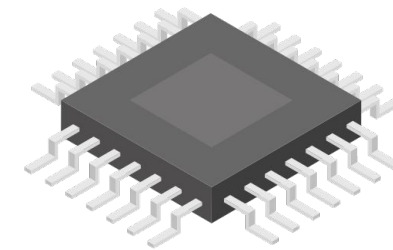




PADO
Labs



Microcontroladores



Prof.º: Pablo Jean Rozário



pablo.jean@padotec.com.br



/in/pablojeanrozario



<https://github.com/Pablo-Jean>

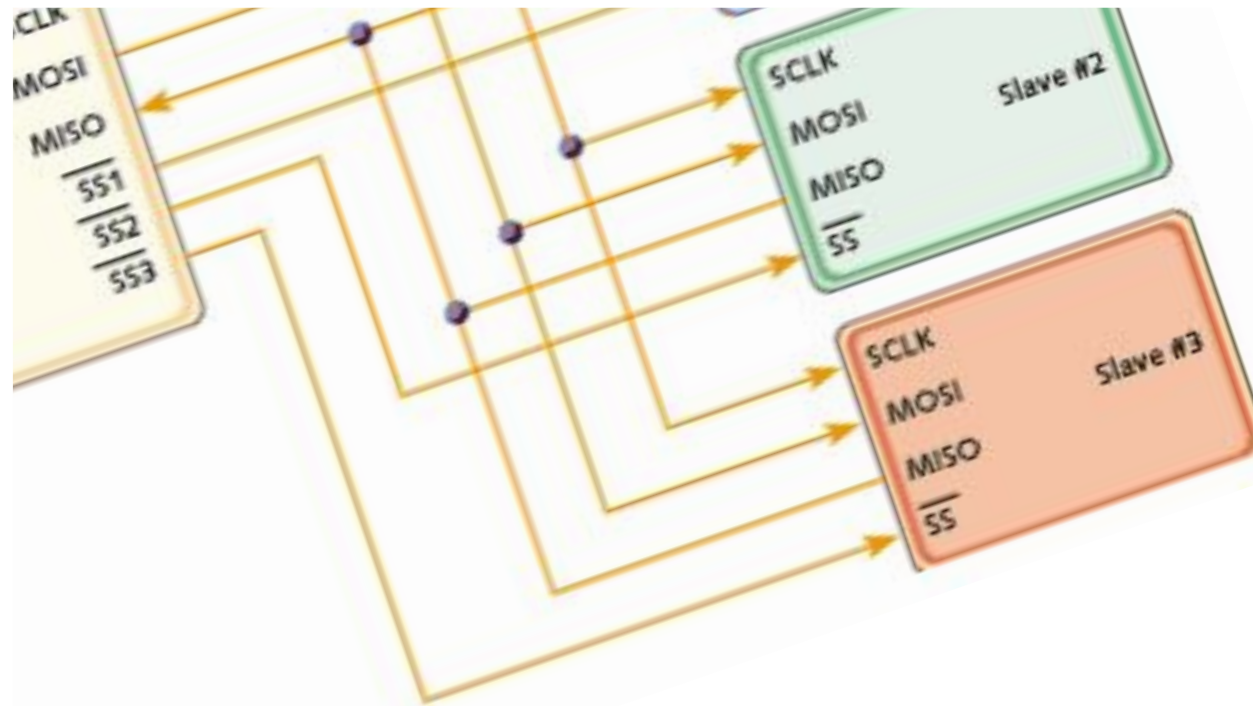
Comunicação SPI

Índice da Aula #5



- Introdução
 - Camada Física da SPI
 - Shift Register
 - Configuração do Clock
 - Modo 3-wire e Daisy Chain
 - Vantagens e Desvantagens
 - Exemplo de sinal
 - SPI no STM32G0
 - Localização dos terminais
 - Funções Utilizadas
- Experimento
 - Interrupções e DMA
 - Lista de Exercícios #6

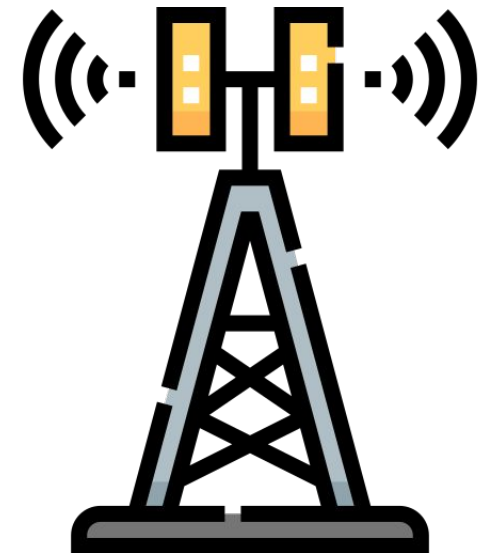
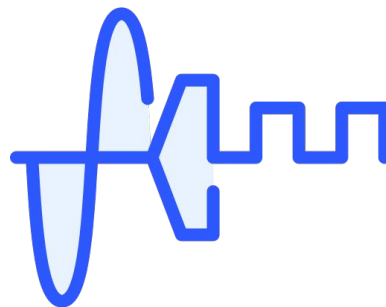
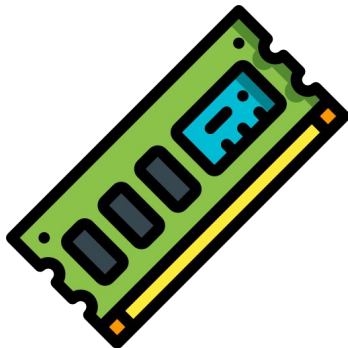
Serial Peripheral Interface



Introdução

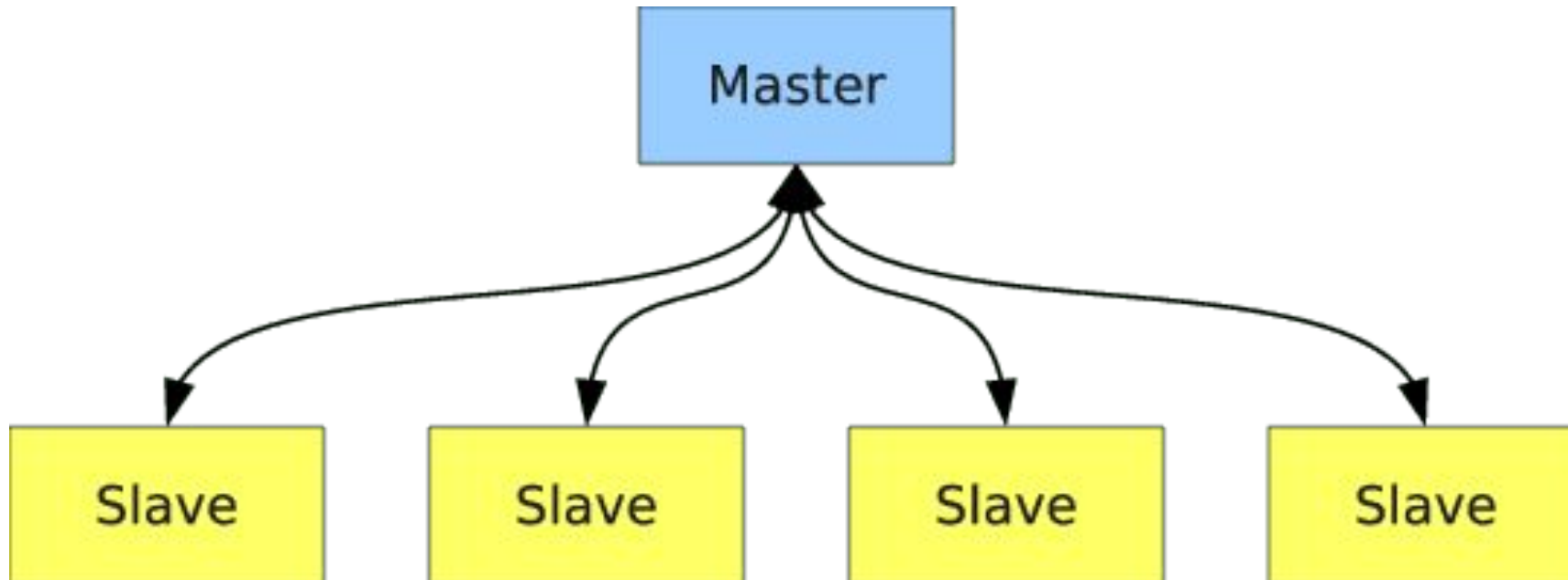
Ao contrário da UART, temos agora a **SPI** (acrônimo para *Serial Peripheral Interface*), uma interface de comunicação muito utilizada para comunicar com periféricos, que comunica *Full-Duplex*.

Alguns periféricos podem ser ADCs, DACs, Memórias, rádios, etc.

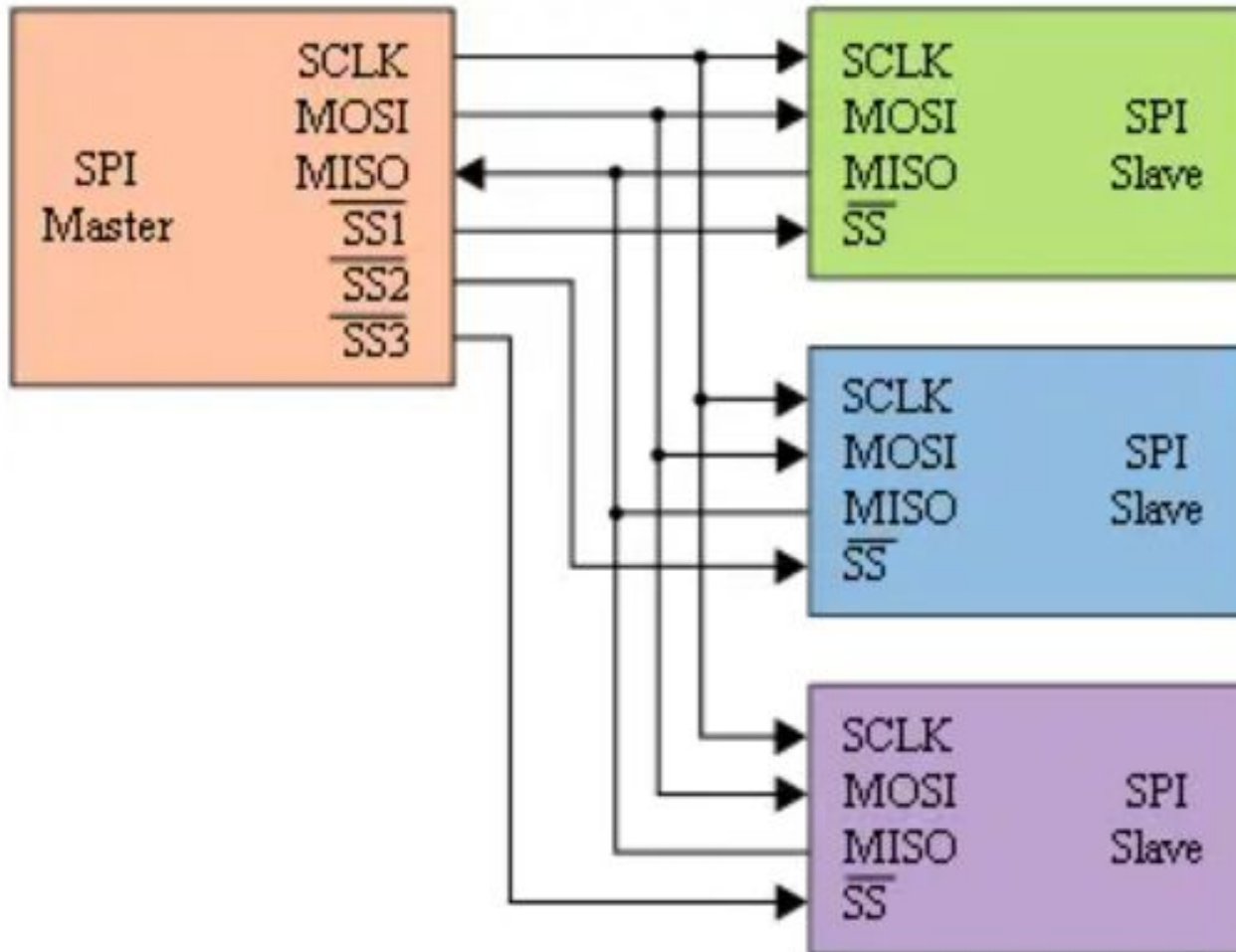


Camada Física - Estrutura

A SPI tem por característica a presença de um *Master*, que tem controle sobre o *barramento*, e os *Slaves* que estão conectados ao mestre, com a topologia:



Camada Física - Estrutura



Com mais detalhes,
temos a estrutura
com os terminais

Camada Física - Terminais

SCLK: Terminal controlado pelo *Master*, que tem por finalidade sincronizar a troca de informações.

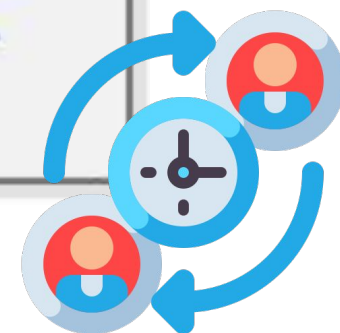
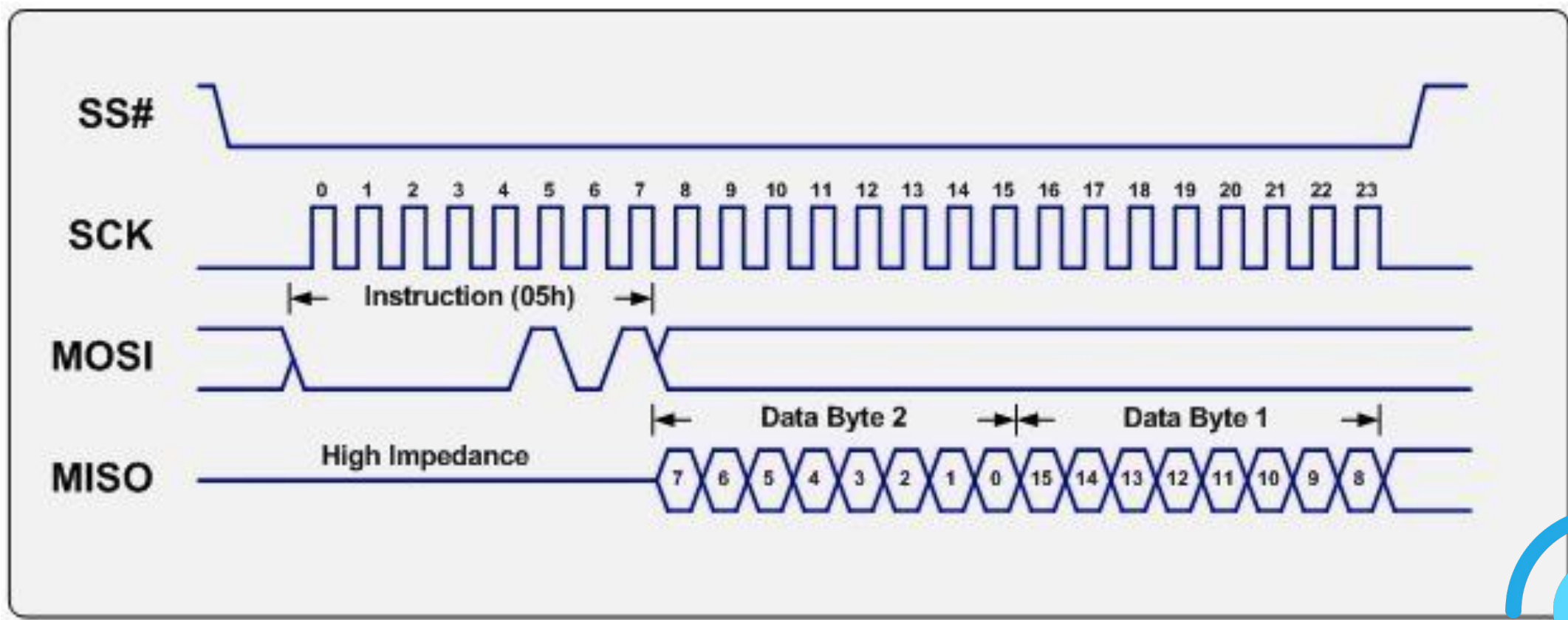
MISO: (Master-Input Slave-Output) entrada de dados do *Master* e saída do *Slave*.

MOSI: (Master-Output Slave-Input) saída de dados do *Master* e entrada do *Slave*.

SS/CS: (Slave Select/Chip Select) utilizado pelo *Master* para selecionar qual *Slave* irá comunicar. (geralmente ativo em nível 0)

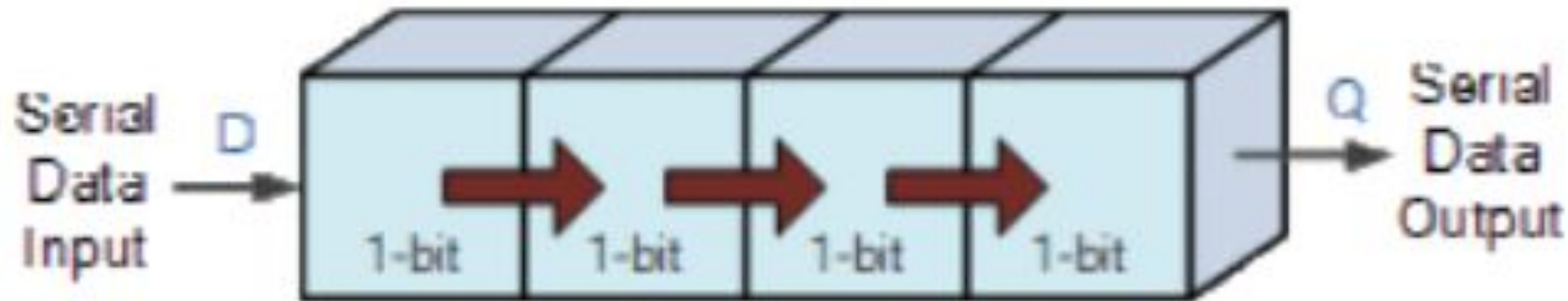


SPI - *Framing*



SPI - *Shift Register*

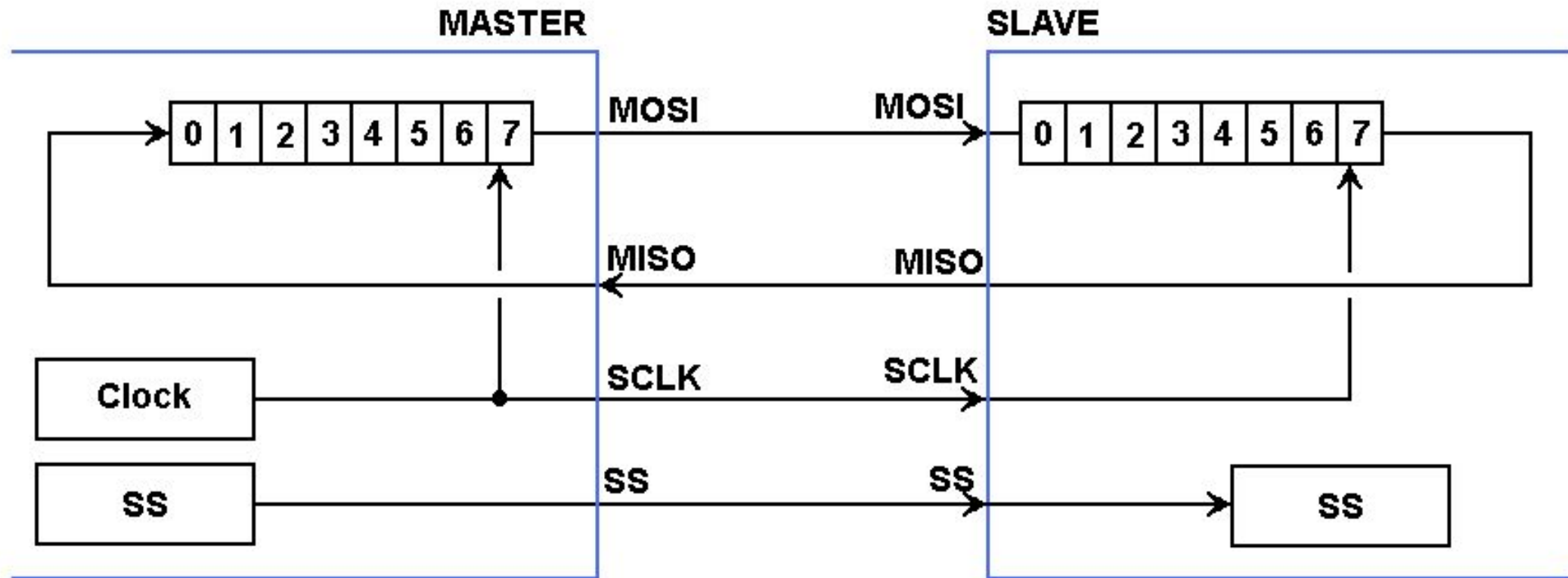
O princípio básico de operação da SPI é o *Shift Register*, que nada mais é do que um dispositivo que irá converter uma informação **paralela** em **serial**.



O dispositivo basicamente “**empurra**” os bits em uma direção e recebe informações do outro.

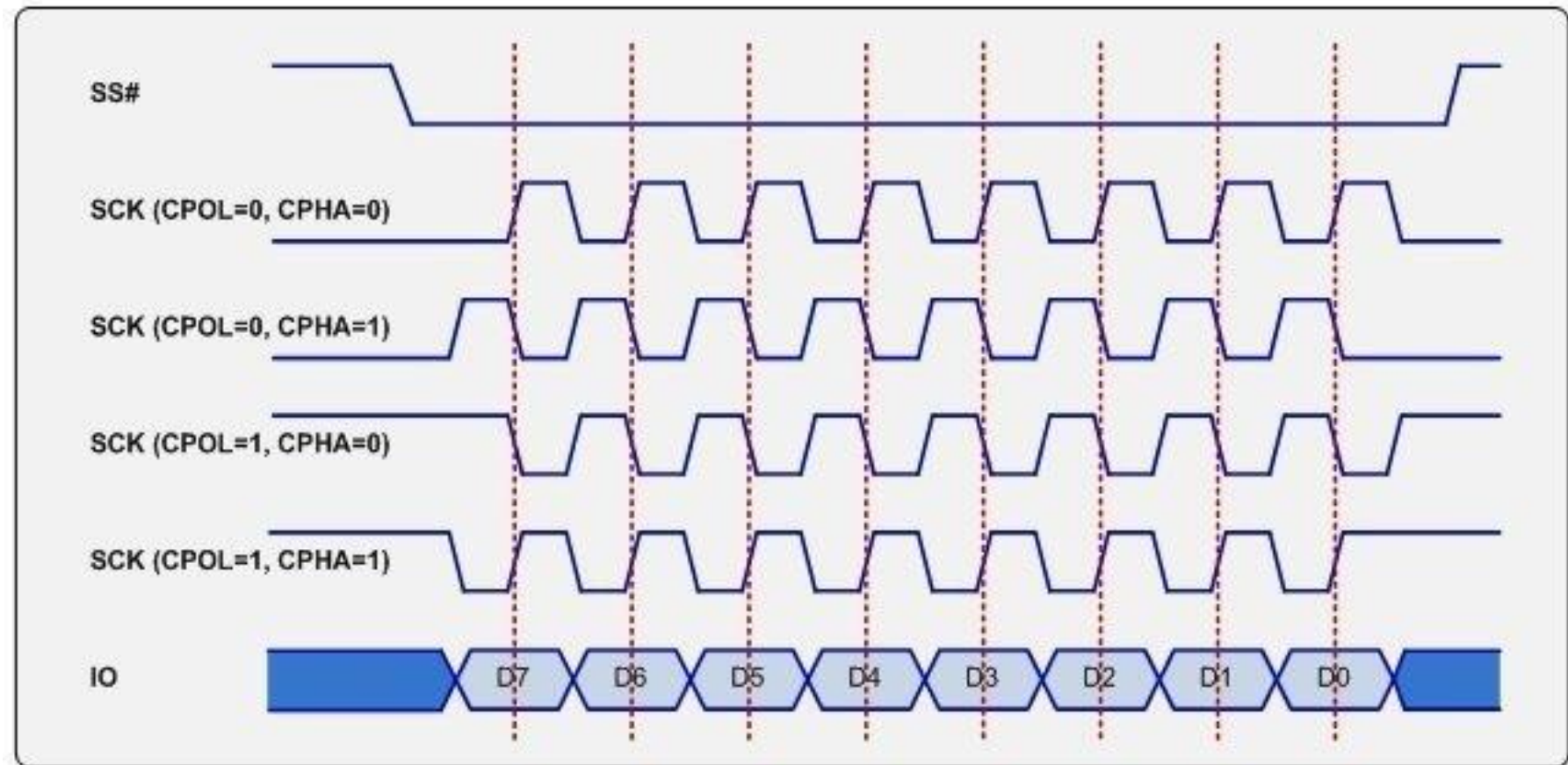
SPI - *Shift Register*

No periférico, a implementação básica segue o diagrama abaixo



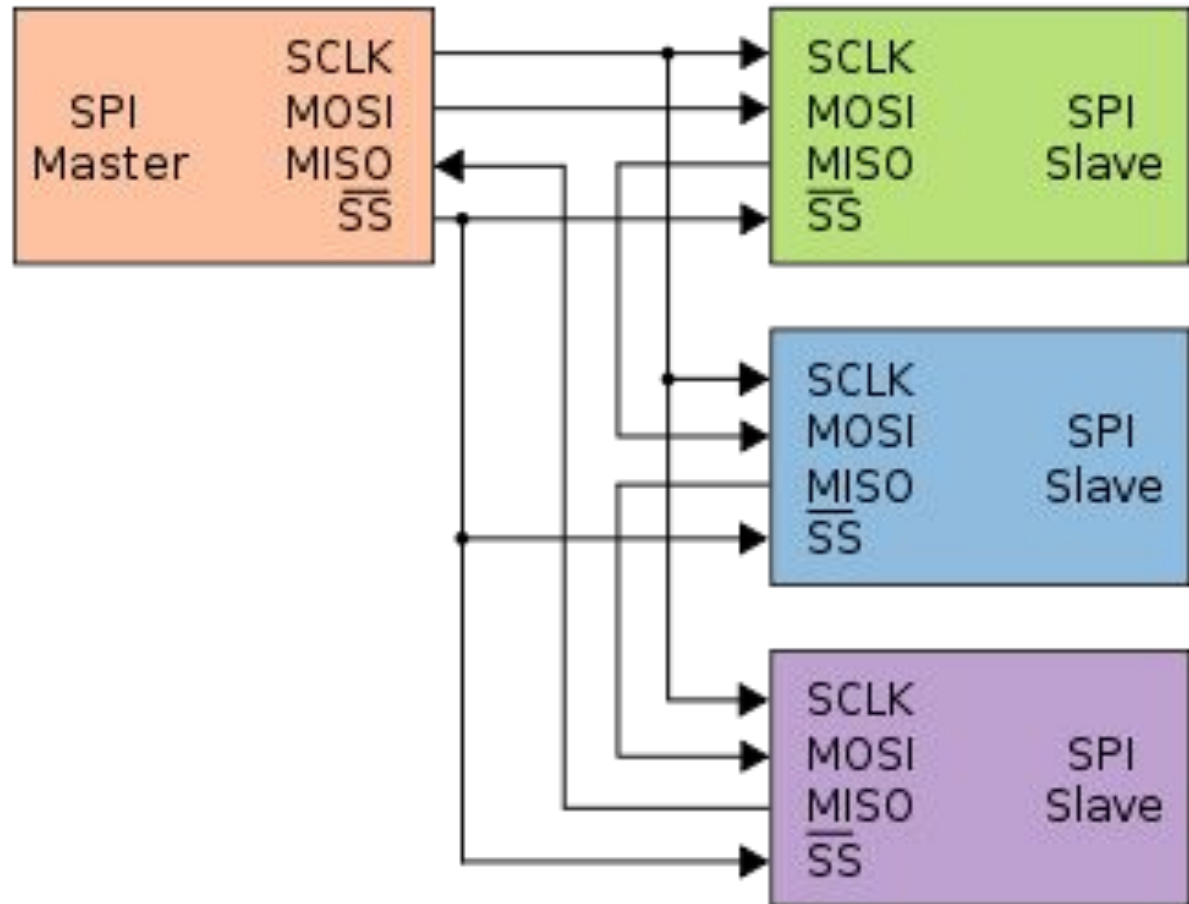
SPI - Configuração do Clock

Há dois parâmetros que é utilizado para configurar o *clock*, sendo o **CPOL** (Polaridade) e **CPHA** (Fase). Com a seguinte regra.



SPI - Daisy Chain

Podemos conectar os *Slaves* em modo **Daisy-Chain** para economizar terminais do *Master*.



SPI - Vantagens

Algumas **vantagens** da interface **SPI** são:

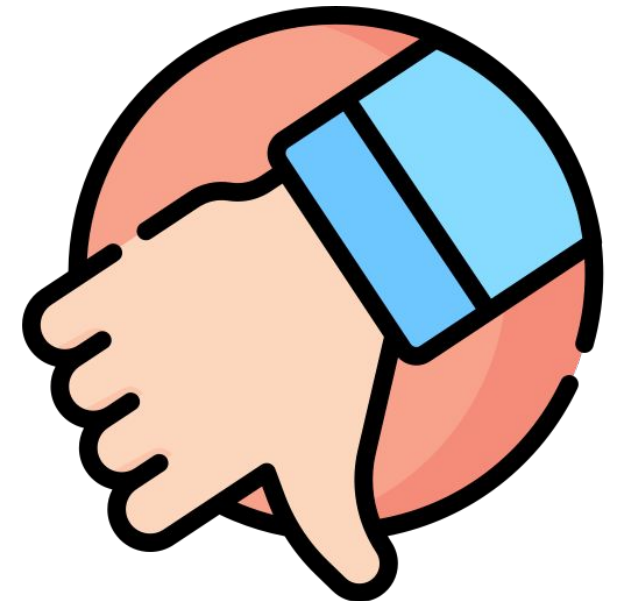
- Comunicação *Full-Duplex*
- Drivers *Push-Pull* que provém integridade de sinal e alta velocidade
- Flexibilidade nos protocolos
- Interface de Hardware simples
- *Entre outras*



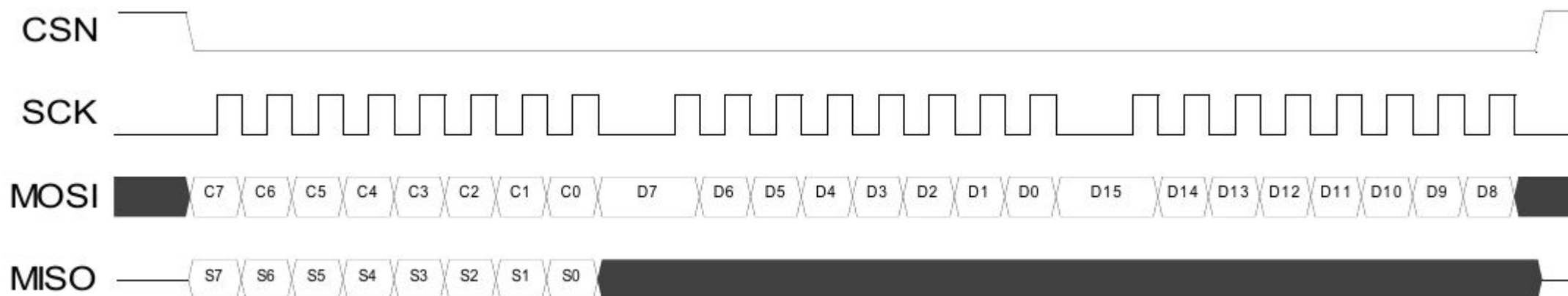
SPI - Desvantagens

E algumas **desvantagens** da interface **SPI** são:

- Requer mais terminais que o I2C
- Sem endereçamento
- Não possui *flow control*
- Sem sinal de *acknowledge* (permitindo que o Master envie nada a lugar nenhum)
- Suporta apenas um *Master*, em geral
- Suporta distâncias muito pequenas

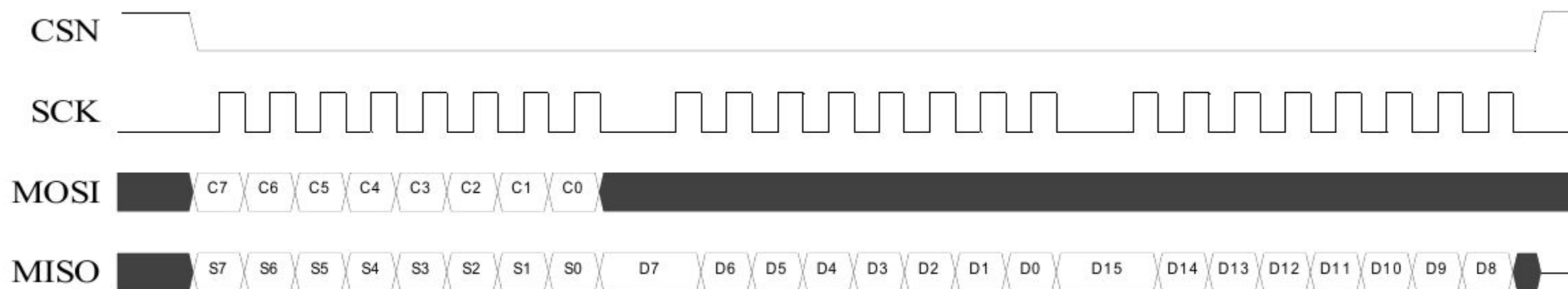


SPI - Exemplo de Sinal - nRF24



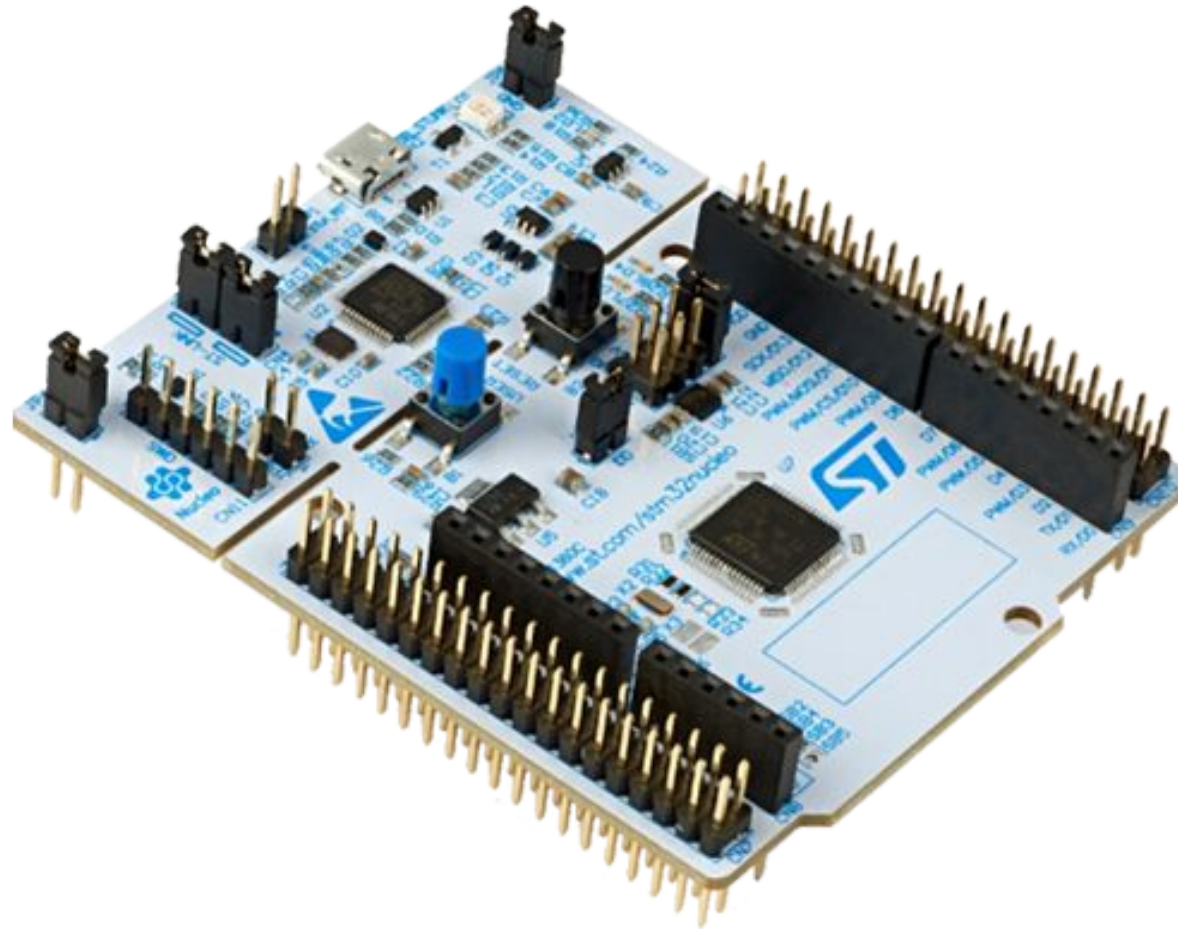
Operação de Escrita no nRF24L01.

SPI - Exemplo de Sinal - nRF24



Operação de Leitura no nRF24L01.

SPI no STM32G0

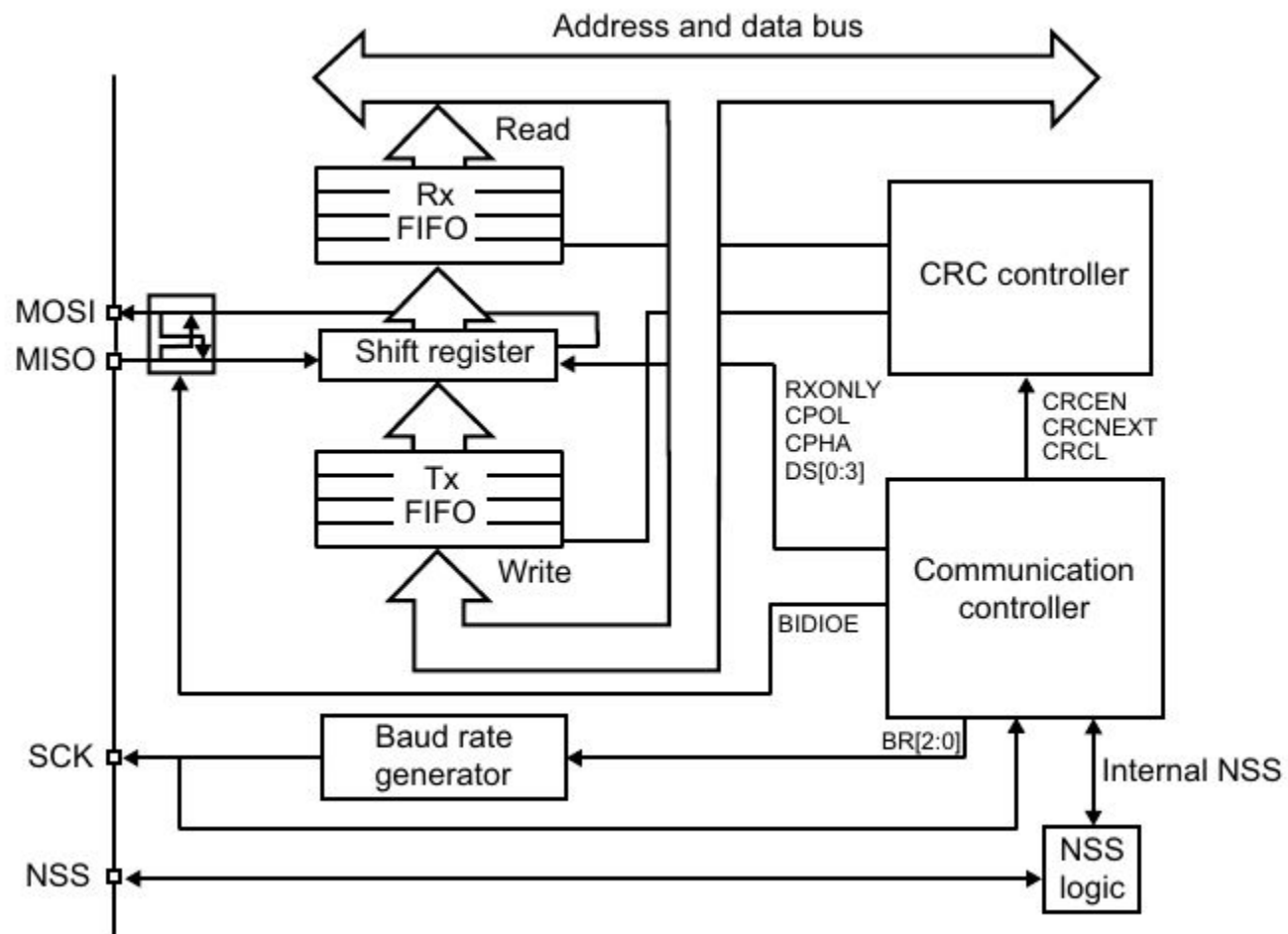


STM32 - Características da SPI



- Operação como *Master* ou *Slave*
- Modo *Full* ou *Half Duplex*
- Entre 4 à 16 bits de dados
- Capaz de operar em modo *Multimaster*
- SS operado por *hardware*
- CPOL e CPHA configuráveis
- suporta checagem por CRC
- *Errors* com interrupção
- Para mais: consulte **RM0444**

STM32G0 - Diagrama do SPI



STM32G0 - Localização

Utilize o documento **STM32G0B1xB/xC/xE**, e consulte a **tabela 13 à 20**, na **página 56 à 63**. Onde é possível observar na coluna de *Alternate Functions* onde a SPI está localizada.

Table 16. Port B alternate function mapping (AF)

Port	AF8	AF9	AF10	AF11	AF12
PB0	USART5_TX	-	LPUART2_CTS	-	-
PB1	USART5_RX	-	LPUART2_RTS_DE	-	-
PB2	-	-	-	-	-
PB3	I2C2_SCL	<u>SPI3_SCK</u>	-	-	-
PB4	I2C2_SDA	<u>SPI3_MISO</u>	-	-	-
PB5	USART5_RTS_DE_CK	<u>SPI3_MOSI</u>	-	-	-
PB6	USART5_CTS	TIM4_CH1	LPUART2_TX	-	-
PB7	-	TIM4_CH2	LPUART2_RX	-	-
PB8	USART6_TX	TIM4_CH3	-	-	-
PB9	USART6_RX	TIM4_CH4	-	-	-
PB10	-	-	-	-	-

STM32G0B - Funções para SPI



Para **TRANSMITIR** um vetor, utilizamos:

```
1 // Envia um array contendo os dados a serem enviados, onde hspi refere
2 // ao handle da interface SPI, gerado pelo proprio CubeMX,
3 // pData e o ponteiro do vetor contendo os dados, Size e a
4 // quantidade de bytes de pData e Timeout e o tempo maximo para
   aguardar
5 // a conclusao da transmissao
6 HAL_SPI_Transmit(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size
, uint32_t Timeout);
```



STM32G0B - Funções para SPI



Para **RECEBER** um vetor, utilizamos:

```
1 // Le um array da SPI, onde hspi refere ao handle da interface ,  
2 // gerado pelo proprio CubeMX, pData e o ponteiro do vetor que recebera  
3 // os dados, Size e a quantidade de bytes que pretende-se receber  
4 // e Timeout e o tempo maximo para aguardar a conclusao da recepcao  
5 HAL_SPI_Receive(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size,  
uint32_t Timeout);
```

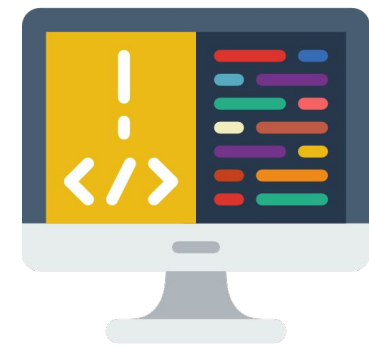


STM32G0B - Funções para SPI

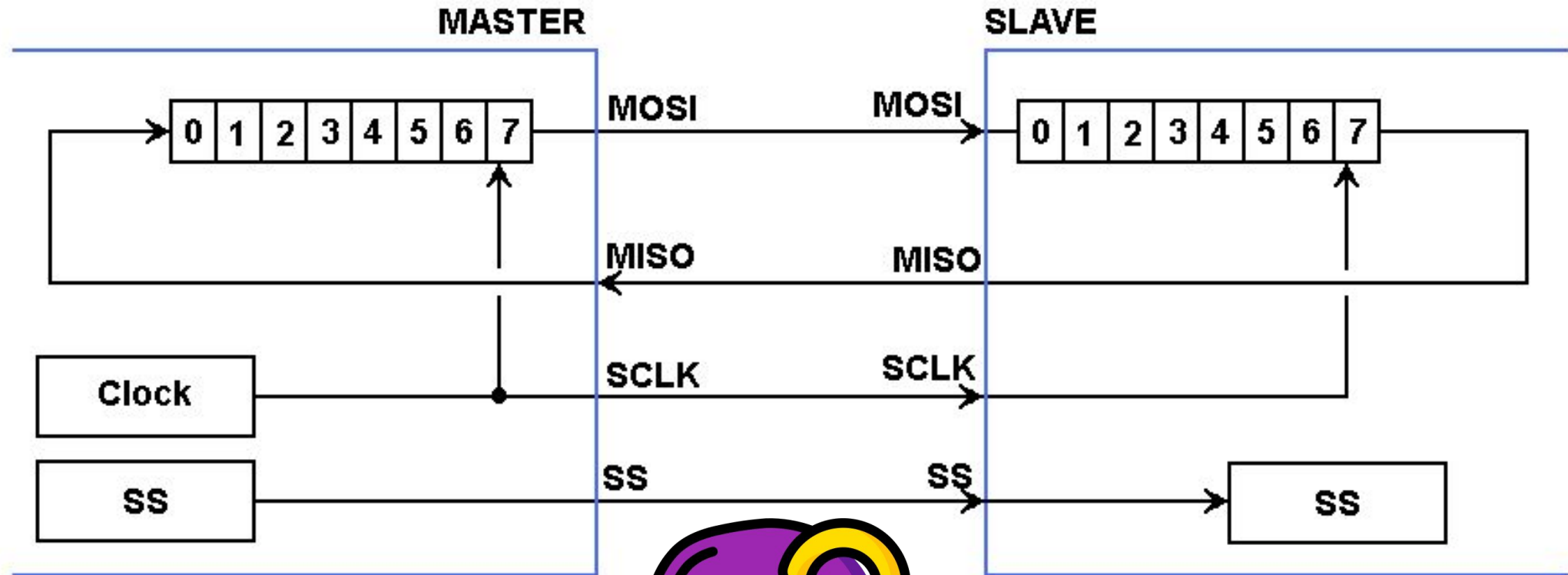


Lembram que o SPI é **Full-Duplex** por causa do *Shift-Register*?
Então:

```
1 // Escreve e le um vetor de dados pela interface SPI, os parametros
2 // sao os mesmos para as outras funcoes, com excessao da adicao do
3 // ponteiro para a transmissao (pTxData) e outro para recepcao
4 // (pRxData).
5 HAL_SPI_TransmitReceive(SPI_HandleTypeDef *hspi, uint8_t *pTxData,
uint8_t *pRxData, uint16_t Size, uint32_t Timeout)
```



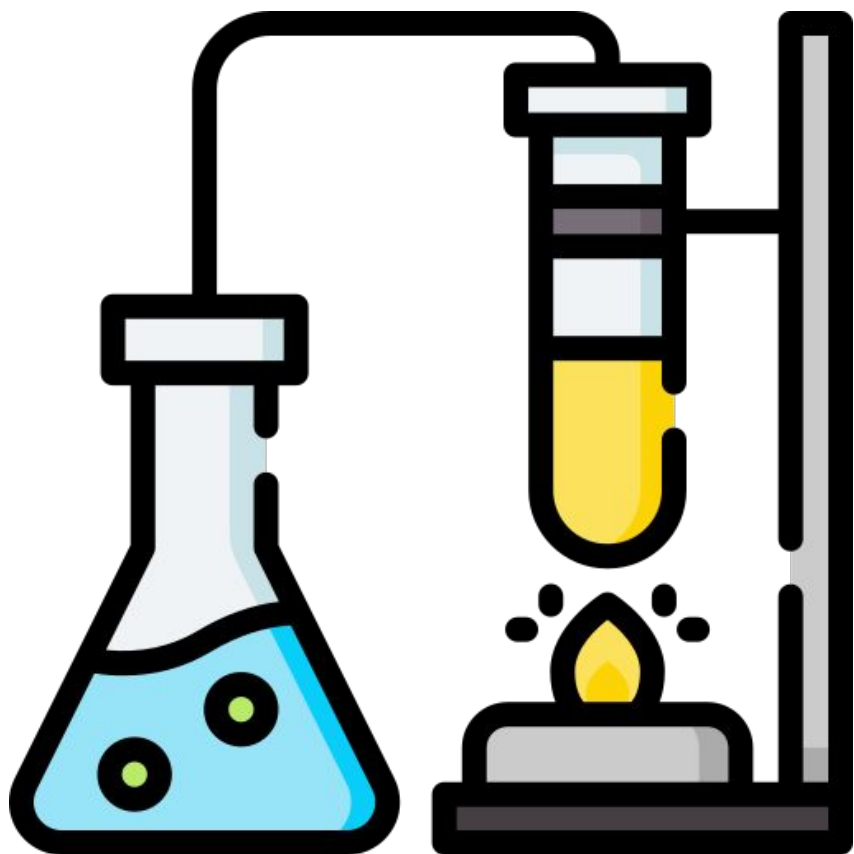
Don't Forget



STM32G0 - Experimentação



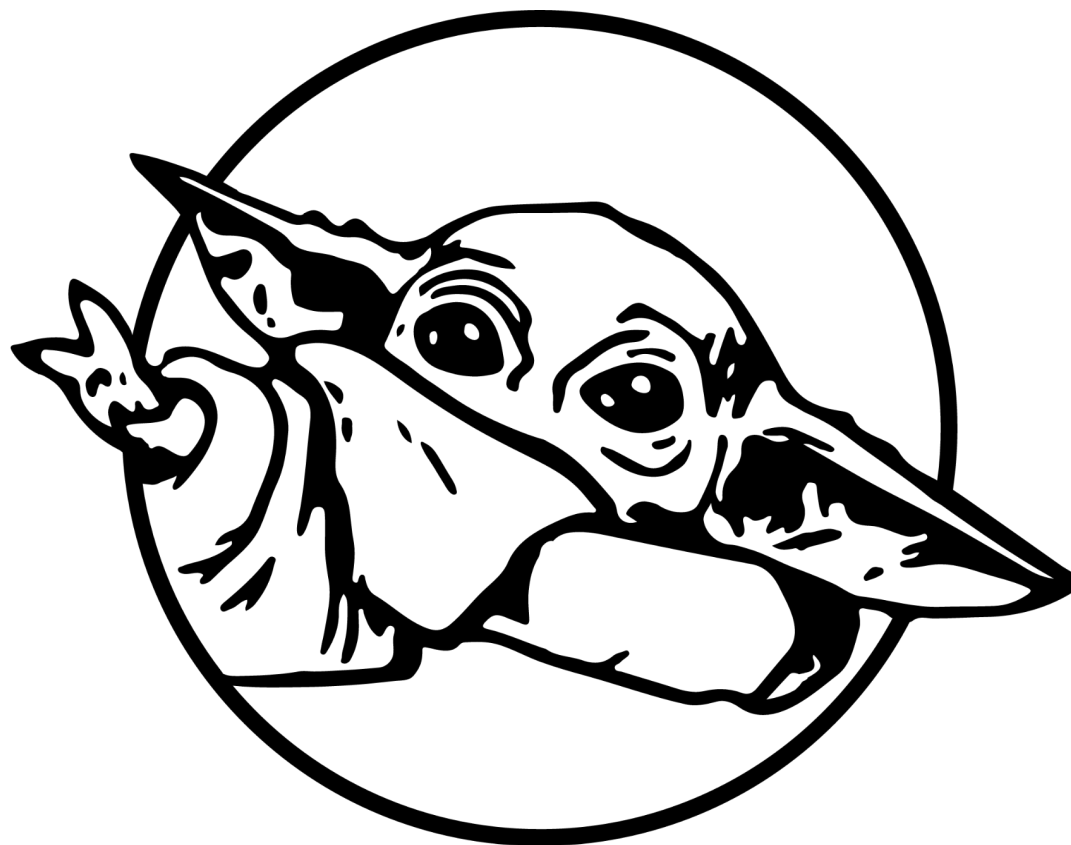
Vamos analisar agora um circuito funcionando com a SPI e analisar o *frame* gerado pela interface em um analisador lógico.



STM32G0 - Esperar?



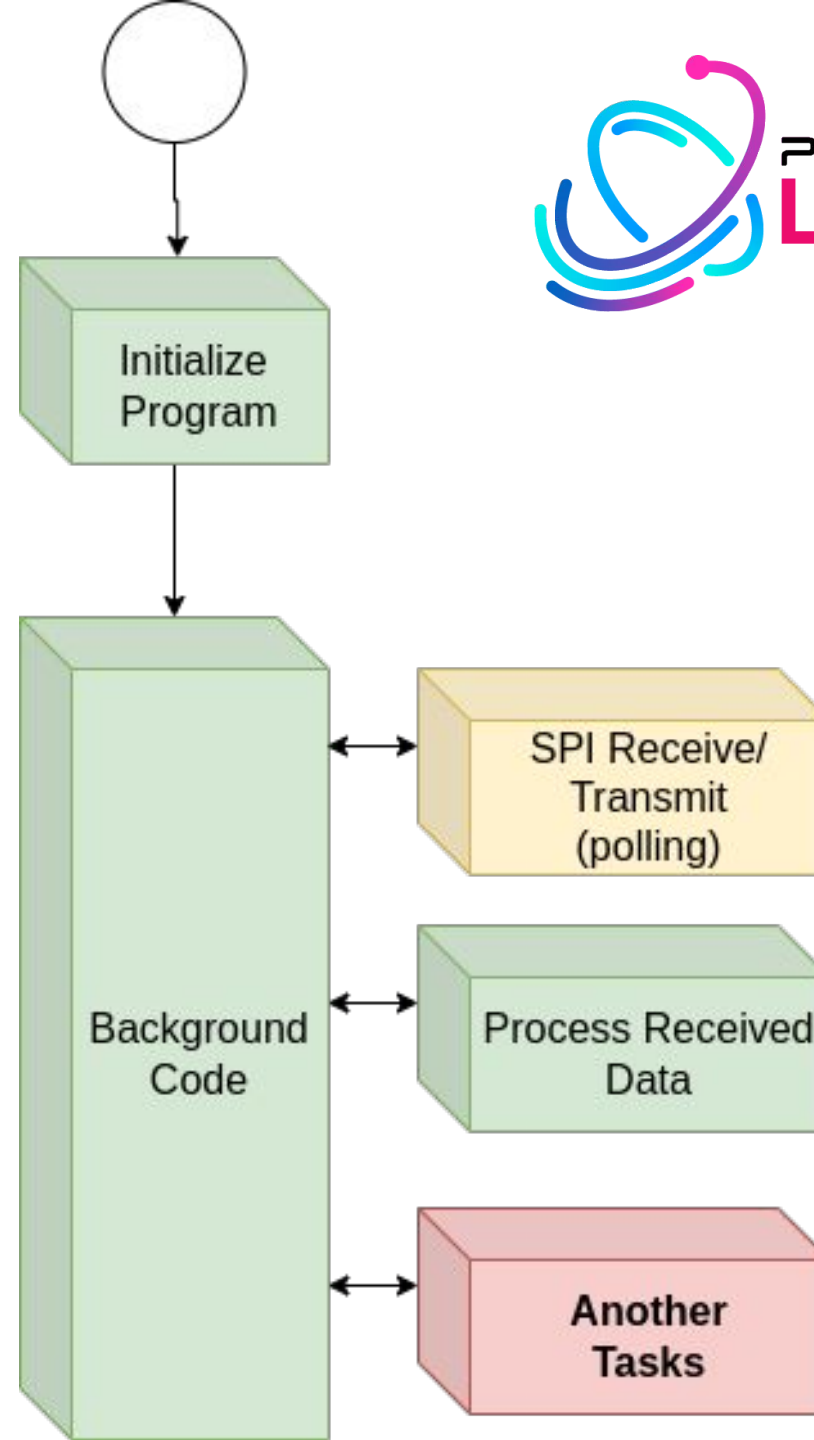
Esperar, não quero. Apressado eu ser.



STM32G0 - Interrupção

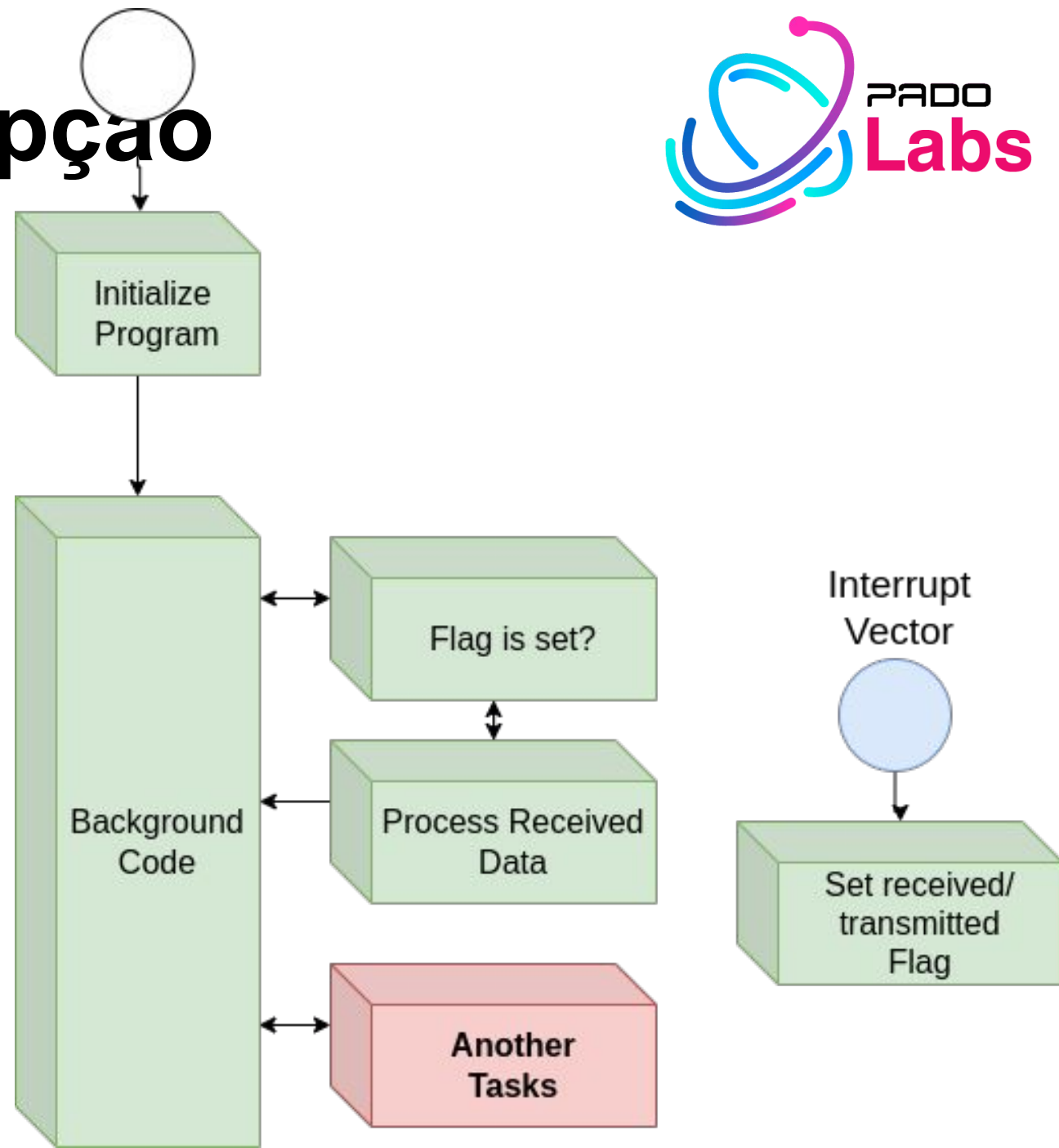


Como já vimos para o **ADC**, para a **UART**, *here we go again*, vamos falar de **Interrupção**.



STM32G0 - Interrupção

Quando utilizamos interrupção, podemos realizar as outras tarefas sem manter a CPU ociosa.

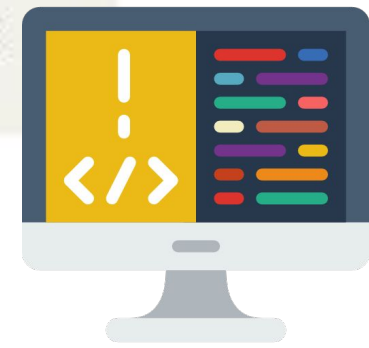


STM32G0 - Funções para IT



```
1 // Envia um array contendo os dados a serem enviados em modo
2 // de interrupcao, onde hspi refere
3 // ao handle da interface SPI, gerado pelo proprio CubeMX,
4 // pData e o ponteiro do vetor contendo os dados e Size e a
5 // quantidade de bytes de pData
6 HAL_SPI_Transmit_IT(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t
Size);
```

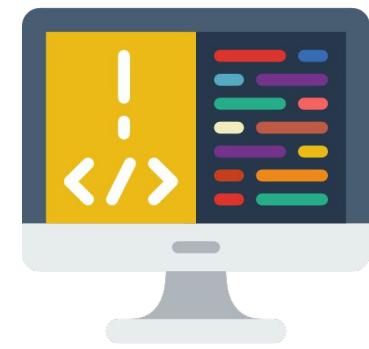
```
1 // Le um array da SPI, onde hspi refere ao handle da interface,
2 // gerado pelo proprio CubeMX, pData e o ponteiro do vetor que recebera
3 // os dados e Size e a quantidade de bytes que pretende-se receber
4 HAL_SPI_Receive_IT(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t
Size);
```



STM32G0 - Funções para IT



```
1 // Escreve e le um vetor de dados pela interface SPI, os parametros
2 // sao os mesmos para as outras funcoes, com excessao da adicao do
3 // ponteiro para a transmissao (pTxData) e outro para recepcao
4 // (pRxData), mas neste caso em modo de interrupcao
5 HAL_SPI_TransmitReceive_IT(SPI_HandleTypeDef *hspi, uint8_t *pTxData,
uint8_t *pRxData, uint16_t Size)
```



STM32G0 - Funções de Callback



```
1 // callback gerado quando ocorre a finalizacao de uma transmissao
2 void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi){
3
4 }
5
6 // callback gerado quando ocorre o termino de uma recepcao solicitada
7 void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef *hspi){
8
9 }
10
11 // callback gerado quando ocorre a finalizacao de uma transmissao/recepcao
12 void HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi){
13
14 }
```



STM32G0 - DMA

Assim como nos periféricos anteriores, podemos utilizar o DMA para realizar as operações da SPI.

Muito útil quando trabalhamos com memórias FLASH SPI e lidamos com grande quantidade de dados.

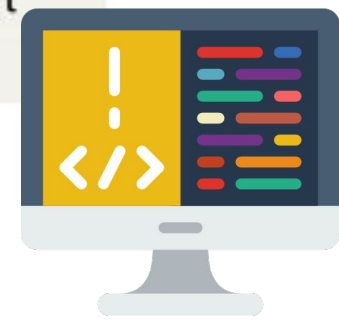


STM32G0 - Funções para DMA



```
1 // Envia um array contendo os dados a serem enviados em modo
2 // de interrupcao , onde hspi refere
3 // ao handle da interface SPI, gerado pelo proprio CubeMX,
4 // pData e o ponteiro do vetor contendo os dados e Size e a
5 // quantidade de bytes de pData
6 HAL_SPI_Transmit_DMA(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t
Size);
```

```
1 // Le um array da SPI, onde hspi refere ao handle da interface ,
2 // gerado pelo proprio CubeMX, pData e o ponteiro do vetor que recebera
3 // os dados e Size e a quantidade de bytes que pretende-se receber
4 HAL_SPI_Receive_DMA(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t
Size);
```

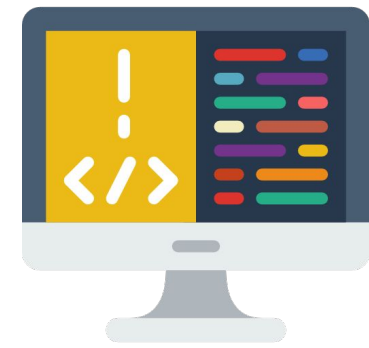


STM32G0 - Funções para DMA



Claro que temos também para a transmissão e recepção

```
1 // Escreve e le um vetor de dados pela interface SPI, os parametros
2 // sao os mesmos para as outras funcoes, com excessao da adicao do
3 // ponteiro para a transmissao (pTxData) e outro para recepcao
4 // (pRxData), mas neste caso em modo DMA
5 HAL_SPI_TransmitReceive_DMA(SPI_HandleTypeDef *hspi, uint8_t *pTxData,
uint8_t *pRxData, uint16_t Size)
```



STM32G0 - Callbacks para DMA



os callbacks para o DMA são os **mesmos** para a interrupção!



STM32G0 - Mais Funções

Para consultar mais funções implementadas pelo **HAL**, utilize a documentação **UM2319:Description of STM32G0 HAL and low-layer drivers**, nos capítulos:

- 46 HAL SPI Generic Driver
- 47 HAL SPI Extension Driver

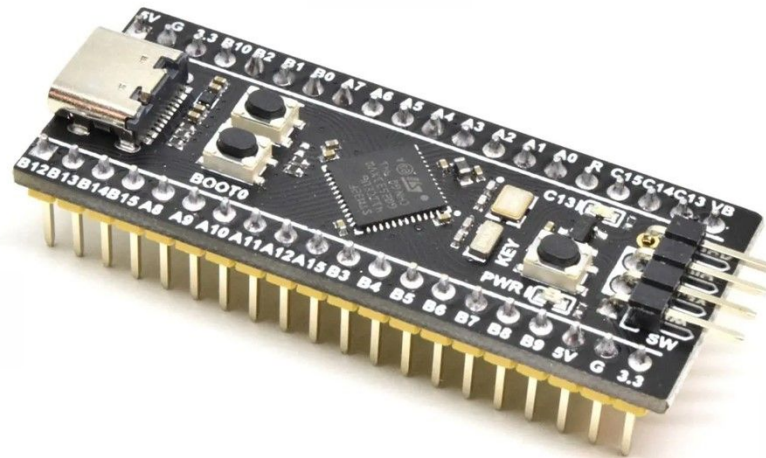


Dúvidas ??

The MCU most people like:



The MCU I like:



Referências



CORELIS. **SPI Tutorial**. 2022. <https://www.corelis.com/education/tutorials/spi-tutorial/> . Acesso em 20 de Fevereiro de 2022.

MAGDY, Khaled. <https://deepbluembedded.com/stm32-spi-tutorial/>. 2021.
<https://deepbluembedded.com/stm32-spi-tutorial/> . Acesso em 13 de Fevereiro de 2021.

NORDIC SEMICONDUCTOR. **nRF24L01+ Single Chip 2.4GHz Transceiver**. 1.0. ed. Otto Nielsens vei 12 7004 Trondheim, 2008.

SACCO, Francesco. **Comunicação SPI – Parte 1**. 2014. <https://www.embarcados.com.br/spi-parte-1/> . Acesso em 13 de Fevereiro de 2021.

ST MICROELECTRONICS. **RM0444 - Reference Manual**. 5. ed. [S.l.], 2020. STM32G0x1 advanced Arm ® -based 32-bit MCUs.

__. **UM2319: Description of STM32G0 HAL and low-layer drivers**. 2. ed. [S.l.], 2020.

__. **UM2324 - User Manual**. 4. ed. [S.l.], 2021. STM32 Nucleo-64 boards (MB1360).

__. **STM32G0B1xB/xC/xE**. 2. ed. [S.l.], 2021. Arm ® Cortex ® -M0+ 32-bit MCU, up to 512KB Flash, 144KB RAM, 6x USART, timers, ADC, DAC, comm. I/Fs, 1.7-3.6V.

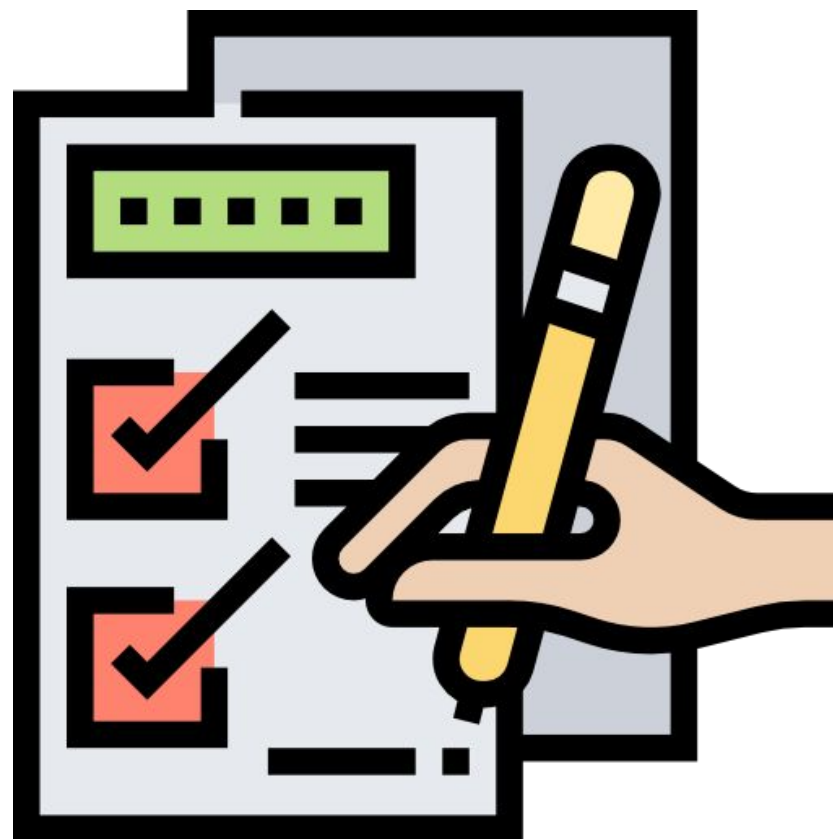
WIKIPEDIA. **Serial Peripheral Interface**. 2021. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface . Acesso em 15 de Fevereiro de 2022.



Mão na Massa

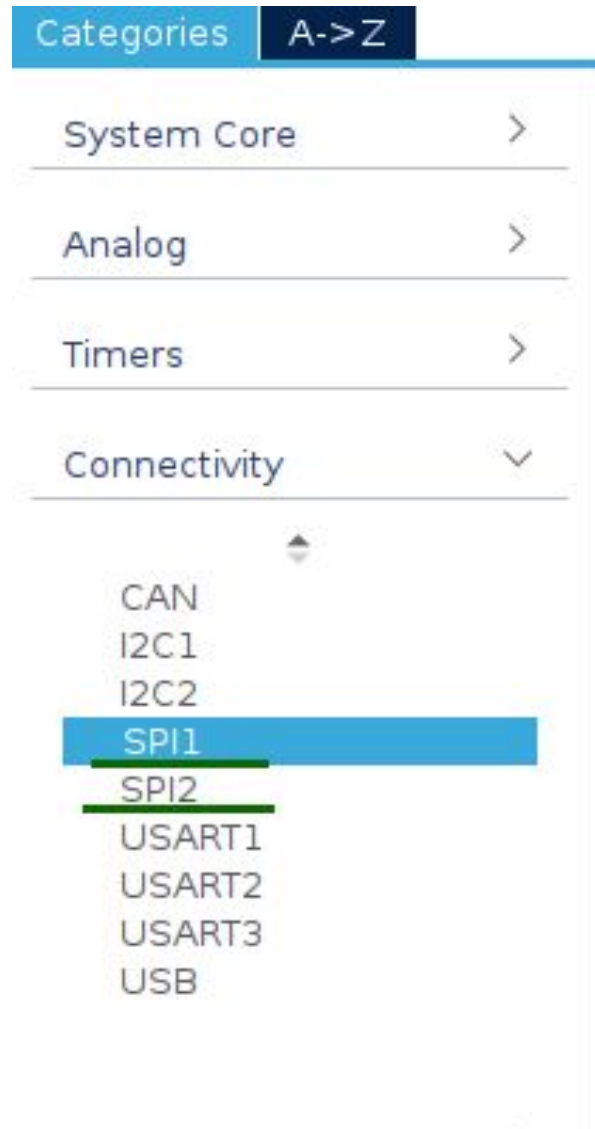


Lista de Exercícios #6



Comunicação SPI

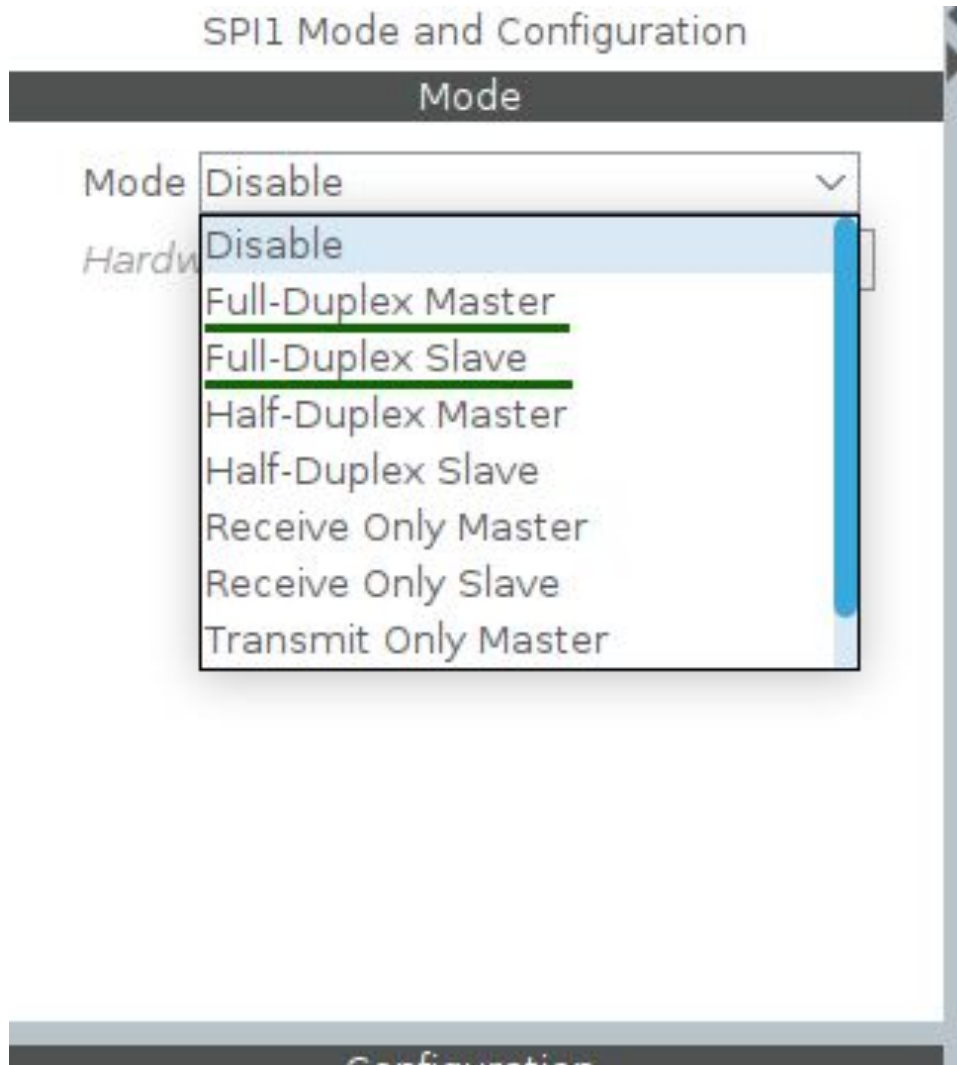
SPI em modo **polling**



Selecionamos o periférico de SPI na seção *Connectivity*.

É possível também escolher os terminais pelos pinos do microcontrolador, habilitando os para SPI ao clicar no terminal desejado.

SPI em modo **polling**



Selecionamento então o modo desejado. O mais comum é o **Full-Duplex Master**.

Caso você utilize o modo *Slave*, selecione **Full-Duplex Slave**.

SPI em modo **polling**

Configuration

Reset Configuration

✓ GPIO Settings

✓ NVIC Settings

✓ DMA Settings

✓ Parameter Settings

✓ User Constants

Configure the below parameters :

Search (Ctrl+F)

⏪

⏩

i

Basic Parameters

Frame For...

Motorola

Data Size

8 Bits

First Bit

MSB First

Clock Parameters

Prescaler (f...

32

* Baud Rate

2.25 MBits/s

Clock Polari...

Low

Clock Phas...

1 Edge

Advanced Parame...

CRC Calcula...

Disabled

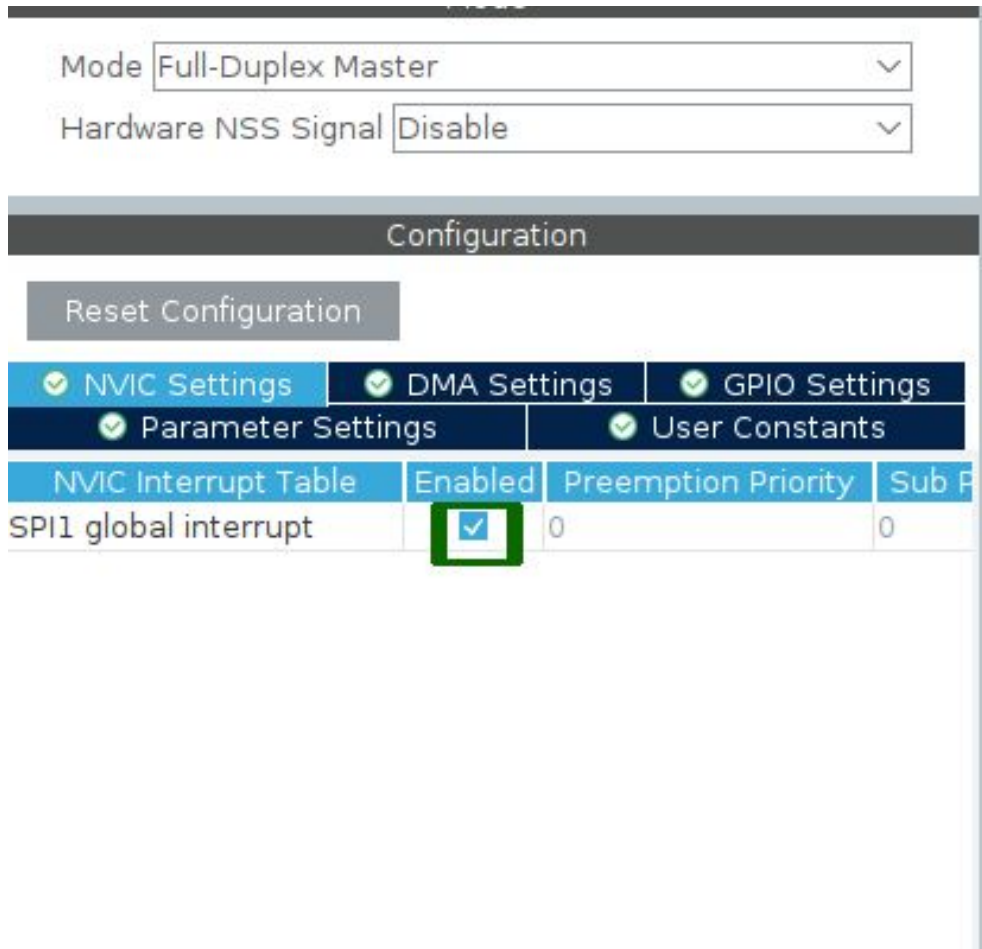
NSS Signal ...

Software

Na tela ao lado temos as configurações para o modo **Full-Duplex Master**.

Ajuste o *Prescaler* para uma frequência compatível com seus periféricos.

SPI em Interrupção

A screenshot of a software configuration interface for SPI. At the top, there are two dropdown menus: "Mode" set to "Full-Duplex Master" and "Hardware NSS Signal" set to "Disable". Below these is a "Configuration" section with a "Reset Configuration" button. A row of tabs includes "NVIC Settings" (selected), "DMA Settings", "GPIO Settings", "Parameter Settings", and "User Constants". Under "NVIC Settings", there is a table with columns "NVIC Interrupt Table", "Enabled", "Preemption Priority", and "Sub P". The first row in the table is "SPI1 global interrupt", with a checked checkbox in the "Enabled" column, and "0" in the "Preemption Priority" and "Sub P" columns. The "Enabled" checkbox is highlighted with a green square.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub P
SPI1 global interrupt	<input checked="" type="checkbox"/>	0	0

Após feitas as configurações anterior, vamos até a aba do NVIC Settings.

E habilitamos a interrupção global do periférico de **SPI**.

SPI em DMA



Vamos até a aba do DMA Settings.

Clicamos em Add e selecionamos os canais da SPI que serão utilizados, no exemplo ao lado, habilitou-se os dois canais da SPI.

Configuration

Reset Configuration

✓ NVIC Settings

✓ DMA Settings

✓ GPIO Settings

✓ Parameter Settings

✓ User Constants

DMA Request	Channel	Direction	Priority
SPI1_RX	DMA1 Chan...	Peripheral T...	Low
SPI1_TX	DMA1 Chan...	Memory To ...	Low

Add

Delete

UART em DMA

SPI1_RX	DMA1 Channel 2	Peripheral To Memory	Low
SPI1_TX	DMA1 Channel 3	Memory To Peripheral	Low

AddDelete

DMA Request Settings

Mode	Normal	Increment Address	<input type="checkbox"/>	Peripheral	<input type="checkbox"/>	Memory	<input checked="" type="checkbox"/>
Data Width	Byte					Byte	

Configuramos os canais de acordo com a necessidade. A configuração mais comum é a padrão.



PADO
Labs