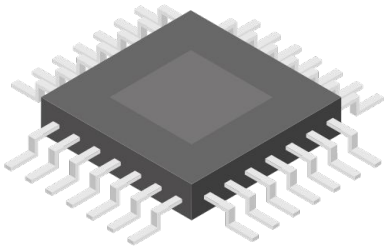
PADO

Labs

Microcontroladores



Prof.º: Pablo Jean Rozário

pablo.jean@padotec.com.br

/in/pablojeanrozario

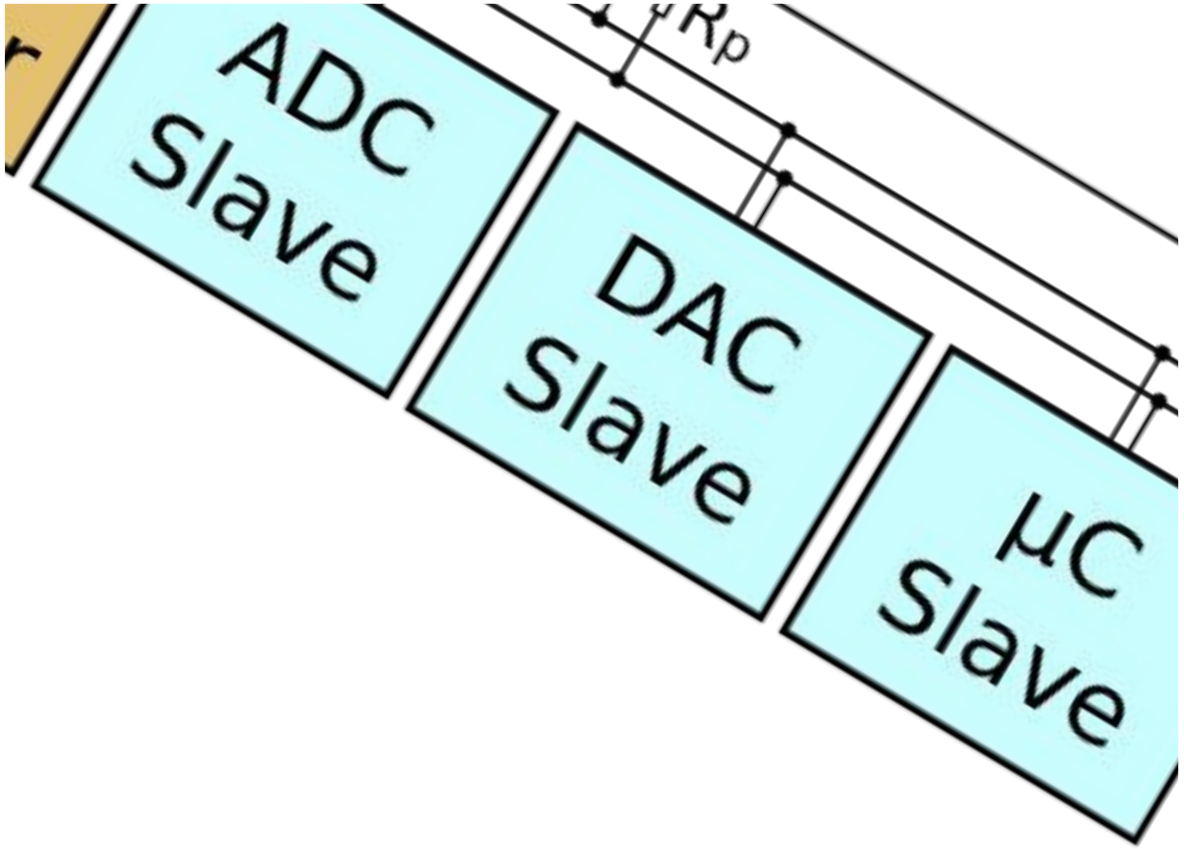
https://github.com/Pablo-Jean

Comunicação I2C

Índice da Aula #7

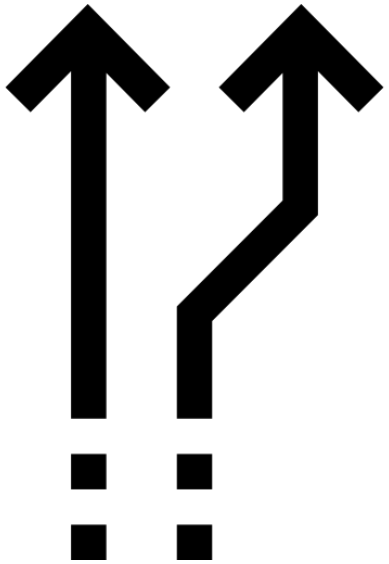
- Introdução
- Camada Física da SPI
- Framing
- Vantagens e Desvantagens
- Exemplo de Aplicação
- I2C no STM32G0B1RE
- Funções Utilizadas
- Interrupções e DMA
- Lista de Exercícios #7

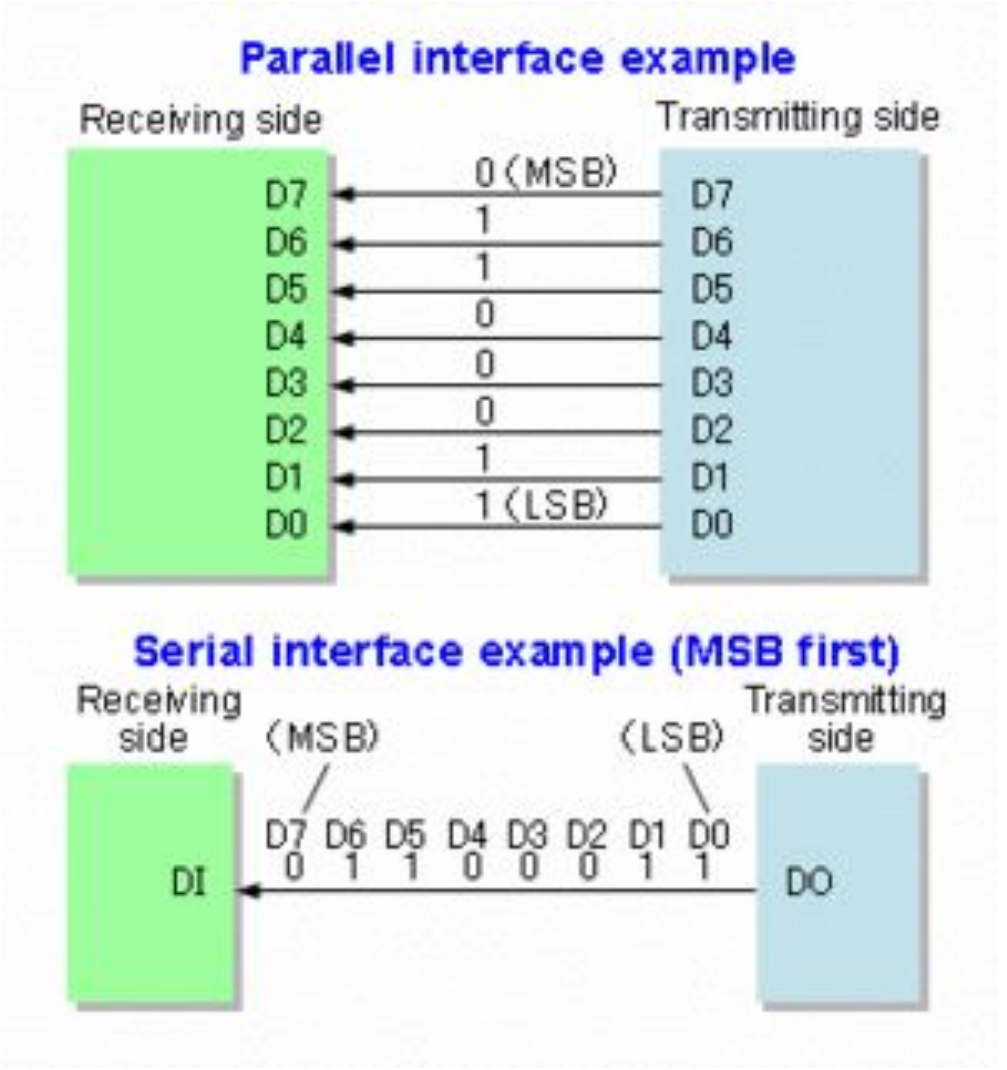
Inter-Integrated Circuit
I2C



Paralelo

Antes de tudo, vamos falar das interfaces paralelas que não vimos antes das seriais.





Introdução

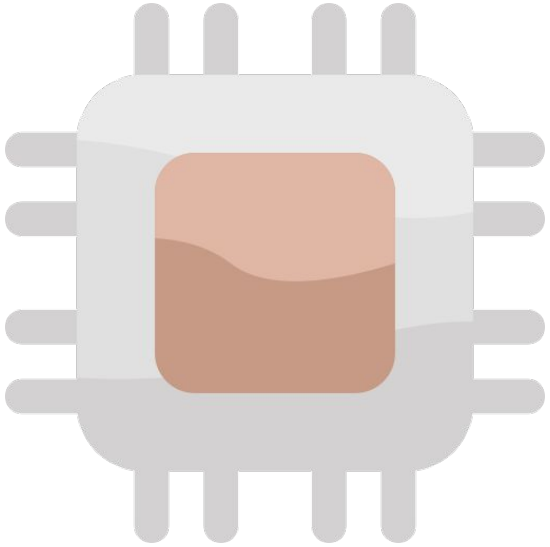
I2C, ou Inter-*Integrated Circuit*, é uma interface de comunicação síncrona **Multi-controllers/Multi-targets**, inventado pela *Philips Semiconductors* para comunicação entre seus dispositivos.

Recentemente o padrão I2C alterou a nomenclatura *Master/Slave* para *Controller/Target*, respectivamente.

Introdução

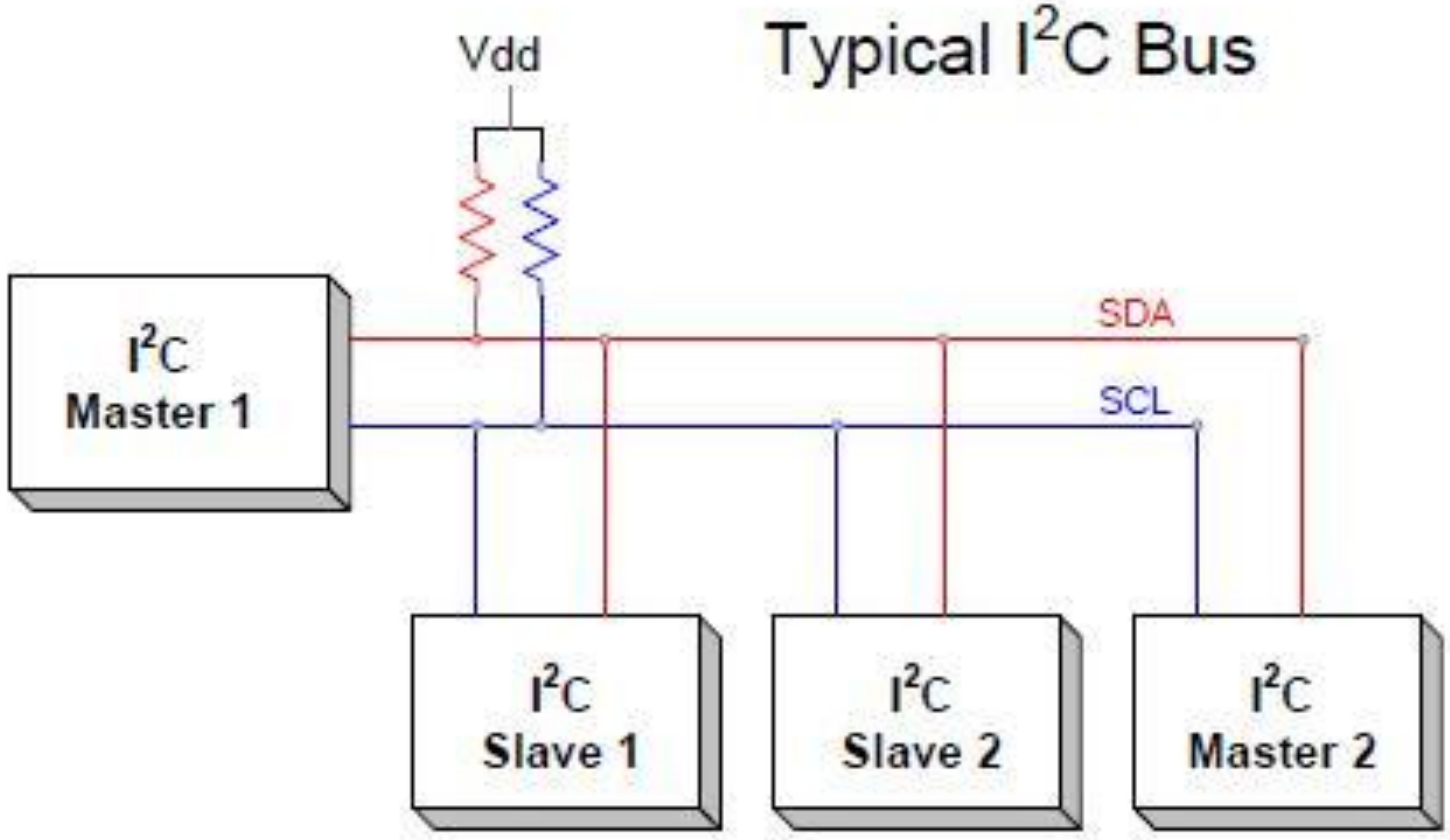
Alguns periféricos que utilizam I2C são:

- Memória EEPROM
- ADCs e DACs;
- Codecs de Áudio;
- Displays LCDs e OLEDs;
- Expansores de IOs;
- Acelerômetros, etc.



Note que é similar a SPI as aplicações, tanto que muito periféricos suportam **SPI** e **I2C**.

Camada Física - Estrutura



Perceba que temos apenas **Dois fios**. Com apenas duas conexões lógicas podemos nos comunicar com até **128 dispositivos**.

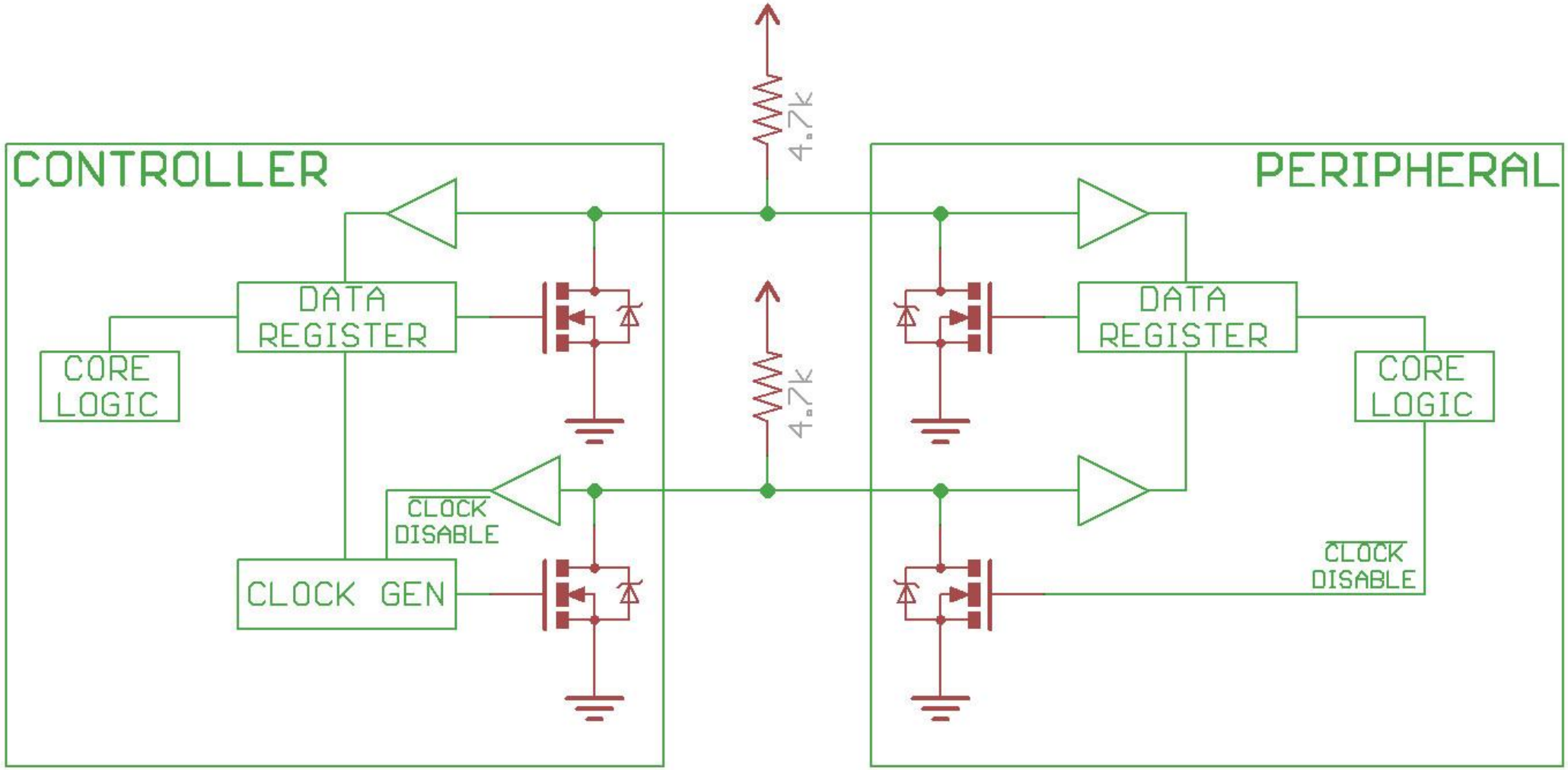
Camada Física - Terminais

SCL: Sinal de clock que é gerenciado pelo *Controller* que esta enviando dados atualmente;

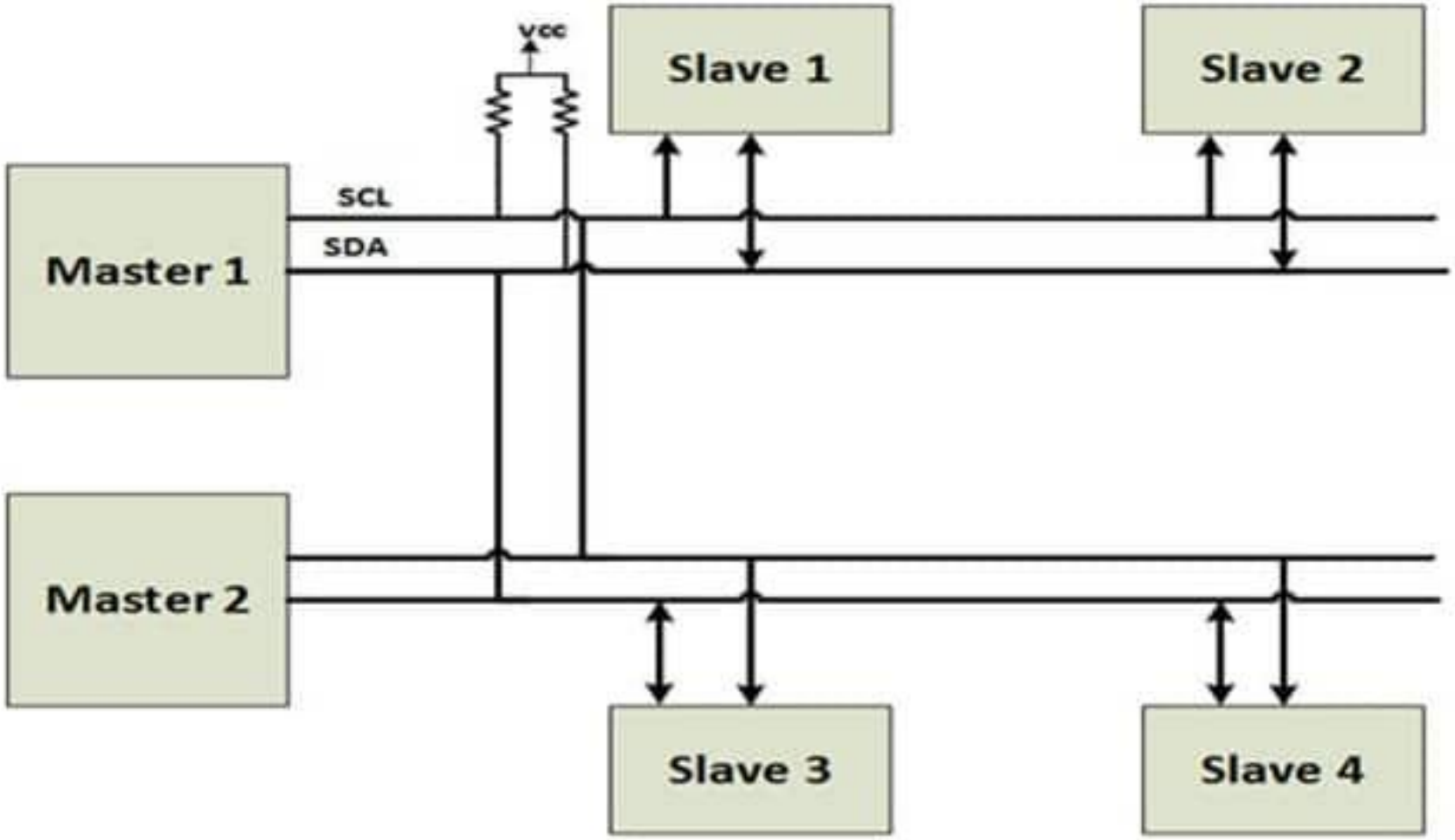
SDA: Sinal de dados bi-direcional, onde é realizado o envio e recebimento de informações.

Os *targets* podem segurar o SCL em nível lógico baixo para atrasar a comunicação com o *Controller*. Isto é chamado de *Clock Stretching*.

Camada Física - Terminais

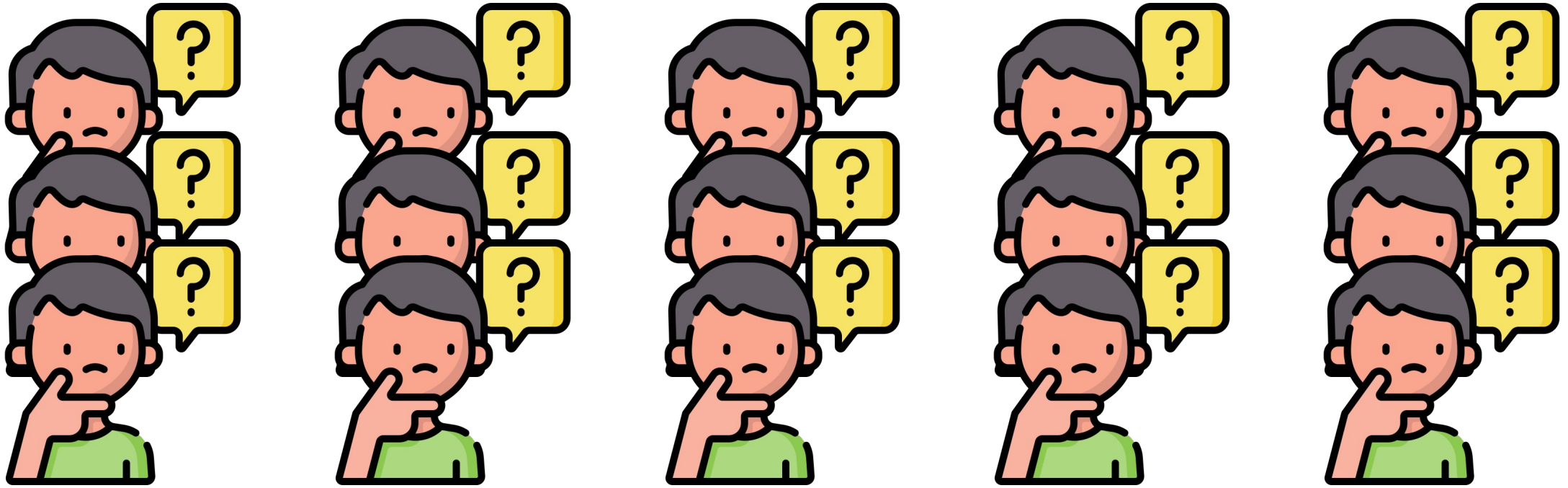


Camada Física - Multicontrollers

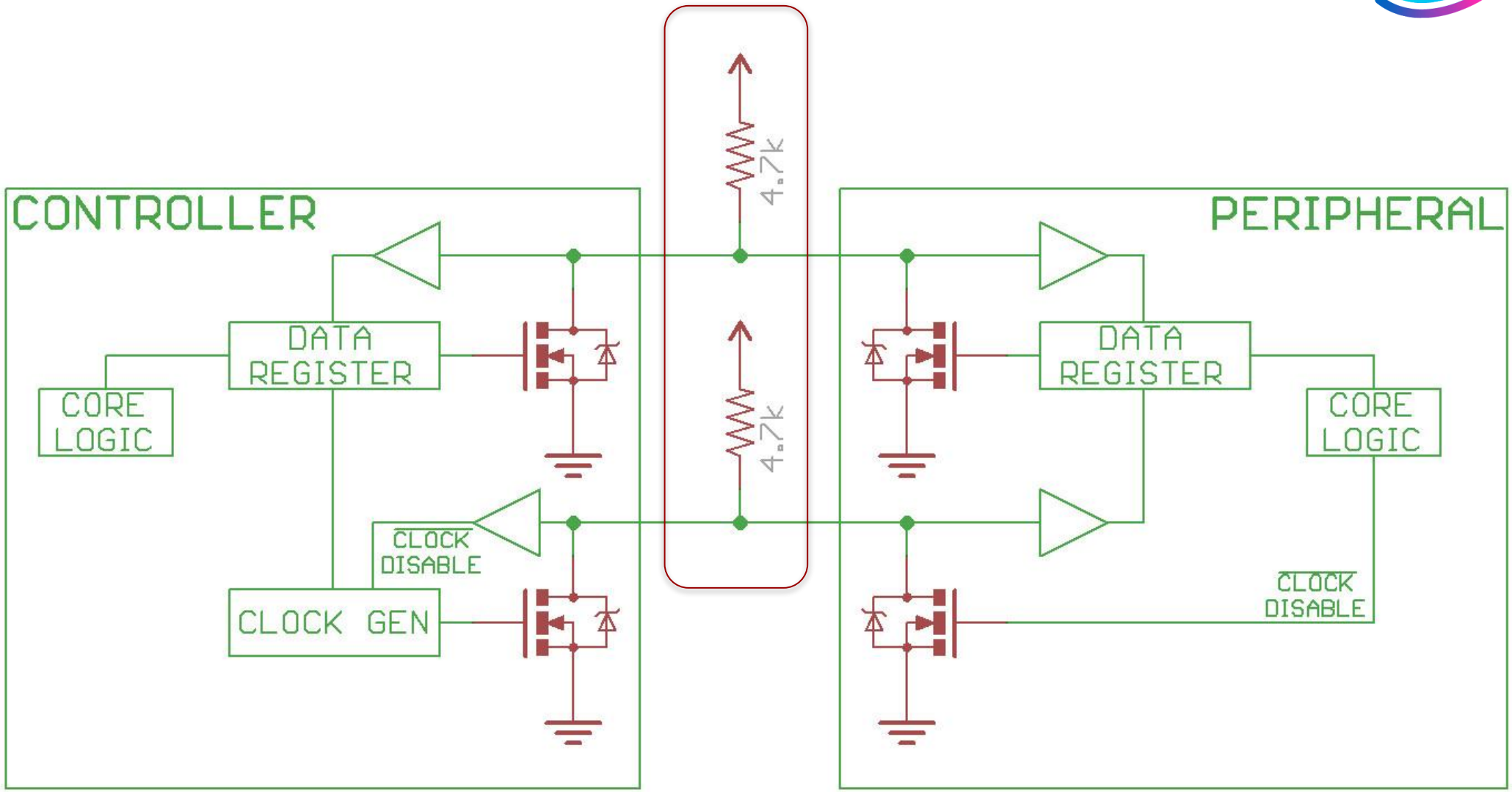


Camada Física - Pull-up Resistors

Com sua percepção, observa perspicazmente que nas linhas de **SDA** e **SCL** temos conectados resistores de Pull-Up, e por que isto acontece?



Camada Física - Pull-up Resistors



Velocidade da I2C

O *data-link* I2C possui padrões de velocidade, sendo estas:

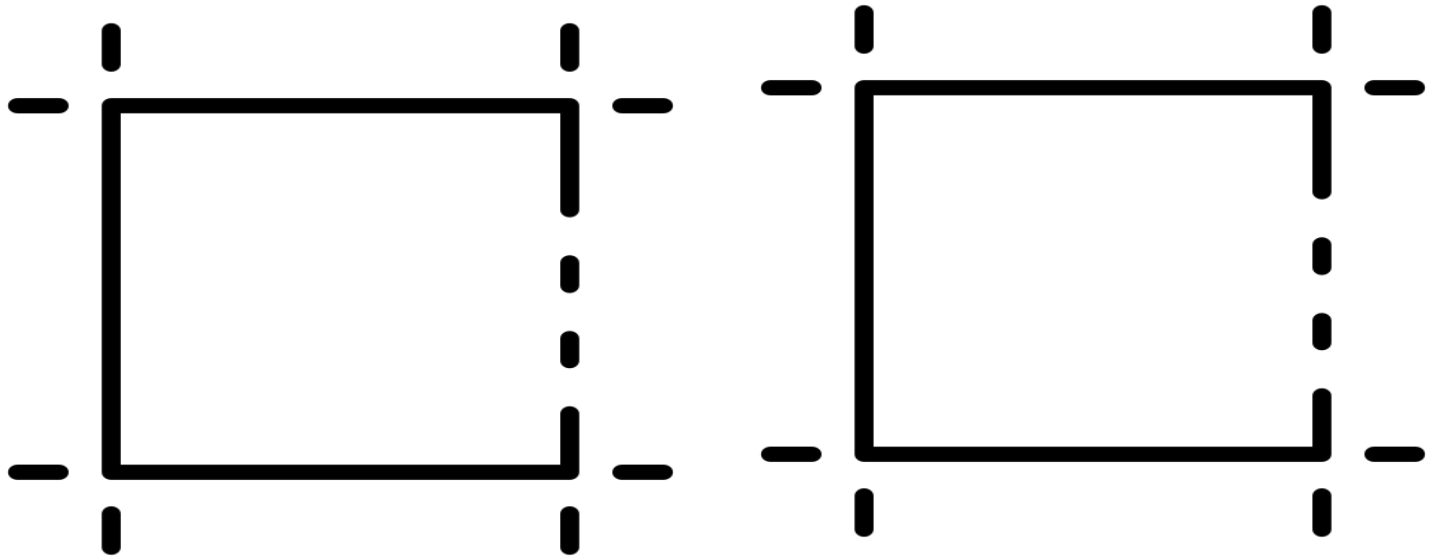
Modo	Velocidade
Standard Mode	100 Kbit/s
Fast Mode	400 Kbit/s
Fast Mode +	1 Mbit/s
High Speed Mode	3.4 Mbit/s
Ultra Fast Mode	5 Mbit/s

I2C - *Framing*

Show, vimos que podemos pendurar uma ***renca*** de devices no barramento, mas como identificamos com quem queremos trocar informações? Sem ocorrer colisões

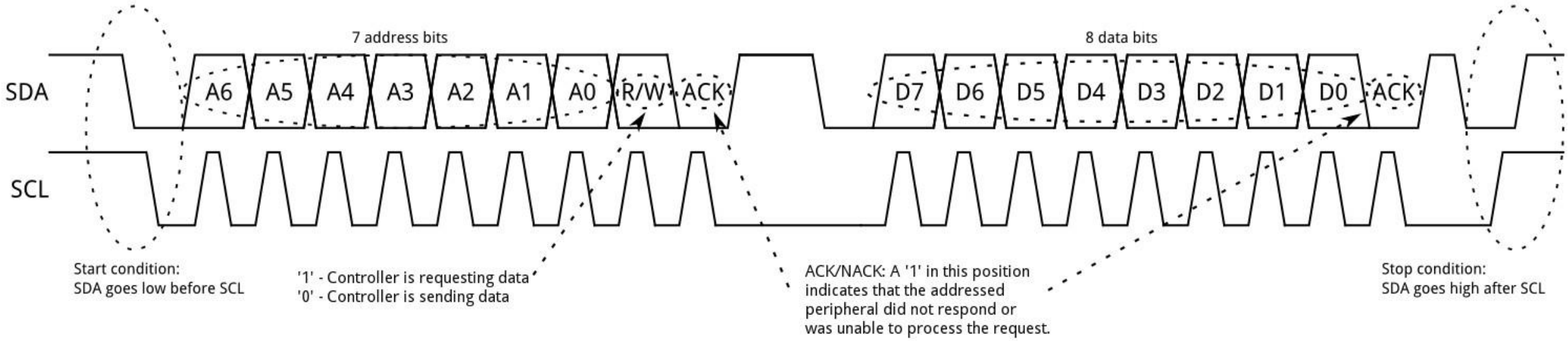
I2C - *Framing*

A comunicação I2C possui um protocolo relativamente complexo, se comparada a SPI (que se quer possui protocolo) e a UART (que possui um protocolo bem simples).



I2C - *Framing*

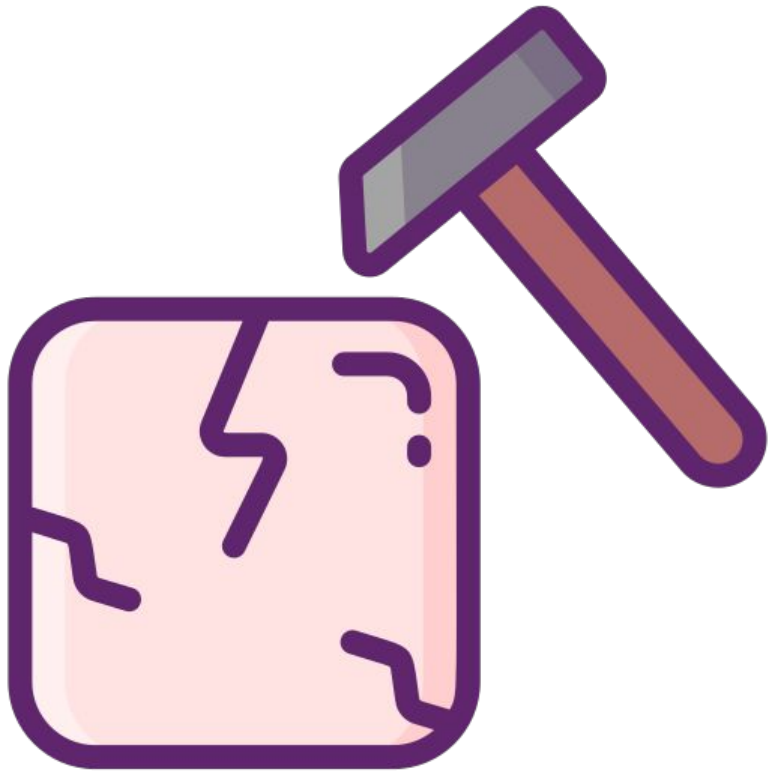
Frames básicos de uma simples mensagem.



I2C - *Framing*

Vamos quebrar o *frame* em 4 partes:

- ***Start Condition***
- ***Address Frame***
- ***Data Frame***
- ***Stop Condition***



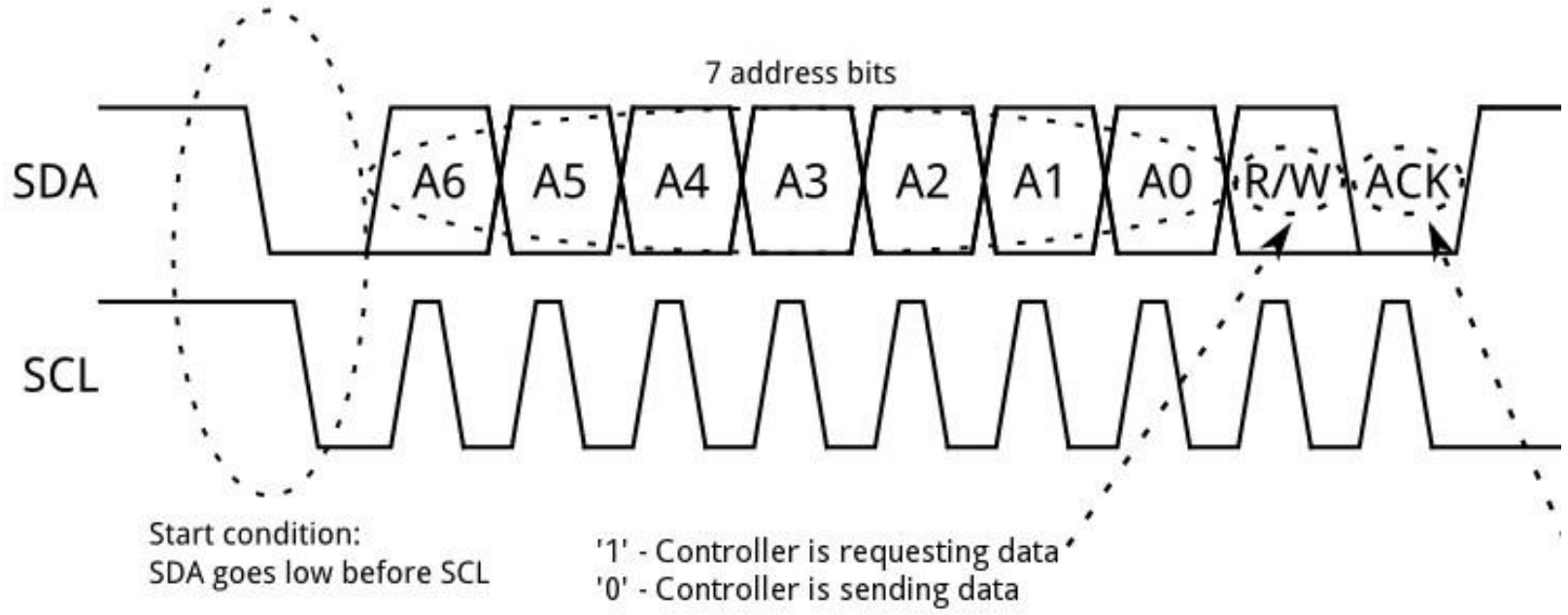
Framing - Start Condition

Sempre que se inicia um *Address Frame,* o *Controller* coloca o **SCL** em nível lógico **baixo**.

Com isto, todos os periféricos ficam atento ao recebimento de um pacote.

Framing - Address Frame

Este é sempre o primeiro *frame* de qualquer mensagem pelo barramento I2C. Sendo enviado os 7 bits que identificar o dispositivo (endereçamento), seguido do bit de *Read/Write*.



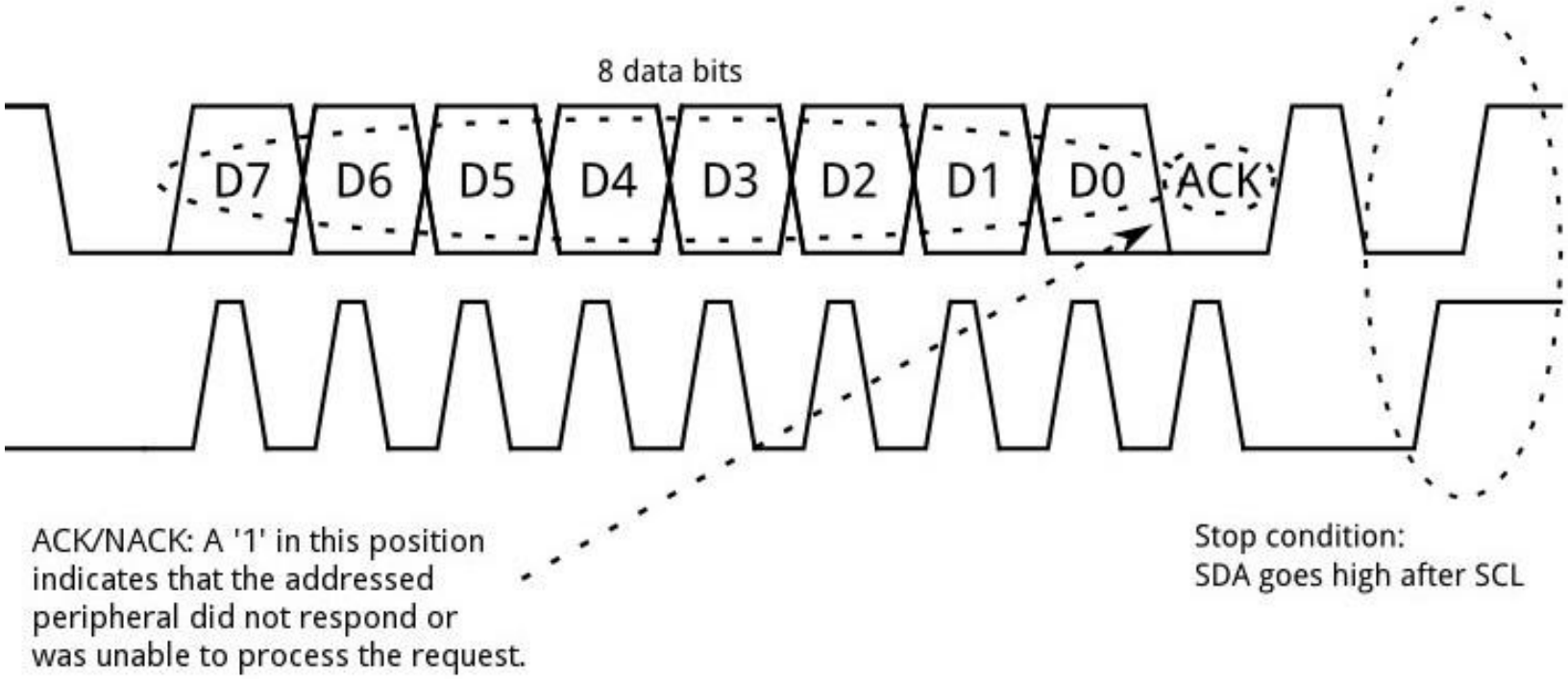
Framing - Address Frame

O nono bit enviado, trata-se do **ACK/NACK**, onde o *target* sinaliza que o byte foi recebido com sucesso. É indicado através do pino de *SDA*, controlado pelo *target*.

Um valor de nível lógico **0**, indica **OK**. Valor de **1**, indica **Falha**.

Framing - Data Frame

Utilizado para enviar e/ou receber dados de um *target*. O *Controller* simplesmente continua a enviar o trem de pulsos em **SCL**, e no terminal de **SDA** são enviado os dados, do *Controller* para o *target*,
ou vice-versa.



Framing - Data Frame

A quantidade de *data frames* é arbitrária, onde a maioria dos periféricos permitem que sejam realizadas **escritas** e **leituras sub-sequentes**, e, inclusive, com incremento de um registrador interno.
Como por exemplo, uma EEPROM I2C.

Framing - Stop Condition

Após todos os *Data Frames* terem sido enviados, é gerado um *Stop condition* por parte do *Controller* para encerrar a comunicação.

Sinal definido por um pulso de **0 para 1** no **SDA** após uma transição de **0 para 1** no **SCL**.

I2C - Vantagens

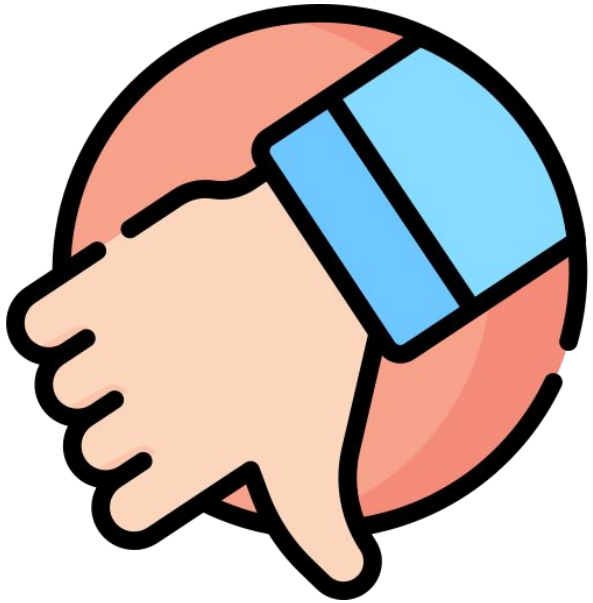
Algumas **vantagens** da interface **I2C** são:

- Suporte a *MultiControllers*
- Apenas duas conexões lógicas, enquanto SPI requer no mínimo 4
- Endereçamento simples
- *Flow control* (ACK, clock stretching, etc)
- Fácil implementar diagnósticos
- Dispositivos podem ser adicionados ou
- removidos com mínimo impacto no hardware.

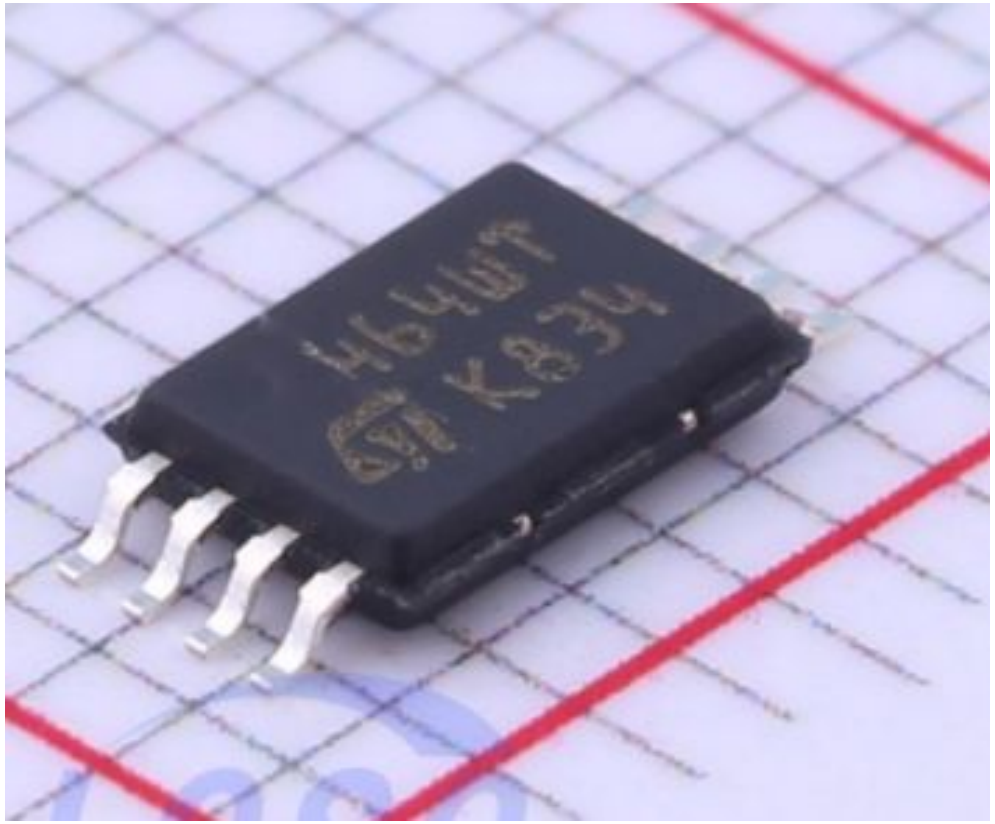
I2C - Desvantagens

E algumas **desvantagens** da interface **I2C** são:

- Comunicação *Half-Duplex*
- Protocolo gerido por uma *stack* de software, gerando mais carga na CPU.



I2C - Aplicação - M24C64



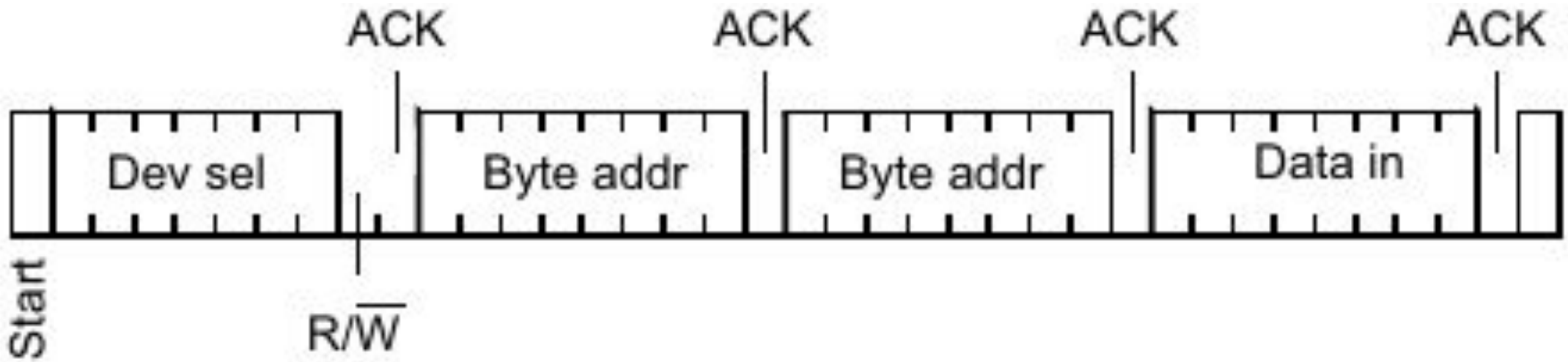
I2C - Aplicação - M24C64

	Device type identifier(1)				Chip Enable address(2)			RW
	b7	b6	b5	b4	b3	b2	b1	b0
Device select code when addressing the memory array	1	0	1	0	E2	E1	E0	RW
Device select code when accessing the Identification page	1	0	1	1	E2	E1	E0	RW

1. The most significant bit, b7, is sent first.
2. E0, E1 and E2 are compared with the value read on input pins E0, E1 and E2.

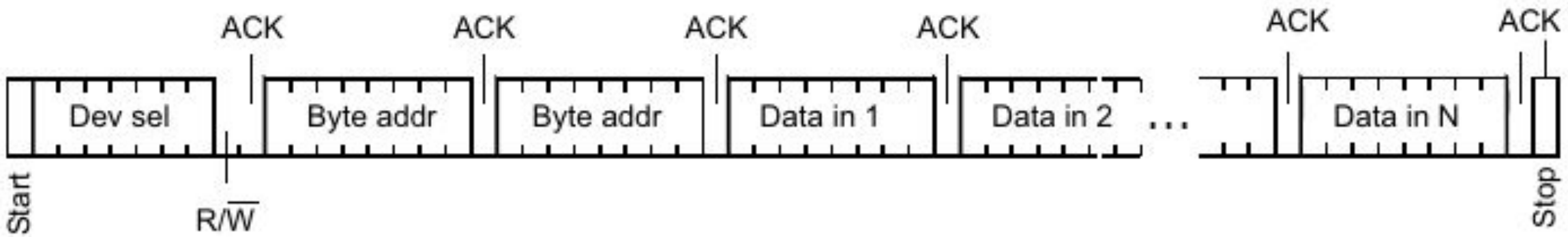
Endereçamento de uma EEPROM deste tipo

I2C - Aplicação - M24C64



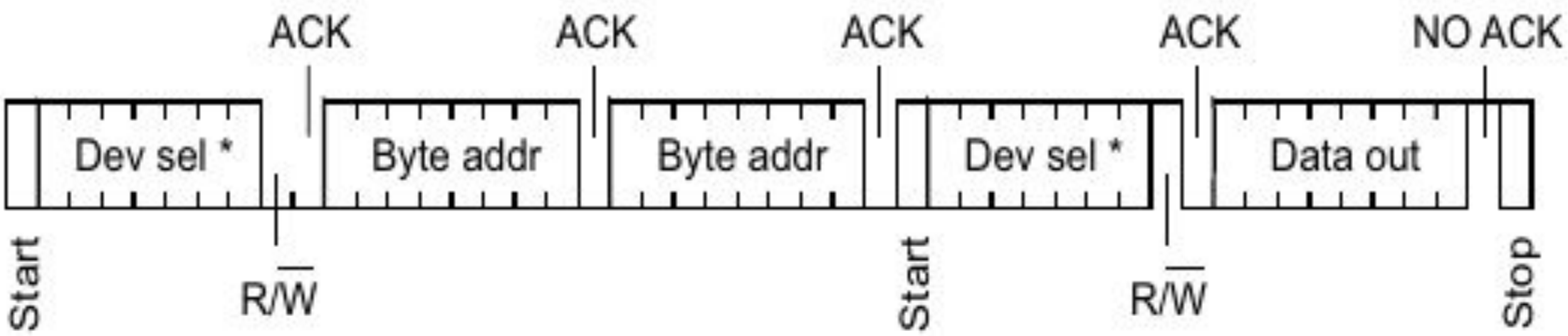
Operação de Byte Write

I2C - Aplicação - M24C64



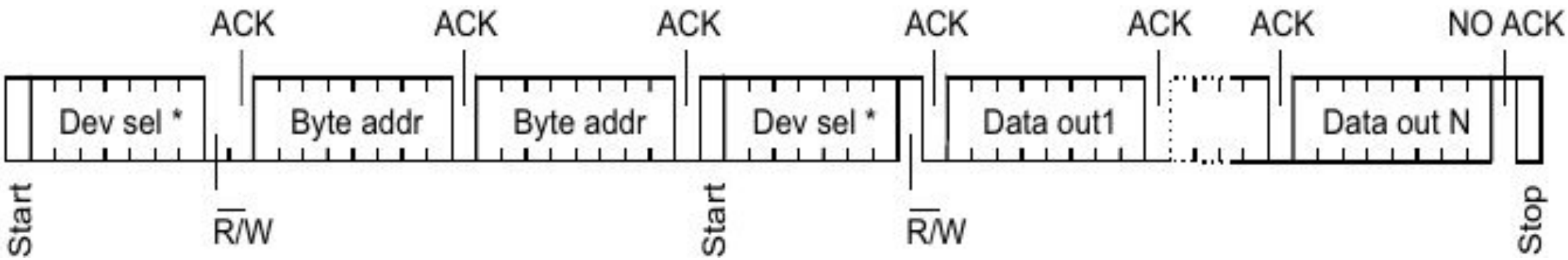
Operação de Page Write

I2C - Aplicação - M24C64



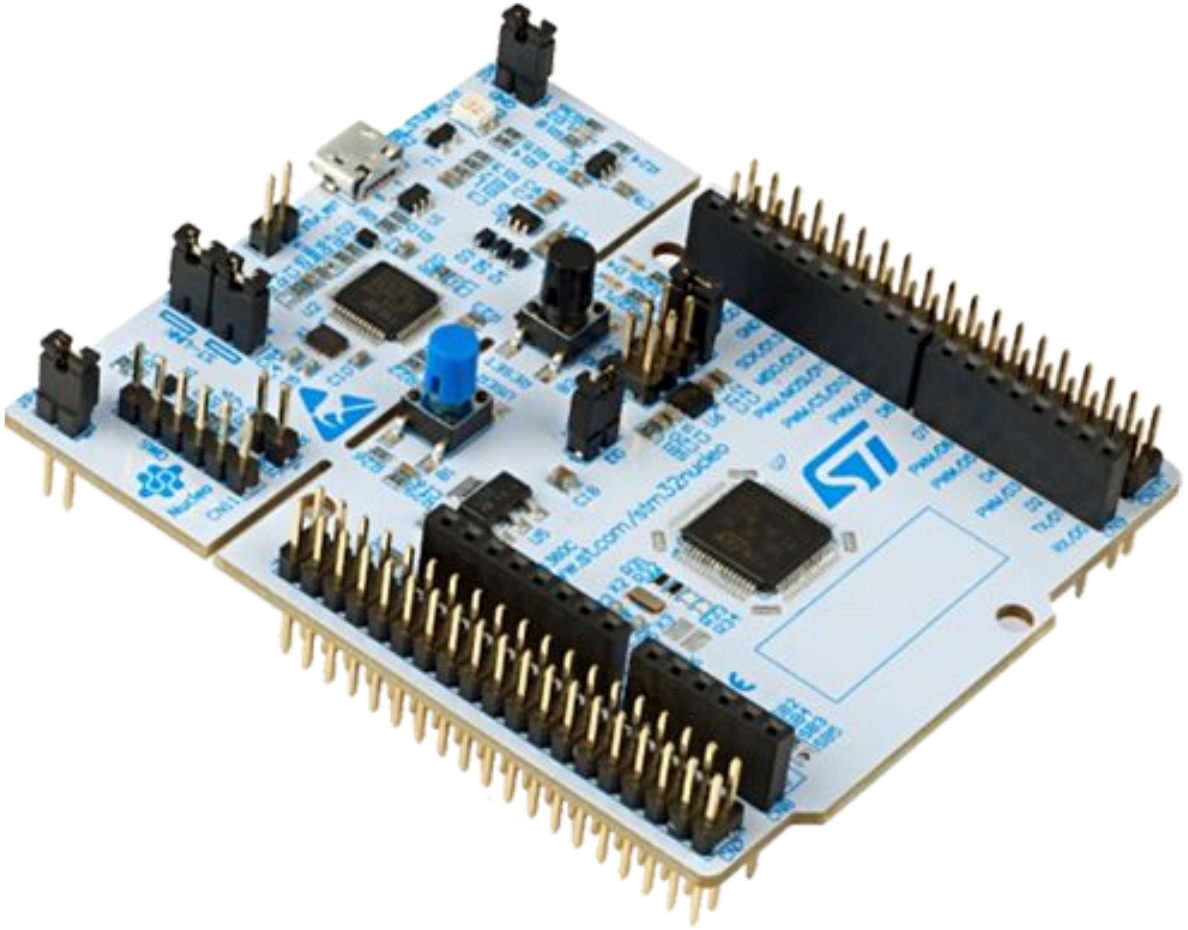
Operação de Random Address Read

I2C - Aplicação - M24C64



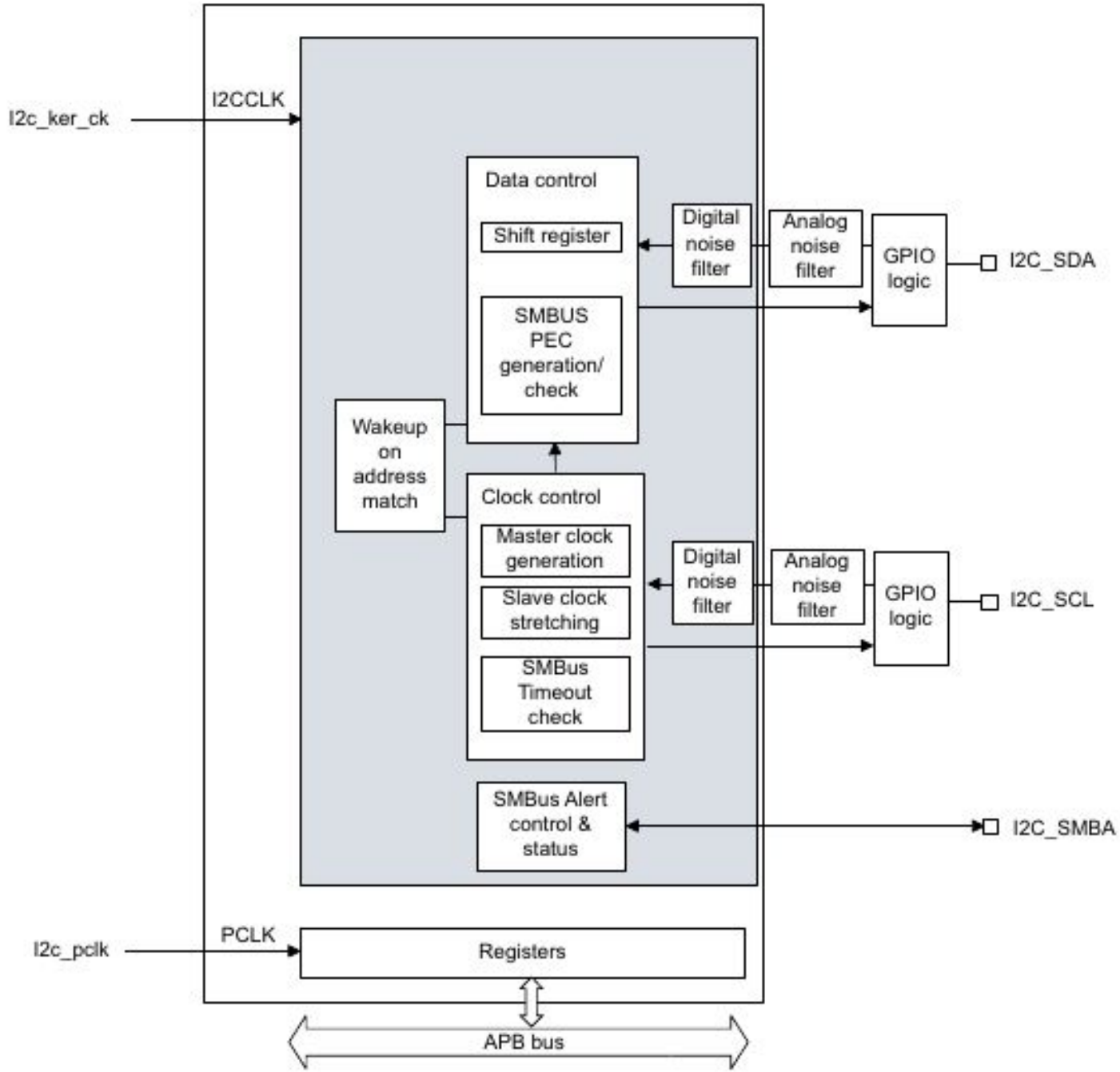
Operação de Sequential Read

I2C no STM32G0



STM32 - Características da I2C

- Opera em Multimaster;
- Endereçamento de *7bits* ou *10bits*;
- Multiplos endereços para Slave de 7bits;
- Software reset
- Entre outros ….
- Para mais: consulte
- **RM0444 Capitulo 32 Inter-integrated circuit (I2C) interface**


I2c_ker_ck
I2CCLK
Data control
Shift register
Digital noise filter
Analog noise filter
GPIO logic
I2C_SDA
SMBUS PEC generation/ check
Wakeup on address match
Clock control
Master clock generation
Slave clock stretching
SMBus Timeout check
Digital noise filter
Analog noise filter
GPIO logic
I2C_SCL
SMBus Alert control & status
I2C_SMBA
I2c_pclk
PCLK
Registers
APB bus

STM32G0 - Localização

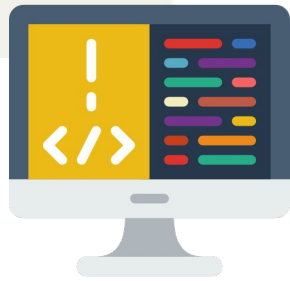
Utilize o documento **STM32G0B1xB/xC/xE**, e consulte a **tabela 13 à 20**, na **página 56 à 63**. Onde é possível observar na coluna de *Alternate Functions* onde a I2C está localizada.

PA8	MCO	SPI2_NSS/ I2S2_WS	TIM1_CH1	-	CRS1_SYNC	LPTIM2_OUT	-	EVENTOUT
PA9	MCO	USART1_TX	TIM1_CH2	-	SPI2_MISO/ I2S2_MCK	TIM15_BKIN	I2C1_SCL	EVENTOUT
PA10	SPI2_MOSI/ I2S2_SD	USART1_RX	TIM1_CH3	MCO2	-	TIM17_BKIN	I2C1_SDA	EVENTOUT
PA11	SPI1_MISO/ I2S1_MCK	USART1_CTS	TIM1_CH4	FDCAN1_RX	-	TIM1_BKIN2	I2C2_SCL	COMP1_OUT
PA12	SPI1_MOSI/ I2S1_SD	USART1_RTS _DE_CK	TIM1_ETR	FDCAN1_TX	-	I2S_CKIN	I2C2_SDA	COMP2_OUT

STM32G0B - Funções para I2C

Para **TRANSMITIR** um dado, utilizamos:

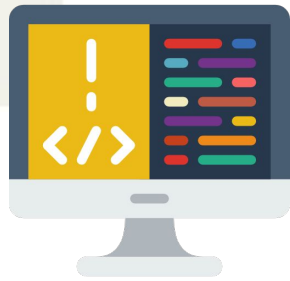
```
1    // Envia um array contendo os dados a serem enviados, onde hi2c refere
2    // ao handle da interface I2C, gerado pelo proprio CubeMX,
3    // onde DevAddress e o endereco do slave deslocado 1 bit a esquerda,
4    // pData e o ponteiro do vetor contendo os dados, Size e a
5    // quantidade de bytes de pData e Timeout e o tempo maximo para
     aguardar
6    // a conclusao da transmissao, em modo Master
7    HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress,
     uint8_t *pData, uint16_t Size, uint32_t Timeout);
```



STM32G0B - Funções para I2C

Para **RECEBER** um dado, utilizamos:

```
9   // Recebe um array contendo os dados a serem recebidos, onde hi2c
    refere
10  // ao handle da interface I2C, onde DevAddress e o endereco do slave
11  // deslocado 1 bit a esquerda, pData e o ponteiro do vetor onde sera
12  // armazenado os dados, Size e a quantidade de bytes a serem lidos e
13  // Timeout e o tempo maximo para aguardar a conclusao da transmissao,
14  // em modo Master
15  HAL_I2C_Master_Receive(I2C_HandleTypeDef *hi2c, uint16_t DevAddress,
    uint8_t *pData, uint16_t Size, uint32_t Timeout);
```



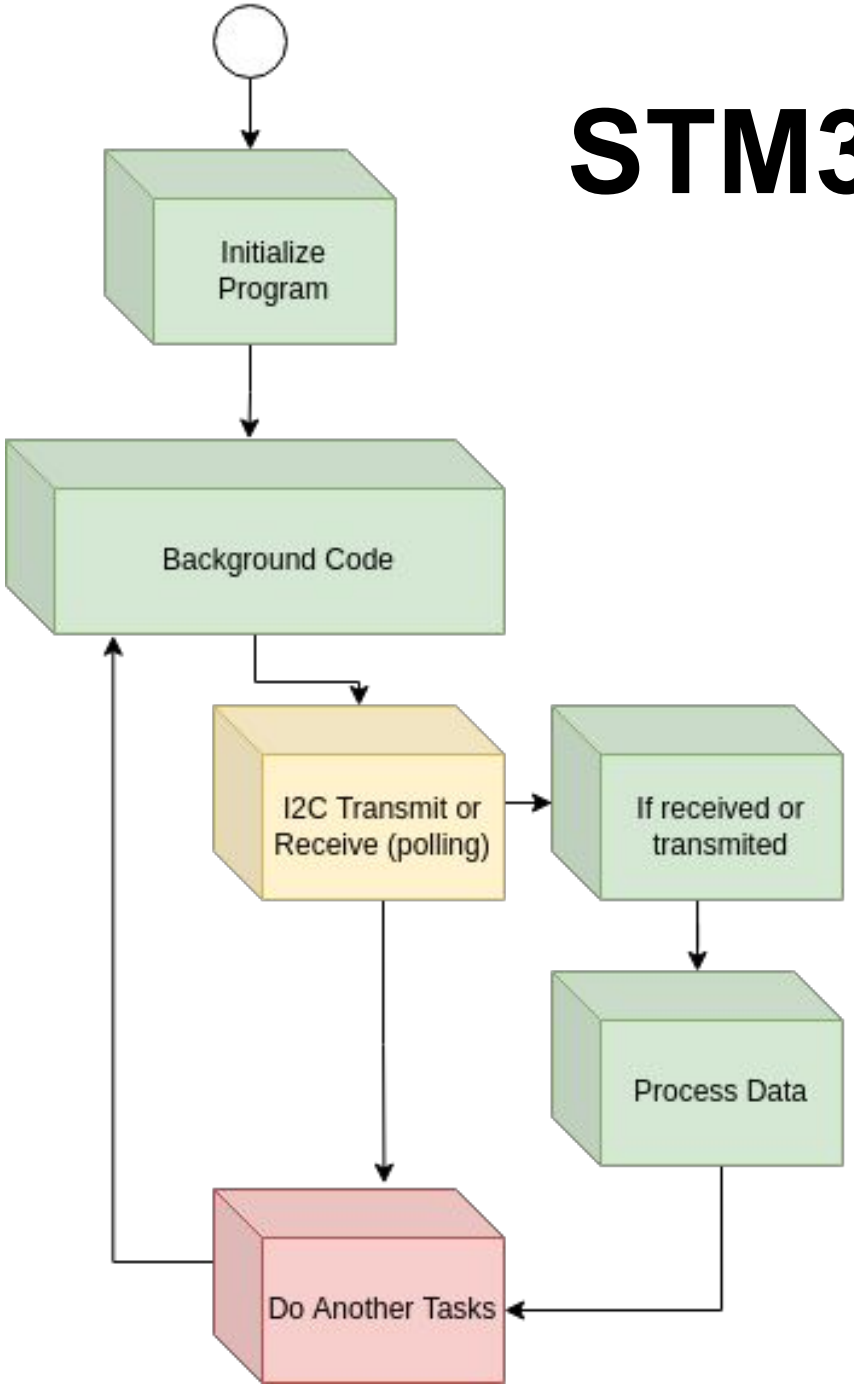
STM32G0B - Funções para SPI

Quando utilizamos a I2C em modo *Slave*

```
1   // Envia um array contendo os dados a serem enviados, onde hi2c refere
2   // ao handle da interface I2C, gerado pelo proprio CubeMX,
3   // onde 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, em modo Slave
6   HAL_I2C_Slave_Transmit(I2C_HandleTypeDef *hi2c, uint8_t *pData,
    uint16_t Size, uint32_t Timeout);
7
8   // Recebe um array contendo os dados a serem recebidos, onde hi2c
    refere
9   // ao handle da interface I2C, onde pData e o ponteiro do vetor onde
    sera
10  // armazenado os dados, Size e a quantidade de bytes a serem lidos e
11  // Timeout e o tempo maximo para aguardar a conclusao da recepcao,
12  // em modo Slave
13  HAL_I2C_Slave_Receive(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t
    Size, uint32_t Timeout);
```

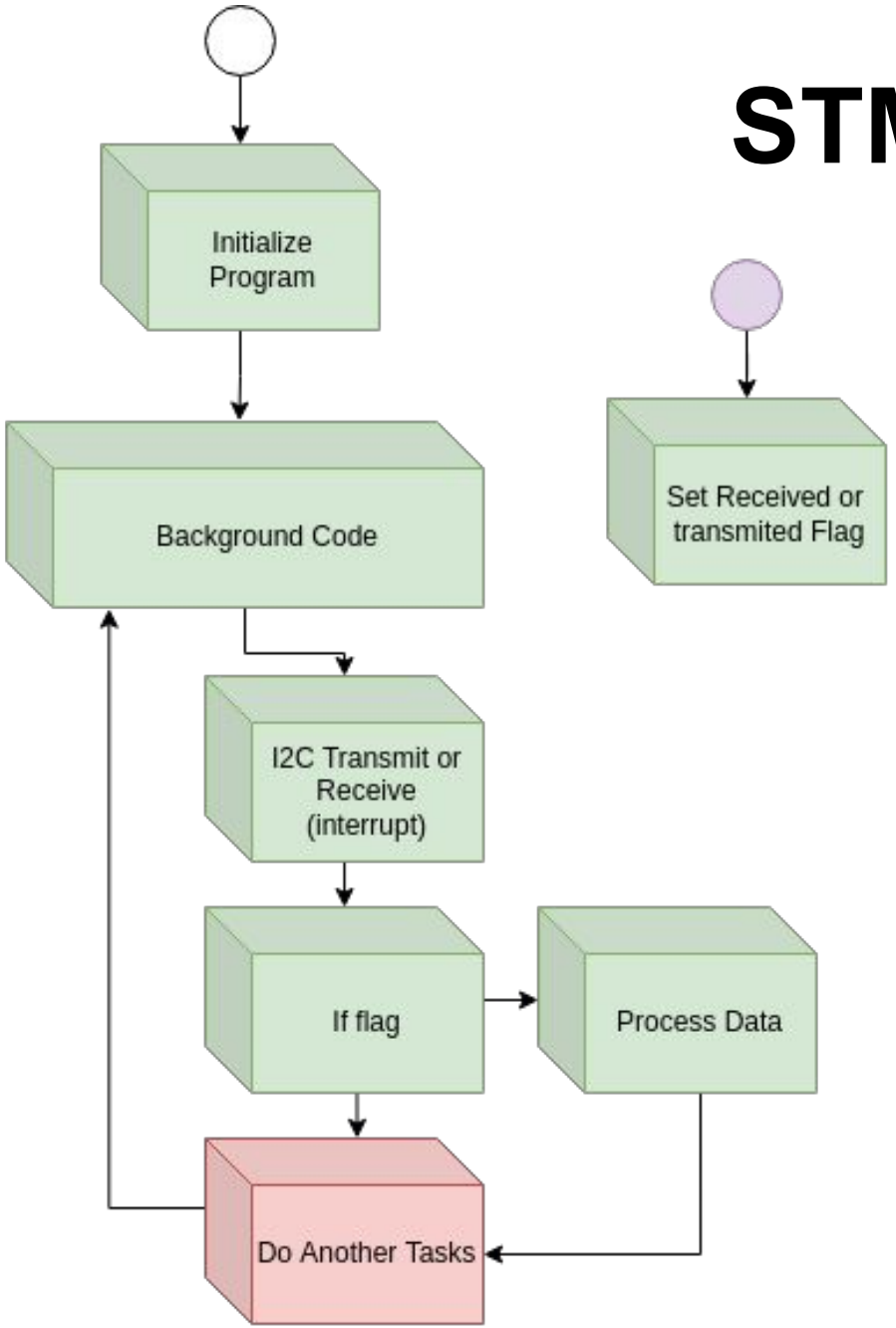
Déjà vu ????

STM32G0 - Interrupção

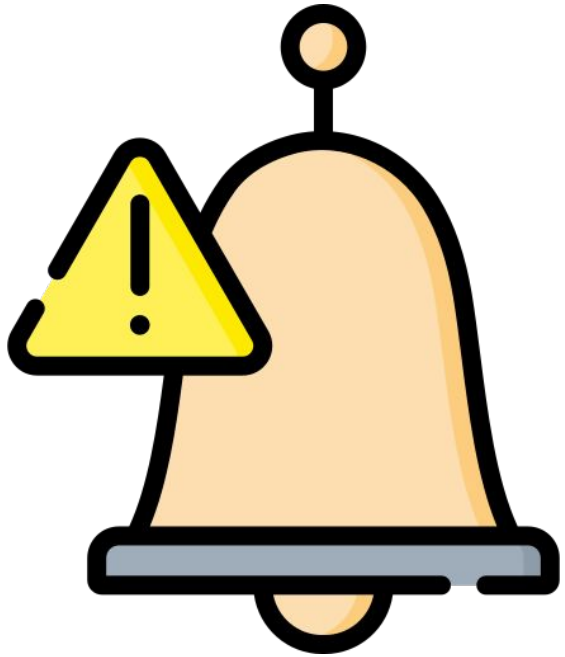


Como já vimos para todos os outros periféricos, falaremos, novamente e mais uma vez de interrupção

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, onde hi2c refere
2  // ao handle da interface I2C, gerado pelo proprio CubeMX,
3  // onde DevAddress e o endereco do slave deslocado 1 bit a esquerda,
4  // pData e o ponteiro do vetor contendo os dados, Size e a
5  // quantidade de bytes de pData, em modo de interrupcao, ao finalizar,
6  // e chamado um callback de transmissao
7  HAL_I2C_Master_Transmit_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress
   , uint8_t *pData, uint16_t Size);
```

STM32G0 - Funções para IT

```
9     // Recebe um array contendo os dados a serem recebidos, onde hi2c
      refere
10    // ao handle da interface I2C, onde DevAddress e o endereco do slave
11    // deslocado 1 bit a esquerda, pData e o ponteiro do vetor onde sera
12    // armazenado os dados, Size e a quantidade de bytes a serem lidos, e
      ao
13    // terminar a operacao, e chamado o callback de recepcao
14    HAL_I2C_Master_Receive_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress,
      uint8_t *pData, uint16_t Size);
```

STM32G0 - Funções para IT

Da mesma forma para o **Slave**

```
1    // Envia um array contendo os dados a serem enviados, onde hi2c refere
2    // ao handle da interface I2C, gerado pelo proprio CubeMX,
3    // onde pData e o ponteiro do vetor contendo os dados, Size e a
4    // quantidade de bytes de pData, em modo de interrupcao, chamando o
5    // callback respectivo ao final
6    HAL_I2C_Slave_Transmit_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData,
     uint16_t Size);
```

STM32G0 - Funções para IT

```
8   // Recebe um array contendo os dados a serem recebidos, onde hi2c
    refere
9   // ao handle da interface I2C, onde pData e o ponteiro do vetor onde
    sera
10  // armazenado os dados, Size e a quantidade de bytes,
11  // em modo de interrupcao, chamando o callback respectivo
12  HAL_I2C_Slave_Receive_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData,
    uint16_t Size);
```

STM32G0 - Funções de Callback

```
1  // Callback chamado quando uma operacao de HAL_I2C_Master_Transmit_IT
2  // e completada
3  void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c){
4
5  }
6
7  // Callback chamado quando uma operacao de HAL_I2C_Master_Receive_IT
8  // e finalizada
9  void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c){
```

STM32G0 - Funções de Callback


```
13    // Callback chamado quando uma operacao de HAL_I2C_Slave_Transmit_IT
14    // e completada
15    void HAL_I2C_SlaveTxCpltCallback(I2C_HandleTypeDef *hi2c){
16
17    }
18
19    // Callback chamado quando uma operacao de HAL_I2C_Slave_Receive_IT
20    // e finalizada
21    void HAL_I2C_SlaveRxCpltCallback(I2C_HandleTypeDef *hi2c){
22
23    }
24
```

STM32G0 - Funções de Callback

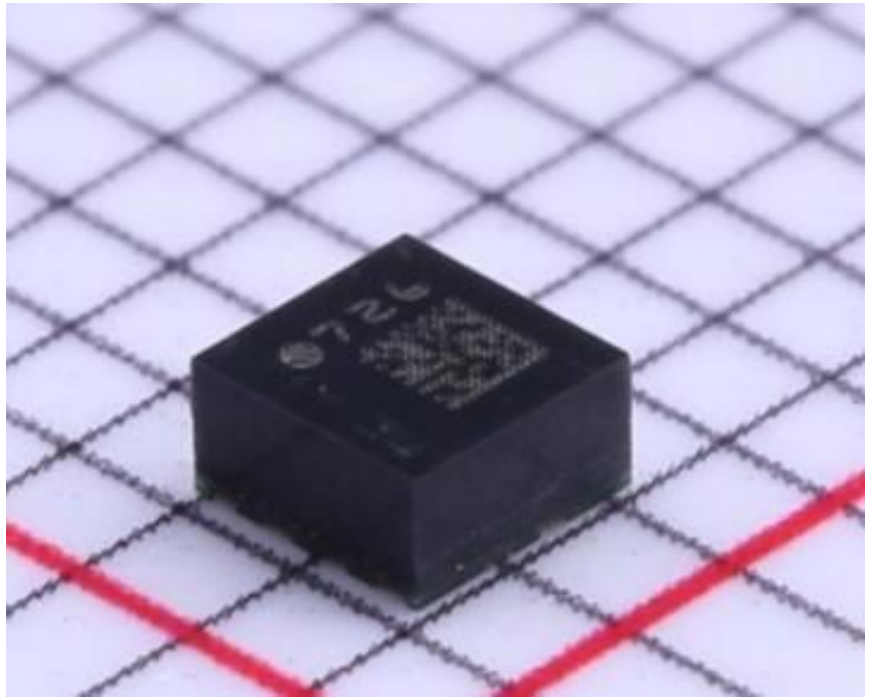
```
25    // Callback chamado quando um Address Frame e recebido pelo periferico
      e
26    // corresponde ao Address que foi definido como Slave
27    // O parametro TransferDirection indica qual a direcao dos dados,
      podendo
28    // assumir o valor de I2C_DIRECTION_RECEIVE ou I2C_DIRECTION_TRANSMIT
29    // AddrMatchCode retorna o endereco do Slave
30    void HAL_I2C_AddrCallback(I2C_HandleTypeDef *hi2c, uint8_t
      TransferDirection, uint16_t AddrMatchCode){
31
32    }
```

Déjà vu ????

STM32G0 - DMA

Assim como nos periféricos anteriores, podemos utilizar o DMA para realizar as operações da **I2C**.

Muito útil quando trabalhamos com **acelerômetros**, para fazer várias leituras de uma vez.



STM32G0 - Funções para DMA

```
1    // Envia um array contendo os dados a serem enviados, onde hi2c refere
2    // ao handle da interface I2C, gerado pelo proprio CubeMX,
3    // onde DevAddress e o endereco do slave deslocado 1 bit a esquerda,
4    // pData e o ponteiro do vetor contendo os dados, Size e a
5    // quantidade de bytes de pData, em modo de DMA, ao finalizar,
6    // e chamado um callback de transmissao
7    HAL_I2C_Master_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint16_t
     DevAddress, uint8_t *pData, uint16_t Size);
8
```

STM32G0 - Funções para DMA

```
9   // Recebe um array contendo os dados a serem recebidos, onde hi2c
    refere
10  // ao handle da interface I2C, onde DevAddress e o endereco do slave
11  // deslocado 1 bit a esquerda, pData e o ponteiro do vetor onde sera
12  // armazenado os dados, Size e a quantidade de bytes a serem lidos, e
    ao
13  // terminar a operacao, e chamado o callback de recepcao
14  HAL_I2C_Master_Receive_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress
    , uint8_t *pData, uint16_t Size);
```

STM32G0 - Funções para DMA

```
1    // Envia um array contendo os dados a serem enviados, onde hi2c refere
2    // ao handle da interface I2C, gerado pelo proprio CubeMX,
3    // onde pData e o ponteiro do vetor contendo os dados, Size e a
4    // quantidade de bytes de pData, em modo de DMA, chamando o
5    // callback respectivo ao final
6    HAL_I2C_Slave_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData,
     uint16_t Size);
```

STM32G0 - Funções para DMA

```
8   // Recebe um array contendo os dados a serem recebidos, onde hi2c
    refere
9   // ao handle da interface I2C, onde pData e o ponteiro do vetor onde
    sera
10  // armazenado os dados, Size e a quantidade de bytes,
11  // em modo de DMA, chamando o callback respectivo
12  HAL_I2C_Slave_Receive_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData,
    uint16_t Size);
```

STM32G0 - Callbacks para DMA

os callbacks para o I2C em modo DMA são os **mesmos** para a interrupção!

STM32G0 - Mais Funções

Para consultar mais funções implementadas pelo **HAL_I2C**, utilize a documentação **UM2319:Description of STM32G0 HAL and low-layer drivers**, nos capítulos:

- **27 HAL I2C Generic Driver**
- **28 HAL I2C Extension Driver**

Dúvidas ??

Referências

NXP SEMICONDUCTORS. **UM10204: I2C-bus specification and user manual**. 7.0. ed. [S.l.], 2021. Acesso em 28 de Dezembro de 2022.

SPARKFUN. **I2C**. 2022. https://learn.sparkfun.com/tutorials/i2c/all . Acesso em 28 de Fevereiro de 2022.

ST MICROELETRONICS. **M24C64-W M24C64-R M24C64-F M24C64-DF: 64-Kbit serial I2C bus EEPROM**. 36. ed. [S.l.], 2018.

__. **RM0444 - Reference Manual**. 5. ed. [S.l.], 2020. STM32G0x1 advanced Arm ®

-based 32-bit MCUs.

__. **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.

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

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

WIKIPEDIA. **I2C**. 2022. https://en.wikipedia.org/wiki/I%C2%B2C . Acesso em 28

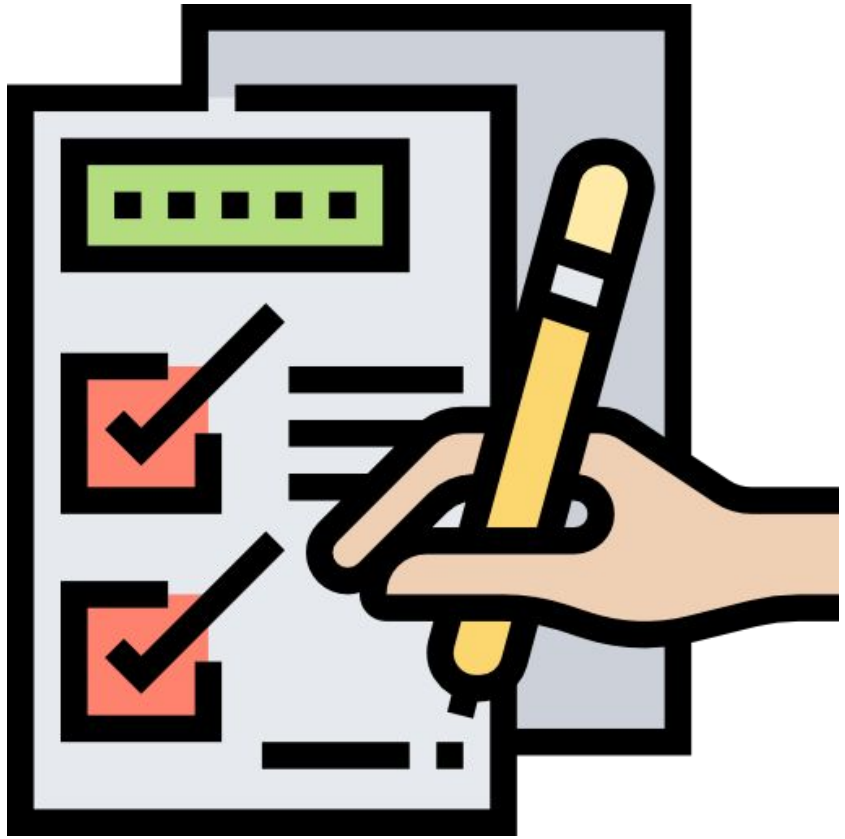
de Fevereiro de 2022.

WORLD, RF Wireless. **Advantages of I2C | disadvantages of I2C**. 2012. https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-I2C.html . Acesso em 28 de Fevereiro de 2022.

Mão na Massa

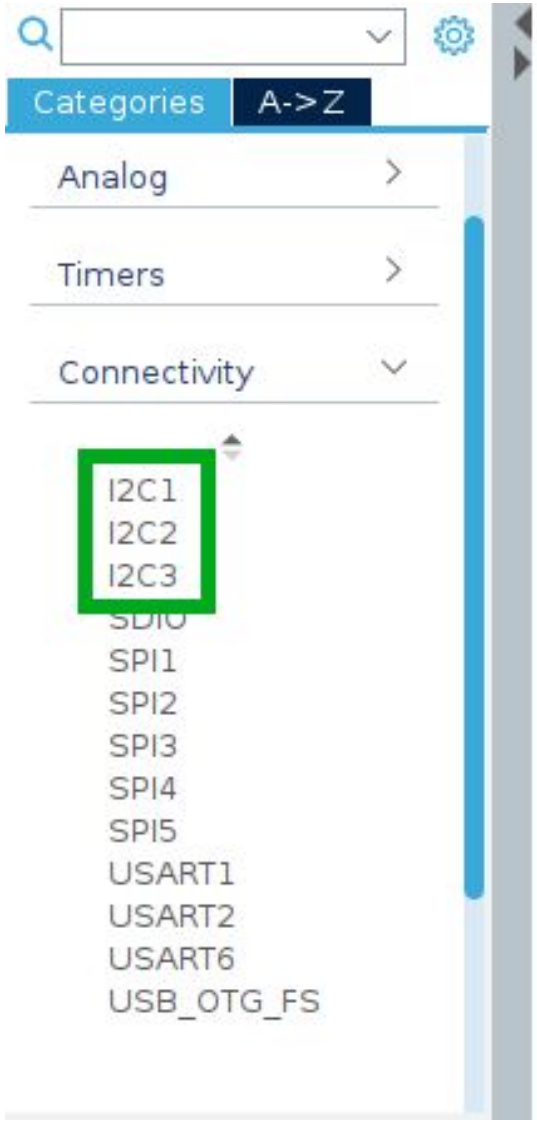
PADO Labs

Lista de Exercícios #7



Comunicação I2C

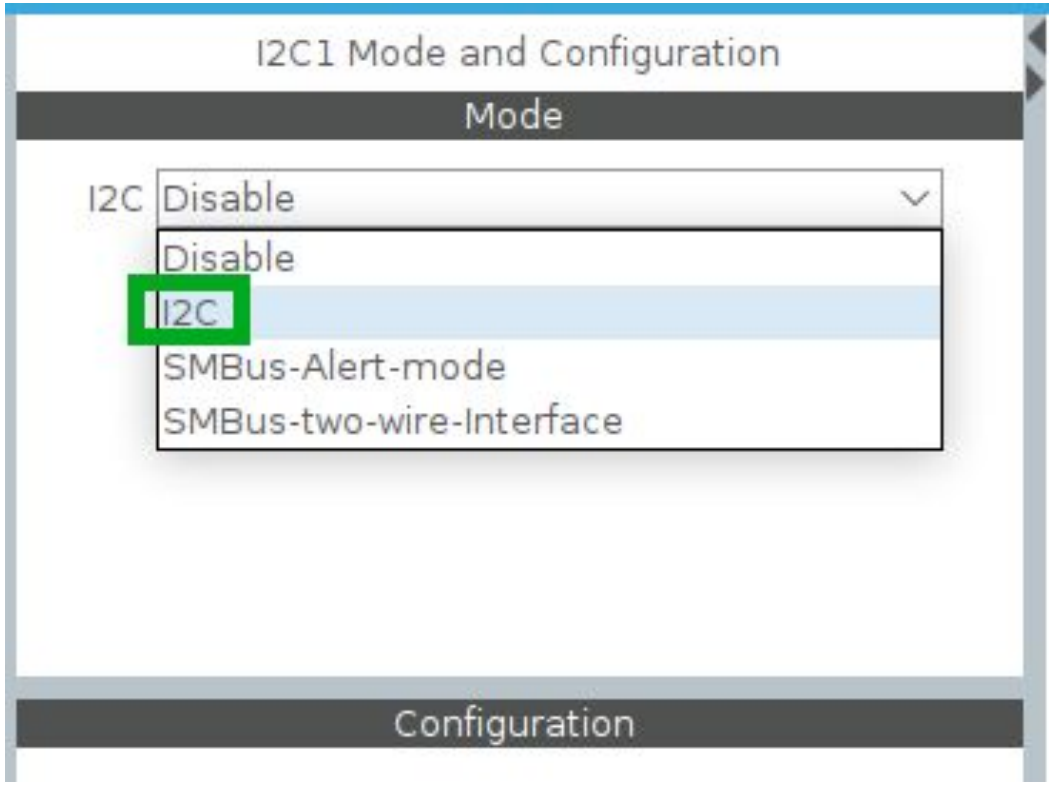
I2C em modo polling



Selecionamos o periférico de I2C na seção *Connectivity.*

É possível também escolher os terminais pelos pinos do microcontrolador, habilitando os para I2C ao clicar no terminal desejado.

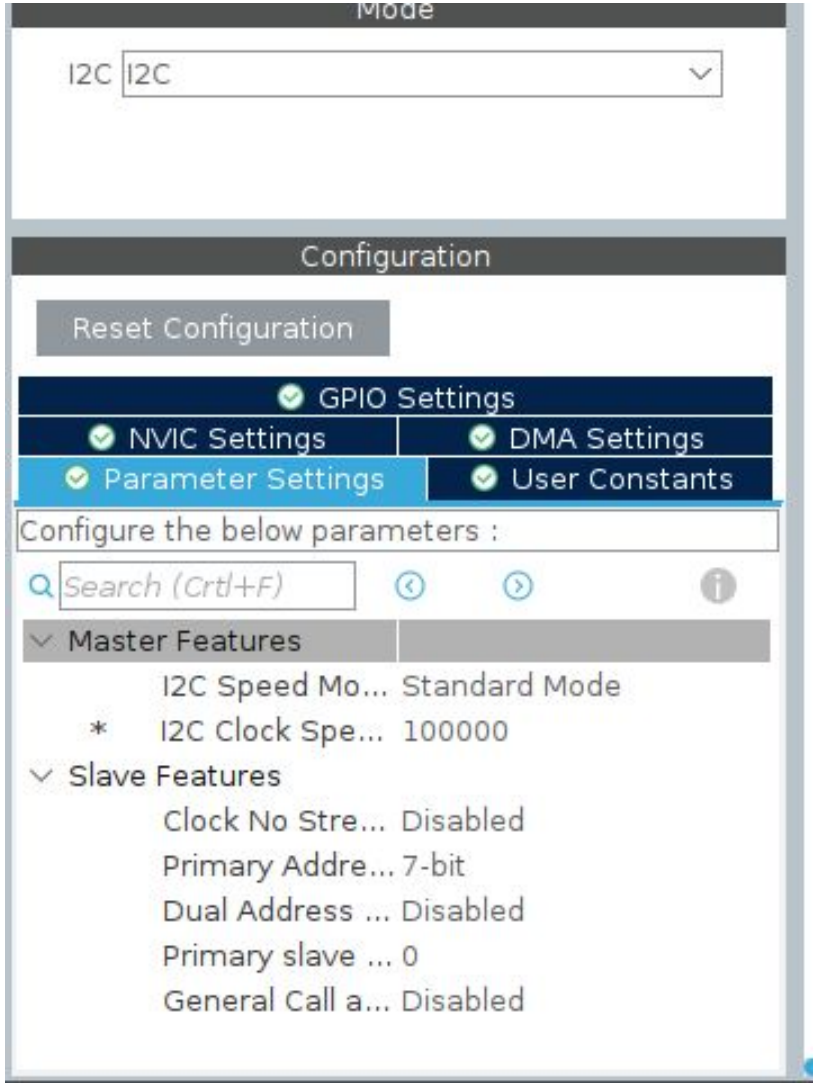
I2C em modo polling



Selecionamento então o modo **I2C**.

Diferente da SPI, aqui ja é habilitado tanto o modo **Master/Controller** quanto **Slave/Target**.

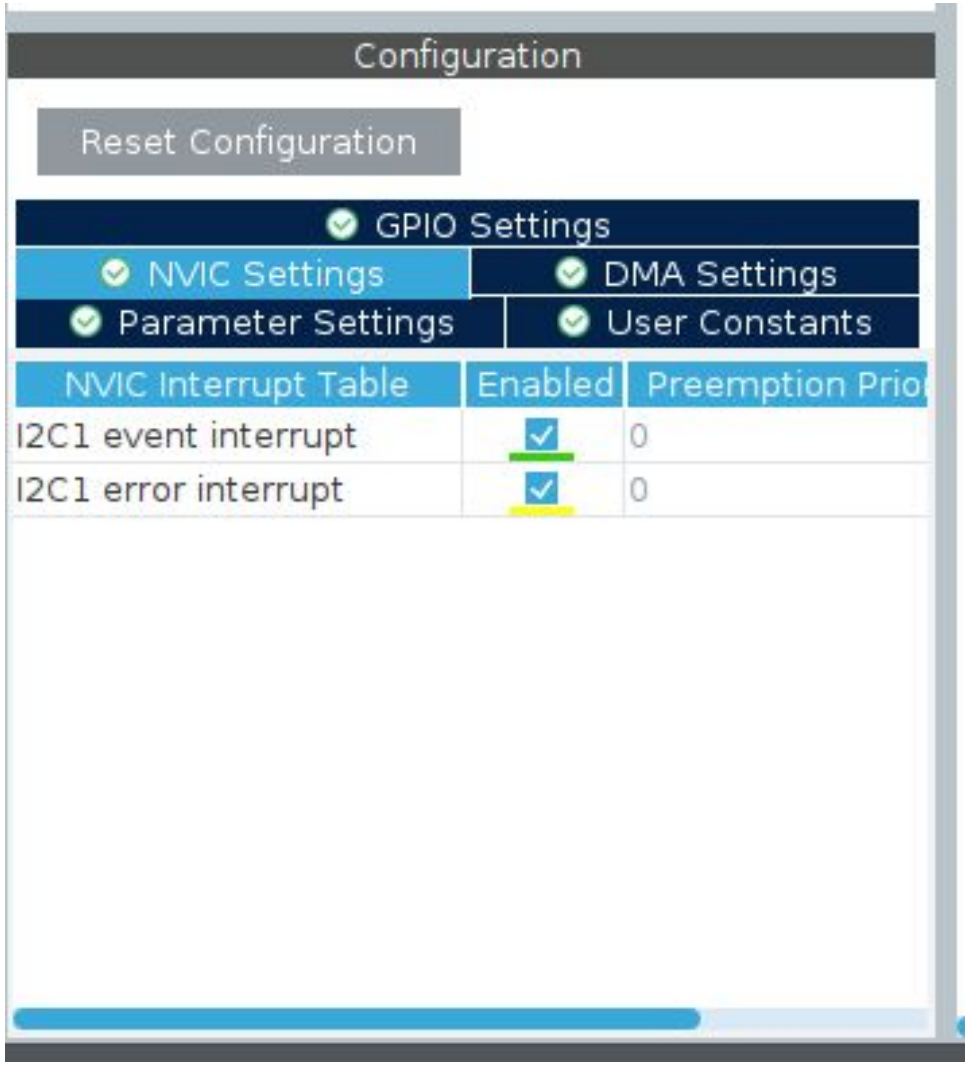
I2C em modo polling



Na tela ao lado temos as configurações para a interface I2C.

Em geral, apenas se define o *Primary Slave* quando se trabalha em modo *target*. Pode-se alterar também o **I2C Speed Mode** para *Fast Mode*.

I2C em Interrupção

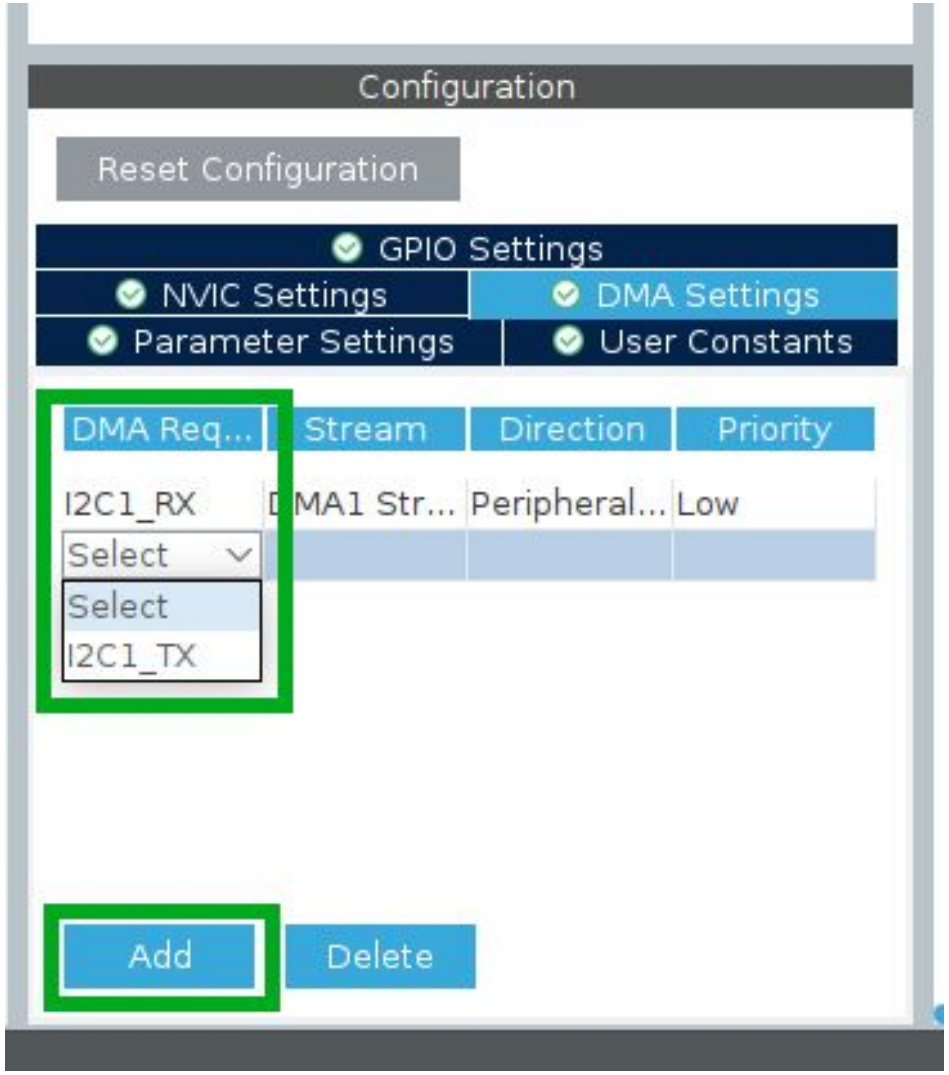


Após feitas as configurações anterior, vamos até a aba do *NVIC Settings*.

E habilitamos a interrupção global do periférico de **I2C.**

Se desejar, pode habilitar também a interrupção para a ocorrência de erros.

SPI em DMA

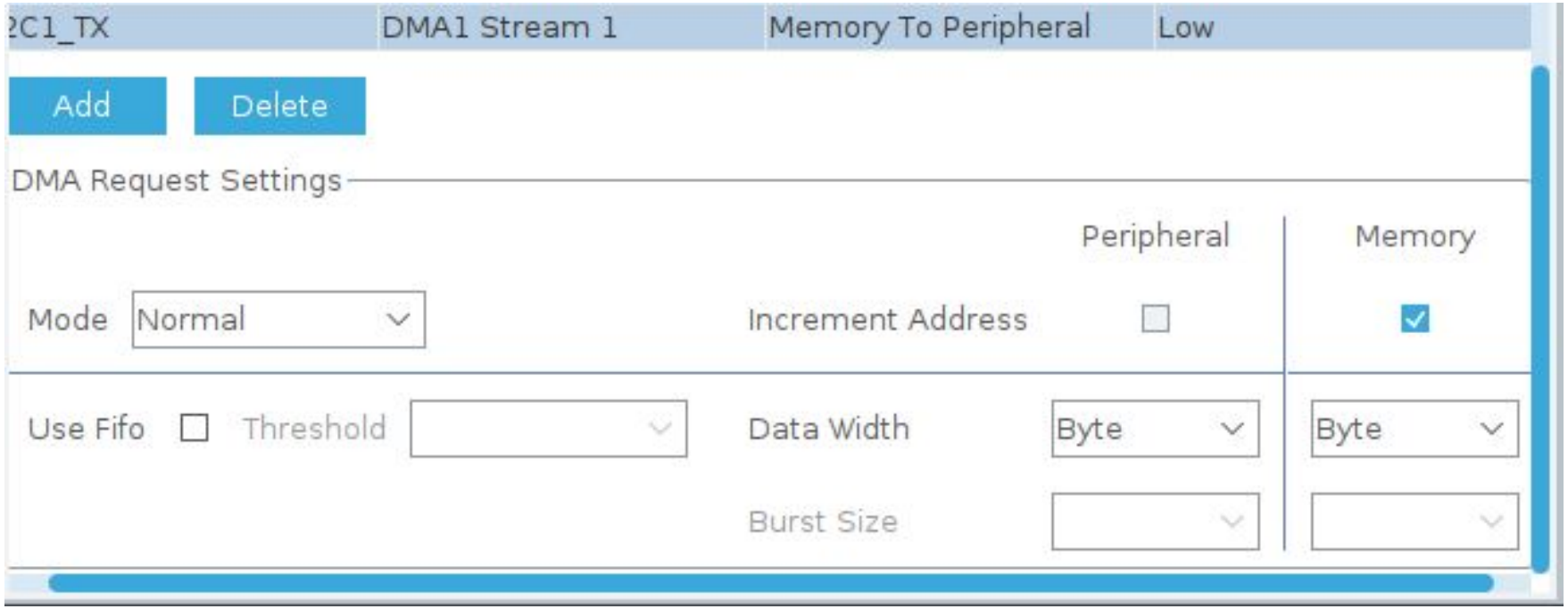


Vamos até a aba do DMA Settings.

Clicamos em Add e selecionamos os canais da I2C que serão utilizados, no exemplo ao lado, habilitou-se os dois canais da I2C.

UART em DMA

Configuramos os canais de acordo com a necessidade. É comum trabalhar com o Modo *Circular*.



PADO

Labs