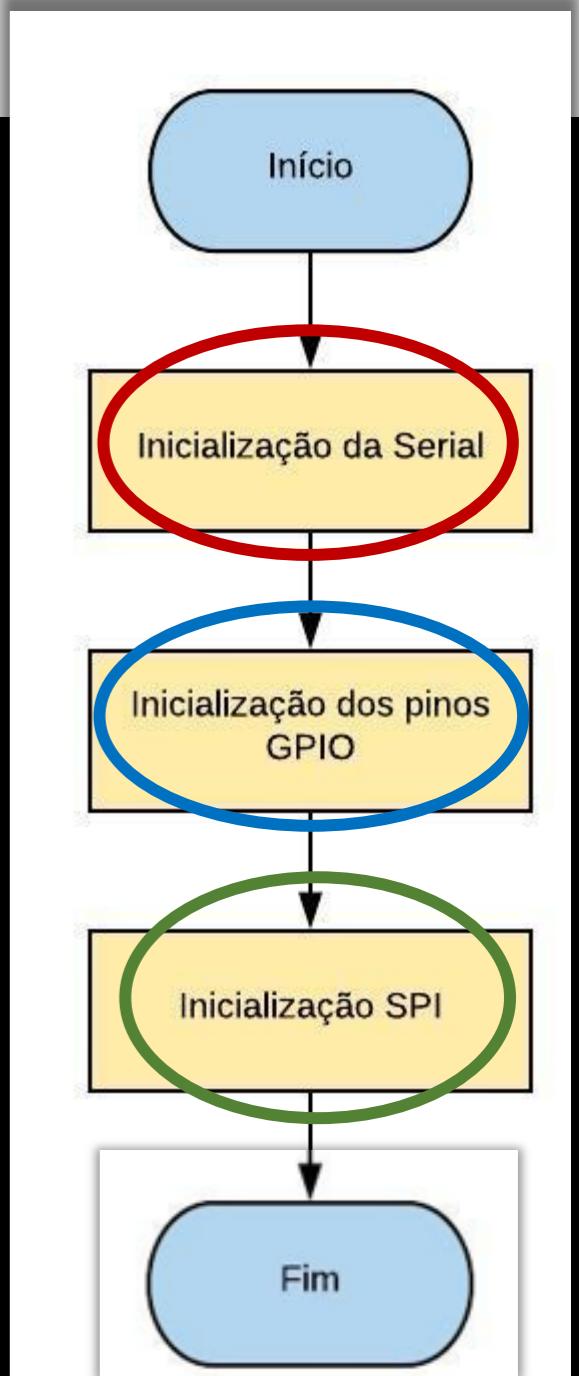
Código Mega setup

```
void setup()
{
    // Iniciamos a velocidade da serial em 115200 para debug
    → Serial.begin(115200);

    → setupGPIOs();
    /*
        // Caso seja necessário saber qual o tamanho da estrutura basta descomentar o
        código abaixo e visualizar no monitor serial
        Serial.println((int)sizeof(digitalPinsUntil33)); // Tamanho de 32 bytes
        Serial.println((int)sizeof(digitalAndADPins)); // Tamanho de 31 bytes
    */
    // Iniciamos a SPI
    → SPI.begin();

    // Setamos o pino do botão como entrada
    → pinMode(eventPin, INPUT);
    // Setamos os pinos slave select como saída
    → pinMode(GPIO_SS_ESP, OUTPUT);
    → pinMode(GPIO_SS_UNO, OUTPUT);

    // Setamos o pino Clock como saída
    → pinMode(GPIO_SCK, OUTPUT);
```



Código Mega setup

```
// Setamos o pino Clock para baixo (SPI_MODE0)
digitalWrite(GPIO_SCK, LOW);

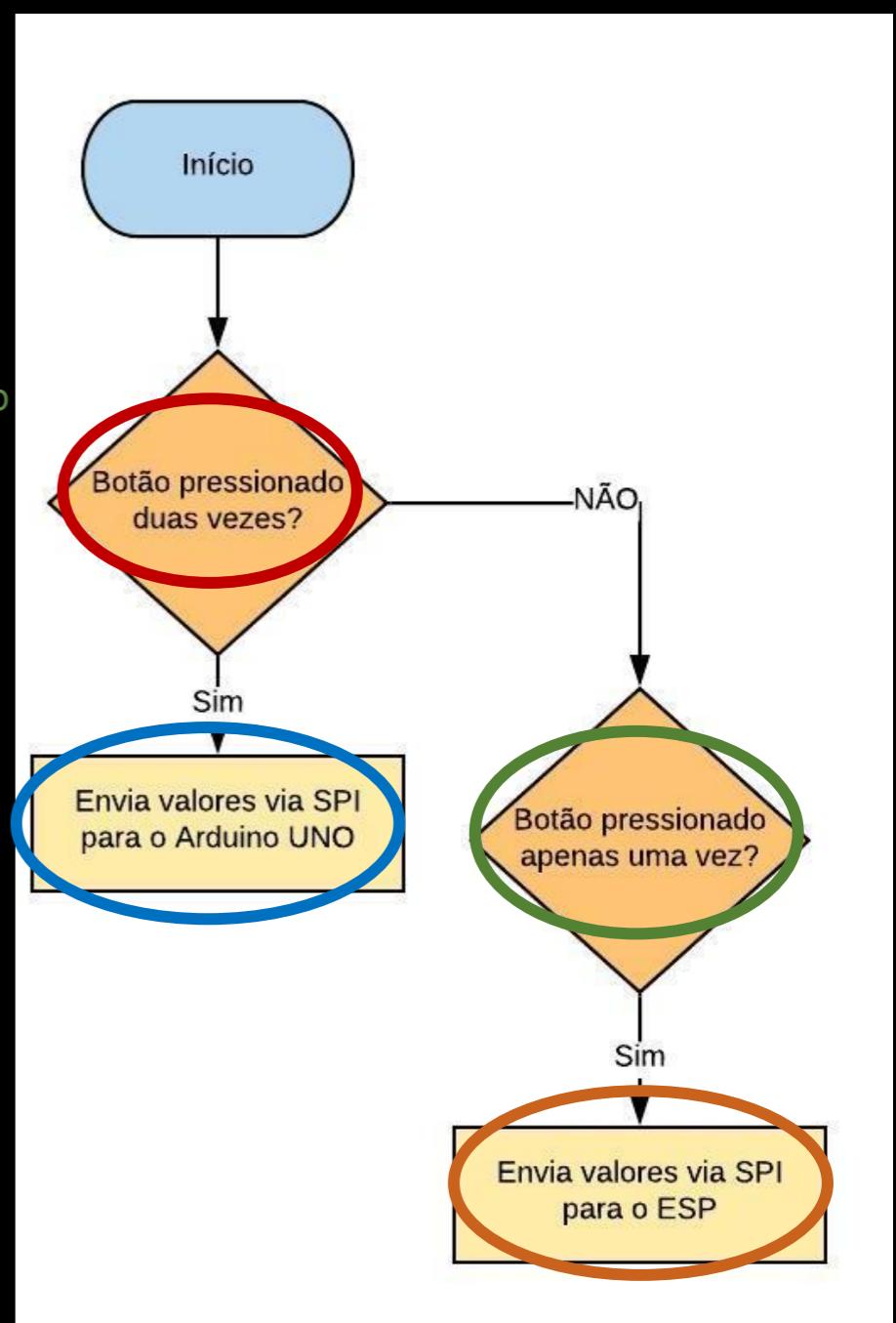
// Deixamos os slaves desabilitado (quando low significa que uma nova
transação será feita)
digitalWrite(GPIO_SS_ESP, HIGH);
digitalWrite(GPIO_SS_UNO, HIGH);

Serial.println("Master Initialized");
delay(10);
}
```



Código Mega loop

```
void loop()
{
    // Se o botão foi pressionado
    if(digitalRead(eventPin))
    {
        // Aguarda soltar o botão
        while(digitalRead(eventPin))
            delay(1);
        // Aguarda 1 segundo, se o botão for pressionado novamente enviamos para o
UNO, senão, enviamos para o ESP32
        → if(buttonPressed(1000))
        {
            // Envia para o arduino
            Serial.println("Enviando para o Arduino UNO");
            → sendValuesToUNO();
        }
        → else
        {
            // Envia para o ESP
            Serial.println("Enviando para o ESP32");
            → sendValuesToESP();
        }
        while(digitalRead(eventPin))// Aguarda soltar o botão
            delay(1);
    }
    → else
        delay(10);
}
```



Código Mega *sendValuesToUNO*

```
// Função que envia as 2 structs para o Arduino UNO
void sendValuesToUNO()
{
    Serial.println("Lendo pinos");

    // Efetuamos as leituras de todos os pinos
    readPins();

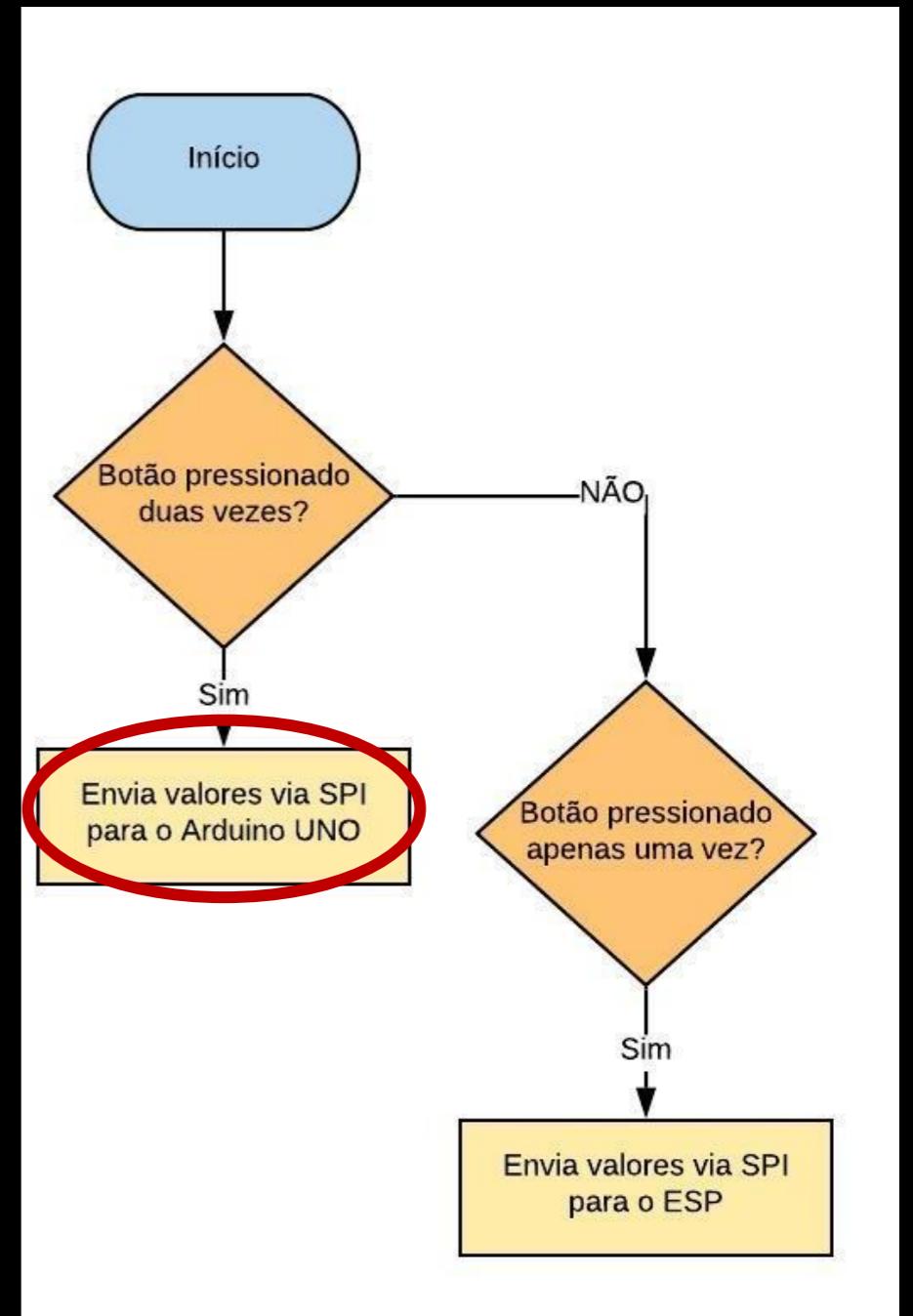
    Serial.println("Enviando parte 1");

    // Enviamos a primeira estrutura em uma transação
    sendStruct(DPins1, GPIO_SS_UNO);
    delay(500);

    Serial.println("Enviando parte 2");

    // Enviamos a segunda estrutura em outra transação
    sendStruct(Dpins2_ADPins1, GPIO_SS_UNO);
    delay(500);

    Serial.println("OK");
}
```



Código Mega *sendValuesToESP*

```
// Função que envia as 2 structs para o ESP32
void sendValuesToESP()
{
    Serial.println("Lendo pinos");

    // Efetuamos as leituras de todos os pinos
    readPins();

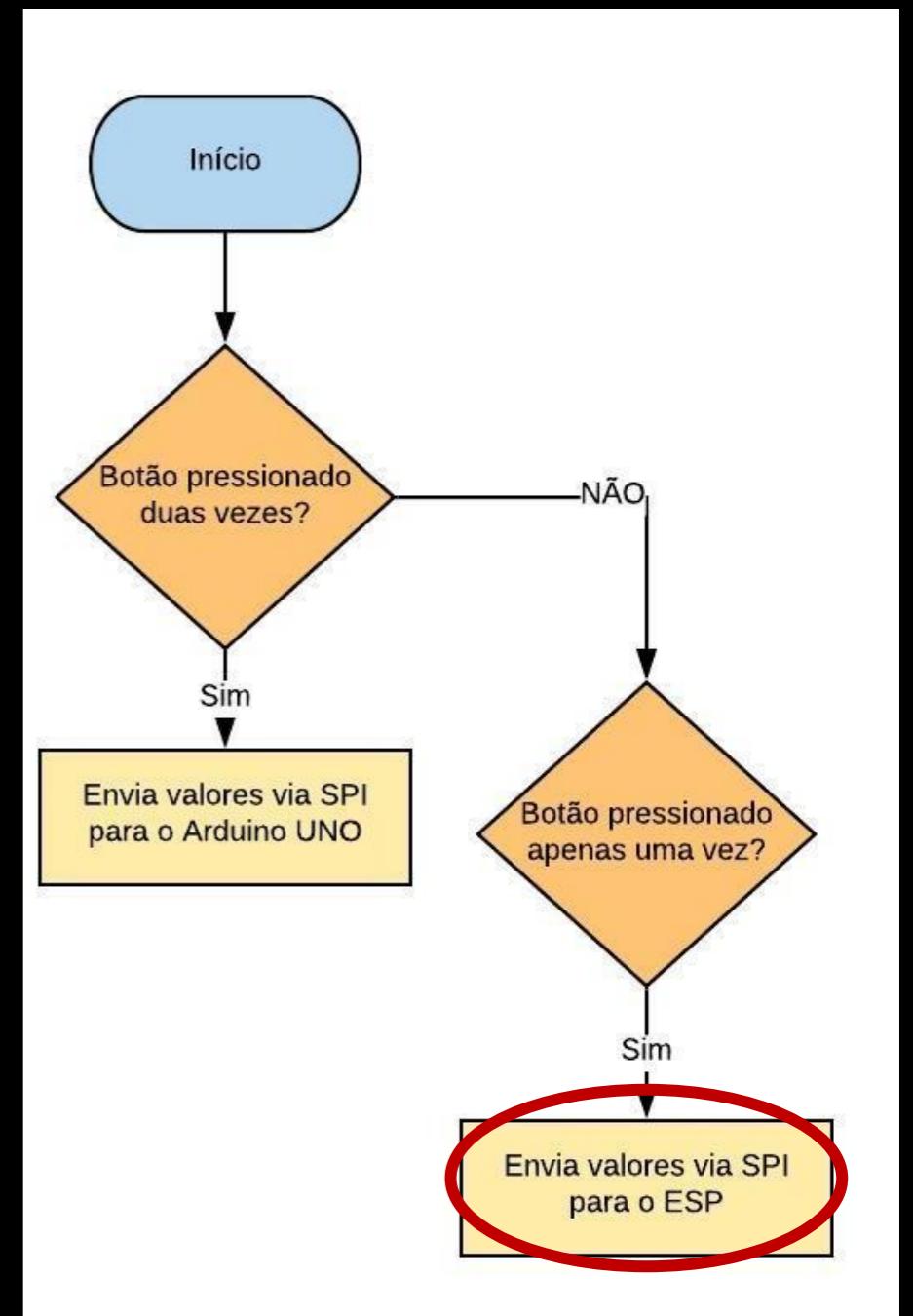
    Serial.println("Enviando parte 1");

    // Enviamos a primeira estrutura em uma transação
    sendStruct(DPins1, GPIO_SS_ESP);
    delay(150);

    Serial.println("Enviando parte 2");

    // Enviamos a segunda estrutura em outra transação
    sendStruct(Dpins2_ADPPins1, GPIO_SS_ESP);
    delay(150);

    Serial.println("OK");
}
```

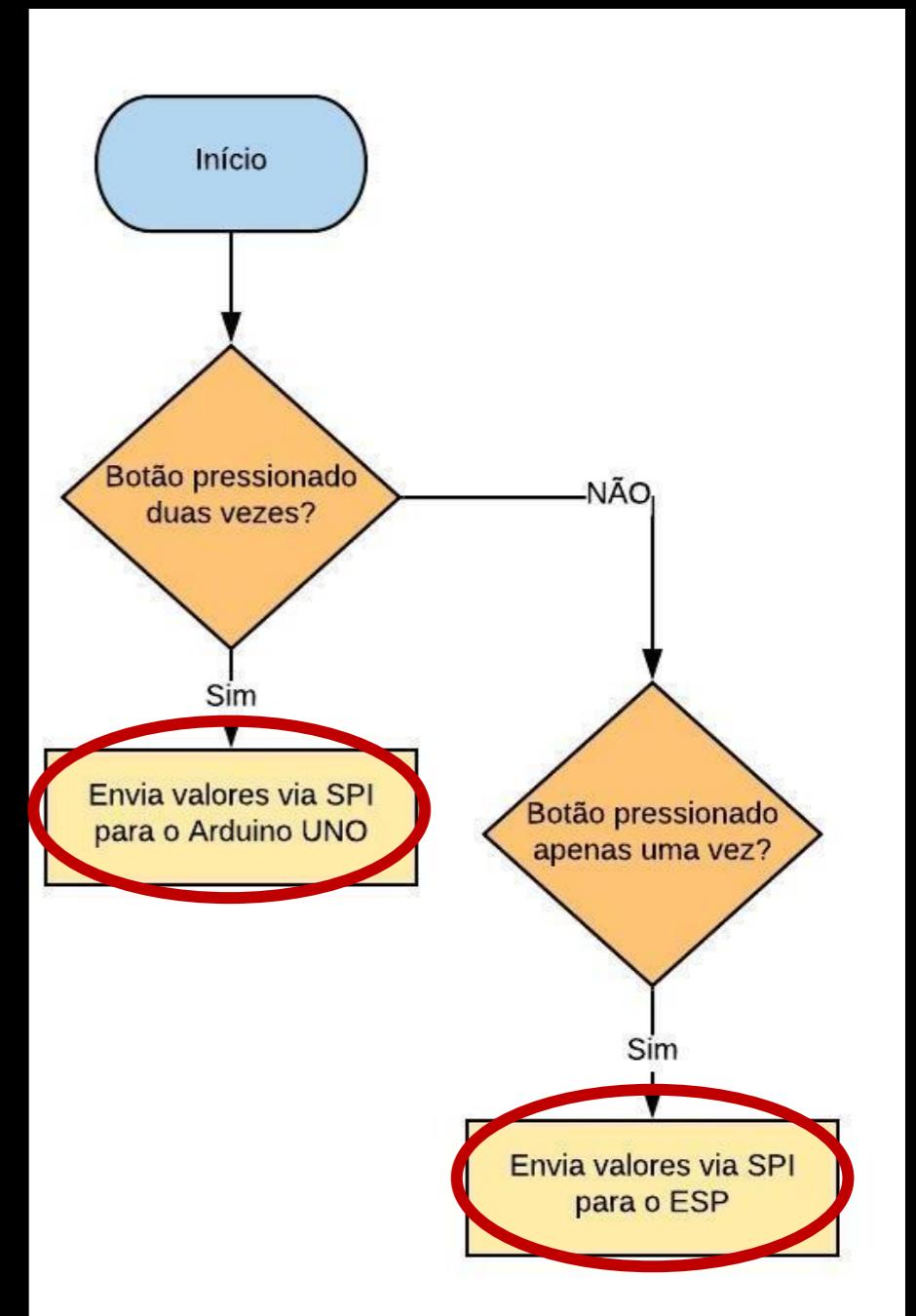


Código Mega sendStruct

```
// Função que envia os valores da estrutura do tipo 1 via SPI
void sendStruct(struct digitalPinsUntil33 dados, int SSPin)
{
    // Selecionamos o slave correspondente ao pino SSPin
(arduino uno ou esp)
    digitalWrite(SSPin, LOW);
    delay(100);

    // Criamos um ponteiro que aponta para o primeiro byte da
estrutura 'dados'
    uint8_t *p = (uint8_t*)&dados;
    // Percorremos byte a byte da estrutura, enviando por vez,
até que todos os 32 bytes sejam enviados
    for(int i=0; i<(int)sizeof(digitalPinsUntil33); i++)
    {
        SPI.transfer(*p++);
        delay(5);
    }

    delay(100);
    // Desativamos o slave correspondente ao pino SSPin
(arduino uno ou esp)
    digitalWrite(SSPin, HIGH);
}
```

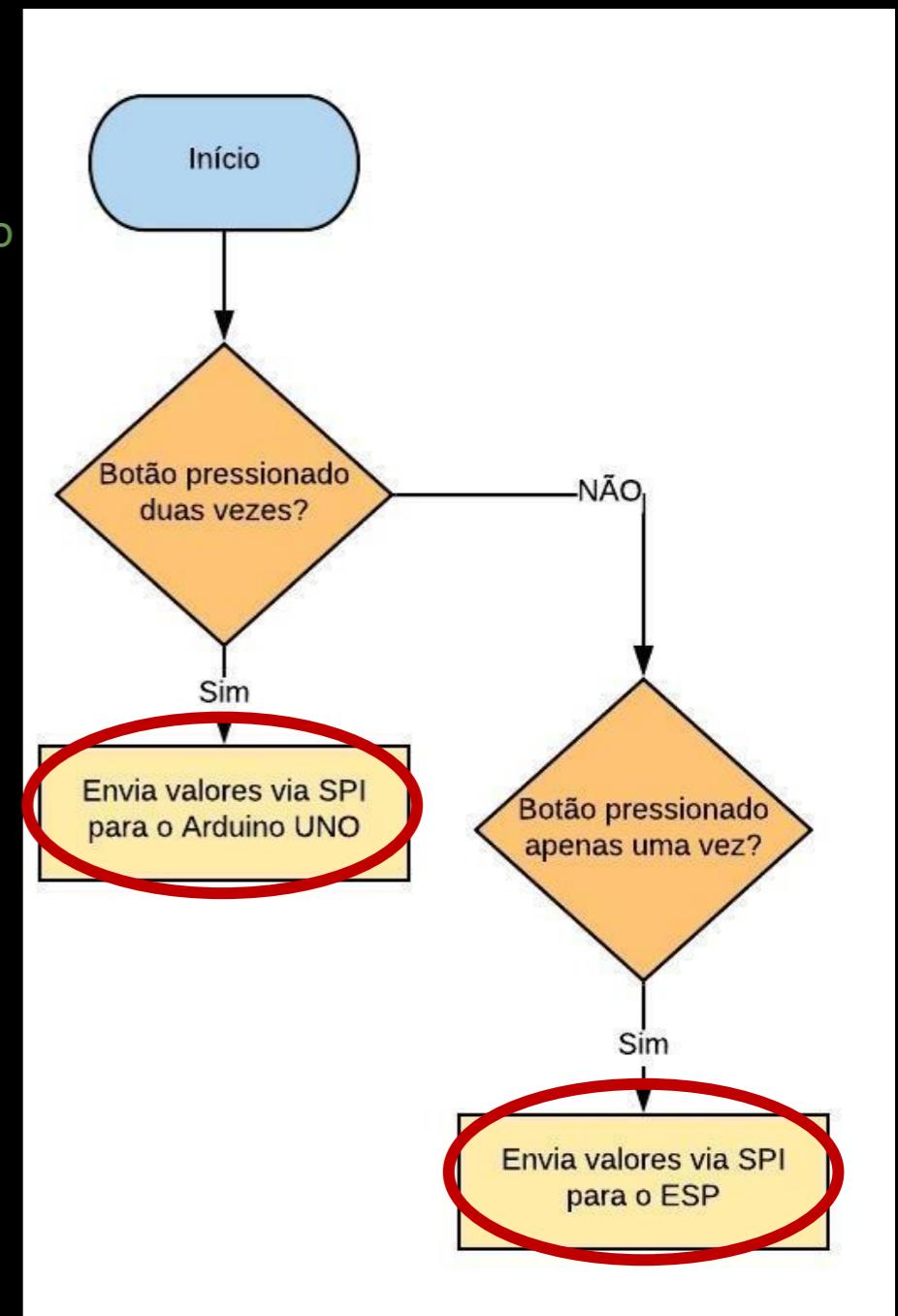


Código Mega sendStruct

```
// Função que envia os valores da estrutura do tipo 2 via SPI
void sendStruct(struct digitalAndADPins dados, int SSPin)
{
    // Selecionamos o slave correspondente ao pino SSPin (arduino uno
    digitalWrite(SSPin, LOW);
    delay(100);

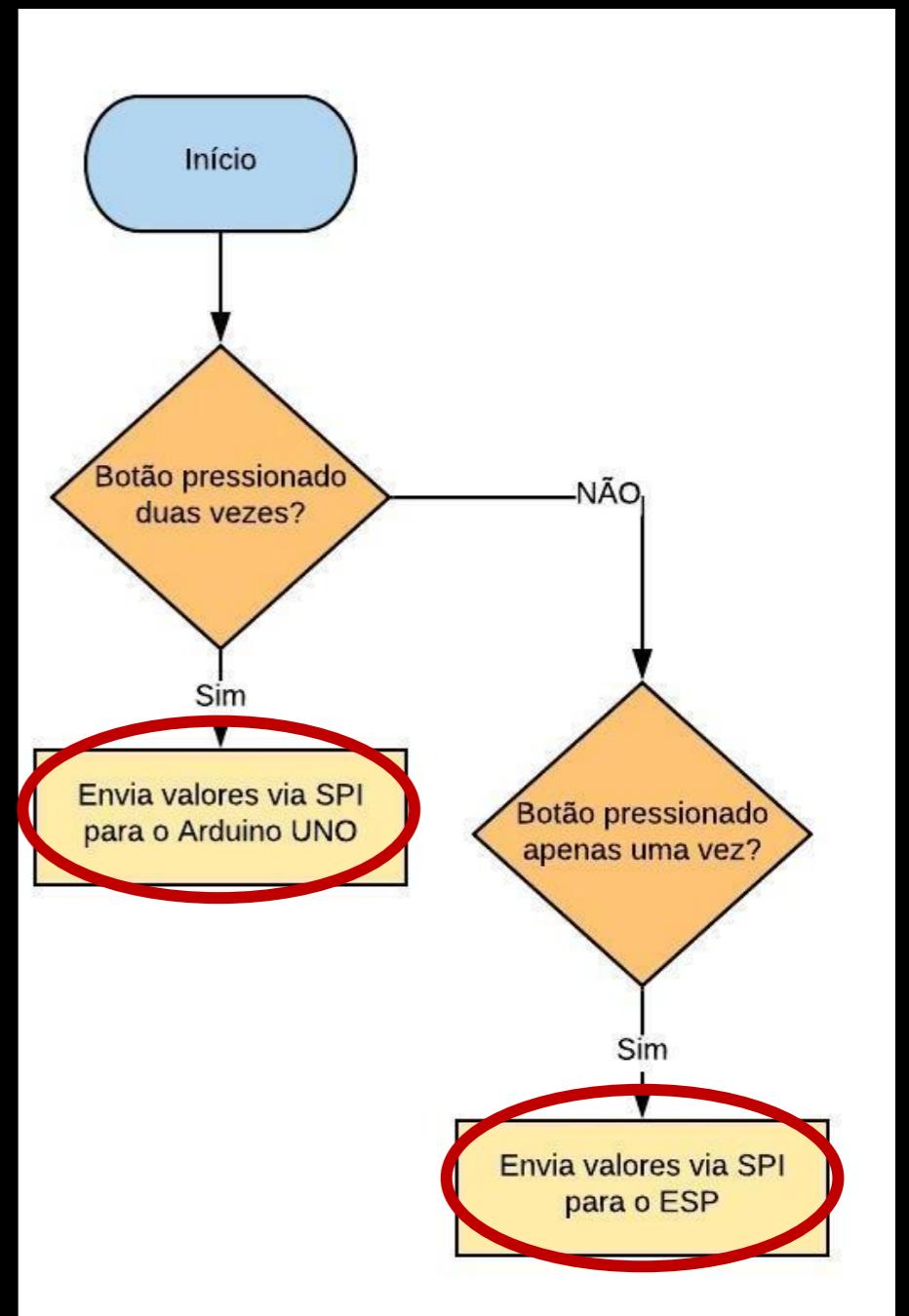
    // Criamos um ponteiro que aponta para o primeiro byte da
estrutura 'dados'
    uint8_t *p = (uint8_t*)&dados;
    // Percorremos byte a byte da estrutura, enviando um por vez,
até que todos os 31 bytes sejam enviados
    for(int i=0; i<(int)sizeof(digitalAndADPins); i++)
    {
        SPI.transfer(*p++);
        delay(5);
    }

    delay(100);
    // Desativamos o slave correspondente ao pino SSPin
    (arduino uno ou esp)
    digitalWrite(SSPin, HIGH);
}
```



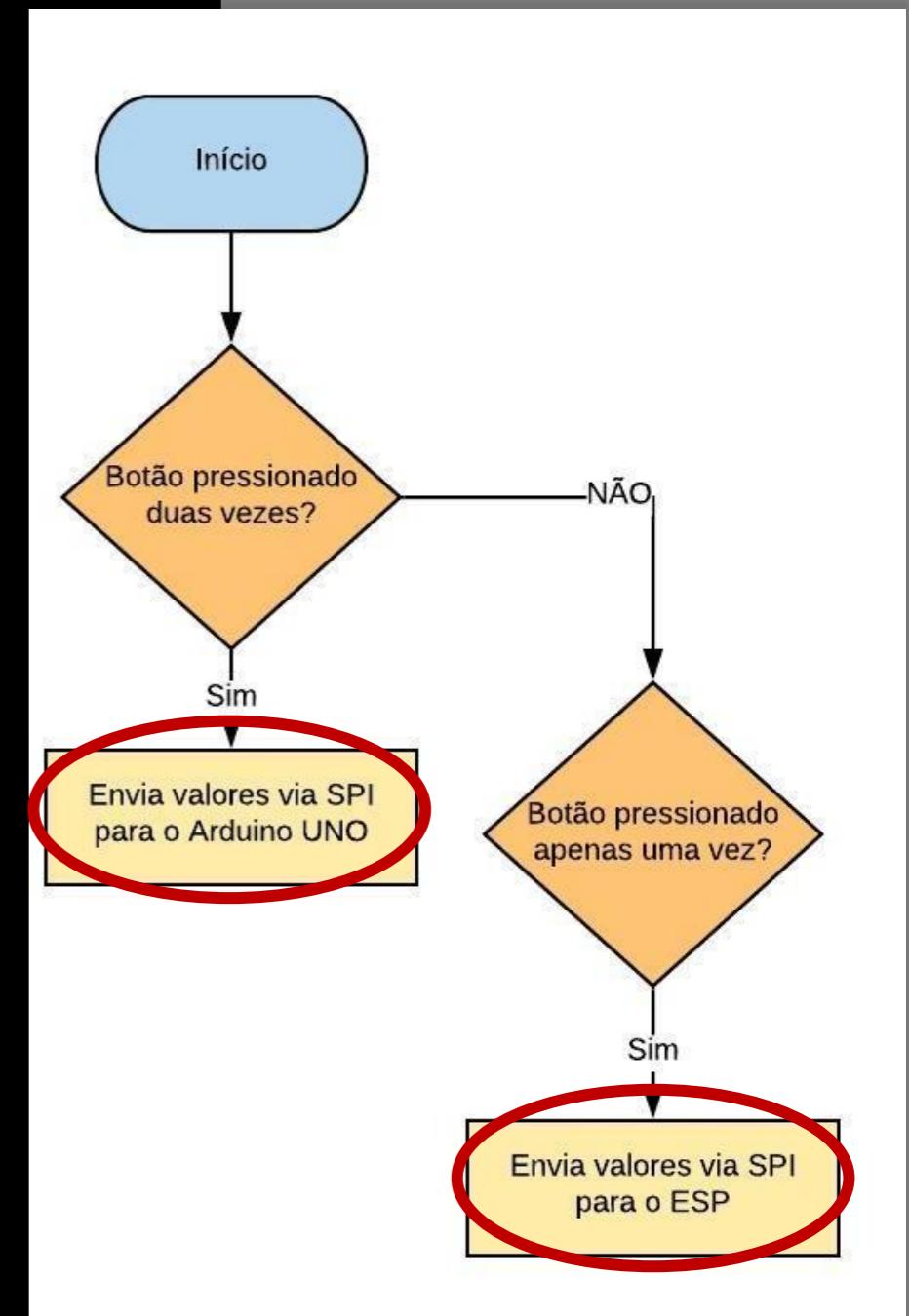
Código Mega readPins

```
// Função que lê os pinos e guarda seus valores nas estruturas
void readPins()
{
    // Estrutura do tipo 1: digitalPinsUntil13
    DPins1.digitalPins[0] = digitalRead(2);
    DPins1.digitalPins[1] = digitalRead(3);
    DPins1.digitalPins[2] = digitalRead(4);
    DPins1.digitalPins[3] = digitalRead(5);
    DPins1.digitalPins[4] = digitalRead(6);
    DPins1.digitalPins[5] = digitalRead(7);
    DPins1.digitalPins[6] = digitalRead(8);
    DPins1.digitalPins[7] = digitalRead(9);
    DPins1.digitalPins[8] = digitalRead(10);
    DPins1.digitalPins[9] = digitalRead(11);
    DPins1.digitalPins[10] = digitalRead(12);
    // O Botão está ligado na gpio13, então não faz parte dos pinos
    válidos
    DPins1.digitalPins[11] = digitalRead(14);
    DPins1.digitalPins[12] = digitalRead(15);
    DPins1.digitalPins[13] = digitalRead(16);
```



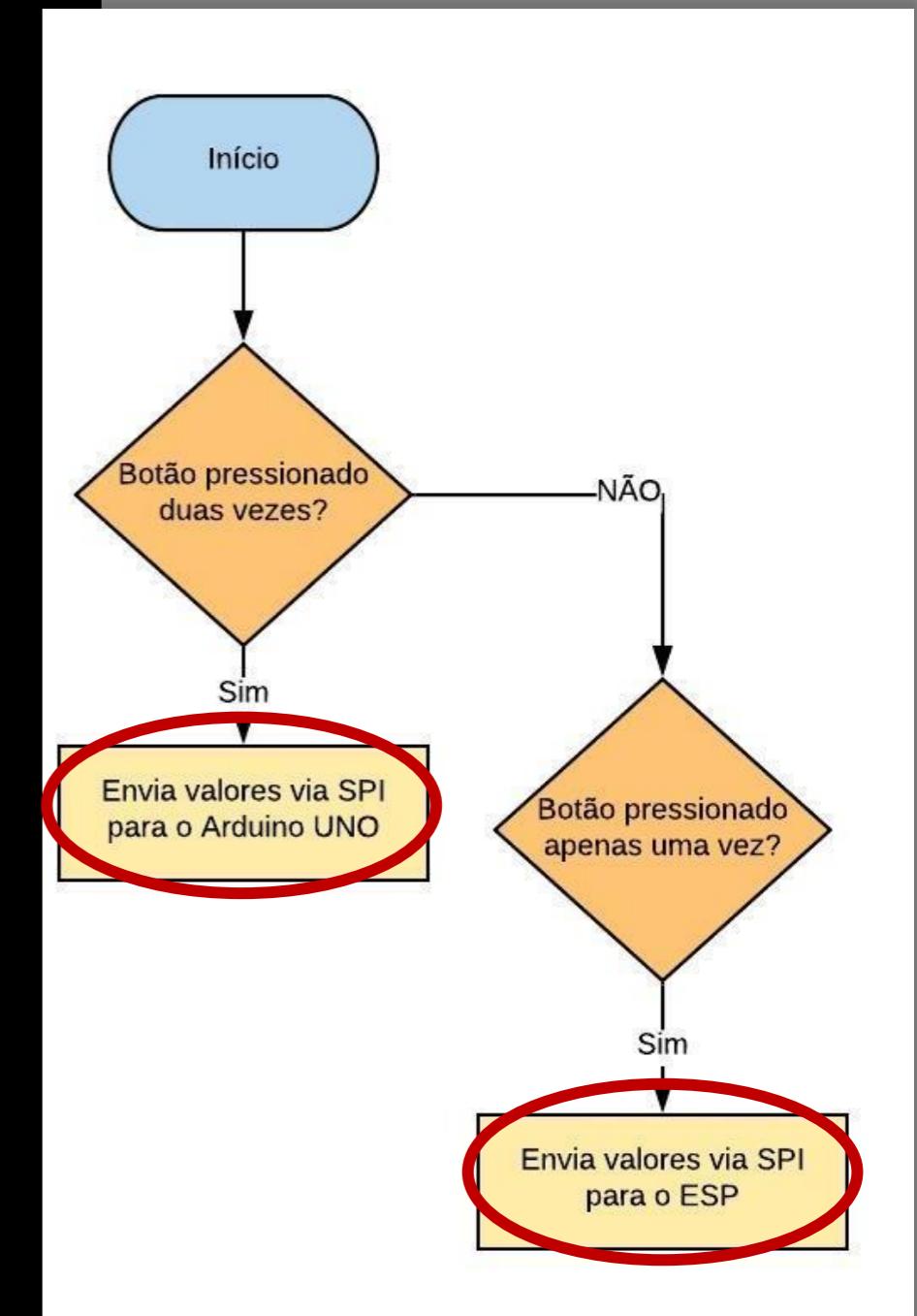
Código Mega readPins

```
DPins1.digitalPins[14] = digitalRead(17);
DPins1.digitalPins[15] = digitalRead(18);
DPins1.digitalPins[16] = digitalRead(19);
DPins1.digitalPins[17] = digitalRead(20);
DPins1.digitalPins[18] = digitalRead(21);
DPins1.digitalPins[19] = digitalRead(22);
DPins1.digitalPins[20] = digitalRead(23);
DPins1.digitalPins[21] = digitalRead(24);
DPins1.digitalPins[22] = digitalRead(25);
DPins1.digitalPins[23] = digitalRead(26);
DPins1.digitalPins[24] = digitalRead(27);
DPins1.digitalPins[25] = digitalRead(28);
DPins1.digitalPins[26] = digitalRead(29);
DPins1.digitalPins[27] = digitalRead(30);
DPins1.digitalPins[28] = digitalRead(31);
DPins1.digitalPins[29] = digitalRead(32);
DPins1.digitalPins[30] = digitalRead(33);
```



Código Mega readPins

```
// Estrutura do tipo 2: digitalAndADPins  
Dpins2_ADPins1.digitalPins[0] = digitalRead(34);  
Dpins2_ADPins1.digitalPins[1] = digitalRead(35);  
Dpins2_ADPins1.digitalPins[2] = digitalRead(36);  
Dpins2_ADPins1.digitalPins[3] = digitalRead(37);  
Dpins2_ADPins1.digitalPins[4] = digitalRead(38);  
Dpins2_ADPins1.digitalPins[5] = digitalRead(39);  
Dpins2_ADPins1.digitalPins[6] = digitalRead(40);  
Dpins2_ADPins1.digitalPins[7] = digitalRead(41);  
Dpins2_ADPins1.digitalPins[8] = digitalRead(42);  
Dpins2_ADPins1.digitalPins[9] = digitalRead(43);  
Dpins2_ADPins1.digitalPins[10] = digitalRead(44);  
Dpins2_ADPins1.digitalPins[11] = digitalRead(45);  
Dpins2_ADPins1.digitalPins[12] = digitalRead(46);  
Dpins2_ADPins1.digitalPins[13] = digitalRead(47);  
Dpins2_ADPins1.digitalPins[14] = digitalRead(48);  
//Dpins2_ADPins1.digitalPins[15] = digitalRead(49);  
Dpins2_ADPins1.ADPins[0] = analogRead(A0);  
Dpins2_ADPins1.ADPins[1] = analogRead(A1);  
Dpins2_ADPins1.ADPins[2] = analogRead(A2);  
Dpins2_ADPins1.ADPins[3] = analogRead(A3);  
Dpins2_ADPins1.ADPins[4] = analogRead(A4);  
Dpins2_ADPins1.ADPins[5] = analogRead(A5);  
Dpins2_ADPins1.ADPins[6] = analogRead(A6);  
}
```

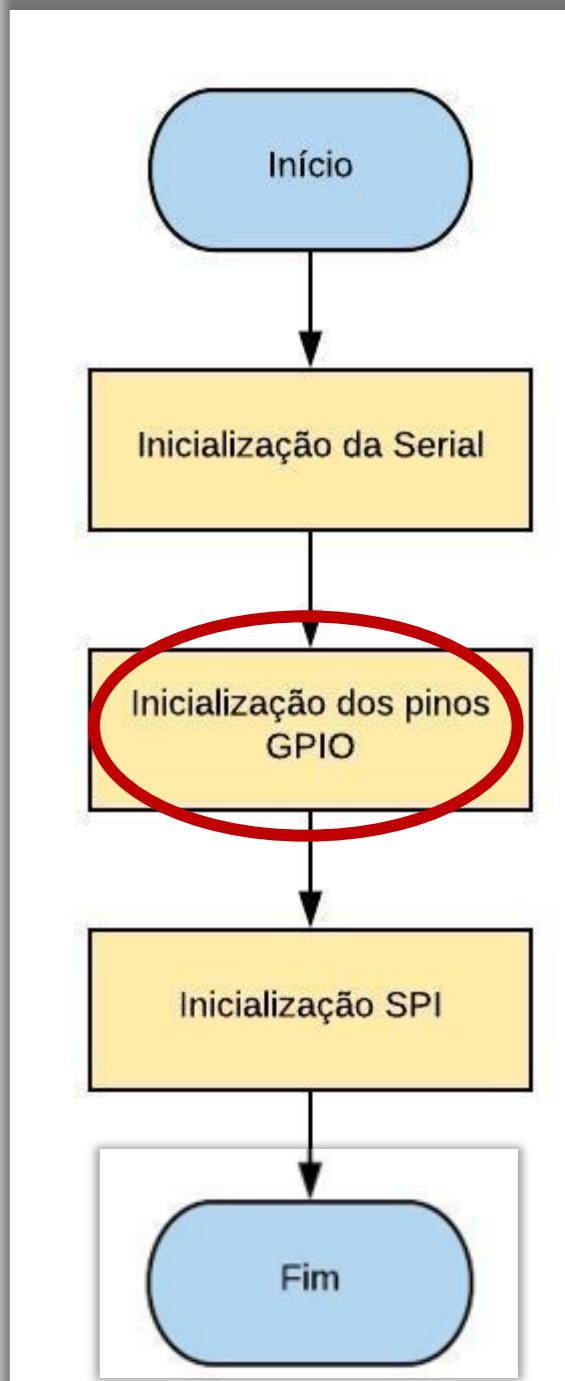


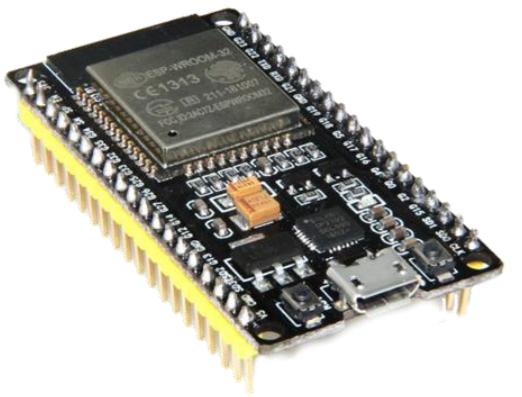
Código Mega *setupGPIOs*

```
// Função que seta os pinos como input
void setupGPIOs()
{
    for(int i=2; i<13; i++)
        pinMode(i, INPUT);

    for(int i=14; i<49; i++)
        pinMode(i, INPUT);

    pinMode(A0, INPUT);
    pinMode(A1, INPUT);
    pinMode(A2, INPUT);
    pinMode(A3, INPUT);
    pinMode(A4, INPUT);
    pinMode(A5, INPUT);
    pinMode(A6, INPUT);
}
```





Código

Código ESP Declarações e Variáveis

```
// Biblioteca SlaveSPI para ESP32
#include "SlaveSPI/SlaveSPI.h"
// Biblioteca SPI
#include <SPI.h>

// Pinos SPI utilizados
#define _MISO    (gpio_num_t)19
#define _MOSI    (gpio_num_t)23
#define _SCK     (gpio_num_t)18
#define _SS      (gpio_num_t)5

// Objeto SlaveSPI, setamos para virtual SPI (VSPI)
SlaveSPI slave(VSPI_HOST);

// Variável usada para que duas tasks não exibam na serial ao mesmo tempo, evitando que os
// valores se embaralhem durante suas exibições
bool serialIsBusy = false;
```

Código ESP *Declarações e Variáveis*

```
// Variáveis que correspondem ao código do Arduino Mega (explicação no código do Arduino Mega)
struct digitalPinsUntil33
{
    uint8_t type;
    uint8_t digitalPins[31];
};

struct digitalAndADPins
{
    uint8_t type;
    uint8_t digitalPins[15];
    uint16_t ADPins[7];
};

struct digitalPinsUntil33 DPins1;
struct digitalAndADPins Dpins2_ADPins1;

void readData(uint8_t type);
```

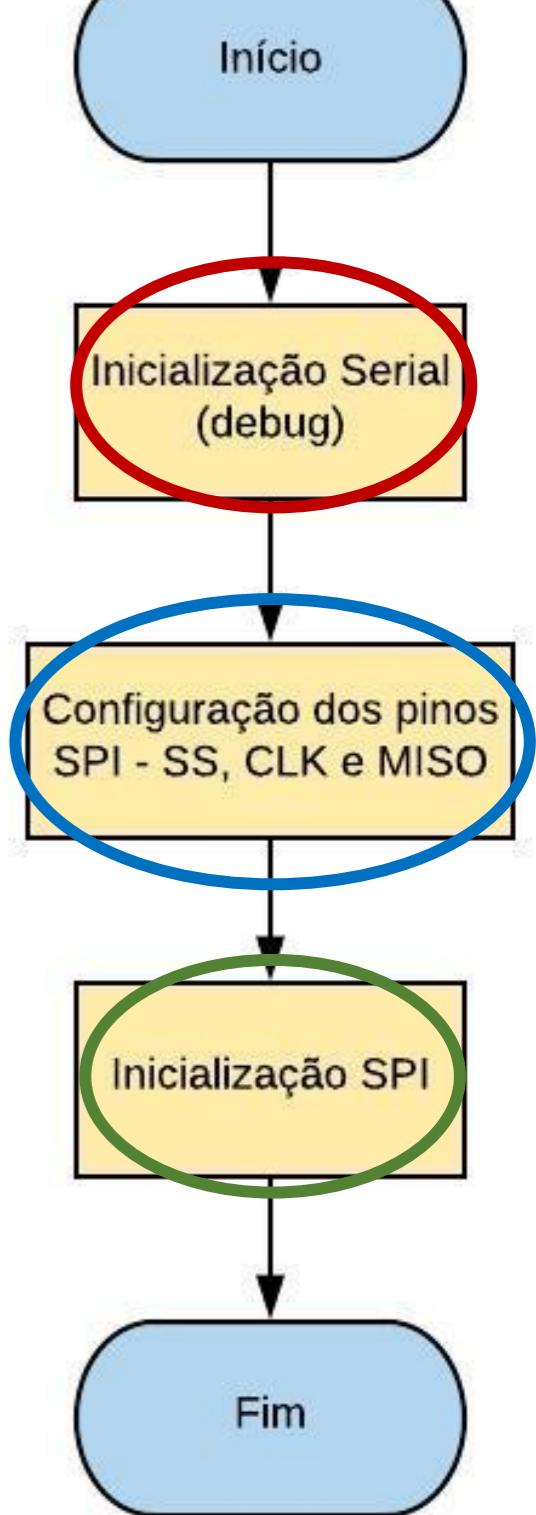
Código ESP Setup

```
void setup()
{
    // Desabilitamos o watchdog de hardware do core 0
    disableCore0WDT();

    // Setamos a velocidade da serial para 115200
    → Serial.begin(115200);

    // Setamos o pino Slave Select como entrada
    pinMode(_SS, INPUT);
    // Setamos o pino Clock como entrada
    pinMode(_SCK, INPUT);
    // Setamos o pino MISO como saída
    pinMode(_MISO, OUTPUT);

    // Iniciamos a SPI setando os pinos, tamanho máximo de transmissões
    // (32 bytes) e setando a função callback (opcional)
    → slave.begin(_MISO, _MOSI, _SCK, _SS, 32, callback_after_slave_tx_finish);
}
```

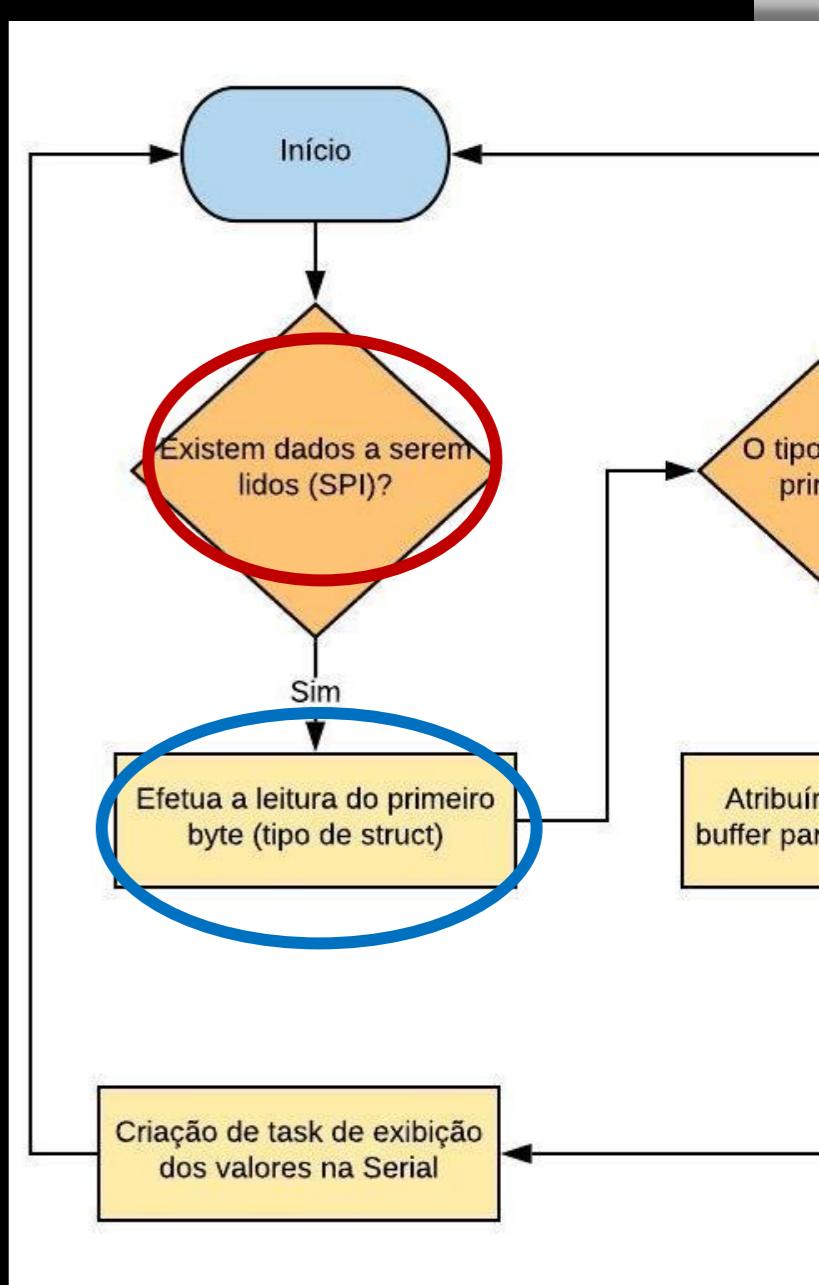


Código ESP Loop

```
void loop()
{
    // Se existem dados a serem lidos
    → if(slave.getInputStream()->length() && digitalRead(_SS) == HIGH)
    {
        uint8_t type;
        // Obtemos o primeiro byte, que indica o tipo de struct recebida
        → type = slave.getInputStream()->getBuffer()[0];

        // Efetuamos a leitura de acordo com o seu tipo
        readData(type);

        slave.flushInputStream();
    }
}
```

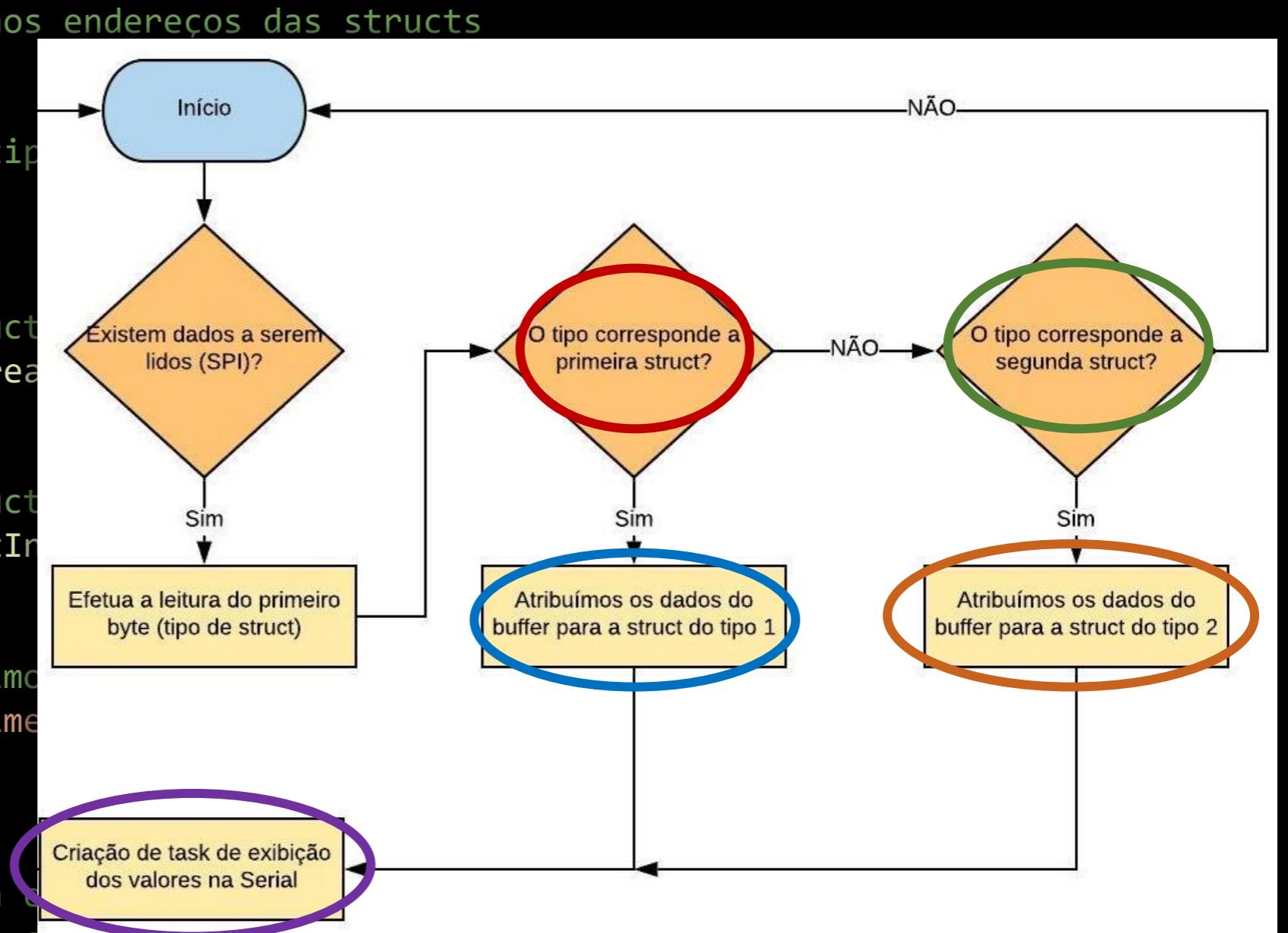


Código ESP *readData*

```
// Função que lê os dados, atribuindo aos endereços das structs
void readData(uint8_t type)
{
    // Executamos um switch/case para o tipo (type)
    switch(type)
    {
        case 1:
            // Atribuimos ao endereço da struct global DPins1, os valores em bytes recebidos por SPI
            memcpy(&DPins1, slave.getInputStream()->getBuffer(), sizeof(digitalPinsUntil33));
            break;
        case 2:
            // Atribuimos ao endereço da struct global Dpins2_ADPPins1, os valores em bytes recebidos por SPI
            memcpy(&Dpins2_ADPPins1, slave.getInputStream()->getBuffer(), sizeof(digitalAndADPins));
            break;
        default:
            // Se o tipo não for 1 ou 2 exibimos erro e abortamos a função
            Serial.println("Erro ao obter primeiro byte!");
            return;
            break;
    }
    // Executamos uma task no core 0 para exibirmos na serial os valores das structs
    // Assim podemos voltar para a rotina de leitura SPI mais rapidamente
    xTaskCreatePinnedToCore(taskPrintValues, "taskPrintValues", 10000, (void*)(int)type, 2, NULL, 0);
}
```

Código ESP *readData*

```
// Função que lê os dados, atribuindo aos endereços das structs
void readData(uint8_t type)
{
    // Executamos um switch/case para o tipo
    switch(type)
    {
        case 1:
            // Atribuimos ao endereço da struct
            memcpy(&DPins1, slave.getInputStream(), 1);
            break;
        case 2:
            // Atribuimos ao endereço da struct
            memcpy(&Dpins2_ADPins1, slave.getInputStream(), 1);
            break;
        default:
            // Se o tipo não for 1 ou 2 exibimos
            Serial.println("Erro ao obter primeiros bytes");
            return;
            break;
    }
    // Executamos uma task no core 0 para exibir os valores
    // Assim podemos voltar para a rotina de leitura SPI mais rapidamente
    xTaskCreatePinnedToCore(taskPrintValues, "taskPrintValues", 10000, (void*)(int)type, 2, NULL, 0);
}
```

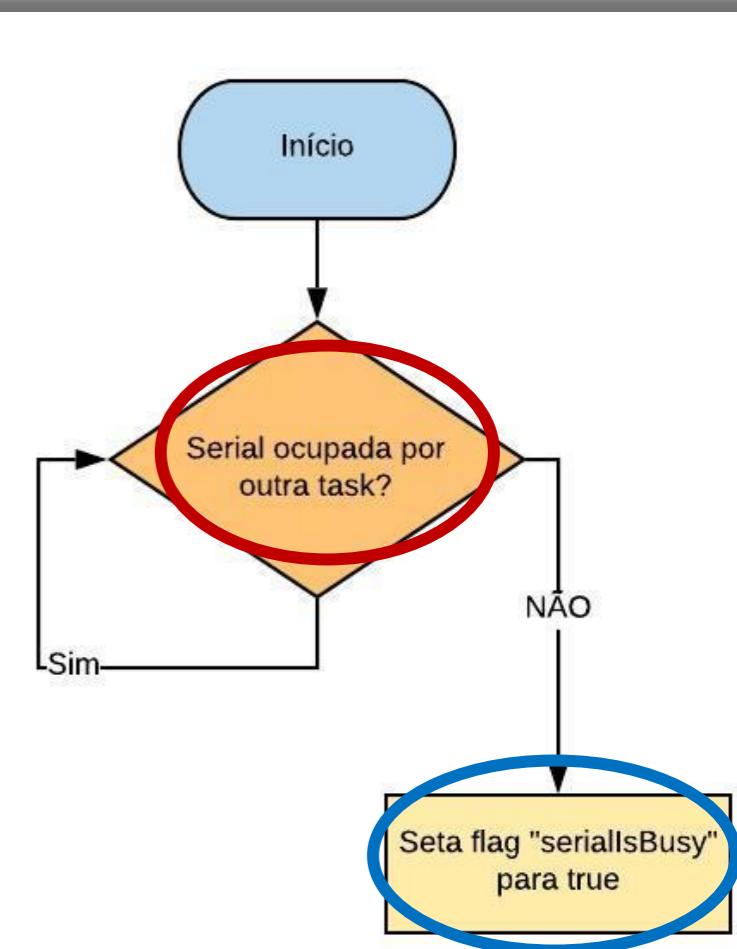


Código ESP *taskPrintValues*

```
// Task que exibe os valores dos pinos de acordo com o tipo de estrutura
void taskPrintValues(void *p)
{
    // Recebemos o tipo por parâmetro
    // 1 - primeira struct
    // 2 - segunda struct
    // Obs. Não é obrigatório enviar as estruturas na ordem, desde que
    seus tipos sejam enviados corretamente
    uint8_t type = *((uint8_t*)(&p));

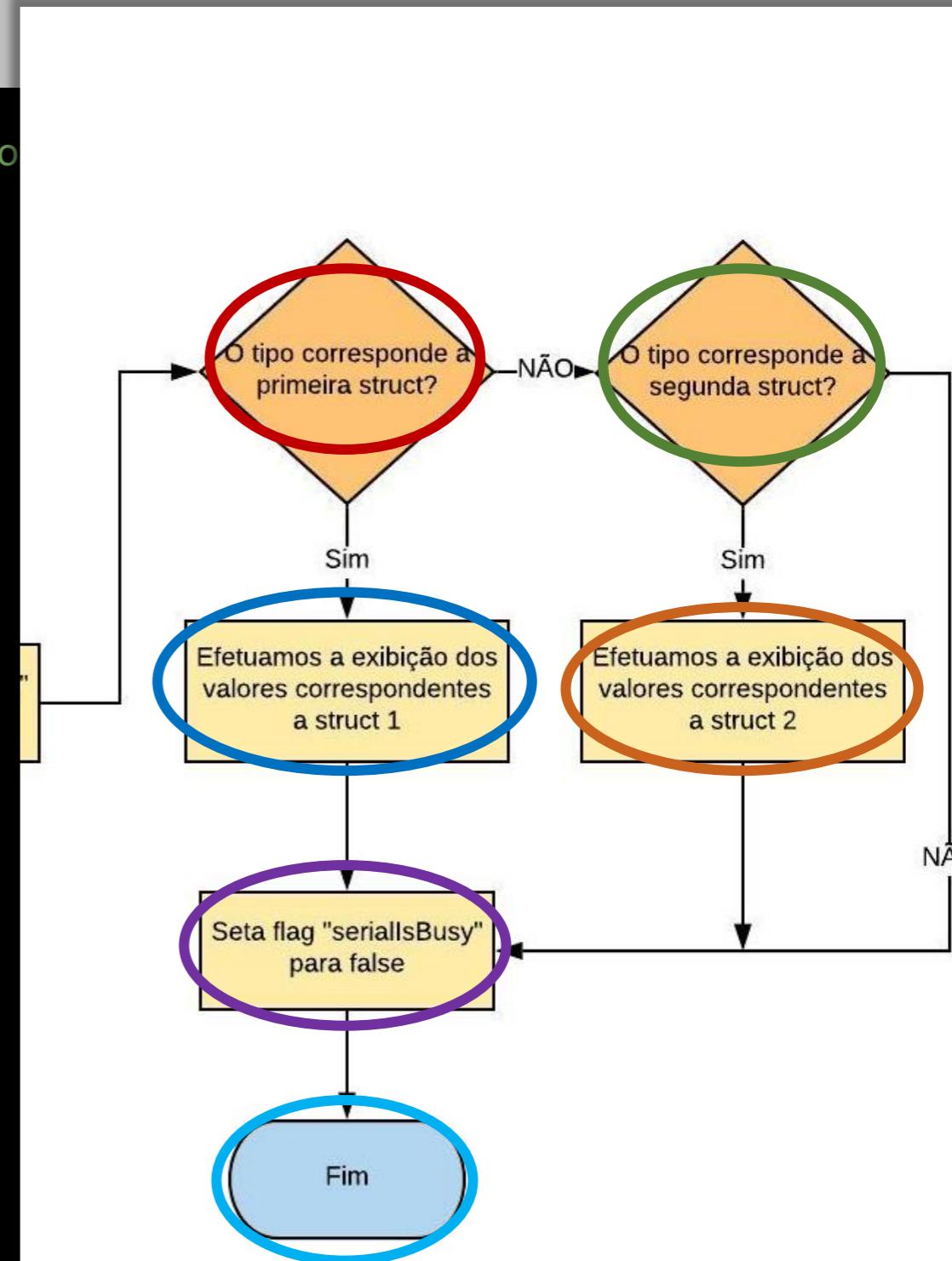
    // Flag que evita imprimir os dados embaralhados (por mais de uma task)
    → while(serialIsBusy)
        delay(1);

    // Setamos a flag para true
    → serialIsBusy = true;
    // Exibimos na serial
    Serial.println("\nTipo:" + String(type));
```



Código ESP taskPrintValues

```
// Para cada tipo, chamamos a função printStatePins enviando  
uma determinada struct  
switch(type)  
{  
    case 1:  
        printStatePins(DPins1);  
        break;  
    case 2:  
        printStatePins(Dpins2_ADPins1);  
        break;  
    default:  
        // Se o tipo não for 1 ou 2, exibimos erro  
        Serial.println("taskPrintValues: Tipo invalido");  
        break;  
  
    // Setamos a flag para false  
    serialIsBusy = false;  
    // Encerramos a task  
    vTaskDelete(NULL);  
}
```

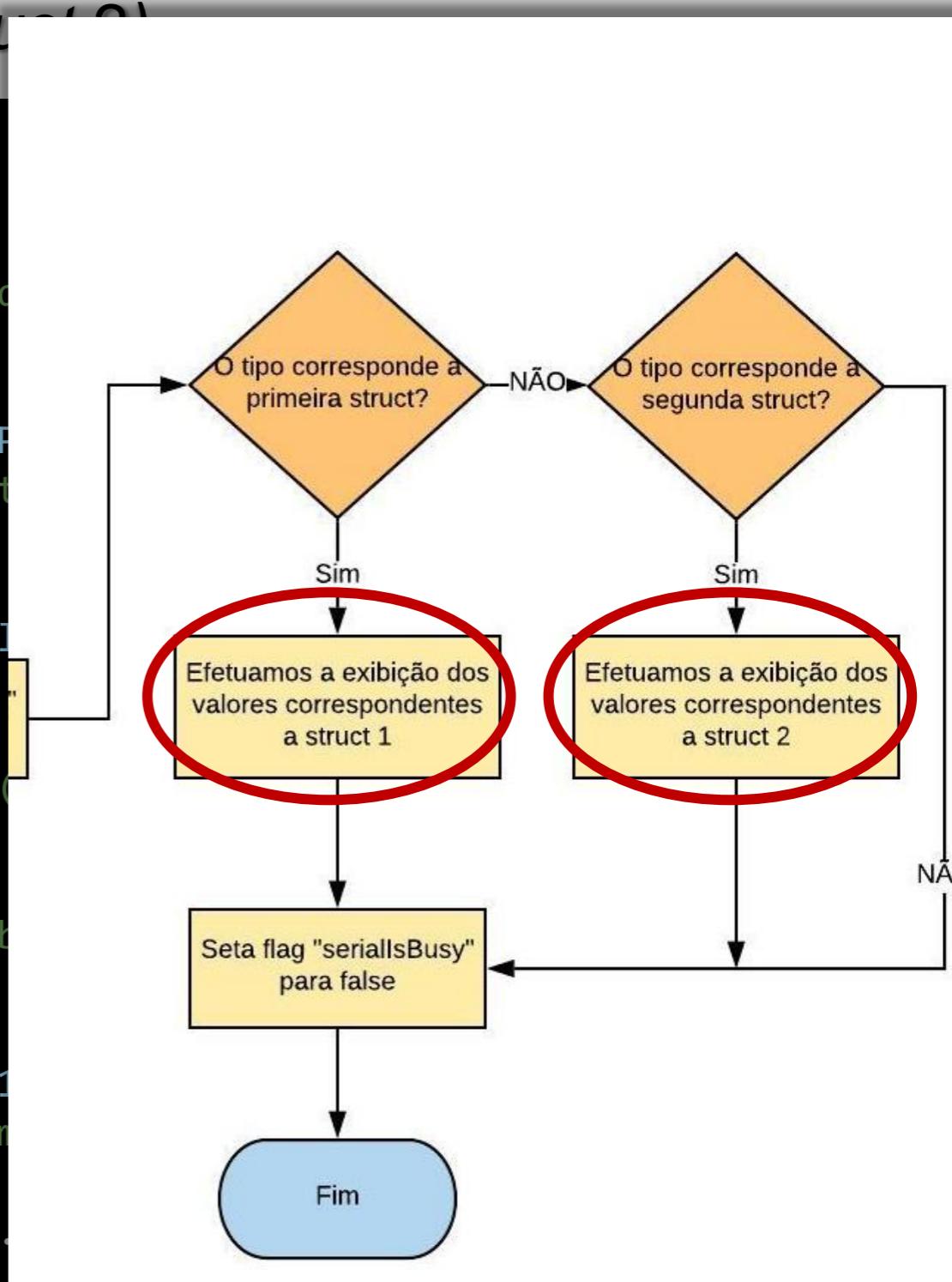


Código ESP *printStatePins (Struct 1 e Struct 2)*

```
// Função que exibe na serial os valores da primeira estrutura (D2 até D33)
void printStatePins(struct digitalPinsUntil33 DPins1)
{
    // Percorremos o vetor de pinos digitais até o GPIO12 exibindo na serial
    // Exemplo de exibição: "D3 = 0"
    for(int i=0; i<11; i++)
        Serial.println("D"+String(i+2)+" = "+String(DPins1.digitalPins[i]));
    // Sabemos que o GPIO13 não faz parte dos pinos válidos, portanto na exibição pulamos do D12 para
    o D14
    for(int i=11; i<31; i++)
        Serial.println("D"+String(i+3)+" = "+String(DPins1.digitalPins[i]));
}
// Sobrecarga da função acima
// Função que exibe na serial os valores da segunda estrutura (D34 até D49 e A0 até A6)
void printStatePins(struct digitalAndADPins Dpins2_AD Pins1)
{
    // Percorremos o vetor com o restante dos pinos digitais exibindo na serial do D34 ao D49
    // Exemplo de exibição: "D34 = 0"
    for(int i=0; i<15; i++)
        Serial.println("D"+String(i+34)+" = "+String(Dpins2_AD Pins1.digitalPins[i]));
    // Percorremos o vetor com os valores AD do A0 ao A6 e exibimos na serial
    for(int i=0; i<7; i++)
        Serial.println("AD"+String(i)+" = "+String(Dpins2_AD Pins1.AD Pins[i]));
}
```

Código ESP printStatePins (Struct 1 e Struct 2)

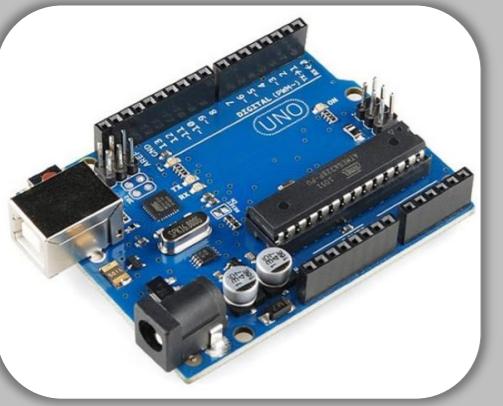
```
// Função que exibe na serial os valores da primeira estrutura
void printStatePins(struct digitalPinsUntil33 DPins1)
{
    // Percorremos o vetor de pinos digitais até o GPIO12 exibindo
    // Exemplo de exibição: "D3 = 0"
    for(int i=0; i<11; i++)
        Serial.println("D"+String(i+2)+" = "+String(DPins1.digitalPins[i]));
    // Sabemos que o GPIO13 não faz parte dos pinos válidos, portanto
    // o D14
    for(int i=11; i<31; i++)
        Serial.println("D"+String(i+3)+" = "+String(DPins1.digitalPins[i]));
}
// Sobrecarga da função acima
// Função que exibe na serial os valores da segunda estrutura
void printStatePins(struct digitalAndADPins Dpins2_AD Pins1)
{
    // Percorremos o vetor com o restante dos pinos digitais exibindo
    // Exemplo de exibição: "D34 = 0"
    for(int i=0; i<15; i++)
        Serial.println("D"+String(i+34)+" = "+String(Dpins2_AD.Pins1[i]));
    // Percorremos o vetor com os valores AD do A0 ao A6 e exibindo
    for(int i=0; i<7; i++)
        Serial.println("AD"+String(i)+" = "+String(Dpins2_AD.Pins1[i]));
}
```



Código ESP *callback_after_slave_tx_finish*

```
// Função callback que é chamada logo após uma transação SPI, não utilizamos neste exemplo
int callback_after_slave_tx_finish()
{
    //Serial.println("Size:"+String(slave.getInputStream()->length()));

    //Serial.println("[slave_tx_finish] slave transmission has been finished!");
    //Serial.println(slave[0]);
}
```



Código

Código UNO Declarações e Variáveis

```
#include <SPI.h>

// Pinos SPI:
// SCK: GPIO13
// MISO: GPIO12
// MOSI: GPIO11
// SS: GPIO10

// Variáveis que correspondem ao código do Arduino Mega (explicação no código do Arduino Mega)
struct digitalPinsUntil33
{
    uint8_t type;
    uint8_t digitalPins[31];
};

struct digitalAndADPins
{
    uint8_t type;
    uint8_t digitalPins[15];
    uint16_t ADPins[7];
};
```

Código UNO *Declarações e Variáveis*

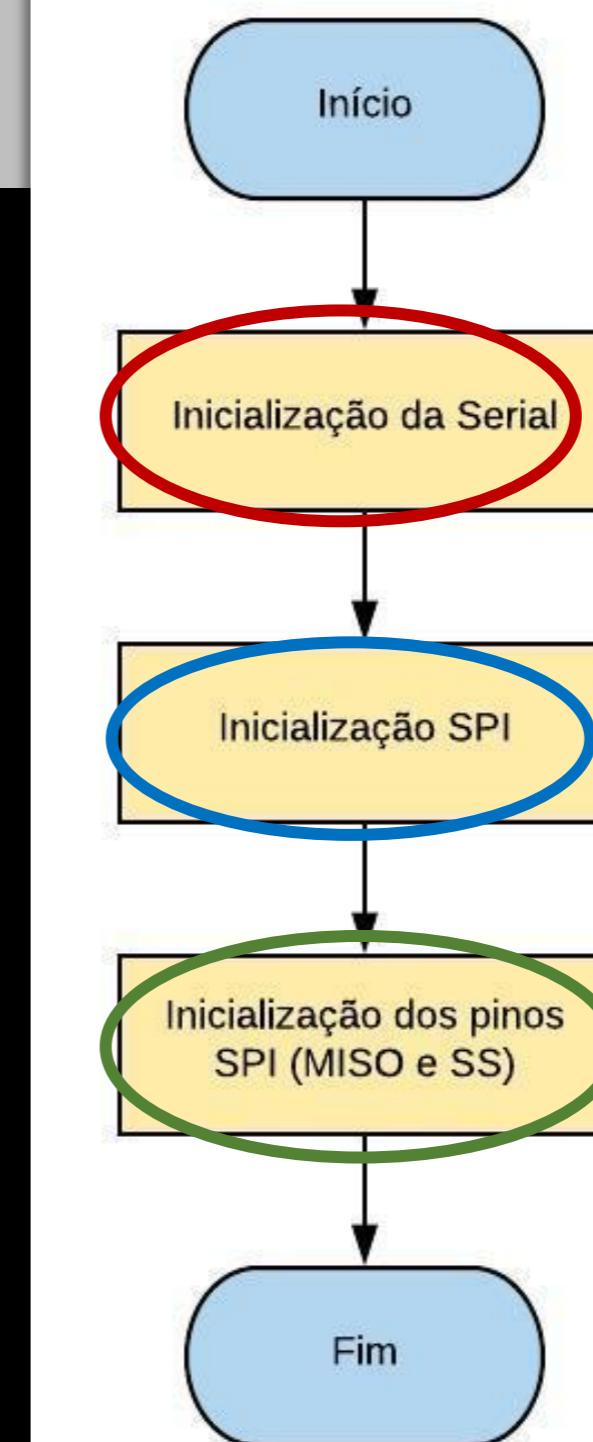
```
struct digitalPinsUntil33 DPins1;
struct digitalAndADPins Dpins2_ADPins1;

// Variável que indica se a primeira struct já foi recebida
bool struct1Received = false;
// Buffer que recebe os dados do Arduino Mega
uint8_t buf[32];
// Variável referente ao tamanho do buffer
int sizeBuff = 0;
```

Código UNO Setup

```
void setup (void)
{
    // Iniciamos a serial (debug)
    → Serial.begin (115200);
    // Setamos o SPI para o modo Slave
    → SPCR |= bit (SPE);
    // Setamos o pino MISO como saída
    → pinMode(MISO, OUTPUT);
    // Setamos o pino SS como entrada
    → pinMode(SS, INPUT);

    // Ativamos a função de interrupção (ISR)
    SPI.attachInterrupt();
    Serial.println("Slave ok");
}
```

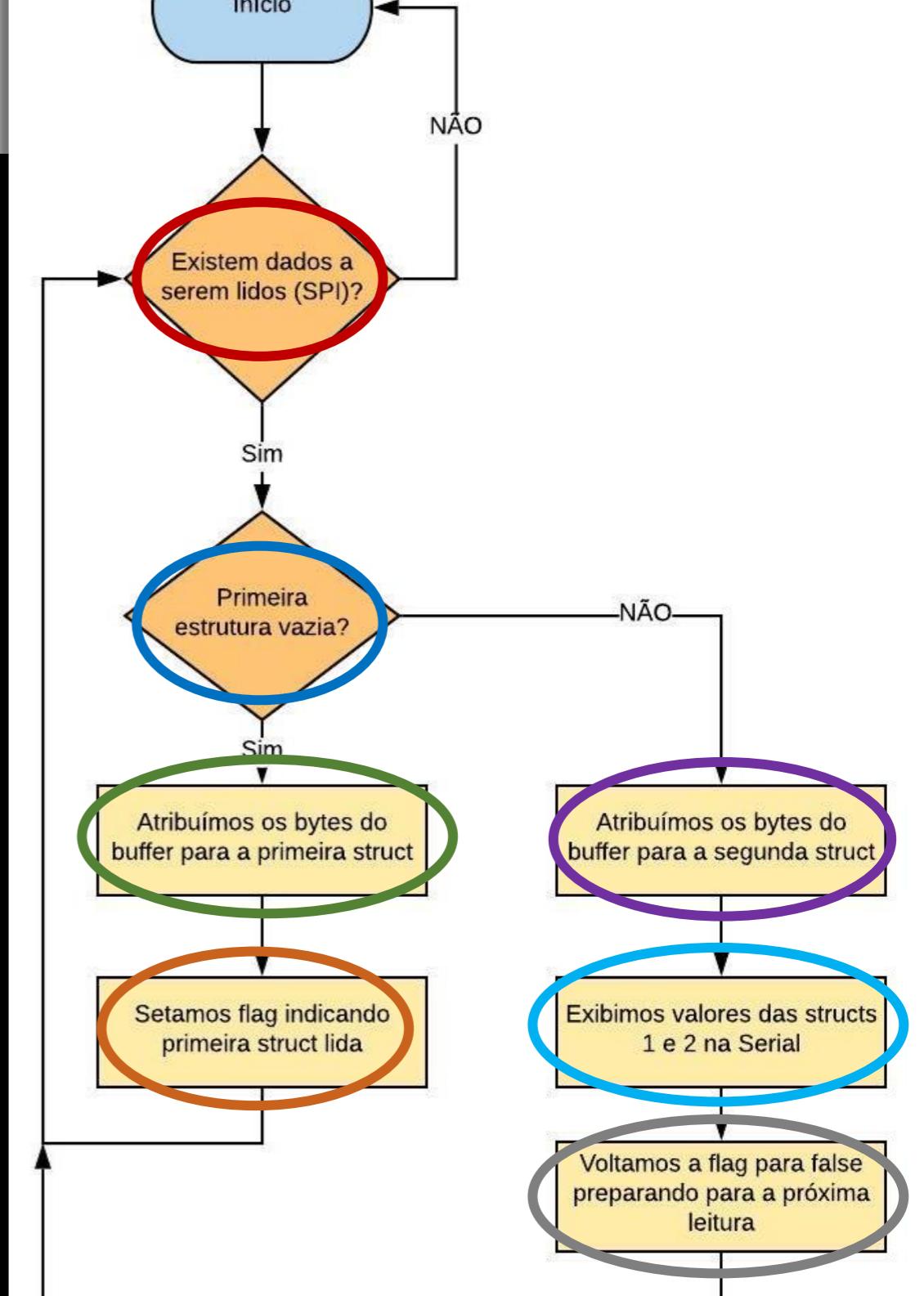


Código UNO Loop

```
void loop (void)
{
    → if (sizeBuff>0 && digitalRead(SS) == HIGH)
    {
        delay(50);
        //Serial.println("Size: "+String(sizeBuff));

    ← if(!struct1Received)
        {
            → memcpy(&DPins1, buf, sizeof(DPins1));
            → struct1Received = true;
        }
        else
        {
            → memcpy(&Dpins2_AD Pins1, buf, sizeof(Dpins2_AD Pins1));
            ← process();
            → struct1Received = false;
        }

        sizeBuff = 0;
    }
}
```



Código UNO ISR

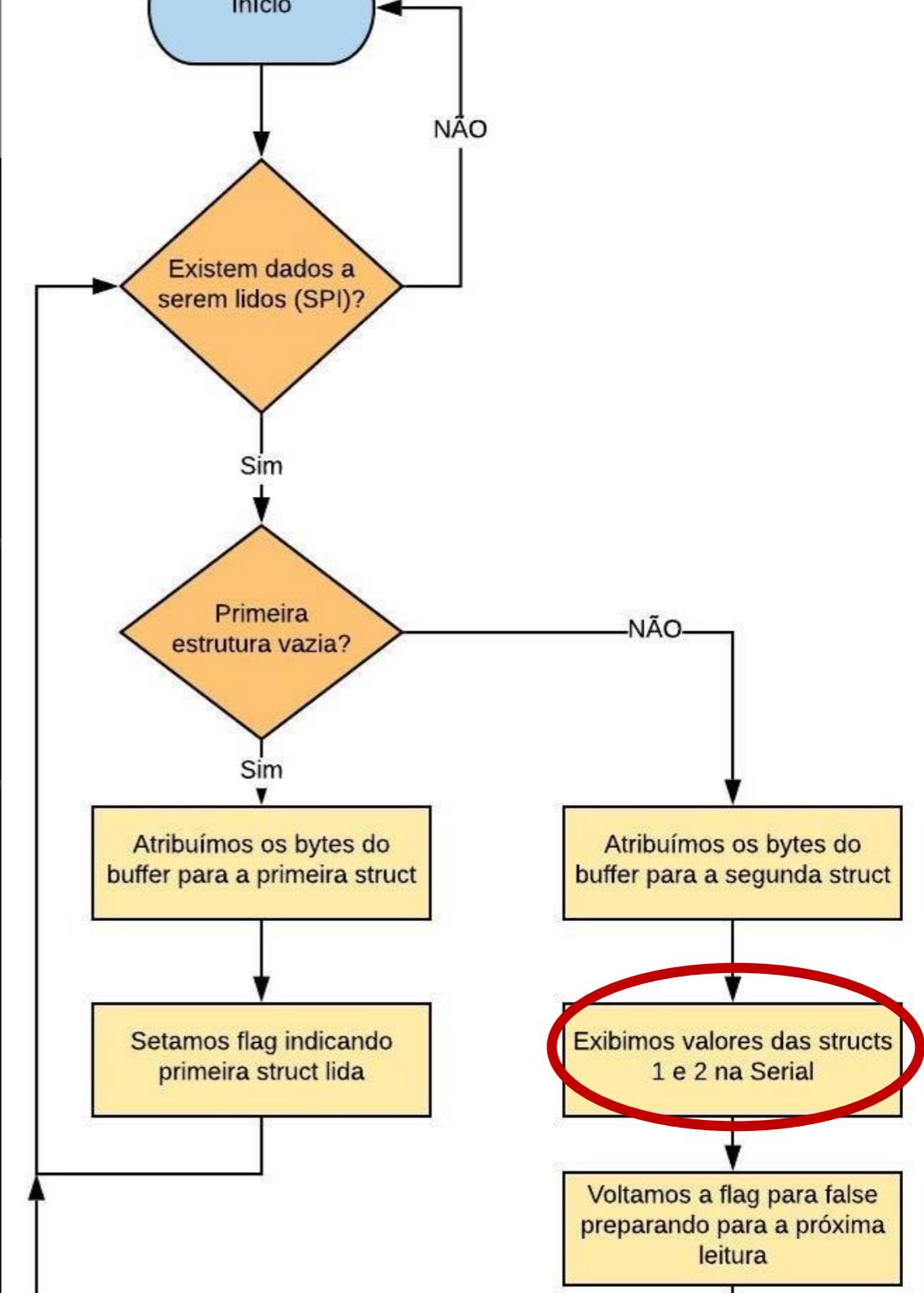
```
// Função de interrupção SPI  
ISR(SPI_STC_vect)  
{  
    // Atribuimos para o buffer o valor de  
    // SPDR (Recebido via SPI)  
    buf[sizeBuff++] = SPDR;  
}
```



Código UNO process

```
// Função que exibe na Serial os valores das structs 1 e 2
// De acordo com o tipo (type)
void process()
{
    if(DPins1.type == 1)
        printStatePins(DPins1);
    else
        Serial.println("Erro ao obter primeiro byte (struct 1)"

    if(Dpins2_AD Pins1.type == 2)
        printStatePins(Dpins2_AD Pins1);
    else
        Serial.println("Erro ao obter primeiro byte (struct 2")
}
```



Código UNO printStatePins (Struct 1 e Struct 2)

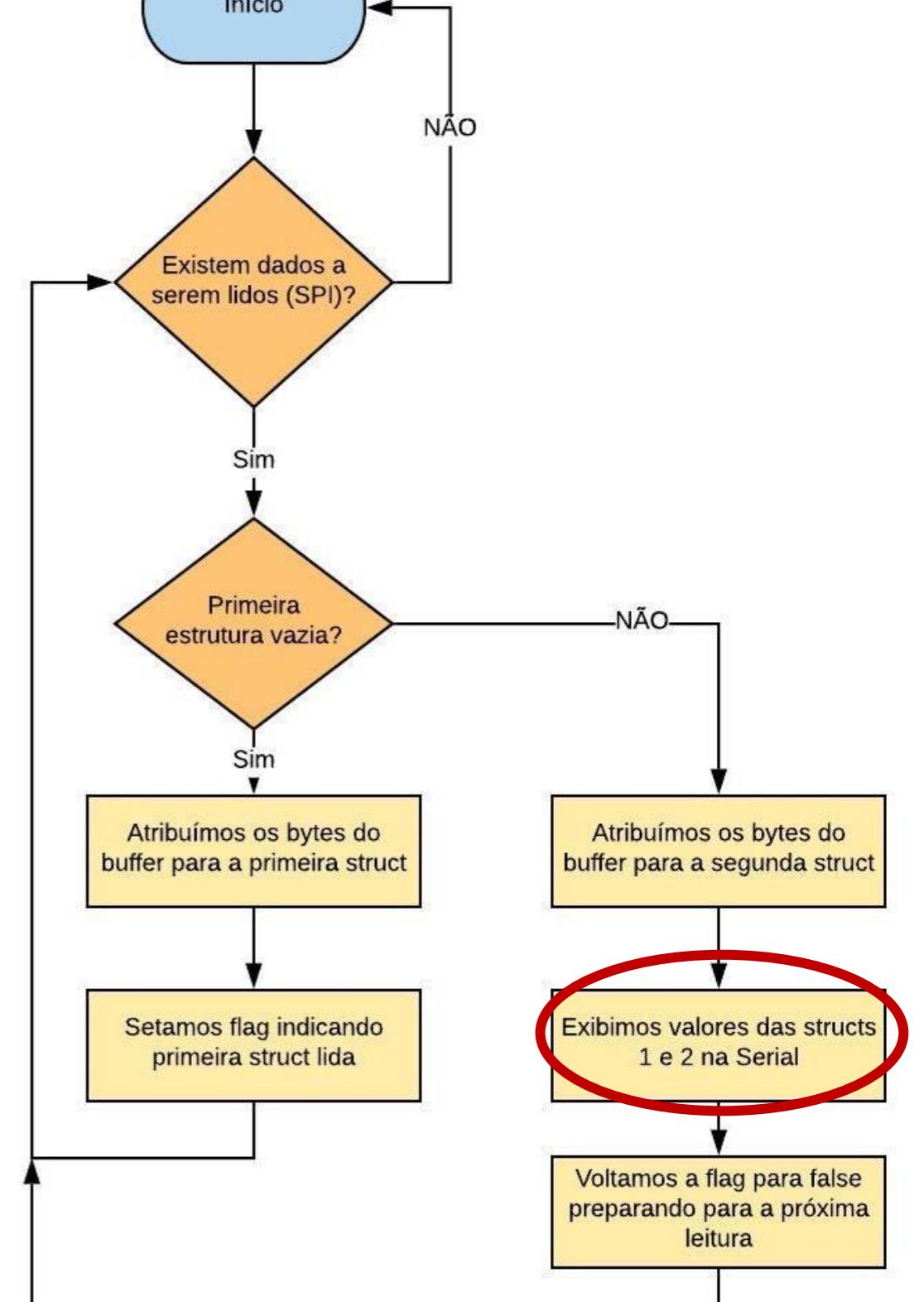
```
// Função que exibe na serial os valores da primeira estrutura (D2 até D33)
void printStatePins(struct digitalPinsUntil33 DPins1)
{
    // Percorremos o vetor de pinos digitais até o GPIO12 exibindo na serial
    // Exemplo de exibição: "D3 = 0"
    for(int i=0; i<11; i++)
        Serial.println("D"+String(i+2)+" = "+String(DPins1.digitalPins[i]));
    // Sabemos que o GPIO13 não faz parte dos pinos válidos, portanto na exibição pulamos do D12 para o D14
    for(int i=11; i<31; i++)
        Serial.println("D"+String(i+3)+" = "+String(DPins1.digitalPins[i]));
}

// Sobrecarga da função acima
// Função que exibe na serial os valores da segunda estrutura (D34 até D49 e A0 até A6)
void printStatePins(struct digitalAndADPins Dpins2_ADPins1)
{
    // Percorremos o vetor com o restante dos pinos digitais exibindo na serial do D34 ao D49
    // Exemplo de exibição: "D34 = 0"
    for(int i=0; i<15; i++)
        Serial.println("D"+String(i+34)+" = "+String(Dpins2_ADPins1.digitalPins[i]));
    // Percorremos o vetor com os valores AD do A0 ao A6 e exibimos na serial
    for(int i=0; i<7; i++)
        Serial.println("AD"+String(i)+" = "+String(Dpins2_ADPins1.ADPins[i]));
}
```

Código UNO printStatePins (Struct 1 e 2)

```
// Função que exibe na serial os valores da primeira estrutura
void printStatePins(struct digitalPinsUntil33 DPins1)
{
    // Percorremos o vetor de pinos digitais até o GPIO12 exibindo
    // Exemplo de exibição: "D3 = 0"
    for(int i=0; i<11; i++)
        Serial.println("D"+String(i+2)+" = "+String(DPins1.digital[i]));
    // Sabemos que o GPIO13 não faz parte dos pinos válidos, portanto
    for(int i=11; i<31; i++)
        Serial.println("D"+String(i+3)+" = "+String(DPins1.digital[i]));
}

// Sobrecarga da função acima
// Função que exibe na serial os valores da segunda estrutura
void printStatePins(struct digitalAndADPins Dpins2_AD Pins1)
{
    // Percorremos o vetor com o restante dos pinos digitais exibindo
    // Exemplo de exibição: "D34 = 0"
    for(int i=0; i<15; i++)
        Serial.println("D"+String(i+34)+" = "+String(Dpins2_AD.Pins1[i]));
    // Percorremos o vetor com os valores AD do A0 ao A6 e exibindo
    for(int i=0; i<7; i++)
        Serial.println("AD"+String(i)+" = "+String(Dpins2_AD.Pins1[i]));
}
```



Em www.fernandok.com

Download arquivos PDF e INO do código fonte

